29th Annual Conference on Neural Information Processing Systems (NIPS 2015)

Pre-Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches (CoCo @ NIPS 2015)

Tarek R. Besold, Artur d'Avila Garcez, Gary F. Marcus, and Risto Miikkulainen (eds.)

Montreal, Canada, 11th & 12th of December 2015

Invited Speakers of CoCo @ NIPS 2015

- Antoine Bordes, Facebook AI Research.
- Rina Dechter, UC Irvine.
- Pedro Domingos, University of Washington.
- Ramanathan V. Guha, Google Inc.
- Gary F. Marcus, NYU and Geometric Intelligence.
- Stephen H. Muggleton, Imperial College London.
- Daniel L. Silver, Acadia University.
- Paul Smolensky, Johns Hopkins University.
- Josh Tenenbaum, Massachusetts Institute of Technology.
- Greg Wayne, Deep Mind, Google Inc.
- Michael Witbrock, Cycorp Inc.

Contents: Contributed Papers

Probability Matching via Deterministic Neural Networks
(*Kharratzadeh & Shultz*)

The Usefulness of Past Knowledge when Learning a New Task in Deep Neural Networks
(*Montone, O'Regan & Terekhov*)

Symbol Grounding in Multimodal Sequences using Recurrent Neural Networks
(*Raue, Byeon, Breuel & Liwicki*)

Extracting Interpretable Models from Matrix Factorization Models
(*Sanchez Carmona & Riedel*)

Early Detection of Combustion Instability by Neural-Symbolic Analysis on Hi-Speed Video
(*Sarkar, Lore & Sarkar*)

Predicting Embedded Syntactic Structures from Natural Language Sentences with Neural Network Approaches
(*Senay, Zanzotto, Ferrone & Rigazio*)

Building Memory with Concept Learning Capabilities from Large-scale Knowledge Base
(*Shi & Zhu*)

Fractal grammars which recover from perturbation
(*Tabor*)

Analogy Making and Logical Inference on Images Using Cellular Automata-based Hyperdimensional Computing
(*Yilmaz*)

Turing Computation with Recurrent Artificial Neural Networks

Giovanni S. Carmantini
School of Computing and Mathematics
Plymouth University, Plymouth, United Kingdom
giovanni.carmantini@gmail.com

Peter beim Graben
Bernstein Center for Computational Neuroscience Berlin
Humboldt-Universität zu Berlin, Berlin, Germany
peter.beim.graben@hu-berlin.de

Mathieu Desroches
Inria Sophia-Antipolis Méditerranée
Valbonne, France
mathieu.desroches@inria.fr

Serafim Rodrigues
School of Computing and Mathematics
Plymouth University, Plymouth, United Kingdom
serafim.rodrigues@plymouth.ac.uk

Abstract

We improve the results by Siegelmann & Sontag [1, 2] by providing a novel and parsimonious constructive mapping between Turing Machines and Recurrent Artificial Neural Networks, based on recent developments of Nonlinear Dynamical Automata. The architecture of the resulting R-ANNs is simple and elegant, stemming from its transparent relation with the underlying NDAs. These characteristics yield promise for developments in machine learning methods and symbolic computation with continuous time dynamical systems. A framework is provided to directly program the R-ANNs from Turing Machine descriptions, in absence of network training. At the same time, the network can potentially be trained to perform algorithmic tasks, with exciting possibilities in the integration of approaches akin to Google DeepMind's Neural Turing Machines.

1 Introduction

The present work provides a novel and alternative approach to the one offered by Siegelmann and Sontag [1, 2] of mapping Turing machines to Recurrent Artificial Neural Networks (R-ANNs). Here we employ recent theoretical developments from symbolic dynamics enabling the mapping from Turing Machines to two-dimensional piecewise affine-linear systems evolving on the unit square, i.e. Nonlinear Dynamical Automata (NDA)[3, 4]. With this in place, we are able to map the resulting NDA onto a R-ANN, therefore providing an elegant constructive method to simulate a Turing machine in real time by a first-order R-ANN. There are two main advantages to the proposed approach. The first one is the parsimony and simplicity of the resulting R-ANN architecture in respect to previous approaches. The second one is the transparent relation between the network and its underlying piecewise affine-linear system. These two characteristics open the door to key future developments when considering learning applications (see Google DeepMind's Neural Turing Machines[5] for a relevant example with promising future integration possibilities) – with the exciting possibility of a symbolic read-out of a learned algorithm from the network weights – and when considering extensions of the model to continuous dynamics, which could provide a theoretical basis to query the computational power of more complex neuronal models.

2 Methods

In this section we outline a mapping from Turing machines to R-ANNs. Our construction involves two stages. In the first stage a Generalized Shift [3] emulating a Turing Machine is built, and its dynamics encoded on the unit square via a procedure called Gödelization, defining a piecewise-affine linear map on the unit square, i.e. a NDA. In the second stage, the resulting NDA is mapped onto a first-order R-ANN. Next, the theoretical methods employed are discussed in detail.

2.1 Turing Machines

A Turing Machine [6] is a computing device endowed with a doubly-infinite one-dimensional tape (memory support with one symbol capacity at each memory location), a finite state controller and a read-write head that follows the instructions encoded by a δ transition function. At each step of the computation, given the current state and the current symbol read by the read-write head, the machine controller determines via δ the writing of a symbol on the current memory location, a shift of the read-write head to the memory location to the left ($\mathcal{L}$) or to the right ($\mathcal{R}$) of the current one, and the transition to a new state for the next computation step. At a computation step, the content of the tape together with the position of the read-write head and the current controller state define a machine configuration.

More formally, a Turing Machine is a 7-tuple $M_{\text{TM}} = (Q, \mathbf{N}, \mathbf{T}, q_0, \sqcup, F, \delta)$, where Q is a finite set of control states, $\mathbf{N}$ is a finite set of tape symbols containing the blank symbol $\sqcup$, $\mathbf{T} \subset \mathbf{N} \setminus \{\sqcup\}$ is the input alphabet, q_0 is the starting state, $F \subset Q$ is a set of 'halting' states and δ is a partial transition function, determining the dynamics of the machine. In particular, δ is defined as follows:

$$\delta : Q \times \mathbf{N} \to Q \times \mathbf{N} \times \{\mathcal{L}, \mathcal{R}\}. \tag{1}$$

2.2 Dotted sequences and Generalized Shifts

A Turing machine configuration can be described by a bi-infinite dotted sequence on some alphabet $\mathbf{A}$; it can then be defined as:

$$s = \ldots d_{i_{-3}} d_{i_{-2}} d_{i_{-1}} . d_{i_0} d_{i_1} d_{i_2} \ldots, \tag{2}$$

where $l = \ldots d_{i_{-3}} d_{i_{-2}}$ describes the part of the tape on the left of the read-write head, $r = d_{i_0} d_{i_1} d_{i_2} \ldots$ describes the part on its right, $q = d_{i_{-1}}$ describes the current state of the machine controller, and the dot denotes the current position of the read-write head, i.e. the symbol to its right. The central dot splits the tape into two one-sided infinite strings α', β, where α' is the left part of the dotted sequence in reverse order. The first symbol in α represents the current state of the Turing Machine, whereas the first symbol in β represents the symbol currently under the controller's head. The transition function δ can be straightforwardly extended to a function $\hat{\delta}$ operating on dotted sequences, so that $\hat{\delta} : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$.

A Generalized Shift acts on dotted sequences, and is defined as a pair $M_{GS} = (\mathbf{A}^{\mathbb{Z}}, \Omega)$, with $\mathbf{A}^{\mathbb{Z}}$ being the space of dotted sequences, $\Omega : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{\mathbb{Z}}$ defined by

$$\Omega(s) = \sigma^{F(s)}(s \oplus G(s)) \tag{3}$$

with

$$F : \mathbf{A}^{\mathbb{Z}} \to \mathbb{Z} \tag{4}$$

$$G : \mathbf{A}^{\mathbb{Z}} \to \mathbf{A}^{e} \tag{5}$$

where σ shifts the symbols to the left or to the right, or does not shift them at all, as determined by the function $F(s)$. In addition, the Generalized Shift can operate a substitution, with $G(s)$ being the function which substitutes a substring of length e in the *Domain of Effect* (DoE) of s with a new substring. Both the shift and the substitution are functions of the content of the *Domain of Dependence* (DoD), a substring of s of length ℓ.

A Turing Machine can be emulated by a Generalized Shift with DoD = DoE = $d_{i_{-2}} d_{i_{-1}} . d_{i_0}$ and the functions F, G appropriately chosen such that $\Omega(s) = \hat{\delta}(s)$ for all s (see [7] for a detailed exposition).

2.3 Gödel codes

Gödel codes (or Gödelizations) [8] map strings to numbers and, in particular, allow the mapping of the space of one-sided infinite sequences to the real interval $[0, 1]$. Let $\mathbf{A}^{\mathbb{N}}$ be the space of one-sided infinite sequences over an alphabet $\mathbf{A}$, s be an element of $\mathbf{A}^{\mathbb{N}}$, r_k the k-th symbol in s, $\gamma : \mathbf{A} \to \mathbb{N}$ a one-to-one function associating each symbol in the alphabet $\mathbf{A}$ to a natural number, and g the number of symbols in $\mathbf{A}$. Then a Gödelization is a mapping ψ from $\mathbf{A}^{\mathbb{N}}$ to $[0, 1] \subset \mathbb{R}$ defined as:

$$\psi(s) := \sum_{k=1}^{\infty} \gamma(r_k) g^{-k}. \tag{6}$$

Conveniently, Gödelization can be employed on a Turing machine configuration, represented as a dotted sequence $\alpha.\beta \in \mathbf{A}^{\mathbb{Z}}$. The Gödel encoding ψ_x and ψ_y of α' and β define a representation of s $(\psi_x(\alpha'), \psi_y(\beta))$ known as symbol plane or symbologram representation, which is contained in the unit square $[0, 1]^2 \subset \mathbb{R}^2$. The choice of encoding ψ_x and ψ_y to use on the machine configurations is arbitrary. Therefore, to enable the construction of parsimonious Nonlinear Dynamical Automata our encoding will assume that β always contains tape symbols only, and that the first symbol of α' is always a state symbol, the rest being tape symbols only. Based on these assumptions, the particular encoding is defined as:

$$\begin{aligned} \psi_x(\alpha') &= \gamma_q(a_1) n_q^{-1} + \sum_{k=1}^{\infty} \gamma_s(a_{k+1}) n_s^{-k} n_q^{-1}, \\ \psi_y(\beta) &= \sum_{k=1}^{\infty} \gamma_s(b_k) n_s^{-k}, \end{aligned} \tag{7}$$

with $n_q = |Q|$, i.e. the number of states in the Turing Machine, $n_s = |\mathbf{N}|$, i.e. the number of tape symbols in the Turing Machine, γ_q and γ_s enumerating Q and $\mathbf{N}$ respectively, and with a_k and b_k being the k-th symbol in α' and β respectively.

2.3.1 Encoded Generalized Shift and affine-linear transformations

The substitution and shift operated by a Generalized Shift on a dotted sequence $s = \alpha.\beta$ can be represented as an affine-linear transformation on $(\psi_x(\alpha'), \psi_y(\beta))$, i.e. the symbologram representation of s. In particular, a substitution and shift on a dotted sequence can be broken down into substitutions and shifts on its one-sided components. In the following, we will show how substitutions and shifts on a one-sided infinite sequence can be represented as affine-linear transformations on its Gödelization. These results will be useful in showing how the symbologram representation of a Generalized Shift leads to a piecewise affine-linear map on a rectangular partition of the unit square. Let $s = d_1 d_2 d_3 \ldots$ be a one-side infinite sequence on some alphabet $\mathbf{A}$. Substituting the n-th symbol in s with $\hat{d}_n$ yields $\hat{s} = d_1 \ldots d_{n-1} \hat{d}_n d_{n+1} \ldots$, so that

$$\begin{aligned} \psi(s) &= \gamma(d_1) g^{-1} + \ldots \gamma(d_{n-1}) g^{-(n-1)} + \gamma(d_n) g^{-n} + \gamma(d_{n+1}) g^{n+1} + \ldots, \\ \psi(\hat{s}) &= \gamma(d_1) g^{-1} + \ldots \gamma(d_{n-1}) g^{-(n-1)} + \gamma(\hat{d}_n) g^{-n} + \gamma(d_{n+1}) g^{n+1} + \ldots, \\ &= \psi(s) - \gamma(d_n) g^{-n} + \gamma(\hat{d}_n) g^{-n}. \end{aligned}$$

As the previous example illustrates, Gödelizing a sequence resulting from a symbol substitution is equivalent to applying an affine-linear transformation on the original Gödelized sequence. In particular, the parameters of the affine-linear transformation only depend on the position and identities of the symbols involved in the substitution. Shifting s to the left by removing its first symbol or shifting it to the right by adding a new one yields respectively $s_l = d_2 d_3 d_4 \ldots$ and $s_r = b\ d_1 d_2 d_3 d_4 \ldots$, where b is the newly added symbol. In this case

$$\begin{aligned} \psi(s_l) &= \gamma(d_2) g^{-1} + \gamma(d_3) g^{-2} + \gamma(d_4) g^{-3} + \ldots \\ &= g\psi(s) - \gamma(d_1), \end{aligned}$$

and

$$\begin{aligned} \psi(s_r) &= \gamma(b) g^{-1} + \gamma(d_1) g^{-2} + \gamma(d_2) g^{-3} + \gamma(d_3) g^{-4} + \ldots \\ &= g^{-1}\psi(s) + \gamma(b) g^{-1}. \end{aligned}$$

Again, the resulting Gödelized shifted sequence can be obtained by applying an affine-linear transformation to the original Gödelized sequence.

2.4 Nonlinear Dynamical Automata

A Nonlinear Dynamical Automaton (NDA) is a triple $M_{NDA} = (X, P, \Phi)$, with P being a rectangular partition of the unit square, that is

$$P = \{D^{i,j} \subset X |\ 1 \leq i \leq m,\ 1 \leq j \leq n,\ m, n \in \mathbb{N}\}, \tag{8}$$

so that each cell $D^{i,j}$ is defined as the cartesian product $I_i \times J_j$, with $I_i, J_j \subset [0, 1]$ being real intervals for each bi-index (i, j), $D^{i,j} \cap D^{k,l} = \emptyset$ if $(i, j) \neq (k, l)$, and $\bigcup_{i,j} D^{i,j} = X$.
The couple (X, Φ) is a time-discrete dynamical system with phase space $X = [0, 1]^2 \subset \mathbb{R}^2$ (i.e. the unit square) and with flow $\Phi : X \to X$, a piecewise affine-linear map such that $\Phi_{|D^{i,j}} := \Phi^{i,j}$. Specifically, $\Phi^{i,j}$ takes the following form:

$$\Phi^{i,j}(\mathbf{x}) = \begin{pmatrix} a_x^{i,j} \\ a_y^{i,j} \end{pmatrix} + \begin{pmatrix} \lambda_x^{i,j} & 0 \\ 0 & \lambda_y^{i,j} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{9}$$

The piecewise affine-linear map Φ also requires a switching rule $\Theta(x, y) \in [\![1, m]\!] \times [\![1, n]\!]$ to select the appropriate branch, and thus the appropriate dynamics, as a function of the current state. That is, $\Phi(x, y) = \Phi^{i,j}(x, y) \iff \Theta(x, y) = (i, j)$.

Each cell $D^{i,j}$ of the partition P of the unit square can be seen as comprising all the Gödelized dotted sequences that contain the same symbols in the Domain of Dependence. That is, for a Generalized Shift simulating a Turing Machine, the first two symbols in α' and the first symbol in β.
The unit square is thus partitioned in a number of I intervals equal to $m = n_q n_s$, and one of J intervals equal to $n = n_s$, with n_q being the number of states in Q and n_s the number of symbols in $\mathbf{N}$, for a total of $n_q n_s^2$ cells. As each cell corresponds to a different Domain of Dependence of the underlying Generalized Shift in symbolic space, it is associated with a different affine-linear transformation representing the action of a substitution and shift in vector space. The transformation parameters $(a_x^{i,j}, a_y^{i,j})$ and $(\lambda_x^{i,j}, \lambda_y^{i,j})$ can be derived using the methods outlined in subsubsection 2.3.1.
Thus, a Turing Machine can be represented as a Nonlinear Dynamical Automaton by means of its Gödelized Generalized Shift representation.

3 NDAs to R-ANNs

The aim of the second stage of our methodology is to map the orbits of the NDA (i.e. $\Phi^{i,j}(x, y)$) to orbits of the R-ANN, which we will denote by $\zeta^{i,j}(x, y)$.

Let $\rho(\cdot)$ denote the proposed map. Its role is to encode the affine-linear dynamics at each $\Phi^{i,j}$ branch in the architecture and weights of the network, and emulate the overall dynamics Φ by suitably activating certain neural units within the R-ANN given the switching rule Θ. Therefore, we generically define the proposed map as follows:

$$\zeta = \rho(\mathcal{I}, \mathcal{A}, \Phi, \Theta), \tag{10}$$

where $\mathcal{I}$ is the identity matrix mapping (identically) the initial conditions of the NDA to the R-ANN and $\mathcal{A}$ is the adjacency matrix specifying the network architecture and weights, which will be explained in subsequent sections. In addition, ρ defines different neural dynamics for each type of the neural units, that is, $\zeta = (\zeta_1, \zeta_2, \zeta_3)$ corresponding to MCL, BSL and LTL, respectively (see below for the definitions of these acronyms). The details of the R-ANN architecture and its dynamics are subsequently discussed.

3.1 Network architecture and neural dynamics

The proposed map, ρ, attempts to mirror the affine-linear dynamics (given by Equation 9) of an NDA on the partitioned unit square (see Equation 8) by endowing the R-ANN with a structure capturing the characteristic features of a piecewise-affine linear system, i.e. a state, a switching rule and a set of transformations.

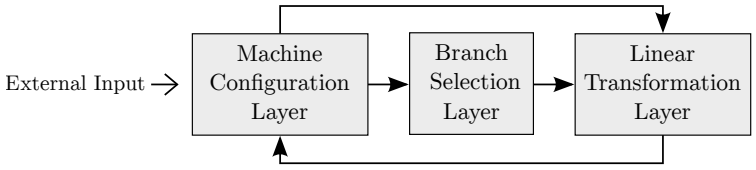


Figure 1. Connectivity between neural layers within the network.

To achieve this, we propose a network architecture with three layers, namely a Machine Configuration Layer (MCL) encoding the state, a Branch Selection Layer (BSL) implementing the switching rule and a Linear Transformation Layer (LTL), as depicted in Figure 1.
The neural units within the various layers make use of either the Heaviside (H) or the Ramp (R) activation functions defined as follows:

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \qquad (11) \qquad\qquad R(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}. \qquad (12)$$

Since Φ is a two-dimensional map, this suggests only two neural units (c_x, c_y) in the MCL layer encoding its state at every step. A set of BSL units functionally acts as a switching system that determines in which cell $D^{i,j}$ the current Turing machine configuration belongs to and then triggers the specific LTL unit emulating the application of an affine-linear transformation $\Phi^{i,j}$ on the current state of the system. The result of the transformation is then fed back to the MCL for the next iteration. On the symbolic level, one iteration of the emulated NDA corresponds to a tape and state update of the underlying Turing machine, which can be read out by decoding the activation of the MCL neurons.

3.1.1 Machine Configuration Layer

The role of the MCL is to store the current Gödelized configuration of the simulated Turing Machine at each computation step, and to synaptically transmit it to the BSL and LTL layers. The layer comprises two neural units (c_x and c_y), as needed to store the Gödelized dotted sequence representing a Turing Machine configuration (see Equation 7).

The R-ANNs is thus initialized by activating this layer, given the NDA initial conditions $(\psi_x(\alpha'), \psi_x(\beta))$ which are identically transformed via $\mathcal{I}$ by the map $\rho(\cdot)$ as follows:

$$(c_x, c_y) = (\psi_x(\alpha'), \psi_x(\beta)) \equiv \zeta_1 = \rho(\mathcal{I}, \cdot, \cdot, \cdot)_{|(\psi_x(\alpha'), \psi_x(\beta))} \qquad (13)$$

At each iteration, the units in this layer receive input from the LTL units, and are activated via the ramp activation function (Equation 12); in other words $\zeta_1 \equiv (c_x, c_y) = (R(\sum_i t_x^i), R(\sum_j t_y^j))$. Finally, the MCL synaptically projects onto the BSL and LTL (refer to Figure 2 for details of the connectivity).

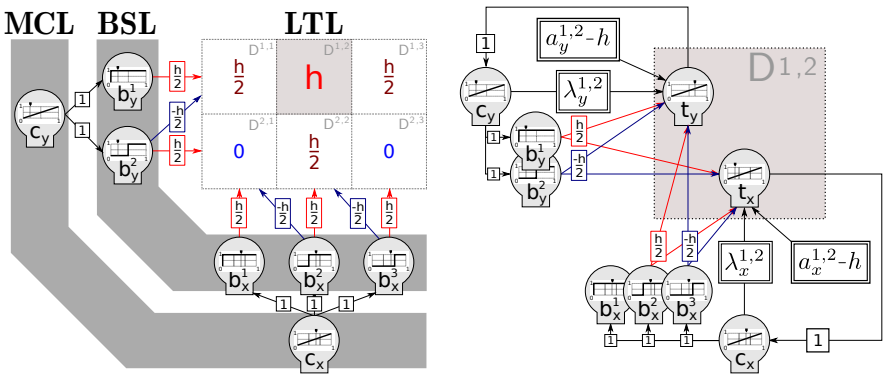
3.1.2 Branch Selection Layer

The BSL embodies the switching rule $\Theta(x, y)$ and coordinates the dynamic switching between LTL units. In particular, if at the current step the MCL activation is $(c_x, c_y) \in D^{i,j} = I_i \times J_j$, with $I_i = [\xi_i, \xi_{i+1})$ being the i-th interval on the x-axis and $J_j = [\eta_j, \eta_{j+1})$ being the j-th interval on the y-axis, the BSL units activate only the $(t_x^{i,j}, t_y^{i,j})$ units in the LTL. In this way, only one couple of LTL units is active at each step. The switching rule is mapped by $\rho(\cdot)$ as follows:

$$\zeta_2(x, y) = \rho(\cdot, \cdot, \cdot, \Theta(x, y) = (i, j)). \qquad (14)$$

The BSL is composed of two groups of Heaviside (Equation 11) units, implementing respectively the x and the y component of the switching rule of the underlying piecewise affine-linear system, namely: i) the b_x group receives input with weight 1 from the c_x unit of the MCL layer, and comprises $n_q n_s$ units (i.e. $b_x^i, 1 \leq i \leq n_q n_s$); ii) the b_y group receives input with weight 1 from c_y and comprises n_s units (i.e. $b_y^j, 1 \leq i \leq n_s$). The activation of the two groups of units is defined as:

$$\begin{aligned} b_x^i &= H(c_x - \xi^i) \qquad \text{with} \qquad \xi^i = \min(I_i), \\ b_y^j &= H(c_y - \eta^j) \qquad \text{with} \qquad \eta^j = \min(J_i). \end{aligned} \qquad (15)$$



(a) Branch Selection Layer (b) Complete branch connection layout

Figure 2. Detailed feedforward connectivity and weights for a neural network simulating a NDA with only 6 branches.

Each b_x^i and b_y^j BSL unit has an activation threshold, defined as the left boundary of the I_i and J_j intervals, respectively, and implemented as input from an always-active bias unit (with weight $-\xi^i$ for the b_x^i unit and $-\eta^j$ for b_y^j). Therefore, an activation of (c_x, c_y) in the MCL corresponding to a point on the unit square belonging to cell $D^{i,j}$, would trigger active all units b_x^k with $k \leq i$. The same would occur for all neural units b_y^k with $k \leq j$.[1]

Each b_x^i unit establishes synaptic excitatory connections (with weight $\frac{h}{2}$) to all LTL units corresponding to cells $D^{k,i}$ (i.e. $(t_x^{k,i}, t_y^{k,i})$) and inhibitory connections (with weight $\frac{-h}{2}$) to all LTL units corresponding to cells $D^{k,i-1}$ (i.e. $(t_x^{k,i-1}, t_y^{k,i-1})$), with $k = 1, \ldots, n_s$; for a graphical representation see Figure 2. Similarly, each b_y^j unit establishes synaptic excitatory connections to all LTL units corresponding to cells $D^{j,k}$ and inhibitory connections to all LTL units corresponding to cells $D^{j-1,k}$, with $k = 1, \ldots, n_q n_s$. Together, the b_x^i and b_y^j units completely counterbalance through their synaptic excitatory connections the natural inhibition (of bias h, which value and definition will be discussed in the following section) of the LTL units corresponding to cell $D^{i,j}$ (i.e. $(t_x^{i,j}, t_y^{i,j})$).

In other words each couple of LTL units $(t_x^{i,i}, t_y^{i,j})$ receives an input of $B_x^i + B_y^j$, defined as follows:

$$\begin{aligned} B_x^i &= b_x^i \frac{h}{2} + b_x^{i+1} \frac{-h}{2}, \\ B_y^j &= b_y^j \frac{h}{2} + b_y^{j+1} \frac{-h}{2}, \end{aligned} \tag{16}$$

where the input sum

$$B_x^i + B_y^j = \begin{cases} h & \text{if } (c_x, c_y) \in D_{i,j} \\ \frac{h}{2} & \text{if } c_x \in I_i, c_y \notin J_j \quad \text{or} \quad c_x \notin I_i, c_y \in J_j \\ 0 & \text{if } (c_x, c_y) \notin D_{i,j} \end{cases} \tag{17}$$

only triggers the relevant LTL unit if it reaches the value h. That is, if $(c_x, c_y) \in D_{i,j}$ then $B_x^i + B_y^j = h$, and the pair $(t_x^{i,i}, t_y^{i,j})$ is selected by the BSL units. Otherwise $(t_x^{i,i}, t_y^{i,j})$ stays inactive as $B_x^i + B_y^j$ is either equal to $\frac{h}{2}$ or 0, which is not enough to win the LTL pair natural inhibition. An example of this mechanism is shown in Figure 2 , where the LTL units in cell $D^{1,2}$ are activated via mediation of $b_x = \{b_x^1, b_x^2, b_x^3\}$ and $b_y = \{b_y^1, b_y^2\}$. Here, both b_x^3 and b_y^2 are not excited since c_x and c_y, respectively, are not activated enough to drive them towards their threshold. However, b_x^2 excites (with weights $\frac{h}{2}$) the LTL units in cell $D^{2,2}$ and $D^{1,2}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cell $D^{2,1}$ and $D^{1,1}$. Equally, b_y^2 excites (with weights $\frac{h}{2}$) the LTL units in cell $D^{2,1}$, $D^{2,2}$ and

[1]Note that the action of the BSL could be equivalently implemented by interval indicator functions represented as linear combinations of Heaviside functions.

$D^{2,3}$ and inhibits (with weights $\frac{-h}{2}$) the LTL units in cells $D^{1,1}$, $D^{1,2}$ and $D^{1,3}$. The b_x^1 and b_y^1 units excite cells $\{D^{2,1}, D^{1,1}\}$ and $\{D^{1,1}, D^{1,2}, D^{1,3}\}$, respectively, but these do not inhibit any cells (due to boundary conditions).

3.1.3 Linear Transformation Layer

The LTL layer can be functionally divided in sets of two units, where each couple applies two decoupled affine-linear transformations corresponding to one of the branches of the simulated NDA. On the symbolic level, this endows the LTL with the ability to generate an updated machine configuration from the previous one. In the LTL, a branch (i, j) of a NDA, $\Phi^{i,j}(x, y) = (\lambda_x^{i,j} x + a_x^{i,j}, \lambda_y^{i,j} y + a_y^{i,j})$, is simulated by the LTL units $(t_x^{i,j}, t_y^{i,j})$. Mathematically, this induces the following mapping:

$$(t_x^{i,j}, t_y^{i,j}) \quad = \quad \zeta_3^{i,j}(x, y) = \rho(\cdot, \cdot, \Phi^{i,j}(x, y), \cdot). \tag{18}$$

The affine-linear transformation is implemented synaptically, and it is only triggered when the BSL units provide enough excitation to enable $(t_x^{i,j}, t_y^{i,j})$ to cross their threshold value and execute the operation. The read-out of this process corresponds to:

$$\begin{array}{rcl} t_x^{i,j} & = & R(\lambda_x^{i,j} c_x + a_x^{i,j} - h + B_x^i + B_y^j), \\ t_y^{i,j} & = & R(\lambda_y^{i,j} c_y + a_y^{i,j} - h + B_x^i + B_y^j). \end{array} \tag{19}$$

A strong inhibition bias h (implemented as a synaptic projection from a bias unit) plays a key role in rendering the LTL units inactive in absence of sufficient excitation. The bias value is defined as follows

$$-\frac{h}{2} \leq -\max_{i,j,k}(a_k^{i,j} + \lambda_k^{i,j}) \quad \text{with} \quad k = \{x, y\}. \tag{20}$$

Hence, each of the BSL inputs B_x^i and B_y^i contributes respectively to half of the necessary excitation ($\frac{h}{2}$) needed to counterbalance the LTL's natural inhibition (refer to Equation 16 and Equation 17).

The LTL units receive input from the two CSL units (c_x, c_y), with synaptic weights of $(\lambda_x^{i,j}, \lambda_y^{i,j})$, and they are also endowed with an intrinsic constant LTL neural dynamics $(a_x^{i,j}, a_y^{i,j})$. If the input from the BSL layer is enough for these neurons to cross the threshold mediated by the Ramp activation function, the desired affine-linear transformation is applied. The read-out is an updated encoded Turing machine configuration, which is then synaptically fed back to the CSL units (c_x, c_y), ready for the next iteration (or next Turing machine computation step on the symbolic level).

3.1.4 NDA-simulating first order R-ANN

The NDA simulation (and thus Turing machine simulation) by the R-ANN is achieved by a combination of synaptic and neural computation among the three neural types (MCL, BSL, and LTL) and with a total of

$$n_{\text{units}} = \underbrace{2}_{\text{MCL}} + \underbrace{n_s + n_s n_q}_{\text{BSL}} + \underbrace{2n_s^2 n_q}_{\text{LTL}} + \underbrace{1}_{\text{bias unit}} \tag{21}$$

neural units, where n_q and n_s are the number of states and the number of symbols in the Turing Machine to be simulated, respectively. These units are connected as specified by an adjacency matrix $\mathcal{A}$ of size $n_{\text{units}} \times n_{\text{units}}$, following the connectivity pattern described in Figure 1 and with synaptic weights as entries from the set

$$\{0, 1, \frac{h}{2}, \frac{-h}{2}\} \cup \{a_k^{i,j} - h \mid i = 1, \ldots, n_q n_s \ , \ j = 1, \ldots, n_s \ , \ k = x, y\},$$

the second component being the set of biases.

An important modelling issue to consider is that of the halting conditions for the ANN, i.e. when to consider the computation completed. In the original formulation of the Generalized Shift, there is no explicit definition of halting condition. As our ANN model is based on this formulation, a deliberate choice has to be made in its implementation. Two choices seem to be the most reasonable. The first one involves the presence of an external controller halting the computation when some conditions are met, i.e. an *homunculus* [4]. The second one is the implementation of a fixed point condition,

intrinsic to the dynamical system, representing a TM halting state as an Identity branch on the NDA. In this way a halting configuration will result in a fixed point on the NDA, and thus on the R-ANN. In other words, the network's computation is considered completed if and only if

$$\zeta_1(x', y') = (x', y'). \tag{22}$$

In the present study we decided to use a fixed point halting condition, but the use of a *homunculus* would likely be more appropriate in other contexts such as interactive computation [9, 10, 11] or cognitive modelling, where different kinds of fixed points are required in order to describe sequential decision problems [12], such as linguistic garden paths [4, 10].

The implementation of the R-ANN defined like so simulates a NDA in real-time and, thus, it simulates a Turing Machine in real time. More formally, it can be shown that under the map $\rho(\cdot)$ the commutativity property $\zeta \circ \rho = \rho \circ \Phi$ is satisfied, which extends the previously demonstrated commutativity property between Turing machines and NDAs [9, 13, 14].

4 Discussion

In this study we described a novel approach to the mapping of Turing Machines to first-order R-ANNs. Interestingly, R-ANNs can be constructed to simulate any piecewise affine-linear system on a rectangular partition of the n-dimensional hypercube by extending the methods discussed

The proposed mapping allows the construction, given any Turing Machine, of a R-ANN simulating it in real time. As an example of the parsimony we claim, a Universal Turing Machine can be simulated with a fraction of the units than previous approaches allowed for: the proposed mapping solution derives a R-ANN that can simulate Minsky's 7-states 4-symbols UTM [15] in real-time with 259 units (as per Equation 21), approximately 1/3 of the 886 units needed in the solution proposed by Siegelmann and Sontag [1], and with a much simpler architecture.

In future work we plan to overcome some of the issues posed by the mapping and parts of its underlying theory, especially in relation to learning applications. Key issues to overcome are the missing end-to-end differentiability, and the need for a de-coupling of states and data in the encoding. A future development would see the integration of methods of data access and manipulation akin to that in Google DeepMind's Neural Turing Machines [5]. A parallel direction of future work would see the mapping of Turing machines to continuous-time dynamical systems (an example with polynomial systems is provided in [16]). In particular, heteroclinic dynamics [12, 13, 17, 18] – with machine configurations seen as metastable states of a dynamical system – and slow-fast dynamics [19, 20] are promising new directions of research.

References

[1] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[2] H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," *Appl. Math. Lett*, vol. 4, no. 6, pp. 77–80, 1991.

[3] C. Moore, "Unpredictability and undecidability in dynamical systems," *Physical Review Letters*, vol. 64, no. 20, p. 2354, 1990.

[4] P. beim Graben, B. Jurish, D. Saddy, and S. Frisch, "Language processing by dynamical systems," *International Journal of Bifurcation and Chaos*, vol. 14, no. 02, pp. 599–621, 2004.

[5] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[6] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proc. London Math. Soc*, vol. 42, 1937.

[7] C. Moore, "Generalized shifts: unpredictability and undecidability in dynamical systems," *Nonlinearity*, vol. 4, no. 2, p. 199, 1991.

[8] K. Gödel, "Über formal unentscheidbare sätze der *principia mathematica* und verwandter systeme i," *Monatshefte für Mathematik und Physik*, vol. 38, pp. 173 – 198, 1931.

[9] P. beim Graben, "Quantum Representation Theory for Nonlinear Dynamical Automata," in *Advances in Cognitive Neurodynamics ICCN 2007*, pp. 469–473, Springer, 2008.

[10] P. beim Graben, S. Gerth, and S. Vasishth, "Towards dynamical system models of language-related brain potentials," *Cognitive neurodynamics*, vol. 2, no. 3, pp. 229–255, 2008.

[11] P. Wegner, "Interactive foundations of computing," *Theoretical Computer Science*, vol. 192, pp. 315 – 351, 1998.

[12] M. I. Rabinovich, R. Huerta, P. Varona, and V. S. Afraimovich, "Transient cognitive dynamics, metastability, and decision making," *PLoS Computational Biology*, vol. 4, no. 5, p. e1000072, 2008.

[13] P. beim Graben and R. Potthast, "Inverse problems in dynamic cognitive modeling," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, no. 1, p. 015103, 2009.

[14] P. beim Graben and R. Potthast, "Universal neural field computation," in *Neural Fields*, pp. 299–318, Springer, 2014.

[15] M. Minsky, "Size and structure of universal turing machines using tag systems," in *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, vol. 5, pp. 229–238, 1962.

[16] D. S. Graça, M. L. Campagnolo, and J. Buescu, "Computability with polynomial differential equations," *Advances in Applied Mathematics*, vol. 40, no. 3, pp. 330–349, 2008.

[17] I. Tsuda, "Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems," *Behavioral and Brain Sciences*, vol. 24, pp. 793 – 810, 2001.

[18] M. Krupa, "Robust heteroclinic cycles," *Journal of Nonlinear Science*, vol. 7, no. 2, pp. 129–176, 1997.

[19] M. Desroches, M. Krupa, and S. Rodrigues, "Inflection, canards and excitability threshold in neuronal models," *Journal of mathematical biology*, vol. 67, no. 4, pp. 989–1017, 2013.

[20] M. Desroches, A. Guillamon, R. Prohens, E. Ponce, S. Rodrigues, and A. E. Teruel, "Canards, folded nodes and mixed-mode oscillations in piecewise-linear slow-fast systems," *SIAM Review*, vol. in press, 2015.

Lifted Relational Neural Networks

Gustav Šourek
Czech Technical University
Prague, Czech Republic
souregus@fel.cvut.cz

Vojtěch Aschenbrenner
Charles University
Prague, Czech Republic
v@asch.cz

Filip Železný
Czech Technical University
Prague, Czech Republic
zelezny@fel.cvut.cz

Ondřej Kuželka*
Cardiff University
Cardiff, United Kingdom
KuzelkaO@cardiff.ac.uk

Abstract

We propose a method combining relational-logic representations with neural network learning. A general lifted architecture, possibly reflecting some background domain knowledge, is described through relational rules which may be handcrafted or learned. The relational rule-set serves as a template for unfolding possibly deep neural networks whose structures also reflect the structures of given training or testing relational examples. Different networks corresponding to different examples share their weights, which co-evolve during training by stochastic gradient descend algorithm. Discovery of notable latent relational concepts and experiments on 78 relational learning benchmarks demonstrate favorable performance of the method.

1 Introduction

Lifted models also known as *templated* models have attracted significant attention recently [10] in areas such as statistical relational learning. Lifted models define patterns from which specific (ground) models can be unfolded. For example, a lifted Markov network model [18] may express that *friends of smokers tend to be smokers* and such a pattern then constrains the probabilistic relationships in all sets of vertices corresponding to particular friends-smokers in the derived ground Markov network. The lifted patterns are typically encoded in relational logic-based languages.

Here we contribute a method for (deep) lifted feed-forward neural network learning, in which the ground network structure is unfolded from a set of weighted rules in relational logic. The relational rules are instantly interpretable and can be handcrafted by a domain expert or learned, e.g. through techniques of Inductive Logic Programming (ILP) [5]. Weights of the ground neural networks are determined by the weighted relational rules and can be learned by stochastic gradient descend algorithm. This means that weights between different ground neurons constructed from the same relational rule are tied in our framework, similarly to how weights are shared in lifted graphical models in statistical relational learning or how weights are tied together by application of filters in convolutional neural networks in deep learning. A salient property of our approach distinguishing it from previous studies on adapting neural networks for relational learning is that the ground network structure depends not only on the relational rule set but also on a particular example, i.e., different networks are constructed for different examples to exploit their particular relational properties. However, the different networks share their weights as these are all bound to the relational rules, and so weight-updates performed for one training example are reflected in networks produced for other examples, which allows the model to learn directly from relational data.

*Corresponding author.

The main advantage of the presented approach is that it can effectively learn weights of latent non-ground relational structures, which is close in principle to predicate invention in ILP [5]. This is a difficult task for existing lifted systems based on probabilistic inference because there one typically needs to run expensive expectation maximization algorithms in order to learn parameters when latent structures are present. On the other hand, deep neural networks, which we exploit in our work, have been shown to effectively learn latent structures, although obviously only in the ground non-relational settings. By combining relational logic with deep neural networks, we obtain a framework flexible enough to learn weights of latent relational structures, which we also verify experimentally. While there have been several works combining propositional or relational logic with neural networks [21, 3, 6], none of the existing methods is able to learn weights of latent non-ground relational structures [1].

2 Lifted Relational Neural Networks

A lifted relational neural network (LRNN) $\mathcal{N}$ is a set of weighted definite clauses, i.e. pairs (R_i, w_i) where R_i is a function-free definite clause and w_i is a real number. When $\mathcal{N}$ is a set of weighted definite clauses, $\mathcal{N}^*$ will denote the corresponding set of the definite clauses without weights, i.e. $\mathcal{N}^* = \{C : (C, w) \in \mathcal{N}\}$. The set $\mathcal{N}$ must satisfy the following *non-recursiveness*[2] requirement: there must exist a strict ordering $\prec$ of predicates such that if there is a rule with a predicate p_1 in the head and a predicate p_2 in the body then $p_1 \prec p_2$.

Given a LRNN $\mathcal{N}$, let $\mathcal{H}$ be the least Herbrand model of $\mathcal{N}^*$. We define *grounding of the LRNN* $\mathcal{N}$ as $\overline{\mathcal{N}} = \{(h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta, w) : (h \leftarrow b_1 \wedge \cdots \wedge b_k, w) \in \mathcal{N}$ and $\{h\theta, b_1\theta, \ldots, b_k\theta\} \subseteq \mathcal{H}\}$. That is, $\overline{\mathcal{N}}$ is defined as the set of ground definite clauses which can be obtained by grounding rules from the LRNN and which are active in the least Herbrand model of $\mathcal{N}^*$ (a rule is active in $\mathcal{H}$ if its body is true in $\mathcal{H}$). As already outlined in Introduction, LRNNs are templates for creating *ground* neural networks. The requirement that ground rules should be active in $\mathcal{H}$ is beneficial for practice because it provides us with flexibility in controlling complexity and tractability of the constructed ground neural networks.

Example 1. Let

$$\begin{aligned}\mathcal{N} =\{&(\mathit{mother}(C, M) \leftarrow \mathit{parent}(C, M) \wedge \mathit{female}(M), 1),\\ &(\mathit{father}(C, F) \leftarrow \mathit{parent}(C, F) \wedge \mathit{male}(F), 2),\\ &(\mathit{female}(\mathit{alice}), 1), (\mathit{parent}(\mathit{bob}, \mathit{alice}), 1), (parent(\mathit{eve}, \mathit{alice}), 1)\}.\end{aligned}$$

Then for its grounding we have

$$\begin{aligned}\overline{\mathcal{N}} =\{&(\mathit{mother}(\mathit{bob}, \mathit{alice}) \leftarrow \mathit{parent}(\mathit{bob}, \mathit{alice}) \wedge \mathit{female}(\mathit{alice}), 1),\\ &(\mathit{mother}(\mathit{eve}, \mathit{alice}) \leftarrow \mathit{parent}(\mathit{eve}, \mathit{alice}) \wedge \mathit{female}(\mathit{alice}), 1),\\ &(\mathit{female}(\mathit{alice}), 1), (\mathit{parent}(\mathit{bob}, \mathit{alice}), 1), (\mathit{parent}(\mathit{eve}, \mathit{alice}), 1)\}.\end{aligned}$$

Notice that $\overline{\mathcal{N}}$ does not contain the predicates $\mathit{male}/1$ or $\mathit{father}/2$ as there are no ground atoms based on them in the least Herbrand model of $\mathcal{N}$.

Definition 1. *Let $\mathcal{N}$ be a LRNN, and let $\overline{\mathcal{N}}$ be its grounding. Let $g_\vee$, $g_\wedge$ and $g^*_\wedge$ be families of multivariate functions with exactly one function for each number of arguments. The* ground neural network *of $\mathcal{N}$ is a feedforward neural network constructed as follows.*

- *For every ground atom h occurring in $\overline{\mathcal{N}}$, there is a neuron A_h, called* atom neuron*. The activation functions of atom neurons are from the family $g_\vee$.*
- *For every ground fact $(h, w) \in \overline{\mathcal{N}}$, there is a neuron $F_{(h,w)}$, called* fact neuron*, which has no input and always outputs a constant value.*

[1] Exemplification of latent non-ground relational concept learning and a more detailed description of Lifted Relational Neural Networks can be found in [20].

[2] The reason why we do not allow recursion will be clearer when we explain weight learning in the next section. Here, we just note that rule sets without recursion will allow us to directly exploit gradient descent training of feed-forward neural networks.

- *For every ground rule* $h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta \in \overline{\mathcal{N}}^*$*, there is a neuron* $R_{h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta}$*, called* rule neuron*. It has the atom neurons* $A_{b_1\theta}, \ldots, A_{b_k\theta}$ *as inputs, all with weight* 1*. The activation functions of rule neurons are from the family* $g_\wedge$*.*
- *For every rule* $(h \leftarrow b_1 \wedge \cdots \wedge b_k, w) \in \mathcal{N}$ *and every* $h\theta \in \mathcal{H}$*, there is a neuron* $Agg^{h\theta}_{(h \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$*, called* aggregation neuron*. Its inputs are all rule neurons* $R_{h\theta' \leftarrow b_1\theta' \wedge \cdots \wedge b_k\theta'}$ *where* $h\theta = h\theta'$ *with all weights equal to* 1*. The activation functions of the aggregation neurons are from the family* $g^*_\wedge$*.*
- *Inputs of an atom neuron* $A_{h\theta}$ *are the aggregation neurons* $Agg^{h\theta}_{(h \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$ *and fact neurons* $F_{(h\theta, w)}$*. The weights of the input neurons are the respective* w*'s.*

Example 2. Let us consider the following LRNN

$$\mathcal{N} = \{(\mathit{foal}(A) \leftarrow \mathit{parent}(A,P) \wedge \mathit{horse}(P), w_m), (\mathit{foal}(A) \leftarrow \mathit{sibling}(A,S) \wedge \mathit{horse}(S), w_n),$$
$$(\mathit{horse}(\mathit{dakotta}), w_1), (\mathit{horse}(\mathit{cheyenne}), w_2), (\mathit{horse}(\mathit{aida}), w_3),$$
$$(\mathit{parent}(\mathit{star}, \mathit{aida}), w_6), (\mathit{parent}(\mathit{star}, \mathit{cheyenne}), w_5), (\mathit{sibling}(\mathit{star}, \mathit{dakotta}), w_4)\}.$$

The LRNN $\mathcal{N}$ and its ground neural network are shown in Fig. 1.

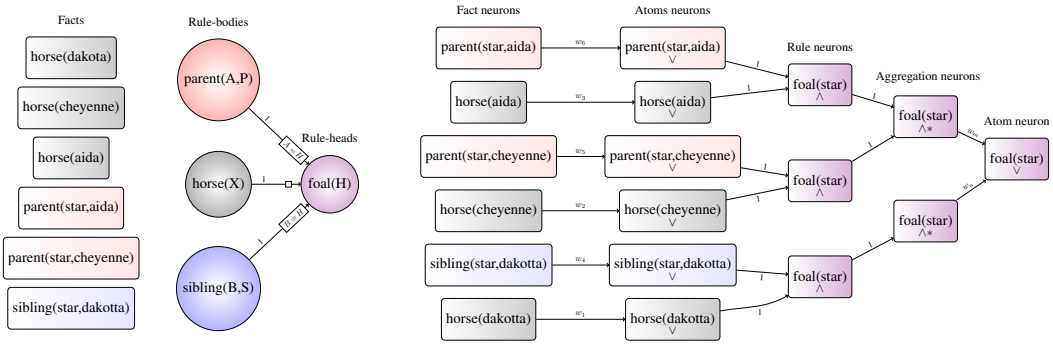


Figure 1: Depiction of the rule-based template (left) of LRNN $\mathcal{N}$ from Ex. 2, and its corresponding ground neural network $\overline{\mathcal{N}}$ (right), with colors denoting the predicate signatures, rectangular nodes corresponding to ground and circular to lifted literals, respectively.

What distinguishes LRNNs from ordinary neural networks the most is the following property. Having a pre-trained LRNN $\mathcal{N}$ described by some general rules, we can extend it with description of a particular case to obtain a ground neural network and then use the latter for prediction. This is similar in spirit to lifted graphical models.

Example 3. For instance, $\mathcal{N}$ may describe general rules for explosiveness of molecules (e.g., represented by a predicate *explosive*) and $\mathcal{M}_1$ and $\mathcal{M}_2$ may be sets of (weighted) facts describing two particular molecules. Then to use the LRNN $\mathcal{N}$ for predicting whether $\mathcal{M}_1$ and $\mathcal{M}_2$ are explosive, we can simply construct ground NNs of $\overline{\mathcal{N} \cup \mathcal{M}_1}$ and $\overline{\mathcal{N} \cup \mathcal{M}_2}$, and compute the output of the respective atom neurons $\mathit{explosive}^1 \in \overline{\mathcal{N} \cup \mathcal{M}_1}$ and $\mathit{explosive}^2 \in \overline{\mathcal{N} \cup \mathcal{M}_2}$. As a distinctive feature of lifted models, the two ground LRNNs for the two example molecules may have very different size and structure because the least Herbrand models of $\mathcal{N}^* \cup \mathcal{M}_1^*$ and of $\mathcal{N}^* \cup \mathcal{M}_2^*$, which determine the structures of the ground LRNNs, may be very different (because the structure and the size of the molecules described by $\mathcal{M}_1$ and $\mathcal{M}_2$ are different). An illustration of this effect, for two example molecules and a template $\mathcal{N}$ from Fig. 2, is displayed in Fig. 3.

Depending on the used families of activation functions $g_\vee$, $g_\wedge$ and $g^*_\wedge$, we can obtain neural networks with different behavior. For intuitiveness, in order for rules $(h \leftarrow b_1 \wedge \cdots \wedge b_k, w)$ to behave similarly to "if-then" rules, we should prefer the outputs of rule neurons to be *high* (e.g. close to 1) if and only if all the inputs from the atom neurons corresponding to the literals from the body of the rule have high outputs. Similarly, we should prefer the output of the atom neurons, which should intuitively behave similarly to disjunction, to be high if and only if at least one of the rule neurons or fact neurons, which are inputs for the given atom neuron, has high output. Logical operators from various fuzzy logics [11] may serve as an inspiration for selecting suitable activation functions.

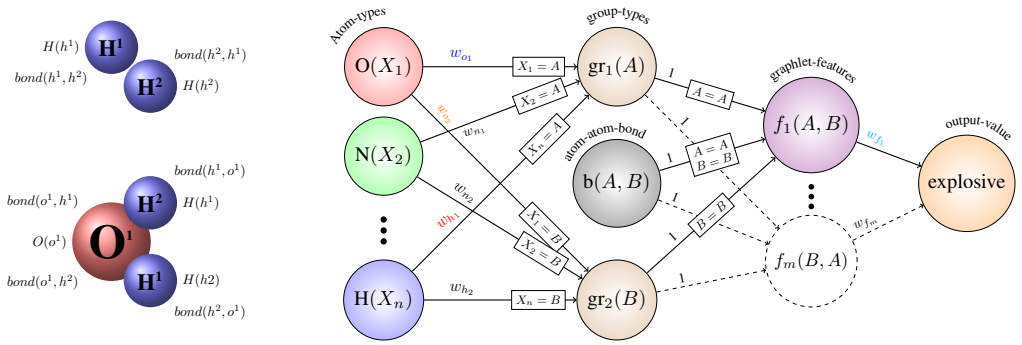


Figure 2: Two example molecules (left), described by surrounding sets of ground facts $\mathcal{M}_1$ and $\mathcal{M}_2$, are being merged with the lifted LRNN $\mathcal{N}$, composed of general weighted rules loosely pointing to explosiveness of molecules (right), to form two ground networks displayed in Fig. 3. The rules in $\mathcal{N}$ provide adaptive means to create latent groups (gr_i) of atom types ($O \dots H$) that, through a bond predicate ($b(A, B)$) connecting couples of atoms, form relational features (e.g., $f_1(A, B) \leftarrow gr_1(A) \wedge bond(A, B) \wedge gr_2(B)$), which set the basis for the final explosiveness output. For the sake of space we assume a single relational (graphlet) feature f_1 only.

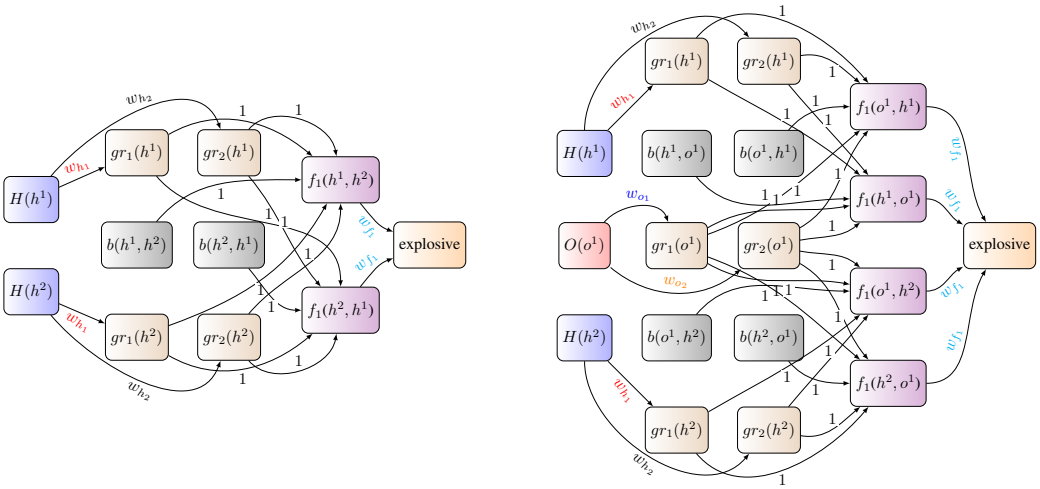


Figure 3: Two groundings $\overline{\mathcal{N} \cup \mathcal{M}_1}$ and $\overline{\mathcal{N} \cup \mathcal{M}_2}$ formed by merging the two example molecules with the LRNN $\mathcal{N}$ from Fig. 2. The shared predicate signatures and weights tied by the template are denoted by colors. For the sake of space we display only ground rule sets instead of complete ground networks (i.e., fact and aggregation neurons are omitted), Fig. 1 illustrates the (direct) correspondence of such a set to a full ground neural network.

Example 4. In Goedel fuzzy logic, conjunction $b_1 \wedge \dots \wedge b_k$, where b_i are fuzzy logic literals, is given as $\min_i b_i$ and disjunction $b_1 \vee \dots \vee b_k$ is given as $\max_i b_i$. To emulate reasoning in Goedel logic, we could simply set $g_\wedge(b_1, \dots, b_k) = \min_i b_i$, $g^*_\wedge(b_1, \dots, b_m) = \max_i b_i$, and $g_\vee(b_1, \dots, b_m) = \max_i b_i$. Here, the output of any rule neuron $R_{h \leftarrow b_1 \wedge \dots \wedge b_k}$ is the minimum value which makes the fuzzy truth value of the implication $h \leftarrow b_1 \wedge \dots \wedge b_k$ equal to 1 in the Goedel fuzzy logic. Likewise, the output of any aggregation neuron is the minimum value which makes the fuzzy truth value of all the respective ground implications equal to 1 simultaneously. This way, LRNNs can emulate fuzzy logic programming.

Next, we introduce two particular collections of activation functions inspired by fuzzy logic which will be used in the experiments (note that the activation functions shown in the above example would not be very suitable for gradient-based learning).

Definition 2 (Max-Sigmoid Activation Functions)**.** *The Max-Sigmoid (MS) collection of activation functions is composed of the following three families of functions:* $g_\wedge(b_1,\ldots,b_k) = \mathit{sigm}\left(\sum_{i=1}^{k} b_i - k + b_0\right)$, $g^*_\wedge(b_1,\ldots,b_m) = \max_i b_i$, *and* $g_\vee(b_1,\ldots,b_k) = \mathit{sigm}\left(\sum_{i=1}^{k} b_i + b_0\right)$.

The rationale for this family of activation functions is as follows. As already mentioned, the activation function $g_\wedge$ should have high output if and only if all its inputs are high. To achieve this, we can crudely approximate Lukasiewicz fuzzy conjunction, which is given as $\max\{0, b_1+\cdots+b_k-k+1\}$, by the function $\mathit{sigm}\,(b_1+\cdots+b_k-k+b_0)$. The activation function $g^*_\wedge$ outputs the value equal to the highest of its inputs. Example 5 illustrates that this can be seen as finding the best "match" of a pattern (rule). The activation function $g_\vee$ should have high output if at least one of the inputs is high or if all inputs are somewhat high. To satisfy this, we can crudely approximate Lukasiewicz fuzzy disjunction, which is given as $\min\{1, b_1+\cdots+b_k\}$ by the function $\mathit{sigm}\,(b_1+\cdots+b_k+b_0)$. Example 6 illustrates the intuition for the activation function $g_\vee$.

Example 5. Let us consider the LRNN

$$\begin{aligned}\mathcal{N} =\{&(\mathit{hasBrightEdge} \leftarrow \mathit{isBright}(E), 1), (\mathit{isBright}(E) \leftarrow \mathit{edge}(E,U,V) \wedge \mathit{bright}(U) \wedge \mathit{bright}(V),\\ &1), (\mathit{bright}(U) \leftarrow \mathit{yellow}(U), 2), (\mathit{bright}(U) \leftarrow \mathit{red}(U), 1), (\mathit{bright}(U) \leftarrow \mathit{blue}(U), 0.5)\}.\end{aligned}$$

Let us also have a set $\mathcal{G}$ describing a graph with colored vertices.

$$\begin{aligned}\mathcal{G} =\{&(\mathit{edge}(e_1,v_1,v_2),1), (\mathit{edge}(e_2,v_2,v_3),1), (\mathit{edge}(e_3,v_3,v_4),1), (\mathit{edge}(e_4,v_4,v_1),1),\\ &(\mathit{red}(v_1),1), (\mathit{blue}(v_2),1), (\mathit{yellow}(v_3),1), (\mathit{yellow}(v_4),1)\}\end{aligned}$$

The output of the atom neuron $A_{\mathit{hasBrightEdge}}$ will only depend on the "brightest edge", i.e. in this case on the edge e_3. The output would be the same for any other colored graph $\mathcal{G}'$, which would also contain an edge connecting two yellow vertices. Thus, for instance, if we considered some physicochemical property of atoms (e.g. their partial charge) instead of brightness of colors, and molecules instead of colored graphs, the corresponding networks could detect presence of a molecular substructure similar to a prescribed pattern.

Example 6. Let us have the LRNN

$$\begin{aligned}\mathcal{N} =\{&(\mathit{highPressure}(X) \leftarrow \mathit{stressed}(X), 1), (\mathit{highPressure}(X) \leftarrow \mathit{obese}(X), 1),\\ &(\mathit{highPressure}(X) \leftarrow \mathit{exercises}(X), -1)\}\end{aligned}$$

and the set of weighted facts $\mathcal{P} = \{(\mathit{stressed}(\mathit{alice}), 1),\ (\mathit{obese}(\mathit{alice}), 1),\ (\mathit{stressed}(\mathit{bob}), 1),\ (\mathit{exercises}(\mathit{bob}), 1)\}$. Outputs of aggregation neurons corresponding to rules from $\mathcal{N}$ with the same predicate in the head are combined using the activation functions $g_\vee$. Intuitively, rules and facts with the same predicate in the head can be seen as forming a logistic regression on the values given by the aggregation neurons from the lower layers. When the LRNN has just one layer, as in this example, one can achieve the same effect using techniques from *propositionalization* [12] – treating the bodies of the rules as features and feeding them as attributes to a logistic regression classifier. However, as soon as the LRNN has more layers, this effect cannot be emulated using propositionalization. In this particular example, if we construct the ground LRNN of $\overline{\mathcal{N} \cup \mathcal{P}}$ then the output of the atom neuron $A_{\mathit{highPressure}(\mathit{alice})}$ will be higher than the output of the atom neuron $A_{\mathit{highPressure}(\mathit{bob})}$ (because *alice* is stressed and obese whereas *bob* is just stressed and exercises).

The Max-Sigmoid activation function is obviously not the only one possible. It is useful when we are interested in detecting one or more patterns (such as the existence of an edge as bright as possible in Example 5) but less useful in situations similar to the one depicted in the next example.

Example 7. Let us consider the following simple LRNN for predicting individuals infected by flu

$$\mathcal{N} =\{(\mathit{hasFlu}(A) \leftarrow \mathit{friends}(A,B) \wedge \mathit{hasFluDiagnosed}(B), 1)\}$$

and a set of weighted ground facts $\mathcal{P}$ about a group of people and their friendships. If we constructed the ground neural networks of $\overline{\mathcal{N} \cup \mathcal{P}}$ using the activation functions from the Max-Sigmoid family then the prediction of whether an individual has flu would be entirely based on the existence of at least one person who already had flu diagnosed. It would be obviously more meaningful to base the predictions on the fraction of one's friends who had flu diagnosed.

A family of activation functions which are more appropriate in situations similar to to the one described in the above example is given by the next definition.

Definition 3 (Avg-Sigmoid Activation Functions)**.** *The Avg-Sigmoid (AS) collection of activation functions is composed of the following three families of functions:* $g_\wedge(b_1,\ldots,b_k) = sigm\left(\sum_{i=1}^{k} b_i - k + b_0\right)$, $g^*_\wedge(b_1,\ldots,b_m) = \frac{1}{m}\sum_{i=1}^{m} b_i$, *and* $g_\vee(b_1,\ldots,b_k) = \sum_{i=1}^{k} b_i + b_0$.

Another advantage of the Avg-Sigmoid family of activation functions over the Max-Sigmoid family is also that the functions from the Avg-Sigmoid family are everywhere differentiable (which simplifies learning). We note that other activation function families based on combinations of different aggregation functions might also be exploited for LRNN learning. Further learning scenarios and LRNN modeling constructs can be found in [20].

3 Weight Learning

Let us have a LRNN $\mathcal{N}$ and a set of training examples $\mathcal{E} = \{\mathcal{E}^1,\ldots,\mathcal{E}^m\}$ where each $\mathcal{E}^j$ is some structure represented by a set of weighted propositions (e.g., left part of Fig. 2), i.e. a LRNN containing only facts[3]. Let us also have a set $\mathcal{Q} = \{\{(q^1_1,t^1_1),\ldots,(q^1_{k_1},t^1_{k_1})\},\ldots,\{(q^m_1,t^m_1),\ldots,(q^m_{k_m},t^m_{k_m})\}\}$ where q^j_i are ground atoms, which we call *training query atoms*, and t^j_i are their *target values*. For any query atom q^j_i, let y^j_i denote the output of the atom neuron $A_{q^j_i}$ in the ground neural network of $\overline{\mathcal{N}\cup\mathcal{E}^j}$. The goal of the learning process is to find weights w_h of the rules (and possibly facts) in $\mathcal{N}$ minimizing cost J on the training query atoms $J(\mathcal{Q}) = \sum_{j=1}^{m}\sum_{i=1}^{k_j} cost(y^j_i,t^j_i)$ where *cost* is some predefined cost function which measures the discrepancy between the output of the atom neurons of the training query atoms and their desired target values. Similarly to conventional NNs, weight adaptation is performed by gradient descent steps $w_h \leftarrow w_h - \gamma\frac{\partial J(\mathcal{Q})}{\partial w_h}$ where γ is some given learning rate. The main difference is that in the case of LRNNs, the ground neural networks may be very different for different learning examples $\mathcal{E}^j$. However, this is not a fundamental problem because the weights for all the ground neural networks $\overline{\mathcal{N}\cup\mathcal{E}^j}$ are fully specified in the LRNN $\mathcal{N}$. Moreover, the weights from $\mathcal{N}$ can be repeated multiple times within a single $\overline{\mathcal{N}\cup\mathcal{E}^j}$, but since recursion is not allowed, the same weight can appear at most once on any simple path from a fact neuron to an atom neuron. Therefore it is possible to learn the weights using conventional online stochastic gradient descent algorithm[4], except that the increments for the shared weights must be accumulated, which is a simple consequence of linearity of partial differentiation.

Specifically, our weight-learning algorithm works as follows. First, it grounds the given LRNN $\mathcal{N}$ w.r.t. every example $\mathcal{E}^j$ from the dataset which gives it a set of ground neural networks $\overline{\mathcal{N}\cup\mathcal{E}^j}$ with shared weights (it keeps the information about the origin of each weight so that it could update the respective weights in the template in each step of the iteration). It then iterates over the ground networks in a random order, computes gradient of the error function for the current particular example given the current weights in the template, updates the weights accordingly and continues iterating these steps (i.e., the standard stochastic gradient descent procedure). In order to reduce the risk of getting stuck in poor quality local optima, we also employ a restart strategy for this algorithm. A more detailed version of LRNN weight learning can be found in [20].

4 Related Work

The main inspiration for the work presented in this paper are lifted graphical models such as Markov logic networks [18] or Bayesian logic programs [9]. However, none of these existing lifted graphical models is particularly well suited for learning parameters of latent relational structures. Our approach is also generally related to prior art in combining logical rules with neural networks, also known as neural-symbolic integration [4], such as in the KBANN system. While the KBANN [21] also constructs the network structure from given rules, these rules are propositional rather than relational and do not serve as a lifted template. Therefore it is impossible to learn relational latent

[3]The restriction of learning from facts only is actually not necessary but it will simplify this presentation.

[4]Learning is slightly more complicated for LRNNs with the Max-Sigmoid family of activation functions because the max operator introduces non-differentiable points to the optimization problem.

structures such as soft clustering of first-order-logic constants. A more recent system CILP++[6] utilizes a relational representation, which is however converted into a propositional form through a propositionalization technique [12]. This again means that latent non-ground relational structures cannot be learned by CILP++ either. A somewhat more closely related paper of FONN method [3] also designs a technique forming a network from relational rule set however this rule set is flat, producing only 1-layer (shallow) networks in which relational patterns are not hierarchically aggregated. While there are many other approaches of neural-symbolic integration aiming at relational (and first-order) representations [1], e.g. based on the CORE method [8], they typically search for a uniform model of the logic program in scope and thus principally differ from the presented lifted modeling approach.

While standard feed-forward neural networks can be seen as a special case of LRNNs, since any such a fixed neural architecture can be encoded in a corresponding ground rule set with respective activation functions, a salient aspect of our method is that it allows for learning from structured (relational) examples, rather than just attribute vectors. There has been previous work on adapting neural networks to cope with certain facets of relational representations. For example, extension to multi-instance learning was presented in [17]. A similarly directed work [2] facilitated aggregative reasoning to process sets of related tuples from relational database as a sequence through recurrent neural network structure, which was also presented for more general structures in [19]. These approaches are principally different from the presented method as they do not follow the lifted modeling strategy to cope with variations in structure of relational samples.

5 Experiments

In this section we describe experiments performed on 78 datasets of organic molecules: Mutagenesis dataset [15], four datasets from the predictive toxicollogy challenge and 73 NCI-GI datasets [16]. The Mutagenesis dataset contains 188 molecules with labels denoting their mutagenicity. A number of the results published on the mutagenesis dataset use extended set of features, providing additional expert knowledge on properties of molecules, degrading the role of learning capabilities in relational models (i.e. the expert features are discriminative enough by themselves). We do not use any of the extra features as we utilize only atom-bond information. The predictive toxicology challenge dataset (PTC) [7] is composed of four datasets of molecules labeled by their toxicity for female rats (fr), mouse (fm) and male rat (mr) and mouse (mm). Each of the NCI-GI datasets contains several thousands of molecules labeled by their ability to inhibit growth of different types of tumors. We compare performance of LRNNs to state-of-the-art relational learners kFOIL [13] and nFOIL [14], where kFOIL combines relational rule learninng with support vector machines and nFOIL combines relational rule learning with naive Bayes learning.

For LRNNs we use a simple hand-crafted template which is principally identical to the template discussed in Figure 2. Using such a generic template for all the datasets, we make sure that there is no additional expert knowledge involved [5]. The idea is that in the process of learning, useful latent relational concepts are created within the neural network by the means of weight adaptation rather than by explicit enumeration, in contrast to propositional approaches and ILP [5]. Indeed, none of the rules used in this template is useful on itself for prediction as a hard logic rule without weight adaptation.

To set the parameters of LRNNs we use the empirical risk minimization principle on the training cross-validation folds to select the parameters such as step size, restarts, number of iterations, etc. This way we obtain unbiased estimates of performance of our methods since test data is never involved in parameter selection. The time for training a LRNN was in the order of few hours for the larger NCI-GI datasets. The results of the experiments are summarized in Figure 4. LRNNs perform clearly the best of the algorithms in terms of accuracy as they have lower prediction error than kFOIL and nFOIL on significant majority of datasets. We also tried to compare LRNNs with another recent algorithm combining logic and neural networks called CILP++ [6], but we were not able to set up a proper relational representation for CILP++ and thus direct comparison remains as a part of our future work.

[5] I.e., the template does not relate to toxicity or any other specific property of molecules and might be as well used for other classification tasks, too.

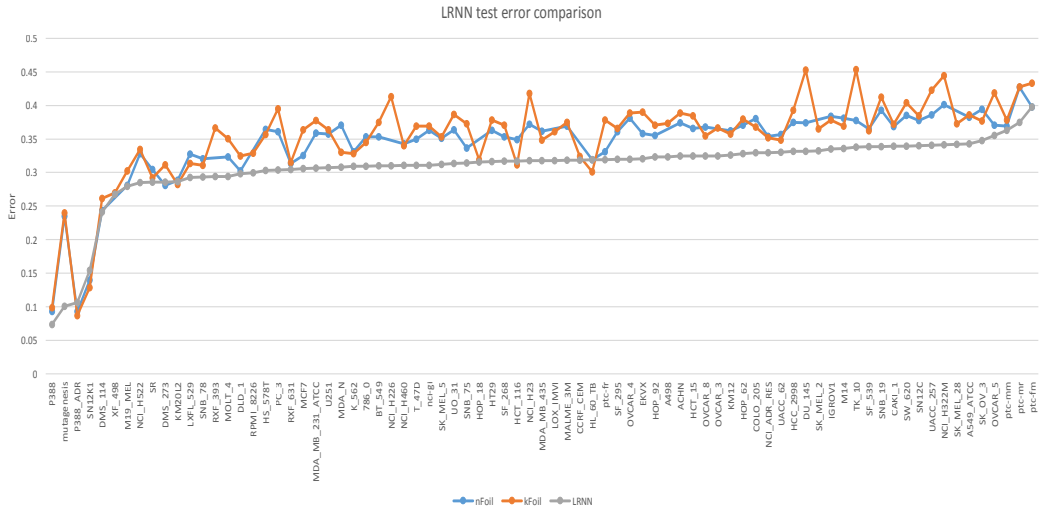


Figure 4: Prediction errors of LRNNs, kFOIL and nFOIL measured by cross-validation on 78 datasets of organic molecules.

In order to test the discovery of latent relational concepts, we performed an additional experiment with the Mutagenesis dataset. The relational concepts we were interested in were chains of varying lengths (up to 5 atoms). We trained the resulting LRNN to optimize the template's weights, however here we were more interested in extracting the learned patterns. We determined the chains of atoms which gave the highest output for the learned latent predicates. We obtained the following atom chain structures: C-C-F, N-O, C-Cl, C-Br, C-C-O, O-N-C. At least some of these structures appear to be directly relevant for the mutagenicity as they contain organic structures containing halogen atoms (Br, F and Cl). The other structures may be relevant to mutagenicity in combination with other structures.

Further, we have also investigated and confirmed the capability of LRNNs to learn proper weights of the latent non-ground relational concepts. This can be demonstrated e.g. on soft clustering of FOL predicates, a demonstration of which can be found in [20], together with evaluation and a closer description of the latent relational concept learning with Lifted Relational Neural Networks.

6 Conclusions

In this paper, we have introduced a method combining relational-logic representations with feedforward neural networks. The introduced method is close in spirit to lifted graphical models as it can be viewed as providing a lifted template for construction of ground neural networks. The performed experiments indicate that it is possible to achieve state-of-the-art predictive accuracies by weight learning with very generic templates and that it is able to induce notable latent relational concepts. There are many directions for future work including structure learning, transfer learning or studying different collections of activation functions. An important future direction is also the question of extending LRNNs to support recursion.

Acknowledgments

GS and FŽ are supported by Cisco sponsored research project "Modelling network traffic with relational features". While with CTU, OK was supported by the Czech Science Foundation through project P202/12/2032 and now by a grant from the Leverhulme Trust (RPG-2014-164).

References

[1] Sebastian Bader and Pascal Hitzler. Dimensions of Neural-symbolic Integration - A Structured Survey. *arXiv preprint*, page 28, 2005.

[2] H Blockeel and W Uwents. Using neural networks for relational learning. In *ICML-2004 Workshop on Statistical Relational Learning and its Connection to Other Fields*, 2004.

[3] M Botta, Giordana A, and R Piola. Combining first order logic with connectionist learning. In *Proceedings of the 14th International Conference on Machine Learning*, 1997.

[4] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. 2012.

[5] Luc De Raedt. *Logical and Relational Learning*. Springer, 2008.

[6] Manoel VM França, Gerson Zaverucha, and Artur S dAvila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104, 2014.

[7] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[8] Steffen Hölldobler, Yvonne Kalinke, and Hans Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11(1):45–58, 1999.

[9] Kristian Kersting and Luc De Raedt. Towards combining inductive logic programming with bayesian networks. In *Inductive Logic Programming, 11th International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001, Proceedings*, pages 118–131, 2001.

[10] A Kimmig, L Mihalkova, and L Getoor. Lifted graphical models: a survey. *Machine Learning*, 99(1):1–45, 2015.

[11] George Klir and Bo Yuan. *Fuzzy sets and fuzzy logic*, volume 4. Prentice Hall New Jersey, 1995.

[12] Mark-A Krogel, Simon Rawles, Filip Železný, Peter A Flach, Nada Lavrač, and Stefan Wrobel. *Comparative evaluation of approaches to propositionalization*. Springer, 2003.

[13] N. Landwehr, A. Passerini, L. De Raedt, and P. Frasconi. kFOIL: learning simple relational kernels. In *AAAI'06: Proceedings of the 21st national conference on Artificial intelligence*, pages 389–394. AAAI Press, 2006.

[14] Niels Landwehr, Kristian Kersting, and Luc De Raedt. Integrating naive bayes and foil. *The Journal of Machine Learning Research*, 8:481–507, 2007.

[15] Huma Lodhi and Stephen Muggleton. Is mutagenesis still challenging. *ILP-Late-Breaking Papers*, 35, 2005.

[16] Liva Ralaivola, Sanjay J. Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Netw.*, 18(8):1093–1110, 2005.

[17] J Ramon and L De Raedt. Multi instance neural networks. In *Proceedings of the ICML Workshop on Attribute-Value and Relational Learning*, 2000.

[18] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[19] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 20(1):61–80, January 2009.

[20] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, and Ondrej Kuzelka. Lifted Relational Neural Networks. *arXiv preprint*, 2015.

[21] Geofrey G Towell, Jude W Shavlik, and Michiel O Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the eighth National conference on Artificial intelligence*, pages 861–866. Boston, MA, 1990.

Relational Knowledge Extraction from Neural Networks

Manoel Vitor Macedo França
Department of Computer Science
City University London
London, United Kingdom EC1V 0HB
manoel.franca@city.ac.uk

Artur S. d'Avila Garcez
Department of Computer Science
City University London
London, United Kingdom EC1V 0HB
a.garcez@city.ac.uk

Gerson Zaverucha
Prog. de Eng. de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil 21941–972
gerson@cos.ufrj.br

Abstract

The effective integration of learning and reasoning is a well-known and challenging area of research within artificial intelligence. Neural-symbolic systems seek to integrate learning and reasoning by combining neural networks and symbolic knowledge representation. In this paper, a novel methodology is proposed for the extraction of relational knowledge from neural networks which are trainable by the efficient application of the backpropagation learning algorithm. First-order logic rules are extracted from the neural networks, offering interpretable symbolic relational models on which logical reasoning can be performed. The well-known knowledge extraction algorithm TREPAN was adapted and incorporated into the first-order version of the neural-symbolic system CILP++. Empirical results obtained in comparison with a probabilistic model for relational learning, Markov Logic Networks, and a state-of-the-art Inductive Logic Programming system, Aleph, indicate that the proposed methodology achieves competitive accuracy results consistently in all datasets investigated, while either Markov Logic Networks or Aleph show considerably worse results in at least one dataset. It is expected that effective knowledge extraction from neural networks can contribute to the integration of heterogeneous knowledge representations.

1 Introduction

Integrating learning and reasoning efficiently and accurately has a vast track of research and publications in artificial intelligence [1, 2, 3, 4]. This integration can be done at different stages of learning, from data pre-processing, feature extraction, the learning algorithm, up to reasoning about learning. Neural-symbolic systems seek to integrate learning and reasoning by combining neural networks and symbolic knowledge representations using, e.g., propositional logic or first-order logic.

Relational learning can be described as the process of learning a first-order logic theory from examples and domain knowledge [5, 6]. Differently from propositional learning, relational learning does not use a set of attributes and values. Instead, it is based on objects and relations among objects, which are represented by constants and predicates, respectively. Relational learning has had applications in bioinformatics, graph mining and link analysis [7, 8].

Inductive Logic Programming (ILP) [5] performs relational learning either directly by manipulating first-order rules or through a process called propositionalization [9, 10, 11], which brings the relational task down to the propositional level by representing subsets of relations as features that can be used as attributes. In comparison with direct ILP, propositionalization normally exchanges accuracy for efficiency [12], as it enables the use of fast attribute-value learners [13, 10, 9], although the translation of first-order rules into features can cause information loss [14].

Much work has been done combining relational learning tasks with propositional learners, including decision trees or neural networks [15, 16, 17, 18, 19]. In this paper, we are interested in the, less investigated, inverse problem: how to extract first-order logic descriptions from propositional learners, in particular, neural networks, trained to solve relational learning tasks?

We extend the well-known CILP neural-symbolic system [3] to allow the extraction of meaningful first-order logic rules from trained neural networks. Propositionalization and subsequent attribute-value learning can destroy the original relational structure of the task at hand, so much so that the provision of interpretable relational knowledge following learning is made impossible [14]. In this paper, we show that by adapting the first-order version of the CILP system, called CILP++ [13], so as to enable the application of a variation of the well-known TREPAN knowledge extraction algorithm [20], a revised set of first-order rules can be extracted from trained neural networks efficiently and accurately, enabling first-order logical reasoning about what has been learned by the network. The result is a neural network, trained using an efficient backpropagation learning algorithm, and capable of receiving "first-order" examples as input and producing first order rules as output. The ability to perform reasoning directly opens a number of research and application possibilities integrating reasoning and learning [21, 7, 8, 22].

We have compared relational knowledge extraction in CILP++ with state-of-the-art ILP system Aleph [23] and Markov Logic Networks (MLN's) [15] on the Mutagenesis [7], UW-CSE [8], Alzheimer-amine [21] and Cora [22] datasets. Results indicate that the relational theories extracted from CILP++ have high fidelity to the trained neural network models, and that the use of neural networks can provide considerable speed-ups while achieving comparable accuracy and area under ROC curve results.

The choice of using MLN's and Aleph for empirical comparisons is due the nature of their methodology for tackling relational learning, which are distinctively different: MLN's take a probabilistic approach for the relational learning problem, by attempting to find a distribution model that fits the ground atoms of a hypothesis interpretation as best as possible, while Aleph performs relational learning by searching the Herbrand space [5] of possible literals for a given dataset.

The remainder of the paper is as follows: section 2 introduces CILP++, the neural-symbolic system that uses the proposed approach in this paper for relational knowledge extraction from trained neural networks. Section 3 presents obtained experimental results with CILP++ on the Mutagenesis, UW-CSE, Alzheimer-amine and Cora datasets, comparing against MLN's and Aleph. Lastly, section 4 discusses outcomes from the experiments performed and also does an overview of other systems that are closely related with the work being presented in this paper.

CILP++ is available from Sourceforge (`https://sourceforge.net/projects/cilppp/`) and experimental settings will be made available for download.

2 Relational Learning with Neural Networks

This section introduces the proposed system for relational knowledge extraction from trained neural networks. It starts by presenting each module of the CILP++ system and how they can be adapted to allow direct first-order knowledge extraction and inference from the trained models.

2.1 Bottom Clause Propositionalization

Relational learning with CILP++ starts by applying bottom clause propositionalization (BCP) onto the first-order examples set. Each first-order example, in the form of a instantiated target clause, e.g. $target(a_1, \ldots, a_n)$, is converted into a numerical vector that a neural network can use as input. In order to achieve this, each example is transformed into a bottom clause and mapped onto features

on an attribute-value table, and numerical vectors are generated for each example. Thus, BCP has three steps: bottom clause generation, feature generation and attribute-value mapping.

Firstly, before describing each BCP step, we present three first-order concepts which are used in this work: *clause*, *relational domain* and *bottom clause*.

– A *clause* is a definition of relations between facts, with structure

$$p_t(V_{1_t},\ldots,V_{n_t})\text{ :- }p_1(V_{1_1},\ldots,V_{n_1}),p_2(V_{1_2},\ldots,V_{n_2}),\ldots,p_m(V_{1_m},\ldots,V_{n_m}),$$

where $\{p_i, 1 \leq i \leq m\} \bigcup \{p_t\}$ is a set of predicates (relations), V_j is a set of variables, $p_1(V_{1_1},\ldots,V_{n_1}),p_2(V_{1_2},\ldots,V_{n_2}),\ldots,p_m(V_{1_m},\ldots,V_{n_m})$ are literals and **:-** represents implication. Literals on the left-hand side of the consequence operator are known as head literals and literals on the right-hand side are known as body literals.

– A *relational domain* is a tuple $< E, BK >$, where: E is a set of ground atoms of target concept(s) (i.e. first-order logic examples), in which labels are truth-values; and BK is a set of clauses known as background knowledge, which can be *facts* (grounded single-literal clauses that define what is known about a given task) or *clauses*, as define above.

– A *bottom clause* is a boundary in the hypothesis search space during ILP learning [24], and is built from one random positive example, background knowledge and language bias (a set of clauses that define how clauses can be built in an ILP model). A bottom clause is the most specific clause (with most literals) that can be considered a candidate hypothesis.

Having introduced clauses and relational domain, we are in position to describe BCP. In the first step of BCP, bottom clause generation, each example e_i from a first-order example set E is given to the bottom clause generation algorithm [25] to create a corresponding bottom clause set $E_\perp$, containing one bottom clause $\perp_i$ for each example e_i. To do so, a slight modification is needed to allow the same *hash* function to be shared among all examples, in order to keep consistency between variable associations, and to allow negative examples to have bottom clauses as well; the original algorithm deals with positive examples only. An extensive algorithm description is provided in [13].

In order to illustrate each BCP step, we introduce a small family relationship relational domain [26], with background knowledge

$$BK = \{\textit{mother(mom1, daughter1)}, \textit{wife(daughter1, husband1)}, \textit{wife(daughter2, husband2)}\},$$

with one positive example and one negative example *motherInLaw(mom1, husband1)* and *motherInLaw(daughter1, husband2)*, respectively. It can be noticed that the relation between *mom1* and *husband1*, which the positive example establishes, can be alternatively described by the sequence of facts *mother(mom1, daughter1)* and *wife(daughter1, husband1)* in the background knowledge. This states semantically that *mom1* is a mother-in-law because *mom1* has a married daughter, namely, *daughter1*. Applied to this example, the bottom clause generation algorithm would create a clause $\perp_i = \textit{motherInLaw}(A,B) \leftarrow \textit{mother(A, C)}, \textit{wife(C, B)}$. Comparing $\perp$ with the sequence of facts above, we notice that $\perp_i$ describes one possible meaning of mother-in-law: "*A* is a mother-in-law of *B* if *A* is a mother of *C* and *C* is wife of *B*", i.e. the mother of a married daughter is a mother-in-law. To generate features from bottom clauses, BCP generates one bottom clause for each (positive or negative) example e, which we denote as $\perp_e$. At the end of the first step of BCP, we end with a bottom clause set containing both positive and negative examples:

$$\begin{aligned} E_\perp = \{&\textit{motherInLaw}(A,B) : -\textit{mother}(A,C), \textit{wife}(C,B);\\ &\sim \textit{motherInLaw}(A,B) : -\textit{wife}(A,C)\} \quad .\end{aligned}$$

In the second step, feature generation, a feature table F is generated from $E_\perp$. Earlier versions of CILP++ used bottom clause literals directly as features, but this approach can lead to inconsistencies if knowledge is to be extracted from models which used such features [27]. In order to tackle this, an adapted version of the first-order feature generation algorithm presented in [16] has been used to generate independent propositional features which represent first-order descriptions.

For illustrating the second step of BCP, consider the following bottom clause $R_\perp$:

$$motherInLaw(A,B) : -parent(A,C), wife(C,B), \\ wife(A,D), brother(B,D)$$

Semi-propositionalization [16] is used to generate a set of first-order features for $R_\perp$. First-order features are sets of literals that share variables that are not inside any head literal. Those variables are known as local variables. From the family relationship example, the following features are obtained:

$$F_1 = \{parent(A,C), wife(C,B)\}$$
$$F_2 = \{wife(A,D), brother(B,D)\}$$

BCP treats each decomposition as a feature and in the example above, two clauses would be generated from the decomposition of $R_\perp$:

$$L_1(A,B) : -parent(A,C), wife(C,B)$$
$$L_2(A,B) : -wife(A,D), brother(B,D)$$

Therefore, $R_\perp$ can be rewritten as the following semi-propositional rule $R'_\perp$:

$$motherInLaw(A,B) : -L_1(A,B), L_2(A,B)$$

If the only example to be propositionalized by BCP is r, the feature table F would, at the end, contain only two elements: $L_1(A,B)$ and $L_2(A,B)$.

Lastly, in the third step of BCP, the feature table F is applied onto E in order to generate binary vectors that a neural network can process. The algorithm, implemented on CILP++, is as follows:

1. Let $|F|$ be the number of elements in F;
2. Let E_v be the set of binary vectors, converted from E, initially empty;
3. For each example $e_i \in E$ do
   (a) For each feature $f_j \in F$ do
      i. Query E against the correspondent first-order description L_j of f_j against the relational domain background knowledge BK;
      ii. If query succeeds, assign 1 to the position j binary vector v_i; if not, assign 0 instead;
   (b) Associate a label 1 to v_i if e_i is a positive example, and -1 otherwise;
   (c) Add v_i to E_v;
4. Return E_v.

Continuing the family relationship example: $|F|$ is equal to 2, since there is only two features in the table: $L_1(A,B)$ and $L_2(A,B)$. Since r contain both features on its bottom clause, $v_r = (1,1)$. See [13] for more details.

2.2 Neural Network Learning and Relational Knowledge Extraction

CILP++ uses resilient backpropagation [28], with *early stopping* [29] for learning. Resilient backpropagation takes into account only the sign of the partial derivative over all training examples (not the magnitude), and acts independently on each weight. For each weight, if there was a sign change of the partial derivative of the total error function compared to the last iteration, the update value for that weight is multiplied by a factor $\eta-$, where $\eta- < 1$. If the last iteration produced the same sign, the update value is multiplied by a factor of $\eta+$ where $\eta+ > 1$. The update values are calculated

for each weight in the above manner, and finally each weight is changed by its own update value, in the opposite direction of that weight's partial derivative, so as to minimize the total error function. We set $\eta+$ and $\eta-$ through validation.

With early stopping, when the validation error measure starts to increase, training is stopped. We have used a more permissive version of early stopping [29], which does not halt training immediately after the validation error increases. It stops when a combined measure of both number of consecutive epochs with increasing validation set error and absolute value of current validation set error reaches a certain threshold.

Following network training, in order to perform relational knowledge extraction, an adapted version of the TREPAN rule extractor [20] is applied to the trained neural network. TREPAN is originally a m-of-n propositional tree inducer which uses a learned neural network as oracle and through a set of examples S, possibly distinct from the example set used for training the neural network, a decision tree is recursively built, based on an information gain-based heuristic. We adapted TREPAN in order to allow the generation and query of first-order rules into Prolog [30], a well-known general purpose logic programming. Several simplifications have also been done in order to improve efficiency and readability. The adapted pseudo-algorithm for TREPAN can be seen on Algorithm 1, based on the original TREPAN algorithm [20]. Changes from the original are highlighted with an underline.

Algorithm 1 Adapted TREPAN algorithm

Inputs: training set E, feature set F
for all example $e \in E$ **do**
  class label for $e = Oracle(E)$
end for
initialize the root of the tree, T, as leaf node
initialize *queue* with tuple $< T, E, \{\} >$
while *queue* not empty and $\underline{size(T) < tree\_size\_limit}$ **do**
  remove node N from head of *queue*
  $examples_n$ = example set stored with N
  $constraints_n$ = constraint set stored with N
  use F to build set of candidate splits for node
  use $examples_n$ and calls to $Oracle(constraints_n)$ to evaluate splits
  S = best binary split
  search for best m-of-n split, S', using S as seed
  make N an internal node with split S'
  for all possible outcomes $s \in S'$ **do**
    make C, a new child node of N
    $constraints_c = constraints_n \cup \{S' = s\}$
    use calls to $Oracle(constraints_c)$ to determine if C should remain a leaf
    if not **then**
      $examples_c$ = members of $examples_n$ with outcome s on split S'
      put $< C, example_c, constraints_c >$ into *Queue*
    end if
  end for
end while
$TH = empty$
for all path p in T **do**
  Create a rule r containing conjunctions of each m-of-n split conditions s inside p
  Add all possible m combinations of the n conditions inside s as disjunctive body literals in r
  Add r to TH
end for
perform a decreasing ordering on TH on the number of examples covered by each path p
return a first-order theory TH

The adapted version of TREPAN presented on Algorithm 1 have the following differences when compared to original TREPAN:

- Line 7: Tree generation has been simplified, only maximum size criterion is used for stopping the process.
- Line 14: The search heuristic for best m-of-n split is now weighted by size of m. The original heuristic value for a given split is now subtracted by m/n.
- Lines 26-32: The m-of-n tree is transformed into a set of (possibly) disjunctive rules, in order to allow first-order inference with logic programming languages such as Prolog.

After extracting rules from the trained network (after obtaining TH), the definitions of the semi-propositional first-order features (L_i clauses) obtained during BCP are added to TH, resulting in a first-order theory that can be used to perform first-order inference. In the following, the well-known east-west trains ILP dataset [31] is used in order to demonstrate how CILP++ performs relational learning, relational knowledge extraction and reasoning.

In the first step of CILP++ (propositionalization with BCP), 20 bottom clauses were generated from the 10 positive and 10 negative examples of eastbound and westbound trains. From those bottom clauses, 41 features were obtained by using semi-propositionalization. Therefore, 41 input neurons will be created in CILP++'s initial neural network structure, each one representing one feature. A small sample of bottom clauses generated, the features generated with BCP and the resulting initial neural network structure are presented in Figure 1.

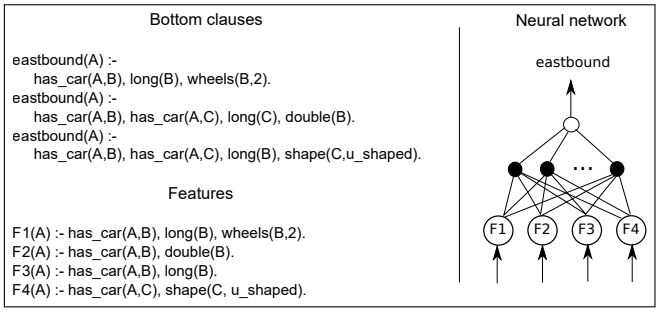


Figure 1: Bottom clauses and BCP features example for the east-west trains dataset.

After neural network training, the adapted TREPAN rule extractor algorithm (Algorithm 1) is used to generate first-order rules from the network. Leave-one-out cross-validation was used, i.e., 20 folds have been generated from the 20 first-order examples. Figure 2 shows the resulting first-order theory. The first part of the generated theory is the extracted TREPAN rules, whilst the second part is the added semi-propositional clauses generated by BCP.

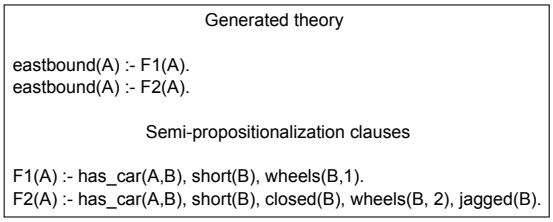


Figure 2: Theory obtained with CILP++ for the east-west trains dataset[1]

The obtained results for the east-west trains are:

- **Accuracy**: 90% for the trained network and 90% accuracy for the extracted rules (the extracted rules were evaluated in Prolog, by querying the first-order examples);
- **Fidelity**: 95% of fidelity between the trained neural network and the extracted rules. Fidelity is defined as the percentage of examples classified in the same way by both models. It does not matter if the result is a hit or a miss for a given example e: as long as both the neural network and the rules classify e identically, it is considered a hit towards fidelity.

[1]Compare with the rules extracted by LINUS in [16]; our method seems to produce more readable rules.

Table 1: Accuracy results (ACC) with standard deviations, area under ROC curve results (AUC) and runtimes in seconds (RUN) for the Mutagenesis, UW-CSE and Alzheimers-anime datasets

SYSTEM	METRICS	DATASETS			
		Mutagenesis	UW-CSE	Alzheimer-amine	Cora
Aleph	*ACC*	80.85%(±10.51)	85.11%(±7.11)	78.71%(±5.25)	−−
	AUC	0.80(0.02)	0.81(±0.07)	0.79(±0.09)	−−
	RUN	721	875	5465	−−
MLN	*ACC*	62.11%(±0.022)	75.01%(±0.028)	50.01%(±0.002)	70.10%(±0.191)
	AUC	0.67(±0.022)	0.76(±0.12)	0.51(±0.02)	0.809(±0.001)
	RUN	4341	1184	9811	97148
CILP++	*ACC*	91.70%(±5.84)	77.17%(±9.01)	79.91%(±2.12)	68.91%(±2.12)
	AUC	0.82(±0.02)	0.77(±0.07)	0.80(±0.01)	0.79(±0.02)
	RUN	851	742	3822	11435

Table 2: Rule accuracy results (ACC) with standard deviations and fidelity (FID) for the Mutagenesis, UW-CSE, Alzheimer-anime and Cora datasets

SYSTEM	METRICS	DATASETS			
		Mutagenesis	UW-CSE	Alzheimer-anime	Cora
Aleph	*ACC*	80.85%(±10.51)	85.11%(±7.11)	78.71%(±5.25)	−−
CILP++	*ACC*	77.72(±2.12)	81.98(±3.11)	78.70%(±6.12)	67.98%(±5.29)
	FID	0.89	0.90	0.88	0.82

3 Experimental Results

CILP++ has been tested empirically and results over ten-fold cross validation for the trained neural network can be seen on Table 1. CILP++ is being tested against a well-known ILP system, Aleph [23] and Markov Logic Networks (MLN's) [15]. Four relational domains have been used: Mutagenesis [7], UW-CSE [8], Alzheimers-anime [21] and Cora [22]. The same parameters as [13] have been used for training CILP++. For Aleph, the settings suggested in [18] have been used. For MLN's, the reported results on three publications [15, 32, 33] have been collected. Lastly, on TREPAN, $tree_size_limit$ has been set as 5. All experiments were run on a 3.2 Ghz Intel Core i3-2100 with 4 GB RAM.

Results show that CILP++ has comparable accuracy and AUC measurements with both Aleph and MLN's, while having considerably better runtimes. While CILP++ was able to run and generate competitive results on all tested datasets, Aleph ran out of memory while running Cora. Standard MLN's performed very poorly on Alzheimer-amine [32] and had higher training times.

Table 2 shows comparative results with Aleph for querying the extracted first-order rules from the trained CILP++ neural network on Prolog. Also, fidelity measurements are provided.

Results show that competitive accuracy with Aleph has been maintained after extraction, and also good fidelity measures have been obtained in comparison with the trained neural network. This indicates that CILP++ neural networks are capable of efficiently solve relational tasks with BCP.

4 Concluding Remarks

In this paper, we have presented an integrated and efficient method and system for the extraction of first-order logic rules from neural networks. Experimental results show that the first-order rules extracted from trained neural networks, in terms of accuracy and AUC, are comparable with a well-

known probabilistic system for relational learning, MLN's, and a search-based ILP system, Aleph, while being considerably faster. Those results indicate the promise of CILP++ as a relational learner.

Further comparisons with related work include the analysis of propositionalization-based systems such as LINUS/DINUS/SINUS [9, 11] and RelF[34], which rely on the quality of their feature generation to reduce the information loss of the propositionalization approach and, consequently, within the rules extracted from the learner. Both the LINUS/DINUS/SINUS family of ILP systems and RelF generate a number of constrained first-order features f from the Herbrand base H (H is the set of possible clauses for a given domain knowledge). From the collection of features f, a final set of features F is obtained for representing the training examples, according to a given score function. CILP++, on the other hand, uses the concept of bottom clause, which is a clause that uniquely describes a single relational example. CILP++ uses bottom clauses to train a neural network, and an algorithm based on the concept of semi-propositionalization [16, 27] to generate F.

Approaches based on Bayesian networks [35] also perform relational learning, but represent the learned knowledge without the use of explicit relational rules. Statistical Relational Models [36] contain a rich representation language which combines a frame-based logical representation with probabilistic semantics based on bayesian networks. BLOG [37] is a first-order probabilistic modeling language that specifies probability distributions over possible worlds with varying sets of objects. A BLOG model contains statements that define conditional probability distributions for a certain set of random variables; the model also specifies certain context-specific independence properties. Inference is done on BLOG using Markov Chain Monte Carlo [38] algorithms. In CILP++, inference is deterministic with first-order rules learned from the neural (statistical) model being applicable directly onto a Prolog theorem prover for reasoning.

As future work, a study on how CILP++ deals with noisy datasets (noise in the background knowledge and/or examples) can provide interesting results, due to how backpropagation naturally deals with incomplete data and noisy inputs. Also, an investigation on how CILP++ can be adapted to deal directly with numeric data can overcome a well-known flaw in ILP systems, which is its inability to deal directly with numbers. ILP systems use auxiliary predicates to indicate relations between numeric variables such as greater-than, less-than and so on.

References

[1] S. Hölldobler and Y. Kalinke, "Towards a massively parallel computational model for logic programming," in *In: Proceedings of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pp. 68–77, 1994.

[2] G. G. Towell and J. W. Shavlik, "Knowledge-Based Artificial Neural Networks," *Artif. Intell.*, vol. 70, no. 1-2, pp. 119–165, 1994.

[3] A. S. D. Garcez and G. Zaverucha, "The Connectionist Inductive Learning and Logic Programming System," *Applied Intelligence*, vol. 11, pp. 59–77, 1999.

[4] R. Basilio, G. Zaverucha, and V. Barbosa, "Learning Logic Programs with Neural Networks," in *Inductive Logic Programming*, vol. 2157 of *LNAI*, pp. 15–26, Springer Berlin / Heidelberg, 2001.

[5] S. Džeroski and N. Lavrač, *Relational Data Mining*. Relational Data Mining, Springer, 2001.

[6] L. De Raedt, *Logical and Relational Learning*. Cognitive Technologies, Springer, 2008.

[7] A. Srinivasan and S. H. Muggleton, "Mutagenesis: ILP experiments in a non-determinate biological domain," in *Proceedings of the 4th International Workshop on Inductive Logic Programming, volume 237 of GMD-Studien*, pp. 217–232, 1994.

[8] J. Davis, E. S. Burnside, I. de Castro Dutra, D. Page, and V. S. Costa, "An integrated approach to learning bayesian networks of rules," in *ECML* (J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, eds.), vol. 3720 of *Lecture Notes in Computer Science*, pp. 84–95, Springer, 2005.

[9] N. Lavrač and S. Džeroski, *Inductive logic programming: techniques and applications*. Ellis Horwood series in artificial intelligence, E. Horwood, 1994.

[10] F. Železný and N. Lavrač, "Propositionalization-based Relational Subgroup Discovery With RSD," *Machine Learning*, vol. 62, pp. 33–63, 2006.

[11] S. Kramer, N. Lavrač, and P. Flach, "Relational Data Mining," ch. Propositionalization approaches to relational data mining, pp. 262–286, New York, NY, USA: Springer-Verlag New York, Inc., 2000.

[12] M.-A. Krogel, S. Rawles, F. Železný, P. Flach, N. Lavrač, and S. Wrobel, "Comparative Evaluation Of Approaches To Propositionalization," in *ILP*, vol. 2835 of *LNAI*, pp. 194–217, Springer-Verlag, 2003.

[13] M. V. M. França, G. Zaverucha, and A. dAvila Garcez, "Fast relational learning using bottom clause propositionalization with artificial neural networks," *Machine Learning*, vol. 94, no. 1, pp. 81–104, 2014.

[14] M. V. M. França, A. S. D. Garcez, and G. Zaverucha, "Relational Knowledge Extraction from Attribute-Value Learners," in *2013 Imperial College Computing Student Workshop*, vol. 35 of *OpenAccess Series in Informatics (OASIcs)*, (Dagstuhl, Germany), pp. 35–42, Schloss Dagstuhl, 2013.

[15] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, pp. 107–136, 2006.

[16] N. Lavrač and P. A. Flach, "An extended transformation approach to inductive logic programming," *ACM Trans. Comput. Logic*, vol. 2, no. 4, pp. 458–494, 2001.

[17] B. Kijsirikul and B. K. Lerdlamnaochai, "First-Order Logical Neural Networks," *Int. J. Hybrid Intell. Syst.*, vol. 2, pp. 253–267, Dec. 2005.

[18] A. Paes, F. Železný, G. Zaverucha, D. Page, and A. Srinivasan, "ILP Through Propositionalization and Stochastic k-Term DNF Learning," in *ILP*, vol. 4455 of *LNAI*, (Berlin, Heidelberg), pp. 379–393, Springer-Verlag, 2007.

[19] R. Basilio, G. Zaverucha, and A. S. Garcez, "Inducing Relational Concepts with Neural Networks via the LINUS System," in *In ICONIP*, pp. 1507151–0, 1998.

[20] M. Craven and J. W. Shavlik, "Extracting Tree-Structured Representations of Trained Networks," in *NIPS*, pp. 24–30, 1995.

[21] R. King and A. Srinivasan, "Relating chemical activity to structure: An examination of ILP successes," *New Generation Computing*, vol. 13, no. 3-4, pp. 411–434, 1995.

[22] M. Bilenko and R. J. Mooney, "Adaptive duplicate detection using learnable string similarity measures," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, (New York, NY, USA), pp. 39–48, ACM, 2003.

[23] A. Srinivasan, "The Aleph System, version 5." http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html, 2007. Last accessed on may/2013.

[24] S. Muggleton, "Inverse Entailment and Progol," *New Generation Computing, Special issue on Inductive Logic Programming*, vol. 13, no. 3-4, pp. 245–286, 1995.

[25] A. Tamaddoni-Nezhad and S. Muggleton, "The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause," *Machine Learning*, vol. 76, no. 1, pp. 37–72, 2009.

[26] S. Muggleton and L. D. Raedt, "Inductive Logic Programming: Theory and Methods," *Journal of Logic Programming*, vol. 19, no. 20, pp. 629–679, 1994.

[27] M. V. M. França, G. Zaverucha, and A. S. D. Garcez, "Neural relational learning through semi-propositionalization of bottom clauses," in *AAAI Spring Symposium Series*, 2015.

[28] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp. 295–307, 1988.

[29] L. Prechelt, "Early stopping - but when?," in *Neural Networks: Tricks of the Trade, volume 1524 of LNCS, chapter 2*, pp. 55–69, Springer-Verlag, 1997.

[30] R. A. Kowalski, "The early years of logic programming," *Commun. ACM*, vol. 31, pp. 38–43, Jan. 1988.

[31] J. Larson and R. S. Michalski, "Inductive inference of VL decision rules," *SIGART Bull.*, no. 63, pp. 38–44, 1977.

[32] T. N. Huynh and R. J. Mooney, "Discriminative structure and parameter learning for markov logic networks," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), pp. 416–423, ACM, 2008.

[33] S. Kok and P. Domingos, "Learning the structure of markov logic networks," in *Proceedings of the 22Nd International Conference on Machine Learning*, (New York, NY, USA), pp. 441–448, ACM, 2005.

[34] O. Kuželka and F. Železný, "Block-wise construction of tree-like relational features with monotone reducibility and redundancy," *Machine Learning*, vol. 83, pp. 163–192, 2011.

[35] J. Pearl, *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[36] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *In IJCAI*, pp. 1300–1309, Springer-Verlag, 1999.

[37] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov, "BLOG: probabilistic models with unknown objects," IJCAI, (San Francisco, CA, USA), pp. 1352–1359, Morgan Kaufmann Publishers Inc., 2005.

[38] A. E. Gelfand and A. F. M. Smith, "Sampling-based approaches to calculating marginal densities," *Journal of the American Statistical Association*, vol. 85, no. 410, pp. 398–409, 1990.

Combinatorial structures and processing in Neural Blackboard Architectures

Frank van der Velde
CTIT, CPE-BMS
University of Twente, Enschede
IOP, Leiden
The Netherlands
f.vandervelde@utwente.nl

Marc de Kamps
Institute for Artificial Intelligence
and Biological Systems
University of Leeds
Leeds, United Kingdom
M.deKamps@leeds.ac.uk

Abstract

We discuss and illustrate Neural Blackboard Architectures (NBAs) as the basis for variable binding and combinatorial processing the brain. We focus on the NBA for sentence structure. NBAs are based on the notion that conceptual representations are in situ, hence cannot be copied or transported. Novel combinatorial structures can be formed with these representations by embedding them in NBAs. We discuss and illustrate the main characteristics of this form of combinatorial processing. We also illustrate the NBA for sentence structures by simulating neural activity as found in recently reported intracranial brain observations. Furthermore, we will show how the NBA can account for ambiguity resolution and garden path effects in sentence processing.

1 Introduction

We outline and illustrate the implementation and processing of combinatorial structures in Neural Blackboard Architectures (or NBAs). First, we will discuss the ideas on which NBAs are based. Then, we will illustrate sentence representation and processing in the NBA for sentence structure [1]. Next, we will simulate neural activation patterns that arise in the NBA during sentence processing and relate it to recently reported intracranial brain observations. Finally, we will illustrate and discuss how the NBA handles linguistic phenomena such as unproblematic ambiguities and garden path sentences [2].

2 In situ concept representations

The development of Neural Blackboard Architectures is based on the notion that representations in the brain, such as representations of concepts, are 'in situ', in line with the neural assemblies proposed by Hebb [3]. Figure 1 (based on [4]) illustrates this with the concept *cat*. As Hebb argued, a neural assembly develops over time as neurons that process information about a concept become interconnected. These neurons could be located in different parts of the brain, depending on the nature of the underlying concept. So, in case of a concept like *cat*, they will consist of neurons in the visual and auditory cortex, involved in the perception of cats. They could also consist of neurons involved in actions related to cats (e.g., pronouncing the word *cat*).

Concept representations with neural assemblies could be distributed, but parts of the assembly could also consist of more local representations. They could consist of neurons involved in processing information about the concepts, but also of neurons involved in actions related to concepts. This entails that both incoming and outgoing connections determine the nature of a concept representation [5]. Furthermore, unlike the original assemblies as described by Hebb, concepts as intended here

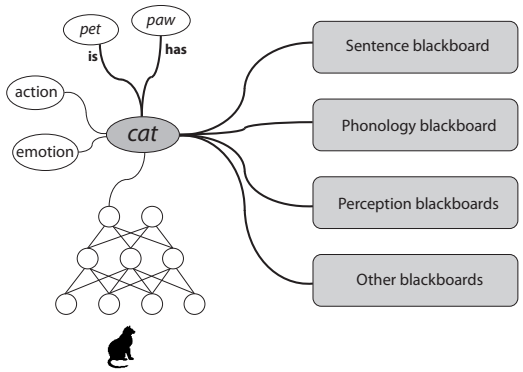


Figure 1: In situ concept representation (based on [4]).

are not only associative neural structures. They also incorporate relations, as illustrated with the relations *is pet* and *has paw* in Figure 1.

The assembly or web-like structure of a concept representation entails that concepts representations are 'in situ' [4]. That is, wherever a concept is activated it always consists of the activation of the assembly of that concept or a part of it. In this view, it is not possible to make a copy of a concept representation and store it elsewhere in the brain to, for example, create a combinatorial structure with it or use it in a reasoning process. In this sense, in situ concept representations are fundamentally different from symbols in symbol architectures [6]. Due to the in situ nature of concepts representations, they are also grounded in perception or action (depending on the nature of the concept).

The in situ nature of neuronal concept representations imposes constraints on the way they can be combined in higher-level cognitive combinatorial structures, such as sentences. In situ concept representation cannot be copied or transported to construct combinatorial structures. Instead, as illustrated in Figure 1, they are embedded in 'neural blackboards' to form combinatorial structures. We assume that there will be specific neural blackboards for specific forms of combinatorial structures, and concept representations will be embedded in a number of them depending on the nature of that concept and the role it plays in specific combinatorial structures. For example, the concept *cat* will be embedded in a phonology blackboard, forming the word *cat* based on familiar phonemes. It will also be embedded in a sentence blackboard, needed to form sentence structures. It could also be embedded in a perception blackboard underlying the combinatorial nature of visual cognition [7]. Other blackboards, e.g., needed for sequential processing or reasoning, could be included as well.

The neural blackboards allow the construction of (potentially novel) combinatorial structures based on (familiar) in situ concept representations, using forms of variable binding (depending on the specific blackboard). They aim to satisfy the following characteristics:

- The concept representations remain in situ (and thus grounded) even when they are a part of (novel) combinatorial structures.
- The combinatorial structures or parts thereof are content addressable, even with novel combinatorial structures (of familiar constituents).
- The variable binding of concepts in blackboards creates the connection paths needed to produce behavior (linking sensory and motor activation), even with (novel) combinatorial structures.

We will discuss and illustrate these characteristics using the sentence blackboard.

3 Sentence neural blackboard

Figure 2 illustrates the structure of the sentence *Sonnet sees (the) dog* in the Neural Blackboard Architecture for sentence structure [1]. Here, we assume that the words *sees* and *dog* are familiar

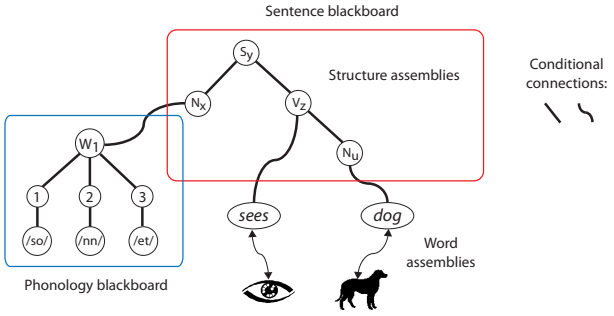


Figure 2: Sentence representation in a sentence neural blackboard.

constituents but *Sonnet* is a new name, not heard before. The representations of *sees* and *dog* consist of neural assembly structures ('word assemblies'), in line with the concept *cat* in Figure 1. For simplicity, we represent them with ovals, but the full connection structure is always implied.

The sentence blackboard allows the creation of a (temporal) connection structure that binds the word assemblies in line with the relations expressed by the sentence. To this end, the NBA consists of 'structure assemblies' such as N_x and N_u in Figure 2 (representing nouns) and S_y (sentence) and V_z (verb). The word assemblies can bind to structure assemblies of their kind, and structure assemblies can bind to each other in line with the sentence structure. The binding process results from specific 'conditional connections' (outlined below).

As noted in [8], humans have the ability to include new words and names in (even novel) sentences structures, exemplified here with the name *Sonnet*. However, the ability to include novel words in sentences is restricted to new words based on familiar phonemes. In the NBA this ability results from a phonology blackboard in which new names can be formed based on familiar phonemes [9]. The phonology blackboard interacts with the sentence blackboard, which allows the incorporation of the new name in a sentence structure [10]. Here, Sonnet activates a set of phonemes (illustrated with the pseudo phonemes *so*, *nn*, and *et*), which bind to a structure assembly in the phonology blackboard (W1). In turn, this structure assembly binds to the sentence structure in the sentence blackboard.

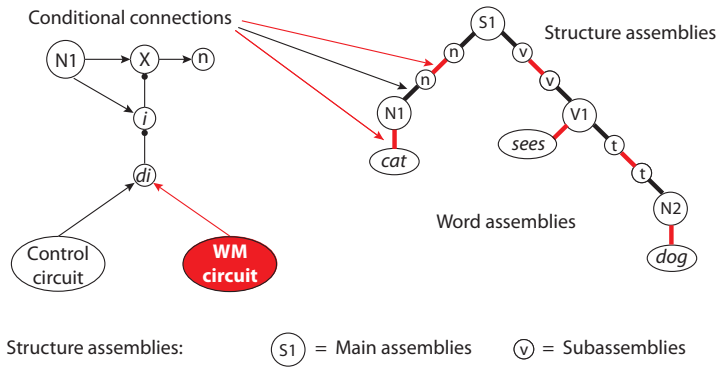


Figure 3: Binding in the sentence neural blackboard.

The binding process in the NBA is illustrated in more detail in Figure 3 with the sentence *cat sees dog*. The conditional connections, here the red and black thick lines, consist of gating circuits. In the NBA, each word assembly (e.g., of a noun) is connected to a set of structure assemblies of the same kind (all N_i assemblies in the case of a noun) with gating circuits. (The number of these connections can be reduced by representing words in the phonology blackboard first, see [9].) In turn, each structure assembly consists of a 'main assembly', such as N1, and (a set of) subassemblies, such as n or t. The connection between a main assembly and a subassembly consists of a gating circuit as well.

Structure assemblies of different kinds, such as V1 and N2, are connected by their subassemblies of the same kind. Here, their t (theme) subassemblies, which represents the fact that a verb can have a theme (object). This connection also consists of a gating circuit.

The gating circuits operate by disinhibition (di), illustrated in Figure 3. So, when N1 is active, it activates a neuron (or neuron population) X and an inhibitory neuron (or population) i. The latter inhibits X, which blocks the flow of activation. But when i itself is inhibited (by neuron or population di) activation can flow from N1 (via X) to n.

Gating circuits can be disinhibited (or 'activated') in one of two different ways. In case of gating circuits between main assemblies and subassemblies the activation results from an external control circuit that activates the di population. This is how syntactical operations affect binding in the blackboard. A control circuit could have recognized that *sees dog* represent a verb and a theme. It then activates all di populations in the gating circuits between all V_i and N_j assemblies and their t assemblies. As a result, the active V_i and N_j will activate their t subassembly.

Gating circuits between subassemblies and between word and main assemblies are activated by (specific) 'working memory' (WM) populations. A WM population remains active for a while, after initial activation, by reverberating activation in the population [11]. An active WM population binds the assemblies to which it is connected in the manner outlined below.

3.1 Variable binding in a connection path

Figure 4 illustrates how binding is achieved in the NBA. The parts (a), (b) and (c) illustrate the same binding process with increasing detail. In (a), the binding between the t subassemblies of V1 (or V1-t) and N2 (N2-t) in Figure 3 is repeated. Figure 4b illustrates that this binding is based on a 'connection matrix', which consists of columns and rows of 'connection nodes' illustrated in Figure 4c.

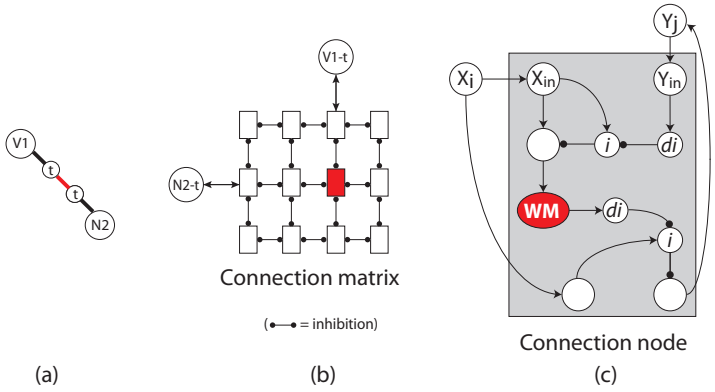


Figure 4: Binding by activity in a Connection Matrix (based on [1]).

Each specific V_i-t and N_j-t pair of subassemblies is interconnected in a specific connection node, located in a connection matrix dedicated to binding V_i-t and N_j-t subassemblies. In general, when two assemblies X_i and Y_j (e.g., V1-t and N2-t) are concurrently active in the processing of a sentence, they activate a WM population in their connection node by means of a gating circuit, as illustrated in Figure 4c. In turn, the active WM population disinhibits a gating circuit by which activation can flow from X_i to Y_j, and another such circuit, not show in (c), by which activation can flow from Y_j to X_i. As long as their WM population is active, X_i and Y_j are 'bound' because activation will flow from one to the other whenever one of them is (initially) activated.

The NBA allows any noun to bind to any verb in any thematic role using dedicated connection matrices. Also, the NBA has structure assemblies that can bind to other structure assemblies, such as S1 in Figure 3, or clause structure assemblies ([1]). In this way, hierarchical sentence structures can be represented, such as relative or complement clauses.

A given word assembly can bind to different structure assemblies at the same time, allowing the creation of sentence structures in which words are repeated (e.g., *cat sees cat*). Yet, word assemblies remain is situ in this way, so words in sentence structures are always accessible and grounded.

The role of the NBA is thus to establish a 'connection path' for any sentence structure that can be formed in a language. Although it has been questioned whether this would be possible [8], an available connection path is in fact a necessary condition for any model of (neuronal) variable binding. That is, whenever two neuronal representations are bound in a functional way, there is some kind of (direct or indirect) connection path between those representations. This path is needed to produce (meaningful) behavior based on the functional binding at hand. For example, to answer questions about the relation expressed by the binding [10]. The ability of the brain to establish such connection paths derives from its 'small-word' like connection structure [12]. The NBA indeed has such a structure [1].

In the NBA the connection matrices are given. But in a pilot simulation [13] we investigated the possible development of such a connection structure. The simulations indicated that a structure like a connection matrix could arise during development, based on initially random activation and connection patterns. But, of course, each node in the connection matrix would also have to have the circuits as illustrated in Figure 4c for the binding process to occur.

3.2 Content addressable sentence structures

The role of the NBA to (temporarily) create a connection path needed for the production of behavior is illustrated in Figure 5. This figure illustrates how the NBA can answer (binding) questions (for a simulation, see [1]).

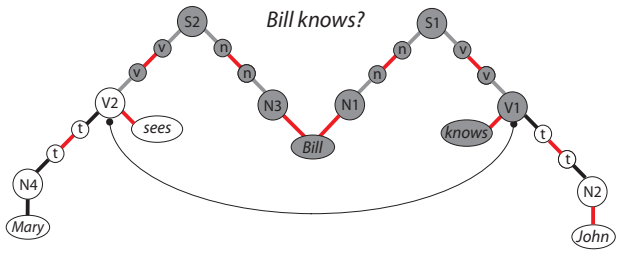


Figure 5: Content addressable sentence structures and competition in the NBA in answering (binding) questions. Grey nodes are active.

The structure of the sentence *Bill knows John*, stored in the NBA, can be used to answer a (binding) question like "What does Bill know?" (or *Bill knows?*). The question activates the assemblies for *Bill* and *knows* and provides information that *Bill* is the subject of *knows*. This results in the activation of the sentence structure *Bill knows*. The question also asks for the theme of the relation *Bill knows*, which can be used to activate the gating circuits for the theme (t) subassemblies. This will initiate a flow of activation from V1 to N2 and thus to *John* in Figure 5, producing the answer to the question.

The NBA can correctly produce the answer even when, for example, *Bill* is a constituent of other sentences as well, as in *Bill sees Mary*. The representation of *Bill* is the same in situ word assembly in both sentences (which directly derives from the notion of in situ representation). As noted above, they are distinguished in the NBA by the binding with different noun main assemblies (here, N1 vs N3). The question *Bill knows?* activates *Bill*, which in turn activates both N1 and N3 to which it is bound. Because the question entails that *Bill* is the subject, this results in the activation of both S1 and S2.

However, the question also identifies *knows* as the verb and not *sees*. This results in the activation of V1 which inhibits the activation of V2, because main assemblies of the same kind inhibit each other in the NBA [1]. In this way, asking for the theme produces the correct answer *John* (bound to *knows*) instead of the incorrect answer *Mary* (bound to *sees*).

Figure 5 illustrates the role of a connection path in producing behavior. The question provides sensory information resulting in the activation of *Bill knows*. The answer (*John*) is produced by a motor program that would be activated by the in situ word assembly of *John*. If there were not a connection path that would link the sensory information (*Bill knows*) to the motor activation (*John*) it is difficult to see how this behavior could be produced. Yet, this kind of behavior is possible for any arbitrary binding of words in (potentially novel) sentence structures or even new words in sentence structures, as in Figure 2. In each of these cases, the behavior of answering a question (probing for relation information or binding) depends on connecting sensory information to motor activation, in a manner that satisfies the binding relations expressed by the relation at hand. The NBA produces this connection path due to its small-world like connection structure and its ability to bind arbitrary relations by the binding process outlined above.

Figure 5 also illustrates the role of content addressability in the NBA. Due to the in situ nature of word (concept) representation, the question *Bill knows?* directly activates the word assemblies *Bill* and *knows*, Because they are bound to the sentence structure in the NBA (i.e., because word representations remain in situ in any sentence structure) the activation of *Bill* and *knows* directly activates the partial sentence structure *Bill knows*.

It seems that the structure of a question itself indicates that sentence structures are content addressable. After all, the question *Bill knows?* presents the words *Bill* and *knows*, indicates that *Bill* is the subject of *knows* and indicates that the response should be the theme. All of this information is precisely the information needed to directly activate the partial sentence structure of *Bill knows* (instead of other partial structures such as *Bill sees*) and to initiate the flow of activation to produce the answer *John*. It is difficult to see how this process could be more direct and thus faster than in this way. Given the evolutionary pressure on fast and yet meaningful behavior, it seems that the NBA fits these constraints in an optimal way.

4 Sentence processing in the NBA

Figure 6 illustrates how sentence processing can proceed in the NBA [14]. Here, sentence processing results from the interaction between a 'control network' and the activation in the neural blackboard. Figure 6 illustrates the processing of *cat sees dog* presented in Figure 3.

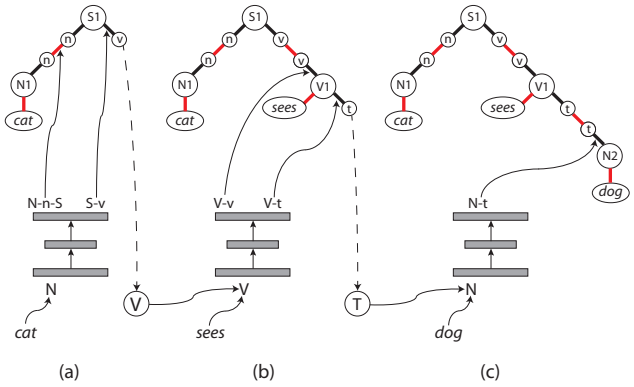


Figure 6: Sentence processing in the NBA (based on [14]).

Sentence processing is incremental in the NBA. The input to the control network consists of word information (e.g. Noun, Verb) and predictions based on the activation in the blackboard that arise during processing. Based on its input, the control network has learned to activate the gating circuits in the NBA needed to create the sentence structure. In (a) *cat* is recognized as a noun and the network activates the binding between S1 and N1 with their n subassemblies. It also activates the v subassembly of S1, which in turn activates a 'prediction node' V, predicting the occurrence of a verb of the main clause. In (b) the control network, based on the combination of the prediction node V and the occurrence of the verb *sees*, activates the v subassembly of V1, which results in the binding of S and V. It also activates the t subassembly of V1 and a prediction node of a theme (T),

anticipating an object of the verb. In (c), the combination of the T node and the noun *dog* results in the binding of V1 and N2 with their t subassemblies.

In [14] we trained a control network with a set of training sentences with different syntactic forms (e.g., subject, theme, relative clause, complement clause). The network could then control a substantially larger set of sentences in which these syntactic forms were combined in different ways. For example, the control network could produce the correct binding operations in sentences with arbitrary deep (center) embeddings. However, the binding process in the NBA starts to break down with these sentences after one or two embeddings, due to competitions with the connection matrices involved [1, 14]. Hence, the NBA could account for both competence and performance aspects of language processing.

4.1 Simulating neuronal activation during sentence processing

Brain activation observed with fMRI indicates that neuronal activation increases for longer and more complex sentences [15]. Observations with intracranial electrodes provided more detailed information [16]. In particular, when (sub) phrases in a sentence are formed (or 'merged' in linguistic terms) neuronal activity first increases but then briefly decreases.

Figure 7 shows the activity in the NBA when the sentence *Ten sad student of Bill Gates will move* from [16] is processed (taking *Bill Gates* as a single noun). Figure 7 (left) shows the basic structure of the sentence in the NBA. Figure 7 (right) shows the sum of all activity (Total) in the NBA, and the sum activities of the main assemblies (MA), the subassemblies (SA), and the WM populations in the connection matrices. All assemblies and gating circuits are modeled with Wilson Cowan population dynamics [17]. MA and SA assemblies have reverberating activity as well (until they are inhibited). Activation values are normalized to fit in one plot.

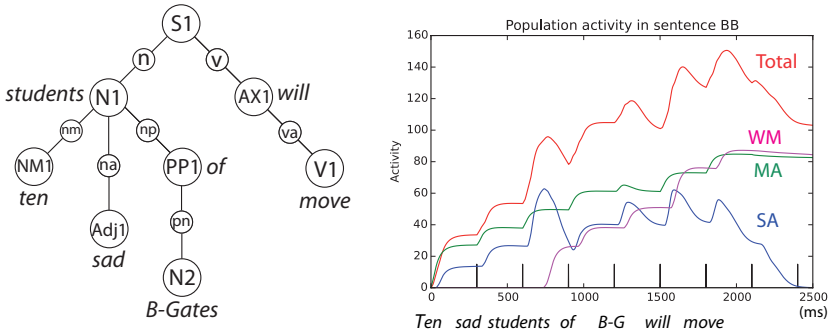


Figure 7: NBA activity in processing *Ten sad student of Bill-Gates will move*.

The sum of all activity resembles the patterns observed in [16]. For example, after *Ten sad students* (forming a phrase) activation increases but then drops, as with other phrases in the sentence. This is in particular due to the subassemblies, which are inhibited by feedback from their connection matrix when a binding has succeeded. But overall activation increases with sentence length and complexity [15].

Figure 8 illustrates the neural activity in the NBA when it resolves an unproblematic ambiguity (UPA) presented in [2]. Here, it is UPA13a (*Bird found in room died*) vs UPA13b (*Bird found in room debris*). The NBA resolves ambiguities by a competition between conflicting WM populations [18]. The NBA structure of both sentences and the conflicts (red dashed lines) are illustrated in the figure (left).

The assumption is that UPA13a is the basis interpretation. So, *found* is the verb of a relative clause bound to *bird*. Simulation shows that this incremental sentence processing proceeds without problems. So, the processing of UPA13b starts in the same way. The figure shows the individual WM activations obtained with UPA13b, which reveal the bindings achieved (the sum of this activity would result in a plot as in Figure 7). When *bird* (N1) is presented it binds to S1, indicated by the black line S1-N1 representing the WM binding population. After the initial activation by the word, this population remains active at a sustained level, so the binding remains intact during (and after)

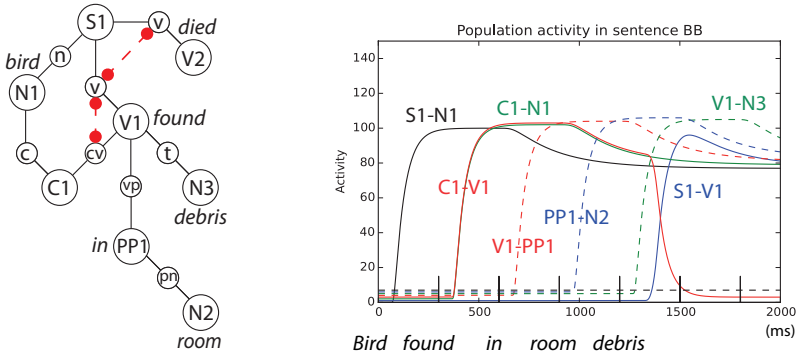


Figure 8: NBA activity in processing *Bird found in room debris*.

the sentence processing. The combination *bird found* is interpreted as *found* (V1) being the verb of a relative clause bound to N1 (as in UPA13a). This binding succeeds, as indicated by the WM populations C1-N1 (green line) and C1-V1 (red line). The bindings of *found in the room* proceed without problems, indicated by the WM populations V1-PP1 (red dash) and PP1-N2 (blue dash).

However, the last word *debris* provides a different interpretation of *found*. Instead of the verb of a relative clause it is the main verb of the sentence. This interpretation initiates the activation of the WM population V1-N3 (green dash), which represents the binding of *debris* as theme of *found*. It also initiates the activation of the WM population S1-V1 (blue line) which represents the binding of *found* as the main verb. This binding is in conflict with the binding C1-V1 (red line). The competition between the WM populations results in the decline of the activation of C1-V1, which resolves the binding conflict. Hence, ambiguity resolution in the NBA results from dynamic competitions in the architecture [18].

Alternatively, UPA13b could be the basis interpretation. Then *found* is the main verb directly. In that case *died* causes a conflict in UPA13a. However, simulations show that this conflict cannot be resolved because V1 blocks the binding of V2 (*died*) to S1. Also, V1 cannot bind anymore as verb to a relative clause of *bird* (N1), because the activations of N1 and V1 have been or are inhibited by other main assemblies (N2 and V2 respectively). Hence, this ambiguity cannot be resolved by the NBA. However, this ambiguity is similar to the classic garden path sentence *The horse raced past de barn fell* [19], which indicates that the NBA can account for both ambiguity resolution and garden path effects [18].

5 Conclusions

Neural Blackboard Architectures (NBAs) can account for variable binding and (novel) combinatorial processing in neural terms. They also satisfy a number of characteristics of brain representation and processing. Conceptual representations are always in situ, also when they are a part of a combinatorial structure. Hence, combinatorial structures are content addressable. This allows their direct activation based on input information. NBAs also provide the connection paths that are needed to account for the production of behavior. The sentence NBA can account for observed patterns of neuronal activation during sentence processing. It can also handle linguistics phenomena such as ambiguity resolution and can account for garden path sentences by the dynamic competitions in the architecture.

Acknowledgments

The work of Frank van der Velde was funded by the project ConCreTe. The project ConCreTe acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 611733. The research of Marc de Kamps has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 604102 (HBP) (Ref: Article II.30. of the Grant Agreement).

References

1] van der Velde, F. & de Kamps, M. (2006). Neural blackboard architectures of combinatorial structures in cognition. *Behavioral and Brain Sciences, 29*, 37-70.

[2] Lewis, R. L. (1993). *An Architecturally-based Theory of Human Sentence Comprehension*. Thesis Carnegie Mellon University, Pittsburgh, PA.

[3] Hebb, D. O. (1949). *The organisation of behaviour*. New York: Wiley.

[4] van der Velde, F & de Kamps, M. (2011). Compositional connectionist structures based on in situ grounded representations. *Connection Science, 23*, 97-107.

[5] van der Velde, F (2015). Communication, concepts and grounding. *Neural networks, 62*, 112 - 117.

[6] Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

[7] Van der Velde, F. & de Kamps, M. (2001). From knowing what to knowing where: Modeling object-based attention with feedback disinhibition of activation. *Journal of Cognitive Neuroscience, 13(4)*, 479-491.

[8] Feldman, J. (2013). The neural binding problem(s). *Cognitive Neurodynamics, 7*, 1-11.

[9] Van der Velde, F. & de Kamps, M. (2006). From neural dynamics to true combinatorial structures (reply to commentaries). *Behavioral and Brain Sciences, 29*, 88-108.

[10] van der Velde, F. & de Kamps, M. (2015). The necessity of connection structures in neural models of variable binding. *Cognitive neurodynamics, 9*, 359-370. DOI 10.1007/s11571-015-9331-7

[11] Amit, D. (1995). The Hebbian paradigm reintegrated: Local reverberations as internal representations. *Behavioral and Brain Sciences, 18*, 617-657.

[12] Shanahan, M. (2010). *Embodiment and the inner life*. Oxford: Oxford University Press.

[13] van der Velde, F. & de Kamps, M. (2011). Development of a connection matrix for productive grounded cognition. *IEEE International Conference on Development and Learning (ICDL)* (Vol 2), 1-6. DOI: 10.1109/DEVLRN.2011.6037343

[14] van der Velde, F & de Kamps, M. (2010). Learning of control in a neural architecture of grounded language processing. *Cognitive Systems Research, 11*, 93-107.

[15] Pallier, C., Devauchelle, A-D. & Dehaene, S. (2011). Cortical representation of the constituent structure of sentences. *PNAS, 108*, 2522-27.

[16] Nelson, M. J., El Karoui, I., Rangarajan, V., Pallier, C., Parvizi, J., Cohen, L., Naccache, L. & Dehaene, S. (2014). Constituent structure representations revealed with intracranial data. *Society for Neuroscience Annual Meeting*. Washington, DC, USA.

[17] Wilson, H. R. & Cowan, J. D (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal, 12*, 1-24.

[18] van der Velde, F. & de Kamps, M. (2015). Ambiguity resolution in a Neural Blackboard Architecture for sentence structure. In Tarek R. Besold & Kai-Uwe Kühnberger (eds.), *Proceedings of the KI 2015 Workshop on Neural-Cognitive Integration*. Dresden, Germany.

[19] Bever, T. G. (1970). The cognitive basis for linguistic structures. In Hayes, J. R., (ed.) *Cognition and the Development of Language*. New York: Wiley.

Request confirmation networks for neuro-symbolic script execution

Joscha Bach[*]
Program for Evolutionary Dynamics
Harvard University
Cambridge, MA 02138
joscha@mit.edu

Priska Herger
micropsi industries GmbH
Berlin, Germany
priska@micropsi-industries.com

Abstract

We propose Request Confirmation Networks (ReCoNs) to combine neural learning with the sequential, hierarchical detection of sensory features, and to facilitate planning and script execution. ReCoNs are spreading activation networks with units that contain an activation and a state, and are connected with typed directed links that indicate partonomic relations and spatial or temporal succession. By passing activation along the links, ReCoNs can perform both neural computations and controlled script execution. We demonstrate an application of ReCoNs in the cognitive architecture MicroPsi2 for an active perception task in a virtual environment.

1 Introduction

MicroPsi is a cognitive architecture that combines neuro-symbolic representations [1] with situated perception and a motivational system [2], [3]. MicroPsi agents live in an open environment that they have to explore and navigate; currently we are using the game environment Minecraft [4]. To this end, agents require mechanisms for bottom-up/top-down perception, reinforcement learning, motivation, decision making and action execution. We are using MicroPsi to study how to combine perceptual and conceptual representations, and to facilitate autonomous learning with full perceptual grounding.

Cognitive architectures with perceptual grounding require a way to combine symbolic and subsymbolic operations: planning, communication and reasoning rely on discrete, symbolic representations, while fine-grained visual and motor interaction requires distributed representations. The standard solution consists in a hybrid architecture that combines a neural network layer with a carefully crafted model that learns to compress the perceptual input, and a layer that facilitates deliberation and control using symbolic operations and uses the extracted regularities [5]. However, while such a dual architecture appears to be a straightforward solution from an engineering point of view, we believe that there is a continuum between perceptual and conceptual representations, i.e. that both should use the same set of representational mechanisms, and primarily differ in the operations that are performed upon them. In our view, symbolic/localist representations are best understood as a special case of subsymbolic/distributed representations, for instance ones where the weights of the connecting links are close to discrete values. Highly localist features often emerge in neural learning, and rules expressed as discrete-valued links can be used to initialize a network for capturing more detailed, distributed features (see KBANN [6]).

MicroPsi's representations combine neural network principles and symbolic operations via recurrent spreading activation networks with directed links. The individual nodes in the network process

[*] http://micropsi.com

information by summing up the activation that enters the nodes through slots, and calculating a function for each of their gates, which are the origin of links to other nodes. Node types differ by the number of gates and slots they have and by the functions and parameters of their gates. Typed links can be expressed by the type of their gate of origin.

The most common node type in earlier MicroPsi implementations is called a concept node. Concept nodes possess seven gate types (with approximate semantics in parentheses): *gen* (associated), *por* (successor), *ret* (predecessor), *sur* (part-of), *sub* (has-part), *exp* (is-exemplar-of), *cat* (is-a). Concept nodes can be used to express hierarchical scripts [7] by linking sequences of events and actions using *por*/*ret*, and subsuming these sequences into hierarchies using *sub*/*sur*.

In MicroPsi2, a perceptual representation amounts to a hierarchical script to test for the presence of the object in the environment. At each level of the hierarchy, the script contains disjunctions and subjunctions of substeps, which bottom out in distributed substeps and eventually in sensor nodes that reflect measurements in the environment and actuator nodes that will move the agent or its sensors. Recognizing an object requires the execution of this hierarchical script. In the past, this required a central executive that used a combination of explicit backtracking and activation propagation. We have replaced this mechanism with a completely distributed mode of execution called request confirmation network that only requires the propagation of activation along the links of connected nodes.

2 Request confirmation networks

Modeling perception in a cognitive architecture requires an implementation of top-down/bottom-up processing, in which sensors delivers cues (bottom-up) to activate higher-level features, while perceptual hypotheses produce predictions of features that have to be verified by active sensing (top-down). When using a neural network paradigm, the combination of bottom-up and top-down processing requires recurrency. Request confirmation networks are a solution to combine constrained recurrency with the execution of action sequences. They provide a neural solution for the implementation of sensorimotor scripts.

Request Confirmation Networks (ReCoN) are implemented within MicroPsi2's formalism but are in fact a more general paradigm for auto-executable networks of stateful nodes with typed links. We therefore provide a general definition that is independent of our implementation.

A request confirmation network can be defined as a set of units $\mathbb{U}$ and edges $\mathbb{E}$ with

$$\mathbb{U} = \{\text{script nodes} \cup \text{terminal nodes}\}$$
$$\mathbb{E} = \{\text{sub, sur, por, ret}\}$$

A script node has a state s where

$$s \in \{\text{inactive, requested, active, suppressed, waiting, true, confirmed, failed}\}$$

and an activation $a \in \mathbb{R}^n$, which can be used to store additional state. (In MicroPsi2, the activation is used to store the results of combining feature activations along the *sub* links). A terminal node performs a measurement and has a state $s \in \{\text{inactive, active, confirmed}\}$, and an activation $a \in \mathbb{R}^n$, which represents the value obtained from the measurement. An edge is defined by $\langle u_s, u_t, type \in \{\text{por, ret, sub, sur}\}, w \in \mathbb{R}^n \rangle$, whereby u_s and u_t denote the source and target unit, *por* connects to a successor node, *ret* to a predecessor node, *sub* to a parent node, and *sub* connects to a child node. w is a link weight with n dimensions that can be used to perform additional computations. Each pair of nodes (u_s, u_t) is either unconnected or has exactly one pair of links of the types *por*/*ret*, or *sub*/*sur*. Each script node must have at least one link of type *sub*, i.e. at least one child that is either a script node or a terminal node. Script nodes can be the source and target of links of all types, whereas terminal nodes can only be targeted by links of type *sub*, and be the origin of links of type *sur*.

ReCoNs solve the problem of constraining the flow of control in a hierarchical script on the level of individual units without using topological information, i.e. without a centralized interpreter or informing individual units about their neighboring elements. To do this, they need to store the execution state within the script elements in a distributed manner; each of these units is a state machine with one of eight states (as defined above). Each unit implements connections to its neighbors using

directional links. Initially, all units are inactive. The execution of a script starts with sending the signal 'request' to its root node. Semantically, the root node represents a hypothesis that is spelled out in the script, and that we want to test; we request the validation of the root node. After the script is executed, it will either be in the state "confirmed" or "failed" (until we turn off the 'request' signal, and the unit becomes inactive again).

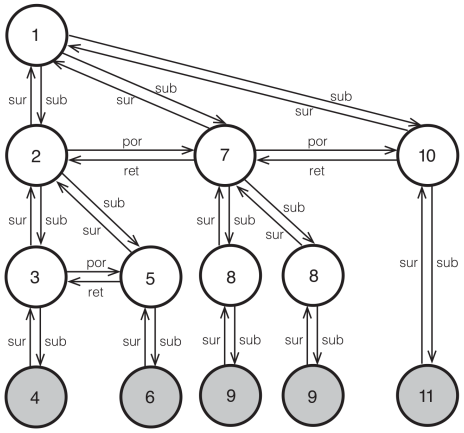


Figure 1: Example of script execution.

During the validation, it will need to validate all of its children, by sending a 'request' signal along its *sub* links. These may either form sequences or alternatives, cf. Figure 1. The former are validated in succession, the latter in parallel. Successive execution requires that units prevent their successors from becoming "active" until it is their turn; they do this by sending an 'inhibit request' signal along their *por* links. If a requested unit receives an 'inhibit request' signal, it becomes "suppressed" until its predecessor becomes "true" and turns off the signal. Active elements tell their parent that it should wait for their validation to finish by sending them a 'wait' message. Each active element will remain in the "waiting" state until either one of its children sends a 'confirm' message (in which case it turns into the state "true", or no more children ask it to wait (in which case the element will turn into the state "failed"). In sequences, we also need to ensure that only the last element of a sequence can confirm the parent request, so each unit sends an 'inhibit confirm' signal via *ret* to its predecessors. The last unit in a sequence will not receive an 'inhibit confirm' signal, and can turn into the state "confirmed" to send the 'confirm' signal to the parent. The execution of the script can be reset or interrupted at any time, by removing the 'request' from its root node.

The ReCoN can be used to execute a script with discrete activations, but it can also perform additional operations along the way. This is done by calculating additional activation values during the request and confirmation steps. During the confirmation step (a node turns into the state "confirmed" or "true"), the activation of that node is calculated based on the activations of its children, and the weights of the *sur* links from these children. During the requesting step, children may receive parameters from their parents which are calculated using the parent activation and the weights of the *sub* links from their parents. This mechanism can be used to adapt ReCoNs to a variety of associative classification and learning tasks.

2.1 A message passing definition of a ReCoN unit

The units of a ReCoN implement a finite state machine – as drawn out in Figure 2. This can be realized by defining the eight discrete states explicitly and using a set of messages that are distributed along the *por*, *ret*, *sub* and *sur* links, depending on the current state of each unit. These messages can be defined as {inhibit_request, inhibit_confirm, wait, confirm, fail}. They are passed as specified in Table 1.

In each state, incoming messages are evaluated and nodes either stay in the current state or switch to the next, if a condition is fulfilled. These conditions are given in Figure 2.

Unit state	POR	RET	SUB	SUR
inactive (ø)	–	–	–	–
requested (R)	inhibit request	inhibit confirm	–	wait
active (A)	inhibit request	inhibit confirm	request	wait
suppressed (S)	inhibit request	inhibit confirm	–	–
waiting (W)	inhibit request	inhibit confirm	request	wait
true (T)	–	inhibit confirm	–	–
confirmed (C)	–	inhibit confirm	–	confirm
failed (F)	inhibit request	inhibit confirm	–	–

Table 1: Message passing in ReCoNs.

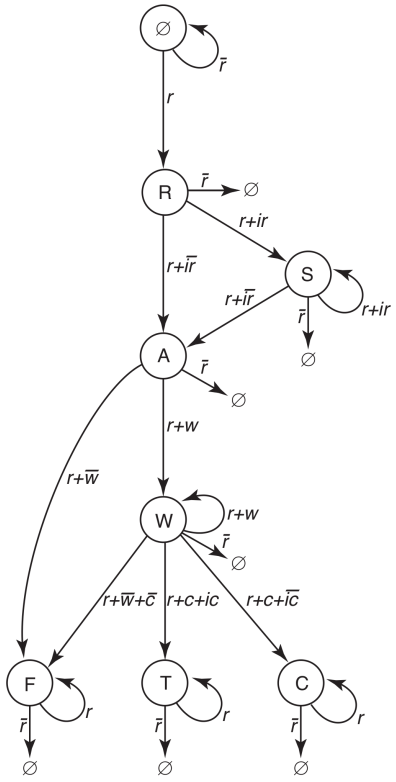


Figure 2: State transitions.

2.2 A neural definition of a ReCoN unit

It is possible to realize a ReCoN unit with an ensemble of artificial neurons. Figure 3 shows an arrangement of ten simple threshold elements with only excitatory and inhibitory links that satisfies these conditions. Let the activation of a neuron be defined as

$$\alpha_j = \begin{cases} \sum(w_{ij} \cdot \alpha_i) & \text{if } w_{ij} \cdot \alpha_i \geq 0 \text{ for all } w_{ij}, \alpha_i \\ 0 & \text{otherwise} \end{cases}$$

i.e. any incoming activation on a link with a negative weight is sufficient to entirely inhibit the neuron. A message is any activation value sent over a link that crosses the boundary between units. Units may send the message 'request' to their children (top-down), 'wait' and 'confirm' to their parents (bottom-up), 'inhibit request' to their successors, and 'inhibit confirm' to their predecessors (lateral inhibition).

We want all activity in a unit to cease as soon as the unit is no longer requested. Thus, all activation in the unit is derived from the activation of the incoming request message. The request activation is directly sent to the neurons IC, IR, and W. IC will inhibit confirm signals by predecessor units, and IR will inhibit requests to child nodes by successor units, before the current unit is confirmed. W informs the parent node that it has active (non-failed) children, by sending the 'wait' message. IR also prevents the unit from confirming prematurely; i.e. before it has received a 'confirm' message by one of its children.

IC then passes on its activation to R, and R sends out a request to the unit's children, unless it is inhibited by the predecessor node. To give predecessor nodes enough time to send the inhibition signal, the flow of activation between IC and R is delayed by DR. F becomes active as soon as the unit has no more active children, and represents the failure state of the unit. Like all states, it must receive its activation from the initial request. It cannot be allowed to fail before the unit has sent a request to its children, so F is inhibited by the predecessors 'inhibit request' message at the neuron R′. F can also not fail before the children cease sending their 'wait' message, so it is inhibited by that message. F must also allow enough time for requested children to respond with this message, so its activation is delayed by the helper neuron DF. Once F becomes active (i.e. the unit fails), it stops the 'request' to the children by inhibiting R, and the 'wait' to the parent, by inhibiting W. (F cannot get its activation directly from R, because F will later inhibit R and would thus remove its own activation; therefore it is activated through the helper neuron R′).

The neuron T represents that the unit has been confirmed (true). It receives its activation from the original request, but is not allowed to become active through the inhibition by IC. Only if IC is turned off by the 'confirm' message of one of its children, T becomes active and will turn off W (the 'wait' message to the parent), R (the request of the children) and IR (the inhibition of the successors, and itself). If the unit has no successors (i.e. receives no 'inhibit confirm' message), it will signal T's value via C as a 'confirm' message to the unit's parent. This is just one of many possible ways in which a ReCoN unit could be realized with artificial neurons.

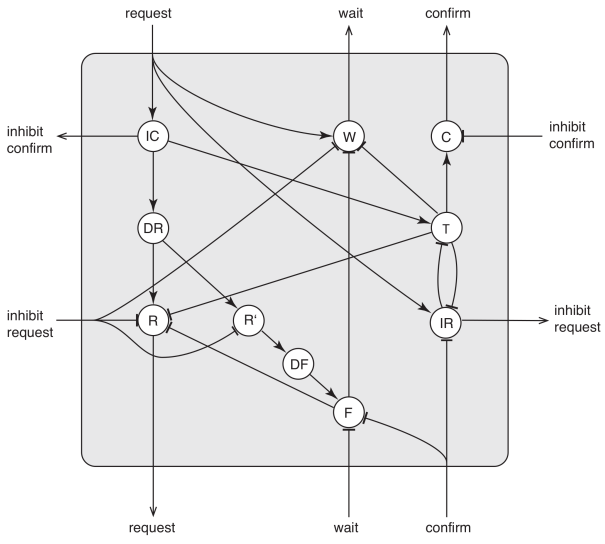


Figure 3: Schematic of a neural ReCoN specification.

In MicroPsi2, the functionality of this circuitry is embedded in a single node, which requires a only one calculation step and reduces memory usage; cf. section 3.1.

3 Implementation

3.1 A compact implementation of ReCoNs

Rather than explicitly modeling the state machine of a ReCoN unit, this implementation makes use of the fact that the state of a ReCoN unit is fully determined by the activation of connected nodes in the previous time step. States are implemented as a set of simple arithmetic rules operating on incoming activation from connected nodes and the node itself. Nodes can link to themselves using "*gen* loops" to store the states "true", "confirmed", and "failed". Activation as well as all link weights are numbers enabling real-value information processing and neural learning.

In this implementation, *por*/*ret* activation $\in \{-1, 0, 1\}$ is used to control the flow of *sub* requests ($\{0, 1\}$) and *sur* confirmation. The real-valued *sur* activations can be interpreted as probabilistic in the range $[0, 1]$ or take on the value -1 to indicate a solid fail. They control *por*/*ret* activation in the next higher layer.

Activation is spread in discrete steps. Each step consists of two phases: Propagation and calculation. Propagation is simply: $\mathbf{z} = \mathbf{W} \cdot \mathbf{a}$ where $\mathbf{W}$ is a matrix of link weights and $\mathbf{a}$ is the activation vector. Calculation is $f_{gate}(f_{node}(\mathbf{z}))$, where f_{gate} is an activation function specified per link type and f_{node} implements the ReCoN behavior. The following definitions describe the behavior of the node functions f_{node} per link type.

$$f^{gen}_{node} = \begin{cases} z^{sur} & \text{if } (z^{gen} \cdot z^{sub} = 0) \vee (\exists \,\text{link}^{por} \wedge z^{por} = 0) \\ z^{gen} \cdot z^{sub} & \text{otherwise} \end{cases}$$

$$f^{por}_{node} = \begin{cases} 0 & \text{if } z^{sub} \leq 0 \vee (\exists \,\text{link}^{por} \wedge z^{por} \leq 0) \\ z^{sur} + z^{gen} & \text{otherwise} \end{cases}$$

$$f^{ret}_{node} = \begin{cases} 1 & \text{if } z^{por} < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$f^{sub}_{node} = \begin{cases} 0 & \text{if } z^{gen} \neq 0 \vee (\exists \,\text{link}^{por} \wedge z^{por} \leq 0) \\ z^{sub} & \text{otherwise} \end{cases}$$

$$f^{sur}_{node} = \begin{cases} 0 & \text{if } z^{sub} \leq 0 \vee (\exists \,\text{link}^{por} \wedge z^{por} \leq 0) \\ (z^{sur} + z^{gen}) \cdot z^{ret} & \text{if } \exists \,\text{link}^{ret} \\ z^{sur} + z^{gen} & \text{otherwise} \end{cases}$$

An undesirable property of this implementation is that new activation is not solely based on incoming activation but also depends on local knowledge about the existence of *por*/*ret* links[1].

3.2 Using ReCoNs for an active perception task

Active perception with ReCoNs is based on learned representations that are encoded in the structure and weights of links. Whenever the system encounters a new situation (for instance, after a locomotion action) it will form a model of its environment as a combination of verifiable hypotheses. At higher levels, these hypotheses can contain object representations and geometric relations between objects, at the lower levels features and eventually input patches that make up the visual structure of these objects. In the basic variant implemented so far, hypotheses are based on a single autoencoder that captures features of the scene as a whole.

Hypotheses are verified by activating their top-most node with a request (i.e. sending activation to its *sub* slot). From there, activation spreads downwards through the sub-hypotheses defined by the node's children, which are verified sequentially. If all existing hypotheses fail, the agent constructs a new one, scanning its visual field and connecting the localized feature nodes into a hierarchy. We

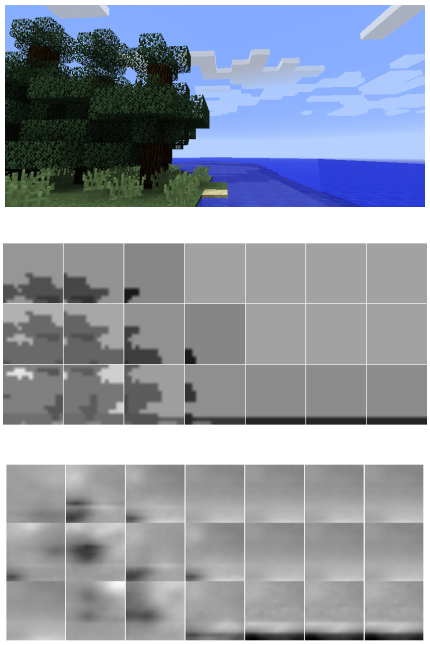
[1] The source code is available from https://github.com/joschabach/micropsi2. Formulas found in the code are slightly more complex as they include additional features like time-based failing and searchability which are not relevant in this context.

built a MicroPsi agent that moves about in a Minecraft world along a transition graph, samples its visual environment and learns to recognize scenes on the graph with a request confirmation network.

The agent does not process the entire visual field at once, but uses a rectangular fovea with a 16×16 sensor matrix. Using actuators, it can move this fovea to fixate a point (and thus a 16×16 patch) in its field of vision. The possible fixation points are constrained to allow for a 50 percent overlap between neighboring patches.

These patches are learned using a denoising autoencoder [8] with zero-masking Bernoulli noise and a corruption level of 0.3. A 2-dimensional perspective projection of the Minecraft block world at a given position serves as input to the autoencoder. Figure 4b shows, for one such position, the visual field as the standard Minecraft client would present it to a human player (top), the projection provided as visual input to the agent (center), and a visualization of the features at that position as learned by the autoencoder (bottom).

(a) Transition graph of the agent.

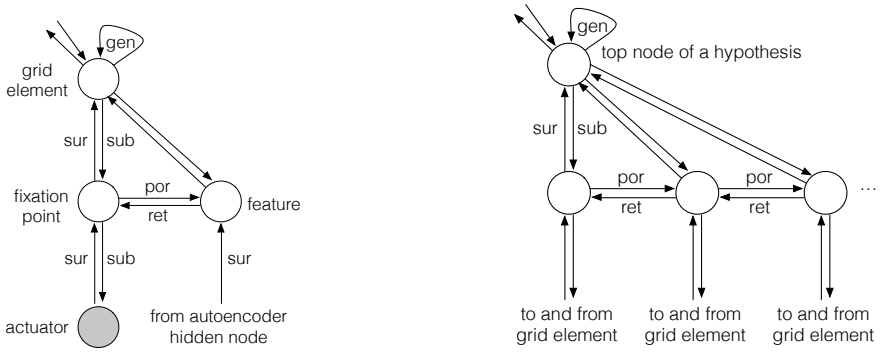


(b) Training data – Minecraft vanilla client (top), visual input (center), learnt features (bottom).

Figure 4: Action space and input sample of a MicroPsi2 agent in Minecraft

The autoencoder hidden nodes play the role of terminal nodes to the ReCoN units. Features detected by the hidden nodes are fed into a localized feature grid, where each combination of feature and fixation point is expressed as a pair of *por*/*ret* connected ReCoN units. Each pair is *sur*/*sub* connected to a parent unit. The first node of the *por*/*ret* sequence is *sub*/*sur* connected to the actuator node that was active when the feature was detected, the second has an incoming *sur* link from one of the hidden nodes of the autoencoder (see Figure 5a). The grid as a whole associates expected features with all possible fixation points. Each grid element represents a feature at a given location in the agent's visual field, together with the action required to reach that location. Since hypotheses are constructed as sequences of grid nodes, which contain actuators for moving the fovea, they are simple sensorimotor scripts. New hypotheses are formed based on those features that are deemed relevant to the current situation. Here, relevance is simply defined by an activation value exceeding a threshold of 0.8. Figure 5b shows the structure and connectivity of such a hypothesis.

To evaluate the functionality of this implementation, we let the agent move around in the Minecraft world using a number of fixed locations. As expected the agent learned hypotheses for all locations, subsequently stopped forming new hypotheses, and successfully recognized all the locations.



(a) An element of the feature grid.

(b) Structure and connectivity of a hypothesis.

Figure 5: Connectivity schemas used for scene recognition.

4 Summary and current work

Request confirmation networks (ReCoNs) provide an elegant way to integrate hierarchical scripts into neuro-symbolic architectures. They combine top-down/bottom-up processing with sequences to form sensorimotor scripts. We gave a definition of a ReCoN unit as a message passing state machine and a second, neural definition using simple threshold elements. The implementation in our cognitive architecture MicroPsi2 uses dedicated nodes as ReCoN units. We tested our model using an autoencoder over visual input as input layer, and let our agent learn and recognize scenes in the game Minecraft. The agent also uses these representations for symbolic operations such as protocol formation and basic policy learning. ReCoNs do not imply a novel way of learning, but a representational formalism – which also limits the ways in which we can quantitatively evaluate them. We are currently working towards combining ReCoN units with long short-term memory [9] to learn the temporal/causal dynamics in Minecraft, for instance, to predict which hypothesis is the most likely candidate at any given time. We are also working on making hypotheses more robust against different types of variance: alternative views of a scene or object, presence /absence of features, and forming hierarchies of hypotheses.

Acknowledgments

We would like to thank Dominik Welland and Ronnie Vuine (micropsi industries) who are vitally involved in our recent and ongoing research and development of request confirmation networks.

References

[1] Ioannis Hatzilygeroudis and Jim Prentzas. Neuro-symbolic approaches for knowledge representation in expert systems. *International Journal of Hybrid Intelligent Systems*, 1(3-4):111–126, 2004.

[2] Joscha Bach. *Principles of synthetic intelligence PSI: an architecture of motivated cognition*, volume 4. Oxford University Press, 2009.

[3] Joscha Bach. Modeling motivation in micropsi 2. In *Artificial General Intelligence*, pages 3–13. Springer, 2015.

[4] Daniel Short. Teaching scientific concepts using a virtual world – minecraft. *Teaching Science - Journal of the Australian Science Teachers Association*, 58(3):55, 2012.

[5] Stefan Wermter and Ron Sun, editors. *Hybrid neural systems*. Springer Science & Business Media, 2000.

[6] Geoffrey G Towell and Jude W Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1):119–165, 1994.

[7] Roger C Schank and Robert P Abelson. *Scripts, plans, and knowledge*. Yale University, 1975.

[8] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[11] Ronald J Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.

Tree-Structured Composition in Neural Networks without Tree-Structured Architectures

Samuel R. Bowman, Christopher D. Manning, and Christopher Potts
Stanford University
Stanford, CA 94305-2150
{sbowman, manning, cgpotts}@stanford.edu

Abstract

Tree-structured neural networks encode a particular tree geometry for a sentence in the network design. However, these models have at best only slightly outperformed simpler sequence-based models. We hypothesize that neural sequence models like LSTMs are in fact able to discover and implicitly use recursive compositional structure, at least for tasks with clear cues to that structure in the data. We demonstrate this possibility using an artificial data task for which recursive compositional structure is crucial, and find an LSTM-based sequence model can indeed learn to exploit the underlying tree structure. However, its performance consistently lags behind that of tree models, even on large training sets, suggesting that tree-structured models are more effective at exploiting recursive structure.

1 Introduction

Neural networks that encode sentences as real-valued vectors have been successfully used in a wide array of NLP tasks, including translation [1], parsing [2], and sentiment analysis [3]. These models are generally either sequence models based on recurrent neural networks, which build representations incrementally from left to right [4, 1], or tree-structured models based on *recursive* neural networks, which build representations incrementally according to the hierarchical structure of linguistic phrases [5, 6].

While both model classes perform well on many tasks, and both are under active development, tree models are often presented as the more principled choice, since they align with standard linguistic assumptions about constituent structure and the compositional derivation of complex meanings. Nevertheless, tree models have not shown the kinds of dramatic performance improvements over sequence models that their billing would lead one to expect: head-to-head comparisons with sequence models show either modest improvements [3] or none at all [7].

We propose a possible explanation for these results: standard sequence models can learn to exploit recursive syntactic structure in generating representations of sentence meaning, thereby learning to use the structure that tree models are explicitly designed around. This requires that sequence models be able to identify syntactic structure in natural language. We believe this is plausible on the basis of other recent research [8, 9]. In this paper, we evaluate whether LSTM sequence models are able to use such structure to guide interpretation, focusing on cases where syntactic structure is clearly indicated in the data.

We compare standard tree and sequence models on their handling of recursive structure by training the models on sentences whose length and recursion depth are limited, and then testing them on longer and more complex sentences, such that only models that exploit the recursive structure will be able to generalize in a way that yields correct interpretations for these test sentences. Our methods extend those of our earlier work in [10], which introduces an experiment and corresponding artificial dataset to test this ability in two tree models. We adapt that experiment to sequence models by

not p_3	$\wedge$	p_3
p_3	$\sqsubset$	p_3 *or* p_2
(*not* p_2) *and* p_6	$\mid$	*not*(p_6 *or* (p_5 *or* p_3))
p_4 *or* (*not*(((p_1 *or* p_6) *or* p_4))	$\sqsubset$	*not*((((*not* p_6) *or* (*not* p_4)) *and* (*not* p_5)) *and* (p_6 *and* p_6))

Table 1: Examples of short to moderate length pairs from the artificial data introduced in [10]. We only show the parentheses that are needed to disambiguate the sentences rather than the full binary bracketings that the models use.

decorating the statements with an explicit bracketing, and we use this design to compare an LSTM sequence model with three tree models, with a focus on what data each model needs in order to generalize well.

As in [10], we find that standard tree neural networks are able to make the necessary generalizations, with their performance decaying gradually as the structures in the test set grow in size. We additionally find that extending the training set to include larger structures mitigates this decay. Then considering sequence models, we find that a single-layer LSTM is also able to generalize to unseen large structures, but that it does this only when trained on a larger and more complex training set than is needed by the tree models to reach the same generalization performance.

Our results engage with those of [8] and [2], who find that sequence models can learn to recognize syntactic structure in natural language, at least when trained on explicitly syntactic tasks. The simplest model presented in [8] uses an LSTM sequence model to encode each sentence as a vector, and then generates a linearized parse (a sequence of brackets and constituent labels) with high accuracy using only the information present in the vector. This shows that the LSTM is able to identify the correct syntactic structures and also hints that it is able to develop a generalizable method for encoding these structures in vectors. However, the massive size of the dataset needed to train that model, 250M tokens, leaves open the possibility that it primarily learns to generate only tree structures that it has already seen, representing them as simple hashes—which would not capture unseen tree structures—rather than as structured objects. Our experiments, though, show that LSTMs can learn to understand tree structures when given enough data, suggesting that there is no fundamental obstacle to learning this kind of structured representation. We also find, though, that sequence models lag behind tree models across the board, even on training corpora that are quite large relative to the complexity of the underlying grammar, suggesting that tree models can play a valuable role in tasks that require recursive interpretation.

2 Recursive structure in artificial data

Reasoning about entailment The data that we use define a version of the *recognizing textual entailment* task, in which the goal is to determine what kind of logical consequence relation holds between two sentences, drawing on a small fixed vocabulary of relations such as entailment, contradiction, and synonymy. This task is well suited to evaluating neural network models for sentence interpretation: models must develop comprehensive representations of the meanings of each sentence to do well at the task, but the data do not force these representations to take a specific form, allowing the model to learn whatever kind of representations it can use most effectively.

The data we use are labeled with the seven mutually exclusive logical relations of [11], which distinguish entailment in two directions ($\sqsubset$, $\sqsupset$), equivalence ($\equiv$), exhaustive and non-exhaustive contradiction ($\wedge$, $\mid$), and two types of semantic independence ($\#$, $\smile$).

The artificial language The language described in [10] (§4) is designed to highlight the use of recursive structure with minimal additional complexity. Its vocabulary consists only of six unanalyzed word types ($p_1, p_2, p_3, p_4, p_5, p_6$), *and*, *or*, and *not*. Sentences of the language can be straightforwardly interpreted as statements of propositional logic (where the six unanalyzed words types are variable names), and labeled sentence pairs can be interpreted as theorems of that logic. Some example pairs are provided in Table 1.

Crucially, the language is defined such that any sentence can be embedded under negation or conjunction to create a new sentence, allowing for arbitrary-depth recursion, and such that the scope of

negation and conjunction are determined only by bracketing with parentheses (rather than bare word order). The compositional structure of each sentence can thus be an arbitrary tree, and interpreting a sentence correctly requires using that structure.

The data come with parentheses representing a complete binary bracketing. Our models use this information in two ways. For the tree models, the parentheses are not word tokens, but rather are used in the expected way to build the tree. For the sequence model, the parentheses are word tokens with associated learned embeddings. This approach provides the models with equivalent data, so their ability to handle unseen structures can be reasonably compared.

The data Our sentence pairs are divided into thirteen bins according to the number of logical connectives (*and, or, not*) in the longer of the two sentences in each pair. We test each model on each bin separately (58k total examples, using an 80/20% train/test split) in order to evaluate how each model's performance depends on the complexity of the sentences. In three experiments, we train our models on the training portions of bins 0–3 (62k examples), 0–4 (90k), and 0–6 (160k), and test on every bin but the trivial bin 0. Capping the size of the training sentences allows us to evaluate how the models interpret the sentences: if a model's performance falls off abruptly above the cutoff, it is reasonable to conclude that it relies heavily on specific sentence structures and cannot generalize to new structures. If a model's performance decays gradually[1] with no such abrupt change, then it must have learned a more generally valid interpretation function for the language which respects its recursive structure.

3 Testing sentence models on entailment

We use the architecture depicted in Figure 1a, which builds on the one used in [10]. The model architecture uses two copies of a single sentence model (a tree or sequence model) to encode the premise and hypothesis (left and right side) expressions, and then uses those encodings as the features for a multilayer classifier which predicts one of the seven relations. Since the encodings are computed separately, the sentence models must encode complete representations of the meanings of the two sentences for the downstream model to be able to succeed.

Classifier The classifier component of the model consists of a combining layer which takes the two sentence representations as inputs, followed by two neural network layers, then a softmax classifier. For the combining layer, we use a neural tensor network (NTN, [12]) layer, which sums the output of a plain recursive/recurrent neural network layer with a vector computed using two multiplications with a learned (full rank) third-order tensor parameter:

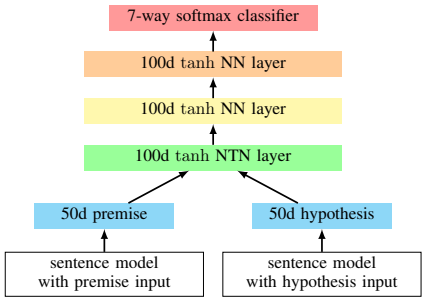
$$\vec{y}_{NN} = \tanh(\mathbf{M} \begin{bmatrix} \vec{x}^{(l)} \\ \vec{x}^{(r)} \end{bmatrix} + \vec{b}) \tag{1}$$

$$\vec{y}_{NTN} = \vec{y}_{NN} + \tanh(\vec{x}^{(l)T} \mathbf{T}^{[1\ldots n]} \vec{x}^{(r)}) \tag{2}$$

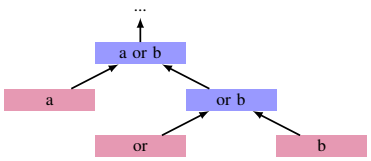
Our model is largely identical to the model from [10], but adds the two additional $\tanh$ NN layers, which we found help performance across the board, and also uses the NTN combination layer when evaluating all four models, rather than just the TreeRNTN model, so as to ensure that the sentence models are compared in as similar a setting as possible.

We only study models that encode entire sentences in fixed length vectors, and we set aside models with attention [13], a technique which gives the downstream model (here, the classifier) the potential to access each input token individually through a soft content addressing system. While attention simplifies the problem of learning complex correspondences between input and output, there is no apparent reason to believe that it should improve or harm a model's ability to track structural information like a given token's position in a tree. As such, we expect our results to reflect the same basic behaviors that would be seen in attention-based models.

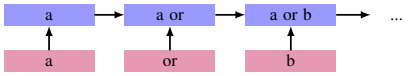
[1] Since sentences are fixed-dimensional vectors of fixed-precision floating point numbers, all models will make errors on sentences above some length, and L2 regularization (which helps overall performance) exacerbates this by discouraging the model from using the kind of numerically precise, nonlinearity-saturating functions that generalize best.



(a) The general architecture shared across models.



(b) The architecture for the tree-structured sentence models. Terminal nodes are learned embeddings and nonterminal nodes are NN, NTN, or TreeLSTM layers.



(c) The architecture for the sequence sentence model. Nodes in the lower row are learned embeddings and nodes in the upper row are LSTM layers.

Figure 1: In our model, two copies of a sentence model—based on either tree (b) or sequence (c) models—encode the two input sentences. A multilayer classifier component (a) then uses the resulting vectors to predict a label that reflects the logical relationship between the two sentences.

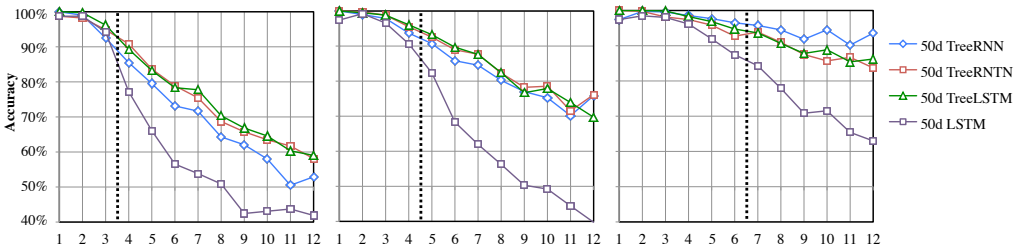
Sentence models The sentence encoding component of the model transforms the (learned) embeddings of the input words for each sentence into a single vector representing that sentence. We experiment with tree-structured models (Figure 1b) with TreeRNN (eqn. 1), TreeRNTN (eqn. 2), and TreeLSTM [3] activation functions. In addition, we use a sequence model (Figure 1c) with an LSTM activation function [14] implemented as in [15]. In experiments with a simpler non-LSTM RNN sequence model, the model tended to badly underfit the training data, and those results are not included here.

Training We randomly initialize all embeddings and layer parameters, and train them using mini-batch stochastic gradient descent with AdaDelta [16] learning rates. Our objective is the standard negative log likelihood classification objective with L2 regularization (tuned on a separate train/test split). All models were trained for 100 epochs, after which all had largely converged without significantly declining from their peak performances.

4 Results and discussion

The results are shown in Figure 2. The tree models fit the training data well, reaching 98.9, 98.8, and 98.4% overall accuracy respectively in the $\leq$6 setting, with the LSTM underfitting slightly at 94.8%. In that setting, all models generalized well to structures of familiar length, with the tree models all surpassing 97% on examples in bin 4, and the LSTM reaching 94.8%. On the longer test sentences, the tree models decay smoothly in performance across the board, while the LSTM decays more quickly and more abruptly, with a striking difference in the $\leq$4 setting, where LSTM performance falls 10% from bin 4 to bin 5, compared to 4.4% for the next worse model. However, the LSTM improves considerably with more ample training data in the $\leq$6 condition, showing only a 3% drop and generalization results better than the best model's in the $\leq$3 setting.

All four models robustly beat the simple baselines reported in [10]: the most frequent class occurs just over 50% of the time and a neural bag of words model does reasonably on the shortest examples but falls below 60% by bin 4.



(a) Training on sz. $\leq$3. (b) Training on sz. $\leq$4. (c) Training on sz. $\leq$6.

Figure 2: Test accuracy on three experiments with increasingly rich training sets. The horizontal axis on each graph divides the test set expression pairs into bins by the number of logical operators in the more complex of the two expressions in the pair. The dotted line shows the size of the largest examples in the training set in each experiment.

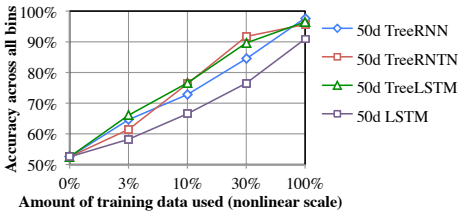


Figure 3: Learning curve for the $\leq$6 experiment.

The learning curve (Figure 3) suggests that additional data is unlikely to change these basic results. The LSTM lags behind the tree models across the curve, but appears to gain accuracy at a similar rate.

5 Conclusion

We find that all four models are able to effectively exploit a recursively defined language to interpret sentences with complex unseen structures. We find that tree models' biases allow them to do this with greater efficiency, outperforming sequence-based models substantially in every experiment. However, our sequence model is nonetheless able to generalize smoothly from seen sentence structures to unseen ones, showing that its lack of explicit recursive structure does not prevent it from recognizing recursive structure in our artificial language.

We interpret these results as evidence that both tree and sequence architectures can play valuable roles in the construction of sentence models over data with recursive syntactic structure. Tree architectures provide an explicit bias that makes it possible to efficiently learn to compositional interpretation, which is difficult for sequence models. Sequence models, on the other hand, lack this bias, but have other advantages. Since they use a consistent graph structure across examples, it is easy to accelerate minibatch training in ways that yield substantially faster training times than are possible with tree models, especially with GPUs. In addition, when sequence models integrate each word into a partial sentence representation, they have access to the entire sentence representation up to that point, which may provide valuable cues for the resolution of lexical ambiguity, which is not present in our artificial language, but is a serious concern in natural language text.

Finally, we suggest that, because of the well-supported linguistic claim that the kind of recursive structure that we study here is key to the understanding of real natural languages, there is likely to be value in developing sequence models that can more efficiently exploit this structure without fully sacrificing the flexibility that makes them succeed.

Acknowledgments

We gratefully acknowledge a Google Faculty Research Award, a gift from Bloomberg L.P., and support from the Defense Advanced Research Projects Agency (DARPA) Deep Exploration and Fil-

tering of Text (DEFT) Program under Air Force Research Laboratory (AFRL) contract no. FA8750-13-2-0040, the National Science Foundation under grant no. IIS 1159679, and the Department of the Navy, Office of Naval Research, under grant no. N00014-13-1-0287. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Google, Bloomberg L.P., DARPA, AFRL, NSF, ONR, or the US government.

References

[1] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, 2014.

[2] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proc. ACL*, 2015.

[3] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. ACL*, 2015.

[4] Jeffrey L. Elman. Finding structure in time. *Cognitive science*, 14(2), 1990.

[5] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proc. IEEE International Conference on Neural Networks*, 1996.

[6] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proc. EMNLP*, 2011.

[7] Minh-Thang Luong Li, Jiwei, Dan Jurafsky, and Eudard Hovy. When are tree structures necessary for deep learning of representations? *Proc. EMNLP*, 2015.

[8] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Proc. NIPS*, 2015.

[9] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *arXiv:1506.02078*, 2015.

[10] Samuel R. Bowman, Christopher Potts, and Christopher D. Manning. Recursive neural networks can learn logical semantics. In *Proc. of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 2015.

[11] Bill MacCartney and Christopher D. Manning. An extended model of natural logic. In *Proc. of the Eighth International Conference on Computational Semantics*, 2009.

[12] Danqi Chen, Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning new facts from knowledge bases with neural tensor networks and semantic word vectors. In *Proc. ICLR*, 2013.

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*, 2015.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8), 1997.

[15] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. In *Proc. ICLR*, 2015.

[16] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv:1212.5701*, 2012.

A Recurrent Neural Network for Multiple Language Acquisition: Starting with English and French

Xavier Hinaut*
Dept. of Informatics, Uni. of Hamburg
Hamburg, Germany
hinaut@informatik.uni-hamburg.de

Johannes Twiefel
Dept. of Informatics, Uni. of Hamburg
Hamburg, Germany
twiefel@informatik.uni-hamburg.de

Maxime Petit
SBRI, INSERM 846
Bron, France
m.petit@imperial.ac.uk

Peter Dominey
SBRI, INSERM 846
Bron, France
peter.dominey@inserm.fr

Stefan Wermter
Dept. of Informatics, Uni. of Hamburg
Hamburg, Germany
wermter@informatik.uni-hamburg.de

Abstract

How humans acquire language, and in particular two or more different languages with the same neural computing substrate, is still an open issue. To address this issue we suggest to build models that are able to process any language from the very beginning. Here we propose a developmental and neuro-inspired approach that processes sentences word by word with no prior knowledge of the semantics of the words. Our model has no "pre-wired" structure but only random and learned connections: it is based on Reservoir Computing. Our previous model has been implemented in the context of robotic platforms where users could teach basics of the English language to instruct a robot to perform actions. In this paper, we add the ability to process infrequent words, so we could keep our vocabulary size very small while processing natural language sentences. Moreover, we extend this approach to the French language and demonstrate that the network can learn both languages at the same time. Even with small corpora the model is able to learn and generalize in monolingual and bilingual conditions. This approach promises to be a more practical alternative for small corpora of different languages than other supervised learning methods relying on big data sets or more hand-crafted parsers requiring more manual encoding effort.

1 Introduction

How do children learn language? In particular, how do they link the structure of a sentence to its meaning? This question is linked to the more general issue: How does the brain link sequences of symbols to internal symbolic or sub-symbolic representations? We propose a framework to understand how language is acquired based on a simple and generic neural architecture [1, 2] which is not hand-crafted for a particular task, but on the contrary can be used for a broad range of applications (see [3] for a review). This idea of "canonical" neural circuits has been coined by several authors:

*www.informatik.uni-hamburg.de/~hinaut ; source code available soon at github.com/neuronalX

it is an aim in computational neuroscience to model generic pieces of cortex [4, 5]. As such simplified canonical circuits we use Echo State Networks (ESN) [1] which are neural networks with a random recurrent layer and a single linear output layer (called "read-out") modified by online or offline learning.

Much research has been done on language processing with neural networks [6, 7, 8] and more recently also with Echo State Networks (ESN) [9, 10]. The tasks used were diverse, from predicting the next symbol (i.e. word) in a sentence to thematic role assignment. In this paper, the task we perform is the latter. Previously the kind of inputs sequence used was mainly based on one-level symbols, i.e. symbols that belong to the same level of abstraction (e.g. only words). However language, like many other cognitive tasks, contains several levels of abstraction, which could be represented hierarchically from phonemes to discourse. Hierarchical processing is a strategy of the brain, from raw perception layers to abstract processing, and even inside the higher-level cognitive computations performed by the prefrontal cortex there exists a hierarchy of processes [11].

Some recent results with end-to-end word recognition from raw audio data with RNN are impressive [12]. This could give some insights on what kinds of features are extracted by the brain during speech processing. However, to uncover language acquisition mechanisms other modelling methods are needed. It is likely that the brain builds hierarchical representations in a more incremental and less supervised way: each newly built abstraction enables the formation of the next higher-order abstraction, instead of abstracting all at once. We hypothesize that the brain canonical circuits, such as the simple version proposed, can deal with different levels of abstraction mixed at the same time. In this paper, we present an initial model version which is able to deal with three kind of symbols: Function Words (FWs), Semantic Words (SWs) and Infrequent function Words (IWs). This model processes IWs and SWs as categories of words, thus they are on a different level of abstraction than FWs, which are processed as words (see Figure 1). Therefore, we have two levels of abstraction. Moreover, we show for the first time that this model is able to learn and generalize over a French corpus, and additionally over two languages at the same time, namely English and French.

2 Reservoir computing and grammatical construction approach

2.1 Echo State Networks (ESN)

The model is based on an Echo State Network (ESN) with leaky neurons. The units of the recurrent neural network have a *leak rate* (α) hyper-parameter, which corresponds to the inverse of a time constant. These equations define the update of the ESN:

$$x(t+1) = (1-\alpha)x(t) + \alpha f(W^{in}u(t+1) + Wx(t)) \tag{1}$$

$$y(t) = W^{out}x(t) \tag{2}$$

with $x(t)$, $u(t)$ and $y(t)$ the reservoir state, the input vector and the read-out vector respectively at time t, α the leak rate, W, W^{in} and W^{out} the reservoir, the input and the output matrices respectively and f is the *tanh* activation function. After collection of all reservoir states the following equation defines how the read-out weights are computed:

$$W^{out} = Y^d[1; X]^+ \tag{3}$$

with Y^d the concatenation of the desired outputs, X the concatenation of the reservoir states over all time steps and M^+ the Moore-Penrose pseudo-inverse of matrix M.

2.2 The sentence comprehension model with grammatical constructions approach

The proposed model processes sequences of symbols as input (namely sequences of words) and generates a dynamic probabilistic estimation of static symbols (namely thematic roles), see Figures 1 and 2. This work is based on a previous approach modelling human language understanding [13, 14], human-robot interaction [15, 16], and language acquisition in a developmental perspective (with incremental and online learning) [17]. Recently an "inverse" version of the model was able to produce sentences depending on the words of focus [18]. The general aim of having an architecture working with robots is to use them to model and test hypotheses about child learning processes of language acquisition [19]. It is also interesting to enhance the Human-Robot Interactions and it has

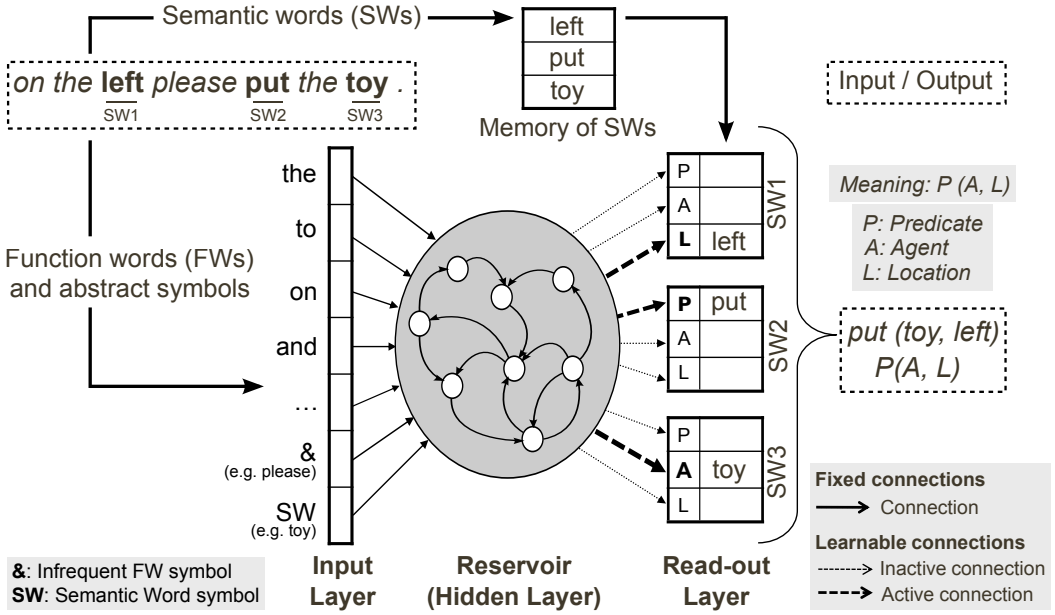


Figure 1: The sentence comprehension model enhanced with IW replacement.

already been implemented in humanoid robotic architectures (iCub and Nao) [15, 20] to enable users to use natural language instead of stereotyped vocal commands during interactions. To illustrate how the system works a video is available at [20].

Mapping the surface form (sequence of words) onto the deep structure (meaning) of a sentence is not an easy task since making word associations is not sufficient. For instance, a system relying only on the order of semantic words (cat, scratch, dog) to assign thematic roles is not enough for the following simple sentences, because even if *cat* and *dog* appear in the same order they have different roles: *"The **cat scratched** the **dog**"* and *"The **cat** was **scratched** by the **dog**"*. It was shown that infants are able to quickly extract and generalize abstract rules [21], and we want to take advantage of this core ability. Thus, to be able to learn such a mapping (from sequence of words to meaning) with a good generalization ability it seems natural to rely on abstractions of sentences rather than sentences themselves. In order to teach the model to extract the meaning of a sentence, we base our approach on the notion of grammatical construction: the mapping between a sentence's form and its meaning [22]. Constructions are an intermediate level of meaning between the smaller constituents of a sentence and the full sentence itself. Based on the cue competition hypothesis of Bates et al. [23] we make the assumption that the mapping between a given sentence and its meaning can rely on the order of words, and particularly on the pattern of function words and morphemes [14]. In our model (see Figure 1), this mapping corresponds to filling in the Semantic Words (SWs) of a sentence into the different slots (the thematic roles) of a basic event structure that could be expressed in a predicate form like *action(object, location)*. This predicate representation enables us to integrate it into the representation of actions in a robotic architecture.

As can be seen in Figure 1, the system processes inputs as follows: from a sentence (i.e. sequence of words) as input (left) the model outputs (right) a command that can be performed by a robot (i.e. thematic roles for each Semantic Word). Before entering the neural network, words are preprocessed, transforming the sentence into a grammatical construction: Semantic Words, i.e. nouns and verbs that have to be assigned a thematic role, are replaced by the *SW* symbol; Infrequent function words (IWs) are replaced by the *&* symbol. The processing of the grammatical construction is sequential: words are given one at a time, and the final thematic roles for each SW is read-out at the end of the sentence. Only the necessary outputs are shown in the figure for this example. In contrast to previous recurrent neural models [7, 9, 6, 10], the proposed model processes grammatical constructions, not sentences, thus permitting to bind a virtually unlimited number of sentences based only on a small training corpus, and enabling to process future sentences with currently unknown semantic words. Therefore, it is suited for modelling developmental language acquisition.

2.3 What to do with unknown symbols?

Children should be able to (at least partially) understand sentences with some unknown words and to extract the meaning of these new words from their environment [19]. Such a capability enables children to understand that in the sentence "The cat shombles the mouse" *shombles* is probably a verb and to potentially maps its meaning from the context. Even if some words are useless to understand the core meaning of a sentence, for instance "Could you & put some water in the cup?", with *&* symbolizing an unknown word (*e.g.* "please"), the child could still understand what is asked and perform the corresponding action. This ability to (partially) understand a sentence with unknown words is probably crucial (1) for the ability of children to bootstrap the language understanding process, and (2) to quickly learn new words and infer their meaning from the context. On the application side, when interacting with a robot through language, speech recognition will be more robust when the number of words that must be recognized is reduced [24]. That is why we propose to include a new kind of input symbol (&) to deal with infrequent words (see subsection 3.3).

3 Methods and experiments

3.1 Bilingual experiment

Our goal here is to see whether a neural network with no imposed structure (a random reservoir) could learn to process both English and French sentences and to provide the corresponding action commands that could be performed by a robot. For the experiments we use the same set of parameters in order to demonstrate that it is not necessary to tune the parameters for each language.

3.2 Natural language material

Corpora were obtained by asking naive users (agnostic about how the system works) to watch several actions in a video and give the commands corresponding to the motor actions performed, as if they wanted a robot to perform the same action. The video used is available online with the first experiments we did with robots [15]. Five users were recruited for each language, each user provided 38 commands: for each language there is a total of 190 sentences. The English corpus is a subset of the one used in [15]. A selection of sentences is provided in Table 1. For instance, for the "Action order" sentences, one can see that the order of actions to be performed does not necessarily correspond to the semantic word order in the sentence. The particular function of the FW "after" is difficult to get for the model because it appears in the middle of both sentences even if the actions to be performed are reversed. Note also that some sentences provided by users are grammatically incorrect (see Table 1). Each corpus is made of grammatical constructions, not sentences: this means that all the SWs, nouns and verbs, that should be attributed a role have been replaced by a common Semantic Word symbol "SW" in the corpus. In this way, we prevent the network to learn semantic information form nouns and verbs. Several sentences may then be recoded with the same grammatical constructions. The ratios of unique grammatical constructions in the corpora are: 0.410 (78/190) for French (FR), 0.616 (117/190) for English (EN) and 0.513 (195/380) for bilingual (FR+EN) corpora.

3.3 Infrequent symbols category

As mentioned in subsection 2.3 it is important, for a child or a robot, to be able to deal with unknown words. Out-of-vocabulary (OOV) words are a general problem in Natural Language Parsing [25]. In comparison to the previous approach developed [15] we implemented an additional method that replaces most infrequent words in the corpus. We used a threshold θ ($\theta = 7$, see subsection 3.5) that defines the limit under which a Function Word (FW) is considered infrequent and replaced by the Infrequent Word (IW) symbol "&". The preprocessing was performed on the whole dataset before performing the simulations. This new method enables us to process unknown words, which is a desirable property for online interaction when the model is implemented in a robotic platform. However, there is no a priori insight that would state whether this infrequent word replacement will enhance or decrease the generalization performances of the neural network model.

Table 1: Some sentence examples from the noisy English corpus.

TYPE	SENTENCE EXAMPLE
Sequence of actions	touch the circle **after** having pushed the cross to the left
	put the cross on the left side and **after** grasp the circle
Implicit reference to verb	**move** the circle to the left **then the cross to the middle**
Implicit reference to verb and object	**put** first the triangle on the middle **and after on the left**
"Crossed reference"	***push*** **the triangle** and *the circle* ***on the middle***
Repeated action	hit **twice** the blue circle
	grasp the circle **two times**
Unlikely action	put the cross to the right and **do a u-turn**
Particular FW	put **both** the circle and the cross to the right

3.4 Implementation details

We use one shot offline learning to get the optimal output weights in order to make generalization performance comparisons between the bilingual and monolingual corpora. The teacher signals for training the read-out layer are provided during the whole sentence: the rationale for that is that a child or a robot has just performed an action and the caregiver (the teacher) describes the actions that have just been performed. Thus the teacher signal is already available when the sentence is provided to the system. As shown previously [13], this provides the nice property of having the read-out units predicting the thematic roles during the sentence; see Figure 2.

The input W_{in} and recurrent W matrices are randomized following these distributions: values are taken with equiprobability in the set $\{-1, 1\}$ for W_{in}, and with a normal distribution with 0 mean and 1 variance for W. Both matrices have a sparsity of 0.1, i.e. only 10% of the connections are non-zero. After random initialization the input matrix W_{in} is scaled with a scalar value called *input scaling* (IS), and the absolute maximum eigenvalue of the recurrent matrix W is scaled by the *spectral radius* (SR) value. All hyper-parameters are described in section 3.5. As can be seen in Figure 1, the inputs consist of a localist representation of the Function Words (different for each language) and in addition the final dot, the IW symbol "&" and the "*SW*" symbol. Thus, the input dimensions for the different corpus are: 31 (28 + 3) for the French (FR) one, 32 (29 + 3) for the English (EN) one and 60 (57 + 3) for the bilingual (FR+EN) one. The total output dimension is 48 (8 SW * 3 roles * 2 actions): we set the maximum number of SW to 8 in this experiment.

3.5 Hyper-parameters

In the experiment we use a reservoir of 500 units in order to keep the simulation to be trained in a few seconds on a basic machine (without GPU computations), and running (after training) in less than a second: thus if used within a robotic platform it enables real-time interaction with the user. A few hyper-parameters were optimized using the *hyperopt* python toolbox [26], namely the *spectral radius* (SR), the *input scaling* (IS), the *leak rate* α and the threshold θ under which Infrequent Words are replaced. A set of rounded parameters were then chosen from the parameter space region leading to good performance: SR=1, IS=0.03, α=0.2 and θ=7. Actually θ could have been set to any value between 7 and 10 because there was no Function Word (FW) with this number of occurrences: this threshold appears to be a natural limit in the density distribution of FW occurrences. Note that for the *spectral radius* we disregarded the upper limit "advised" by the Echo State Property [1], namely 1, when we performed the hyper-parameter search, because as we are using leaky neurons the effective *spectral radius* is different from the "static" one [27].

3.6 Evaluation

In order to evaluate the performance, we record the activity of the read-out layer at the last time step, which is when the final dot is given as input. We first discard the activations that are below a threshold of 0.5. For sentences that do not contain the maximal number of SW (*i.e.* 8) we discard the remaining outputs because no SW in the input sentence could be linked to them: e.g. if there is only four SWs in the sentence, we discard outputs concerning SWs 5 to 8. Unit activations of

discarded SW outputs represent predictions about SWs that will never occur (see Figure 2). Finally, if there is several possible roles for a particular SW we do a winner-take-all and keep the role unit with the highest activation.

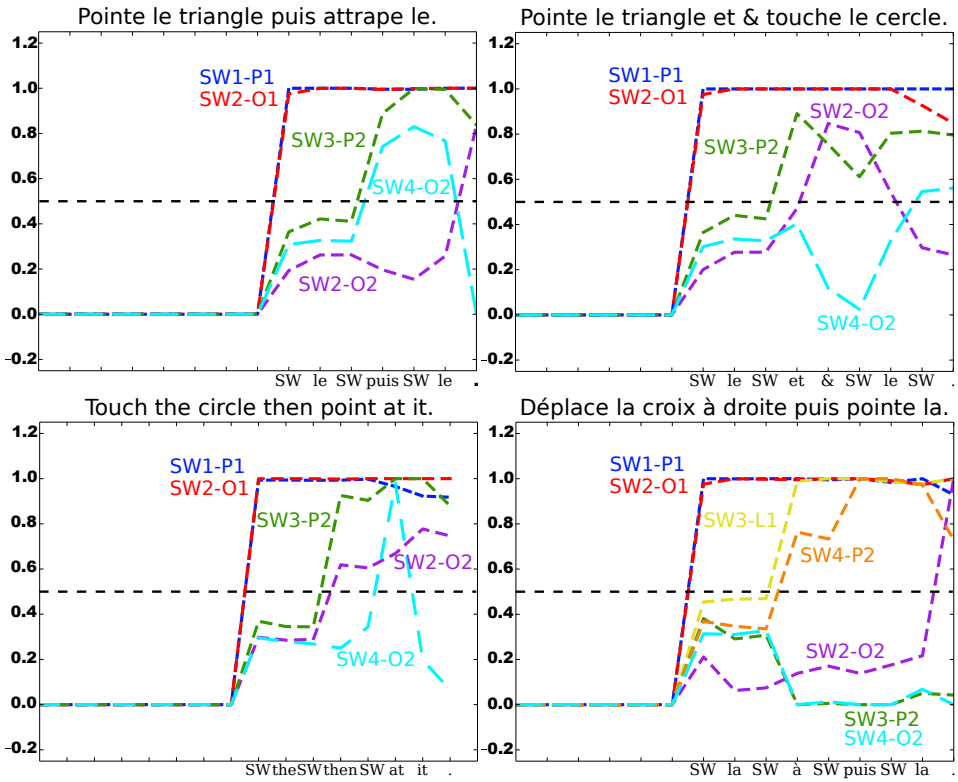


Figure 2: Examples of read-out units activations for different sentences.

4 Results

We start by providing a quantitative analysis (generalization capabilities) of the model for the different corpora, and then we perform a qualitative analysis by examining the output activity of the model for particular sentences.

4.1 Quantitative analysis

First we analyse the generalization errors and standard deviation for a 10-fold cross validation averaged over 50 different network instances. Since there are several thematic roles to be recognized in each sentence, the full meaning of a sentence is correct only if all roles are correct; if one role or more is incorrect the sentence is regarded as recognized incorrectly (*i.e.* sentence error of 1). We provide here the means and standard deviations of the *sentence errors*: 0.158 (+/-0.012) for the FR corpus, 0.214 (+/-0.022) for the EN corpus, and 0.206 (+/-0.013) for the FR+EN corpus. The EN corpus seems to incorporate some slightly more complex sentences than the FR corpus and has less redundant grammatical constructions than the French one: this probably explains the higher generalization error for the EN corpus. Even if results are not directly comparable, the new result for the English corpus outperforms the previous performance of 24.2% obtained[1] in [15].

It is remarkable that the performance for the FR+EN corpus (0.206) is not very impaired compared to the average error of the corpora processed separately (0.186).

[1]Results in [15] were obtained by taking the best of an ensemble of ten reservoirs with twice the number of units (1000 instead of 500) and leave-one-out CV (instead of 10-fold CV). Moreover the training corpus was two times larger.

4.2 Qualitative analysis

In this subsection we use the same instances for the input and reservoir weight matrices, only the read-out weight matrices are different (due to learning on different corpora). We analysed the read-out layer activity for the French and English training corpora and selected some interesting sentences (see Figure 2). For clarity and to limit the number of curves per plot only relevant output units have been kept, others have been discarded. The dotted line indicates the decision boundary threshold for the final extraction of thematic roles. In Figure 2 read-out units activations (i.e. outputs of the model) can be seen for four sentences: three in French[2] and one in English. Sentences before preprocessing are shown on top of each plot; corresponding grammatical constructions processed by the reservoir are shown on the x-axis. The top left plot shows activations for a grammatical construction that was both in the training and testing set of the given cross-validation. For trained grammatical constructions the output activations show online probabilistic estimations for the different roles based on the statistics of the training set. The three remaining plots were taken from grammatical constructions that were not in training set but which generalized correctly. For instance, we can see in the top right plot of Figure 2 that the model is able to generalize to the correct roles even in the presence of an infrequent word (IW symbolized by &). In all plots of the figure, two output roles units are active near the maximum value (*i.e.* 1) since the beginning of the sentence: *SW1-Predicate-1st_action* (SW1-P1; blue curve) unit and *SW2-Predicate-1st_action* (SW2-O1; red curve) unit. This is because for most sentences, the first two SWs are the *predicate* and *object* of the first action; *i.e.* the order of actions is not reverse by words like *before* or *after*.

We choose to focus more on the French language since this is the first time we demonstrate generalization capabilities with this language, and also because it has two interesting properties that English does not have: the words *le* or *la* (*the* in English) are gender specific, and they could be determiners or pronouns. We can see both functions in this sentence: *"Pointe **le** triangle puis attrape **le**."* ("*Point the triangle then catch it.*"): the first *le* is a determiner, and the second one is a pronoun referring to the *triangle*. As we can see in the top left plot of Figure 2, at the time step after the second occurrence of *le* (i.e. at the final dot) there is a particular "re-analysis" of the sentence because this occurrence of *le* is a pronoun, which implies that the object of the second action (O2) is not a potential semantic word (SW4) that could have followed *le*, the O2 is rather the same one as the first action, namely the SW in position 2 (SW2) in the sentence. That is why the activity of the output unit *SW4-Object-2nd_action* (SW4-O2, the unit that binds O2 with SW4; cyan curve), goes down to zero and the activity of the output unit *SW2-Object-2nd_action* (SW2-O2; purple curve) goes up to one. On the contrary, in the top right of Figure 2 the input of the last SW (SW4) confirms the determiner function of *le*, thus the activity of the unit SW4-O2 (cyan curve) increases above the threshold, and the activity of SW2-O2 (purple curve) goes down. The input of the infrequent word symbol "&" seems to "perturb" the on-going predictions compared to the top left plot: these activities may not reflect the statistics of the training corpus since the occurrence of "&" at this precise point in the sequence makes this sequence unique and produces a reservoir state that was not used during training. The bottom right plot of Figure 2 shows similar outputs as the top left plot, but for a sentence containing a location for the first action (SW3-L1; yellow curve). One can see how the following output activities of units SW3-P2 and SW4-O2 are modified in consequence. In the bottom left of Figure 2 is the readout activity for an equivalent English sentence of the French sentence in top left plot. One can see that the unit activations are similar until the last word in the sentence: *it* and *le* respectively. Some differences of units activation could be explained by the fact that the English sentence was not in the training set. This means that we have a model that is able to represent on-going sequences of words in two different languages with the same predictions of roles.

5 Discussion

A neuro-inspired model based on Reservoir Computing that processes sentences word by word with no prior semantic knowledge was used. The only assumptions are that the system is able to distinguish semantic words (SW) from function words (FW), because SWs are related to objects or actions the child or the robot already knows. Nouns and verbs were not distinguished in this SW

[2]Translation of sentences: (top left) "*Point the triangle and catch it.*"; (top right) "*Point the triangle and & touch the circle.*"; (bottom right) "*Move the cross to the left then point at it.*".

abstract symbolic category. The model processed grammatical constructions instead of sentences [22, 19] based on noisy natural language corpora produced by human users.

For the first time we showed that our architecture could process three different kinds of symbolic inputs: function words, semantic words and infrequent words. Moreover we showed that this architecture is able to process and generalize over the French language newly provided. Furthermore, we outperformed previous results obtained in [15] with the English corpus. Generalization performance on these noisy corpora, produced by users, is interesting considering the small corpus we used, about 200 sentences for each language: 84.2% for the French, 78.6% for the English and 79.4% for the bilingual corpora respectively. These figures indicate the percentage of sentences that have all their roles fully recognized, which means that the thematic role performance is higher.

When used in a robotic or other platforms, if a sentence is recognized partially (*i.e.* few roles in the sentence are incorrect), the system may recover based on further contextual and semantic post-processing, thereby reducing the number of sentences not recognized. In fact, these good performances could be enhanced by post-processing because as it has been shown in [13] that most of the sentences not fully recognized have only one or few erroneous roles. Moreover we showed here for the first time that the system was able to understand grammatical constructions that had infrequent unknown function words. This is not only interesting from the point of view of language acquisition [19] but also from the application side because it provides a natural way of dealing with the out-of-vocabulary (OOV) words problem [25]. In further work we will explore distributional encoding of semantic words based only on the context available to the system. For instance we could use *word2vec* representation [28] which is based on huge corpora. In this language acquisition perspective, an issue would be to create such representations with small corpora where not much context information is available, and moreover where this context information is available incrementally.

This bilingual experiment shows that the chosen architecture has interesting properties for multilingual processing. The network was able to learn and generalize without an important drop-off in performance. What is surprising is to have such a high performance for a fixed reservoir size even with an input dimension that doubled in size for the bilingual corpus, compared to the monolingual experiments: the bilingual corpus contained twice more function words than the French one or the English one. Moreover, the reservoir state spaces explored for each of the corpora are quite different, due to the different sets of inputs, nevertheless the linear regression performed by the read-out layer is still able to combine state spaces produced by very different inputs towards the same output roles. Deeper analysis of the reservoir states and read-out weights may provide some more explanation why the bilingual model is working better than one would expect. It is possible that the model benefits from the regularities of the syntax similarities between French and English. Further work is needed to compare this bilingual neural model to other models [29] and to analyse which insights it can give on bilingual language acquisition and second language acquisition [30] (if using an incremental learning). For instance, would a bilingual model that builds its own self-organized input word representations (shared by the two languages) be able to benefit from both languages and generalize better than a monolingual model? It would be also interesting to evaluate the ability of the current model to process grammatical constructions that have parts in French and parts in English.

Acknowledgments

This research was supported by a Marie Curie Intra European Fellowship within the 7th European Community Framework Programme: EchoRob project (PIEF-GA-2013-627156). Authors are grateful to Cornelius Weber and Dennis Hamester for their very useful and interesting feedback.

References

[1] Jaeger, H. (2001) The echo state approach to analysing and training recurrent neural networks. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report, 148, 34.

[2] Dominey, P. F. (1995) Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. Biological Cybernetics, 73, pp. 265–274.

[3] Lukosevicius, M., & Jaeger, H. (2009) Reservoir computing approaches to recurrent neural network training. Computer Science Review 3: 127–149.

[4] Rigotti, M. et al. (2013). The importance of mixed selectivity in complex cognitive tasks. Nature, 497(7451), 585–590.

[5] Maass W., Natschlger T., & Makram H. (2003) A Model for Real-Time Computation in Generic Neural Microcircuits. In Proc. of NIPS 2003, 213–220.

[6] Elman J (1990) Finding structure in time. Cognitive Science 14: 179–211.

[7] Miikkulainen, R. (1996) Subsymbolic case-role analysis of sentences with embedded clauses. Cognitive Sci 20: 47–73.

[8] Wermter, S., Arevian, G., & Panchev, C. (2000) Meaning Spotting and Robustness of Recurrent Networks. In Proc. of IJCNN, pp. III-433-438. Como, Italy.

[9] Tong, M. H. et al. (2007) Learning grammatical structure with Echo State Networks. Neural networks 20: 424–432.

[10] Frank, S. L. (2006). Strong systematicity in sentence processing by an Echo State Network. In Proc. of ICANN 2006, pp. 505–514.

[11] Koechlin, E., & Jubault, T. (2006). Broca's area and the hierarchical organization of human behavior. Neuron, 50(6), 963-974.

[12] Hannun, A. et al. (2014) Deep Speech: Scaling up end-to-end speech recognition. arXiv:1412.5567

[13] Hinaut, X., & Dominey, P. F. (2013) Real-Time Parallel Processing of Grammatical Structure in the Fronto-Striatal System: A Recurrent Network Simulation Study Using Reservoir Computing. PLoS ONE 8(2): e52946.

[14] Dominey, P. F., Hoen, M., & Inui, T. (2006) A neurolinguistic model of grammatical construction processing. Journal of Cognitive Neuroscience, 18(12):2088–2107.

[15] Hinaut, X. et al. (2014) Exploring the Acquisition and Production of Grammatical Constructions Through Human-Robot Interaction. Frontiers in NeuroRobotics 8:16.

[16] Dominey, P. F., & Boucher, J. D. (2005). Developmental stages of perception and language acquisition in a perceptually grounded robot. Cognitive Systems Research, 6(3), 243-259.

[17] Hinaut, X., & Wermter, S. (2014) An Incremental Approach to Language Acquisition: Thematic Role Assignment with Echo State Networks. In Wermter, S. et al., Proc. of ICANN 2014, pp. 33-40.

[18] Hinaut, X. et al. (2015) Cortico-Striatal Response Selection in Sentence Production: Insights from neural network simulation with Reservoir Computing. Brain and Language, vol. 150, Nov. 2015, pp. 54–68.

[19] Tomasello M (2003) Constructing a language: A usage based approach to language acquisition. Cambridge, MA: Harvard University Press. 388 p.

[20] Hinaut, X. et al. (2015) Humanoidly Speaking – How the Nao humanoid robot can learn the name of objects and interact with them through common speech. Video, IJCAI 2015. http://bit.ly/humanoidly-speaking

[21] Marcus, G. F. et al. (1999). Rule learning by seven-month-old infants. Science 283, 77–80.

[22] Goldberg, A. (1995) Constructions: A Construction Grammar Approach to Argument Structure; Fauconnier G, Lakoff G, Sweetser E, editors. Chicago: University of Chicago Press. 265 p.

[23] Bates, E. et al. (1982) Functional constraints on sentence processing: a cross-linguistic study. Cognition 11: 245–299.

[24] Twiefel, J. et al. (2014) Improving Domain-independent Cloud-based Speech Recognition with Domain-dependent Phonetic Post-processing. In Brodley C.E. et al. (eds.). Proc. of AAAI 2014, pp. 1529-1535.

[25] Jurafsky, D., and Martin, J. H. (2009) Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Pearson International, 2nd edition.

[26] Bergstra, J., Yamins, D., & Cox., D. D. (2013) Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms. In SciPy13.

[27] Jaeger, H. et al. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. Neural Networks, 20(3), 335-352.

[28] Mikolov, T. et al. (2013) Distributed Representations of Words and Phrases and their Compositionality. In Proc. of NIPS 2013.

[29] Frank, S. L. (2014). Modelling reading times in bilingual sentence comprehension. In P. Bello et al. (Eds.), Proc. of CogSci 2014, pp. 1860–1861.

[30] Berens, M. S.; Kovelman, I. & Petitto, L.-A. (2013) Should Bilingual Children Learn Reading in Two Languages at the Same Time or in Sequence? Bilingual Research Journal, 36, pp. 35–60.

Efficient neural computation in the Laplace domain

Marc W. Howard, Karthik H. Shankar, and Zoran Tiganj
Department of Psychological and Brain Sciences
Boston University
{marc777,shankark,zorant}@bu.edu

Abstract

Cognitive computation ought to be fast, efficient and flexible, reusing the same neural mechanisms to operate on many different forms of information. In order to develop neural models for cognitive computation we need to develop neurally-plausible implementations of fundamental operations. If the operations can be applied across sensory modalities, this requires a common form of neural coding. Weber-Fechner scaling is a general representational motif that is exploited by the brain not only in vision and audition, but also for efficient representations of time, space and numerosity. That is, for these variables, the brain appears to support functions $f(x)$ by placing receptors at locations x_i such that $x_i - x_{i-1} \propto x_i$. The existence of a common form of neural representation suggests the possibility of a common form of cognitive computation across information domains. Efficient Weber-Fechner representations of time, space and number can be constructed using the Laplace transform, which can be inverted using a neurally-plausible matrix operation. Access to the Laplace domain allows for a range of efficient computations that can be performed on Weber-Fechner scaled representations. For instance, translation of a function $f(x)$ by an amount δ to give $f(x+\delta)$ can be readily accomplished in the Laplace domain. We have worked out a neurally-plausible mapping hypothesis between translation and theta oscillations. Other operations, such as convolution and cross-correlation are extremely efficient in the Laplace domain, enabling the computation of addition and subtraction of neural representations. Implementation of neural circuits for these elemental computations would allow hybrid neural-symbolic architectures that exhibit properties such as compositionality and productivity.

1 Introduction

Cognitive computation in the brain is fast, efficient and flexible. Emulating this ability would result in extremely important technological advances. A general computational framework should be able to operate on a wide range of content without learning each exemplar. Such a framework should generalize across not only different specific operands but also across sensory domains, providing a general computational language for cortical computation. Mathematical operations are an important aspect of symbolic processing. Because of the combinatorics of these problems, learning each set of operands and the appropriate outcome is not feasible.

This paper argues that

1. The brain represents functions of many quantities, including time, using a common form of coding that we refer to as Weber-Fechner scaling.
2. Some of these quantities can be efficiently computed using the Laplace domain and a neurally-plausible mechanism for approximating the inverse Laplace transform.

3. Computational operations, including translation, convolution, and an analog of cross-correlation, can be efficiently computed in a neurally-plausible way with access to the Laplace domain.

This suggests the hypothesis that the brain uses the Laplace domain as a common computational currency across modalities, enabling reuse of the same neural mechanisms for flexible computations on a range of different kinds of information.

1.1 Weber-Fechner scaling of one-dimensional functions in the brain

In this paper we restrict our attention to one-dimensional quantities defined over the positive real line from zero (or some relatively small value) to some large (effectively unbounded) value. We argue that the brain represents functions over variables with these properties using Weber-Fechner scaling. If the ith receptor has a receptive fields centered at x_i, then we define *Weber-Fechner* scaling to mean that

1. the spacing of adjacent receptors is such that $x_i - x_{i-1} \propto x_i$.

2. the width of the receptive field of the unit at x_i should be proportional to x_i.

These two constraints imply a logarithmic scale internal scale for x, which we label $\overset{*}{x}$ to avoid confusion between external physical variables and internal representations. We refer to this coding scheme as a Weber-Fechner scale because it can readily implement the behavioral Weber-Fechner law [2].

There is good evidence that Weber-Fechner scaling is obeyed in the brain in coding extrafoveal retinal position [7, 21]. In the case of vision, Weber-Fechner scaling can be attributed to the structure of the retinal circuitry. However, Weber-Fechner scaling appears to be more general. Neural evidence [15] suggests that Weber-Fechner scaling applies to neural representations of numerosity. For instance, [14] observed approximately Weber-Fechner coding for numerosity in the activity of PFC neurons during the delay period of a working memory task. Different neurons had different preferred numerosities. The width of the tuning curves went up linearly with the cell's preferred numerosity.[1] Weber-Fechner scaling for numerosity cannot be attributed to a property of a physical receptor.

Behavioral [1] and theoretical work [16] suggests that this Weber-Fechner coding scheme should extend also to functions of remembered time. Indeed, a growing body of neurophysiological evidence suggests that representations of time also obey Weber-Fechner scaling. Figure 1 shows evidence illustrating evidence suggesting that the neural representation of time may obey Weber-Fechner scaling. First, the observation of *time cells* suggests that the brain supports functions of past time [12]. During the delay of a memory task, different time cells fire sequentially at circumscribed periods of the delay. At any moment, observation of the set of active cells provides an estimate of the time in the past at which the delay began. Time cells have now been observed in a variety of brain regions with similar qualitative properties that suggest Weber-Fechner coding. It is known that the width of the receptive fields of time cells increases with delay in the hippocampus [5, 11], medial entorhinal cortex [10], mPFC [19] and striatum [13]. Moreover, the density of preferred times decreases with the delay [11, 10, 13, 19]. Collaborative work to quantitatively assess Weber-Fechner coding in a large dataset of hippocampal time cells is ongoing.

If the neural representation of time obeys Weber-Fechner scaling this is a non-trivial computational challenge. A representation of a timeline must update itself in real time. Because the spacing between t_i and t_{i+1} is different than the spacing between t_{i+1} and t_{i+2}, information would have to flow at different rates for different values of t. This seems neurally implausible. We have proposed a solution to this challenge—updating the Laplace transform of history rather than history itself—that we argue also lends itself readily to efficient and flexible computation.

[1] Although they did not assess the spacing in a quantitative way, the number of neurons did go down with preferred numerosity.

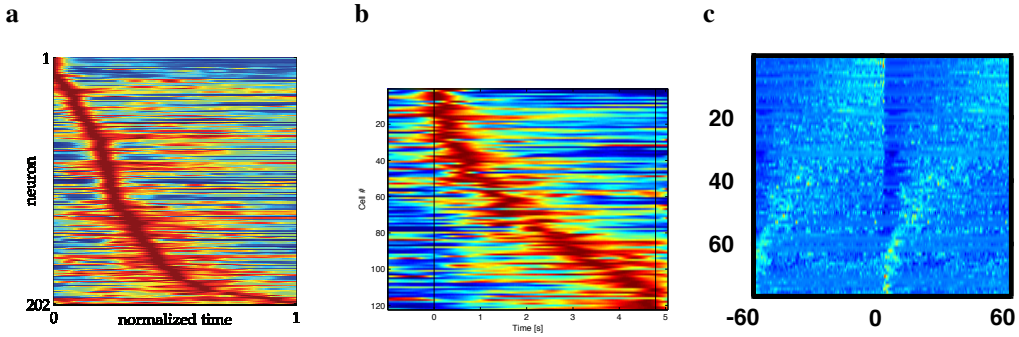


Figure 1: A neural Weber-Fechner scaling for time? In each plot, false color firing rate maps are shown as a function of time for a number of extracellularly-recorded neurons. The neurons are ordered according to their time of peak firing. If the neurons showed linear spacing, these plots would appear as a straight "ridge" of constant width. To the extent the ridges are curved, that implies a decreasing number density of neurons with preferred time of firing, consistent with Property 1. To the extent that the ridges get wider during the delay, this implies an increase in receptive field with preferred time of firing, consistent with Property 2. **a.** Neurons in medial entorhinal cortex during running on a treadmill [10]. The duration of the delay was on average 16 s (see [10] for details). **b.** Neurons in medial PFC during the delay of a temporal discrimination task [19]. **c.** Neurons in the striatum during the 60 s prior to (left) and following reward in a fixed interval procedure [13]. Note that the ordering is such that neurons with time fields earlier in the delay are at the bottom of the figure rather than the top.

2 Constructing Weber-Fechner scale functions of "hidden" variables using the Laplace transform

We have developed a formal mechanism for efficiently representing a Weber-Fechner timeline. The key insight is that while the timeline itself cannot be evolved self-sufficiently in time, the Laplace transform of the timeline can be [16]. The model can be understood as a two-layer feedforward architecture (Fig 2). At each moment a single input node $f(t)$ projects to a set of units $F(s)$ that store the Laplace transform up to the current moment; s indexes the different units. Through a local set of feed forward connections (represented by an operator $\mathbf{L}_k^{-1}$), the second layer approximately inverts the encoded Laplace transform to represent a fuzzy reconstruction of the actual stimulus history itself, $\tilde{f}(\overset{*}{\tau})$. The operator $\mathbf{L}_k^{-1}$ implements the Post inversion formula keeping k terms and can be readily implemented with simple feedforward projections [16].

This simple computational scheme for representing a Weber-Fechner timeline is sufficient to account for canonical behavioral effects in a variety of learning and memory paradigms across species [6]; the long functional time constants necessary to encode $F(s)$ could be computed using known neurophysiological mechanisms [18]. This mechanism can be straightforwardly generalized to represent one-dimensional spatial position, numerosity or any other variable whose time derivative is available at each moment [5]. More precisely, by modulating the differential equations governing the Laplace transform by $\alpha(\tau) = dx/dt$ we can obtain the Laplace transform with respect to x rather than t. This mechanism is sufficient to account for a variety of neurophysiological findings regarding place cells and time cells in the hippocampus [5] and can be generalized to numerosity. For instance, if we initialize a representation with $f(\tau = 0)$ set to a single delta function input, then let it evolve with $\alpha(\tau)$ set to the rate of change of some variable x during an interval T, then at the end of the interval $\tilde{f}(\overset{*}{x}, T)$ will give a scale-invariant estimate of the net quantity x accumulated from time 0 to time T. When $\alpha(\tau)$ is set to zero, the estimate of $\tilde{f}$ stops changing so that α can also be used as a control signal to maintain information in working memory.

As with all path integration models, this approach is subject to cumulative error. That is if $\alpha(\tau) = dx/dt + \eta$, the estimate of $\tilde{f}(\overset{*}{x})$ will grow more imprecise over time. However, note that in the absence of noise, the "blur" in the representation of time, place, and number does not reflect stochastic variability. Rather, the blur is more analogous to a tuning curve with non-zero width.

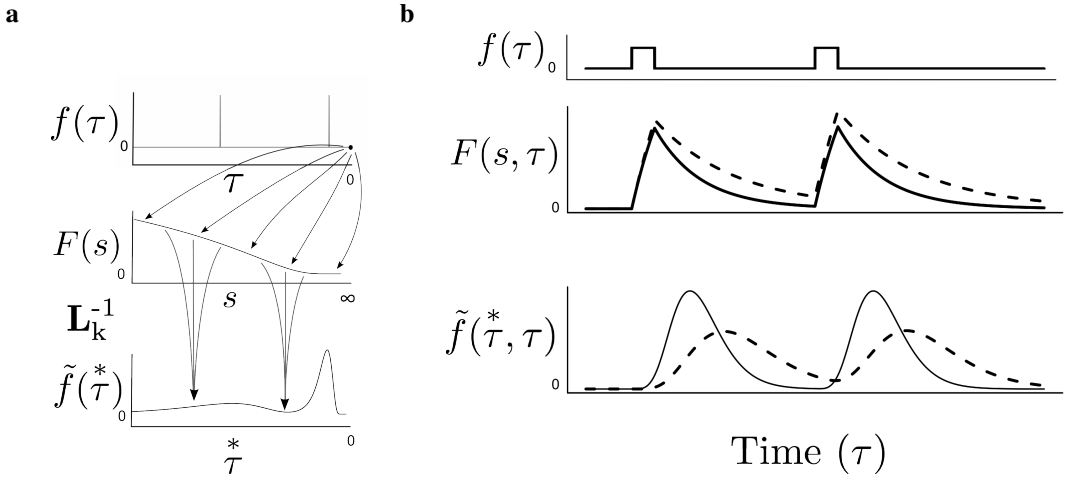


Figure 2: **a.** Schematic of the model for encoding a temporal history $f(\tau)$. At each time step, input from a single node provides input to a set of nodes $F(s)$. Each node of F is indexed by a different value of s which can be identified with the real Laplace variable. Nodes in $F(s)$ project locally to another set of nodes in $\tilde{f}(\tilde{\tau}^*)$ *via* an operator $\mathbf{L}_k^{-1}$. The nodes in $\tilde{f}$ approximate the original function $f(\tau)$. The error in $\tilde{f}(\tilde{\tau}^*)$ is scale invariant. We choose the distribution of nodes across s and thus also $\tilde{\tau}^*$ to implement Weber-Fechner spacing (not shown). **b.** Nodes in $\tilde{f}$ behave like neural time cells. In this plot the input $f(\tau)$ and the activity of two nodes in $F(s)$ with different values of s and two corresponding nodes in $\tilde{f}(\tilde{\tau}^*)$ are shown evolving in time. Note that the units in $F(s)$ behave like charging and discharging capacitors with different rate constants (controlled by their value of s). The units in $\tilde{f}(\tilde{\tau}^*)$ behave like neural time cells, responding a characteristic time after the input. The time at which each unit's activity peaks is controlled by $\tilde{\tau}^* = k/s$.

3 Flexible computations in the Laplace domain

If time, space, and number, as well as sensory representations share a common coding scheme, then mechanisms for computing with representations of that form could be reused across many types of information. Here we sketch neurally implementable mechanisms for three operations in the Laplace domain, translation, convolution, and a close analog of cross-correlation. Of these three, translation is the most thoroughly worked out, with a detailed mapping hypothesis onto neurophysiological mechanisms related to theta oscillations [17]. Translation of functions of time can be used to anticipate the future to inform decision-making in the present; translation of functions of other variables can be used to imagine alternative states of the world to inform decision-making in the world in its current state. Convolution and cross-correlation can be used for the addition and subtraction of functions, respectively (among other uses). Because the Post inversion formula is not well-defined for $-s$, we describe an analog of cross-correlation that can be implemented within the neural framework we have developed.

3.1 A neural mechanism for translation *via* hippocampal theta oscillations.

Access to the Laplace domain facilitates flexible translation of one-dimensional representations. A function $f(x)$ can be translated to obtain $f(x+\delta)$ in the Laplace domain *via* a simple point-wise multiplication with the function $\exp(-s\delta)$ where s is the Laplace domain variable. This can understood in the context of the two layer network as modulation of the synaptic weights in $\mathbf{L}_k^{-1}$ between F and $\tilde{f}$ [22]. Consideration of the computational requirements for translation in the Laplace domain coupled with the hypothesis that hippocampal theta phase precession implements translation leads to several results [17].

The resulting neural model accomplishes translation across scales and at the same time explains and organizes a broad variety of neurophysiological findings related to hippocampal theta oscillations. The hypothesis is that theta oscillations implement translation from zero to some large value within each theta cycle. This successive translation of the present into the past enables prediction of the fu-

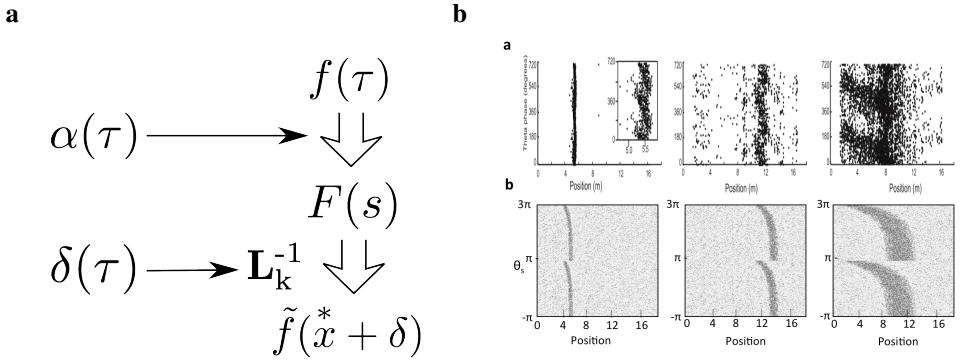


Figure 3: A neurophysiological mechanism for translation of one-dimensional functions exploiting theta oscillations. **a.** A generalized circuit for computing and translating scale-invariant functions of one-dimensional variables. $\alpha(\tau)$ enables the same circuit to represent functions of time or any other one-dimensional quantity for which the time derivative is available. Thus, if $f(\tau)$ can be rewritten as $f\left(x\left(\tau\right)\right)$ and $\alpha(\tau) = dx/d\tau$, then the reconstruction is with respect to x rather than τ and we write $\overset{*}{x}$. δ provides a mechanism to translate the function. **b.** Theta phase precession shows properties resembling translation to different future points of the trajectory within a theta cycle. Top: neurophysiological data from [9]. Place cells from different positions along the dorsoventral axis of the hippocampus have place cells of different size. However, cells at all scales still precess over the same range of phases. Bottom: model predictions show the same qualitative patterns [17].

ture at successively more distant points. This model accounts for the finding that all scales (different values of s) phase precess through the same range of local theta phases (Fig. 3). Moreover, coherent translation *requires* that both past time (controlled by the values of s) and future time (controlled by the rate at which δ changes within a theta cycle) obey Weber-Fechner scaling. Finally, the model predicts that cells coding for predicted events should ramp up their firing from the time at which the prediction becomes available to the time at which the predicted stimulus is obtained, phase precessing through at most one theta cycle. This prediction is analogous to findings for neurons in the ventral striatum [20].

We found good evidence aligning translation of functions of space and time from 0 to some large value of δ to neurophysiological findings during hippocampal theta oscillations. However, translations with other properties could be implemented during other neurophysiological events. For instance, translation by negative values would correspond to search through memory for the past; translation to a single non-zero value of δ (rather than sweeping through a range of values) would facilitate retrieval of a memories at a specific range of past times [4]. In two spatial dimensions, one can imagine a series of translations tracing out an imagined path in a navigation task [8]. In the visual modality, translation could be used to simulate planned (or imagined) eye movements or motion of objects in the world. Although these translations could have somewhat different neurophysiological signatures, they are all computationally related to one another. And in all cases, the translation facilitates decision-making and behavior in the present by enabling examination of imagined states of the world.

3.2 Arithmetic operations on functions through parallel computations

Access to the Laplace domain facilitates operations other than translation. In the same way that point-wise multiplications in the Laplace domain can be achieved in a parallel fashion to implement translation of any function, it is also possible to perform *addition* and *subtraction* operations on any two functions by point-wise parallel computations with similar efficiency in the Laplace domain. For this, we start with a definition of the operations addition and subtraction on numbers represented by distribution functions.

Let $f(x)$ and $g(x)$ be functions representing two distributions of possible values for the number x in the range 0 to x_{max}. Outside this range, the functions are assumed to vanish. We shall define the operation of 'addition' of these two distributions to be $[f + g](x)$ to be the convolution of the two

functions.

$$[f+g](x) \equiv \int_0^\infty f(x')g(x-x')\,dx'$$

The justification for this definition is rather straightforward. By considering the two functions to be Dirac delta functions at two different positions, x_1 and x_2, note that $[f+g]$ is a Dirac delta function at x_1+x_2. Moreover, the addition operation is bilinear with respect to the two functions, and hence the above generalized definition for addition is justified. Importantly, since we have access to the Laplace transform of the functions, namely $F(s)$ and $G(s)$, the addition operation can be performed in the Laplace domain. The Laplace transform of $[f+g]$ is simply the point wise multiplication of $F(s)$ and $G(s)$, which can be computed in a parallel fashion, independently for each s value. Finally, the $\mathbf{L}_{\mathrm{k}}^{-1}$ operator can be employed to invert the Laplace transform of $[f+g]$ and obtain a fuzzy addition operation.

It is easy to convince oneself that subtraction operation can similarly be defined to be[2]

$$[f-g](x) \equiv \int_0^\infty f(x')g(x'+x)\,dx'$$

By defining a reflected function $g_r(x) = g(x_{max}-x)$, it can be seen that the Laplace transform of $[f-g]$ is simply the point wise multiplication of the Laplace transform of $f(x)$ and $g_r(x)$. A point of subtlety here is that for the subtraction operation, we have to consider both positive and negative values of x although the two functions are assumed to be non vanishing only for $x>0$. However, noting that $[f-g](x) = [g-f](-x)$ for positive x, we can perform the subtraction operation for negative x values also. In this entire process, only positive values of s are utilized, and hence the inverse Laplace operator $\mathbf{L}_{\mathrm{k}}^{-1}$ is always well defined and the entire process can be performed in parallel.

We have not yet carefully considered the neurophysiological substrates that could support these arithmetic operations. However, the computational efficiency of performing these operations in the Laplace domain is considerable. Given these considerations, it may be reasonable for the brain to encode the Laplace transform even for variables that are provided along a Weber-Fechner scale due to the property of the sensory receptors.

4 Discussion

We suggest that the brain uses a common form of coding, Weber-Fechner scaling, to represent unbounded one-dimensional quantities. It can be shown that Weber-Fechner scaling is an optimal response to signals that are long-range correlated, which are found throughout the natural world [16]. Weber-Fechner scaling allows for representation of exponential scales with linear resources.

Representation of variables such as time, space and numerosity is greatly facilitated by access to the Laplace transform. Many computations can be efficiently performed in the Laplace domain. For instance, translation of representations of space and time toward the past can be used to estimate the future. Recent work has developed a detailed mapping between a translation operator and hippocampal theta oscillations. We sketched implementable operations for addition and subtraction of functions on a Weber-Fechner scale. These operations could be used for combining functions, or for comparing one function to another. Because the outcome of a specific operation does not need be learned, but can be computed on-line, the existence of these operations provides an important step towards endowing neural systems with the properties of productivity and compositionality that are taken to be essential aspects of symbolic computation and cognition more broadly [3]. For instance, it is clear that arithmetic obeys the properties of compositionality and productivity (modulo edge effects). If the result of an addition operation is a function with the same neural code as the addends, then one can in principle represent an effectively infinite number of possible problems. For instance, given only two input functions f and g one could compute $f+g$, or $(f+g)+g$, or $(f+f)+g$, etc.

There are several design considerations that are important in developing this into a general framework for cognitive computation. The first consideration is whether computation for different information should be performed in a central location, as in a von Neumann architecture or performed

[2]The challenge of this approach is that the Post inversion formula does not work when the transform is growing exponentially as with $-s$. If that were not the case, cross-correlation would suffice to implement subtraction.

locally. The answer may depend on the form of operation. Consider Fig. 3a. Different settings for $\alpha(\tau)$ and different settings for $f(\tau)$ can give rise to a very broad range of representations, corresponding to a broad taxonomy of cells in the hippocampus and related structures [5]. All of these representations can be translated by modulating the same weights used to construct the representation (modulation by δ). Here the control signal for translation is a scalar per representation and the output of the computation can be written to the same cells that are used to hold the representation itself.[3] This means that the cost of local implementation of translation is small per translatable function. In contrast, addition and subtraction operators require additional resources to hold the output of the computation. The storage cost of implementing this operation locally would be relatively substantial. Moreover, because there are many pairwise combinations of representations that might need to be combined, there is in addition a considerable wiring cost associated with local processing. For these reasons addition and subtraction of functions ought not to be performed locally.

Acknowledgments

We acknowledge helpful discussions with Eric Schwartz, Haim Sompolinsky, Kamal Sen, Xuexin Wei, and Michele Rucci. This work was supported by BU's Initiative for the Physics and Mathematics of Neural Systems and AFOSR FA9550-12-1-0369.

References

[1] BALSAM, P. D., AND GALLISTEL, C. R. Temporal maps and informativeness in associative learning. *Trends in Neuroscience 32*, 2 (2009), 73–78.

[2] FECHNER, G. *Elements of psychophysics. Vol. I.* Houghton Mifflin, 1860/1912.

[3] FODOR, J. A., AND PYLYSHYN, Z. W. Connectionism and cognitive architecture: A critical analysis. *Cognition 28*, 1 (1988), 3–71.

[4] FOSTER, D. J., AND WILSON, M. A. Reverse replay of behavioural sequences in hippocampal place cells during the awake state. *Nature 440*, 7084 (2006), 680–3.

[5] HOWARD, M. W., MACDONALD, C. J., TIGANJ, Z., SHANKAR, K. H., DU, Q., HASSELMO, M. E., AND EICHENBAUM, H. A unified mathematical framework for coding time, space, and sequences in the hippocampal region. *Journal of Neuroscience 34*, 13 (2014), 4692–707.

[6] HOWARD, M. W., SHANKAR, K. H., AUE, W., AND CRISS, A. H. A distributed representation of internal time. *Psychological Review 122*, 1 (2015), 24–53.

[7] HUBEL, D. H., AND WIESEL, T. N. Uniformity of monkey striate cortex: a parallel relationship between field size, scatter, and magnification factor. *Journal of Comparative Neurology 158*, 3 (1974), 295–305.

[8] JOHNSON, A., AND REDISH, A. D. Neural ensembles in CA3 transiently encode paths forward of the animal at a decision point. *Journal of Neuroscience 27*, 45 (2007), 12176–89.

[9] KJELSTRUP, K. B., SOLSTAD, T., BRUN, V. H., HAFTING, T., LEUTGEB, S., WITTER, M. P., MOSER, E. I., AND MOSER, M. B. Finite scale of spatial representation in the hippocampus. *Science 321*, 5885 (2008), 140–3.

[10] KRAUS, B. J. *Time and distance coding by the hippocamus and medial entorhinal cortex*. PhD thesis, Boston University, 2012.

[11] KRAUS, B. J., ROBINSON, 2ND, R. J., WHITE, J. A., EICHENBAUM, H., AND HASSELMO, M. E. Hippocampal "time cells": time versus path integration. *Neuron 78*, 6 (2013), 1090–101.

[12] MACDONALD, C. J., LEPAGE, K. Q., EDEN, U. T., AND EICHENBAUM, H. Hippocampal "time cells" bridge the gap in memory for discontiguous events. *Neuron 71*, 4 (2011), 737–749.

[13] MELLO, G. B., SOARES, S., AND PATON, J. J. A scalable population code for time in the striatum. *Current Biology 25*, 9 (2015), 1113–1122.

[14] NIEDER, A., AND MERTEN, K. A labeled-line code for small and large numerosities in the monkey prefrontal cortex. *Journal of Neuroscience 27*, 22 (2007), 5986–93.

[3] This is possible because the original untranslated function can be recovered simply by setting $\delta = 0$.

[15] NIEDER, A., AND MILLER, E. K. Coding of cognitive magnitude: compressed scaling of numerical information in the primate prefrontal cortex. *Neuron 37*, 1 (2003), 149–57.

[16] SHANKAR, K. H., AND HOWARD, M. W. Optimally fuzzy temporal memory. *Journal of Machine Learning Research 14* (2013), 3753–3780.

[17] SHANKAR, K. H., SINGH, I., AND HOWARD, M. W. Neural mechanism to simulate a scale-invariant future. *arXiv preprint arXiv:1503.03322* (2015).

[18] TIGANJ, Z., HASSELMO, M. E., AND HOWARD, M. W. A simple biophysically plausible model for long time constants in single neurons. *Hippocampus 25*, 1 (2015), 27–37.

[19] TIGANJ, Z., KIM, J., JUNG, M. W., AND HOWARD, M. W. Temporal coding across scales in the rodent mPFC. *Cerebral Cortex* (In revision).

[20] VAN DER MEER, M. A. A., AND REDISH, A. D. Theta phase precession in rat ventral striatum links place and reward information. *Journal of Neuroscience 31*, 8 (2011), 2843–54.

[21] VAN ESSEN, D. C., NEWSOME, W. T., AND MAUNSELL, J. H. The visual field representation in striate cortex of the macaque monkey: asymmetries, anisotropies, and individual variability. *Vision Research 24*, 5 (1984), 429–48.

[22] WYBLE, B. P., LINSTER, C., AND HASSELMO, M. E. Size of CA1-evoked synaptic potentials is related to theta rhythm phase in rat hippocampus. *Journal of Neurophysiology 83*, 4 (2000), 2138–44.

Neural Network Model of Semantic Processing in the Remote Associates Test

Ivana Kajić
School of Computing
Plymouth University
Plymouth, Drake Circus, PL4 8AA
United Kingdom
ivana.kajic@plymouth.ac.uk

Thomas Wennekers
School of Computing
Plymouth University
Plymouth, Drake Circus, PL4 8AA
United Kingdom
thomas.wennekers@plymouth.ac.uk

Abstract

The ability to generate novel, unique and useful ideas is an important trait of intelligent behaviour. It is also a virtue of a creative individual in many scientific and artistic domains. In this study we are concerned with the Remote Associates Test (RAT), a task widely used in psychology and neuroscience to study insight and creative problem solving. The RAT is used to assess the ability of an individual to generate novel relationships among familiar words. The test consists of word triplets (e.g. cream, water, skate) and the task is to find a unique word associated with all three words. Here, we aim to identify a basic set of computational mechanisms underlying cognitive processes in the RAT solving. To this end, we propose a multi-layer neural network based on biologically and cognitive realistic mechanisms. The search for a solution in a RAT problem is realised by spreading of activity among word associations in a semantic layer, and the selection of a response by a winner-take-all layer. The model yields human-like performance and distinguishes between easy and difficult RAT problems. The modelling findings are consistent with the existing theories in creativity research, confirming that less stereotypical word associations are important for the good performance on the RAT.

1 Introduction

The adjective "creative" is often attributed to new, valuable and surprising ideas and objects [1]. It is seen as a positive quality of an individual. Various tests and questionnaires have been devised in the attempt of measuring different aspects of creative thinking. Of particular interest to neuroscience and cognitive science have been cognitive abilities such as working memory, sustained attention and cognitive flexibility underlying behaviour assessed with creativity tests [2].

Divergent thinking tests measure the ability to generate new, meaningful and relevant ideas. An example of a task in a divergent thinking test is to think of as many possible practical uses of a certain object, such as a cooking pot or a brick [3]. In contrast to divergent thinking tests, convergent thinking tests have a correct, not necessarily unique, solution. A convergent task commonly used to study insight is the "9 dots" puzzle, where nine dots have been arranged in three rows and the task is to connect them with four straight lines. Another common task is the Remote Associates Test (RAT) used to measure the ability to form novel and meaningful word associations. While these tests are broadly used in creativity research, solving them requires a range of functions [2, 4] which can facilitate or hinder the performance and the process of problem solving. Thus, to understand the assessed process of ideation it is important to identify and understand the underlying cognitive processes.

1.1 The Remote Associates Test

In this study we are concerned with the modelling of cognitive processes underlying solving of the Remote Associates Test (RAT). The RAT has been widely used in cognitive neuroscience and psychology [5–7] to study insight, problem solving and creative thinking since it was conceived in 1962 [8]. It was developed to measure the ability of an individual to form new associations among seemingly unrelated words. The test consists of word triplets and the task is to find a word associated with all three words for every triplet. An example of a RAT problem is a word triplet *cream, water, skate* with a solution *ice*. The performance on the RAT is measured as the number of correctly solved items within a given time limit. Mednick's theory on associative hierarchies [8] suggests that highly creative individuals are able to form rare, uncommon and remote word associations. This is in contrast with associations formed by less creative individuals who respond with fewer, more common and stereotypical associations. In the present study we investigate this idea by analysing how different word pair associations influence the test performance. The simplicity of the paradigm, availability of a battery of tests in various languages and the opportunity to administer the test in neuroimaging studies [4, 5] have made the RAT ubiquitous in the study of associative basis of semantic cognition.

An interesting cognitive aspect of the RAT solving is the semantic search process in which an individual attempts to find a solution to a RAT problem. Insight solutions to a RAT problem occur abruptly, without voluntary control of a person solving the task, and are preceded by unconscious processing [5]. In contrast, analytical solutions are derived gradually as the person attempts to solve a task [9]. Analytical solving usually allows more time to respond, which correlates with better test performance [10, 11]. One of the first proposals of semantic processing in the RAT involving subconscious processing is the spreading activation theory [12]: every cue word activates a subset of words in a semantic network. Overlapping activations of a unit crossing the threshold give rise to a solution. However, this theory does not provide an account of mechanisms which underlie the search process nor does it explain the variety in performance. Better understanding of the analytical RAT solving has been provided by studies analysing human responses when participants solve RAT problems by reporting every word they think of [13, 14]. Common findings confirm that the search process is restricted by all three cues, with one primary cue being used to generate a response. Evidence was provided in favor of a local sequential search process in the RAT [13], meaning that the next guess is chosen based on the previous one.

The RAT has been also investigated in various computational studies [15–18]. Statistical approaches using large corpora of text and natural language processing tools have yielded a performance which is comparable to or even better than the human performance on the test [16, 17]. Mednick's theory on associative hierarchies [8] was scrutinised by identifying relevant properties of a semantic network of an individual [18–20]. Semantic networks of individuals scoring well on the RAT satisfy small-world network properties, where every word in the network can be reached in a few steps by following associated words of any other word. Qualitative differences in associative networks between individuals with low and high creative abilities have been described in [19]. A modelling study investigating the generation of ideas in spontaneous thought in a neural network [20] confirmed that scale-free and small-world network properties are important for efficient search in the memory generating conventional and creative ideas. A Markov Chain Monte Carlo model [15] successfully reproduced experimentally derived human response patterns [13] supporting findings on the local search strategy. While providing valuable insights into the understanding of memory search, these models offers a limited explanation of the cognitive mechanisms involved in the semantic processing in the RAT and their relation to the test performance.

Summarising the existing work, the evidence suggests that both the executive cognitive component guiding the search process and the organisation of the semantic network are important for the semantic processing in the RAT. To identify a minimal set of computational primitives underlying these processes, we propose a neural network in which every layer realises a different function interpreted in the theoretical framework of semantic processing

2 The Model

We propose a neural network model of the RAT solving based on the spreading activation [12] and a winner-take-all (WTA) mechanism. Solving of a RAT problem is realised as a search for a

Algorithm 1 Search by spreading activation and a winner-take-all function

$nr\_visited \leftarrow 0$
$visited = [\,]$
for $i \leftarrow 1, N$ **do**
    $activations_i \leftarrow 0$
for all c in cues **do**
    $I_c \leftarrow 1$
$j \leftarrow -1$
while $nr\_visited < max\_nodes$ **do**
    for all neighbours i of the current node j **do**
        if $w_{ij} > \vartheta_s$ **then**
            $activations_i \leftarrow activations_i + w_{ij} activations_j$
    $activations_i \leftarrow activations_i + I_i$
    $visited \leftarrow visited + j$
    $nr\_visited \leftarrow nr\_visited + 1$
    if $j == target$ **then**
        $break$
    $j \leftarrow WTA(activations)$

solution in a semantic network. To better understand the workings of the neural network, we first present the search algorithm which is then implemented by the network. The equations of the neural network model are presented together with an explanation of how algorithm components map to computations performed by neuron-like units.

2.1 Search Algorithm

The search for a solution is done separately for every RAT problem consisting of three cue words and a target solution. For every problem, the word cues are used to initialise the search process. In every step of the search process, a word is selected as a candidate solution to a problem. Every word is represented as a node in a graph and every node has a level of activity which changes as the search unfolds. Thus, the search for a solution to a RAT problem is modelled as a graph traversal and the direction of the search is determined by the levels of node activities. If the selected node matches the target word, or a certain number of words in the search has been exhausted, the search process terminates.[1] The activity of neighbours of a currently visited node is elevated by the amount proportional to the connection strength between the node and its neighbours. Connection strengths for $N = 5018$ words are derived from a freely available database of word association strengths acquired in a free association task [21]. In the task, the participants were instructed to write a single word which first came to their mind when prompted with a cue word. To obtain the associative strength between word pairs, the number of participants who responded with a specific word to a cue was divided by the number of participants performing the same task. Only responses which were given by at least two participants were considered.

After elevating the activities of neighbouring nodes, a new node will be selected by the WTA function and the old node will be annotated as visited. In the algorithm, the WTA function sorts node indices based on their activities and returns the index of a node with the highest activity level which has not been visited before. In this way, words which have already been considered as a solution will not appear again in the search process. If two nodes have the same level of activity, the first node returned by the sort algorithm will be selected. The search path is created by appending every visited node to the list of previously visited nodes.

In the beginning, the activity of all nodes is set to zero. Every node receives an external input, which is set to one only for the nodes representing the problem cues, and zero for all other nodes. Based on high activity levels, the WTA sequentially selects three problem cues and the activity spreads through the network from the problem cues first. Only after all three cues have been visited, the

[1] Here we do not model the process of determining whether a response is a solution to the RAT problem. Instead, we focus on the process of ideation of possible words, assuming when an individual comes up with a word they will be able to determine if it is a correct solution by comparing it against the task constraints.

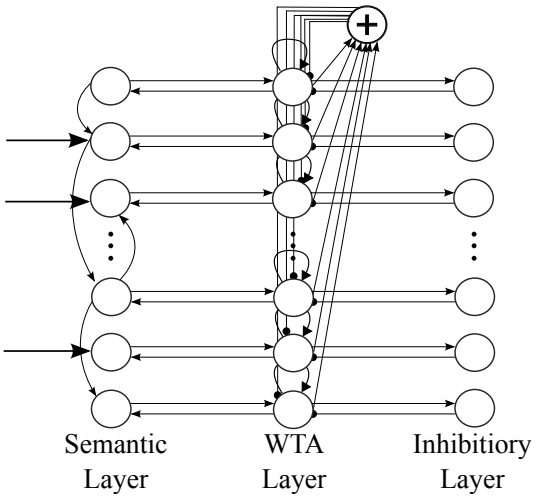


Figure 1: Neural network model of the semantic search in the RAT. The word with the highest activity level in the semantic layer is selected by the WTA layer as a response to a RAT problem. If the selected word does not match the solution the inhibitory layer suppresses the activity of the selected word allowing the WTA to select the next winner.

WTA will select a new node as a candidate solution. The process terminates when the selected response matches the target or when a certain number of nodes has been visited. The pseudocode for the search algorithm is shown in Algorithm 1. In every step the WTA function selects a node j, which is appended to the list of visited nodes after it has been processed. In the first iteration, there is no spreading of the activity from any node as there is no winning node ($j = -1$). This occurs in the second step when the WTA has selected the first problem cue receiving the external input. This condition is handled in the code and omitted here for clarity.

To explore the influence of different word pair association strengths on the RAT performance, we implement spreading of activity only to those neighbours whose connection strength to the processing node is greater than the spreading threshold ϑ_s.[2] Weak connection strengths stand for rare and uncommon associations, which according to Mednick's theory [8] are more likely to be generated by highly creative individuals. Increasing the threshold corresponds to removal of such association pairs, resulting in longer number of steps between two nodes or, for very high thresholds, inability to reach a node.

2.2 Neural Network

A three-layer neural network model is used to simulate solving of RAT problems using the search algorithm described in Algorithm 1. The three layers are the semantic layer, winner-take-all (WTA) layer and inhibitory layer. All three layers have distinct functional roles. The model scheme is shown in Figure 1. The semantic layer represents a semantic network, consisting of a vocabulary and the associative relationships between the words in the vocabulary. The WTA layer selects a winning unit simulating a solution guess to a RAT problem. The spread of activity from a winning unit in the semantic layer is done via feedback connections from the WTA layer to the semantic layer. If the selected word is not a solution to the RAT problem, the inhibitory layer suppresses the activity of the winning unit in the WTA layer. This allows the next unit with the highest activity level to win in the next step. The activity of units in all layers is updated in parallel in each time step.

The activity of units in the first layer can be written as:

[2]This is equivalent to rectifying all $w_{ij} < \vartheta_s$ to zero.

$$a_i(t+1) = a_i(t) + \rho_a\Big(\sum_{j=1}^{N} W_{ij} z_j(t) a_j(t) + I_i(t)\Big) \tag{1}$$

$$z_i(t) = \Theta(w_i(t) - \vartheta_w) \tag{2}$$

$$\|z(t)\| = 1 \tag{3}$$

where $a_i(t)$ describes a non-decaying activity of a unit i at time t where $i = 1, \ldots, N$. W_{ij} is the connection strength from the unit j to the unit i, extracted from the database of free associations [21]. The constant ρ_a is inversely proportional to the stimulus length t_n and allows the analytical derivation of precise values of activity levels (not shown). The binary vector $z(t)$ always has exactly one or zero active elements, with the active element representing the currently visited node (variable j in Algorithm 1) in the semantic layer. The winner is a word considered as a solution to the RAT problem. Consistent with the algorithm, only the activity of nodes whose edges incident to the winning node are greater than the spreading threshold ϑ_s, or the activity of nodes receiving external input, will be elevated. Thresholding is done by setting the weights smaller than ϑ_s in the connection matrix W to zero. The winning node is set based on unit activities in the WTA layer. If a unit in the WTA layer crosses the threshold ϑ_w, the Heaviside step function will toggle the corresponding bit in $z(t)$. Only at the beginning of a simulation, the external input $I(t)$ is used to sequentially increase the activity of cue nodes in the semantic layer. This in turn will elevate the activity of the units in the WTA layer yielding a winner whose feedback connection will activate the spread of activity from a winning unit in the semantic layer. The first three winners are thus going to be the three problem cues. $I_i(t)$ is clamped to one for the duration t_n starting at non-overlapping times t_i for nodes representing problem cues: $t = t_i : I_{cue_i}(t_i) = 1$ where cue_i is the index of a problem cue $i \in \{1, 2, 3\}$. The activity of the units in the WTA layer is described as:

$$w_i(t+1) = w_i(t) + \rho_w\big[c_1 z_i(t) + c_2 \widetilde{a}_i(t) - c_3 y(t) - c_4 r(t) + c_5 \eta_i(t)\big] \tag{4}$$

$$y(t+1) = \sum_{j=1}^{N} z_j(t) \tag{5}$$

Units in this layer receive one-to-one feedforward input $\widetilde{a}_i$ from the units in the first layer. The first WTA unit to cross the threshold ϑ_w will be the one receiving the strongest normalised input $\widetilde{a}_i(t)$.[3]. It will activate the self-excitatory connection of a strength c_1, and the input to the single inhibitory neuron $y(t)$ (a unit with the '+' sign in Figure 1). The inhibitory neuron will project feedback connections to all units in the layer, such that the activity of all the units apart from the winning one will be suppressed and kept below the threshold ϑ_w. The winning neuron will also toggle the corresponding bit in the $z(t)$ as described in equation 2 and elevate activities of its neighbours in the semantic layer. Due to the self-excitation, the activity of a winning unit in the WTA layer will continue to increase until suppressed by a unit in the third layer. The noise term $\eta_i(t)$ randomly drawn from a uniform distribution is added to allow the WTA to select a winning unit when two or more units receive the same input $\widetilde{a}(t)$ from the semantic layer. Finally, after a unit $z_i(t)$ has been active for a certain amount of time t_n it will activate the corresponding inhibitory unit in the third layer:

$$r_i(t+1) = r_i(t) + \Theta\Big(\sum_{j=t-t_n}^{t} z_i(t_j) - t_n\Big) \tag{6}$$

The feedback connections to the WTA layer will inhibit the activity of the winning unit. As input integration in $r(t)$ is non-decaying, the inhibition will be permanent preventing the winning unit from winning again. A new unit will be selected in the semantic layer based on the highest level of activity $\widetilde{a}_i(t)$.

[3] $\widetilde{a}_i(t) = \frac{a_i(t)}{a_k(t)}$ where $k = \arg\max_j a_j(t)$. The activity has been normalised to bound the input range for the units in the WTA layer. Knowing the input range it is possible to analytically derive the parameter values for which the precision of the competition mechanism can be controlled.

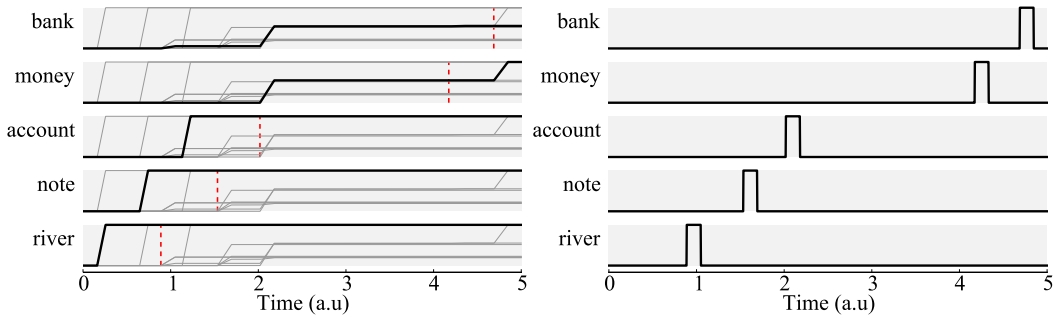


Figure 2: Normalised unit activities in the semantic layer (left) and the winning units (right) for one simulation of a RAT problem with the problem cues: *river, note, account*. The solution *bank* is found as a second response, after *money*. Because the first three winners are the problem cues they are not considered as potential solutions. Red dashed lines in the left plot represent a moment when a winning unit has been selected and determines the onset of spreading activity to its neighbouring units.

Figure 2 shows unit activities in the network over a course of time while solving a RAT problem: *river, note, account*. First three winners in the network are the cues *river, note* and *account* and therefore not considered as solutions. The first response *money* is inhibited as it does not match the solution to this RAT problem. The correct response *bank* is chosen next and the simulation of this RAT problem is terminated. For visual clarity, only a small fraction of activated units in the semantic layer is shown.

3 Results

To test the model performance we use 117 out of 144 RAT items [10] for which the cues and the target are available in the free association database [21]. To obtain problem difficulties, we divide 117 problems into three categories (easy, medium and hard) based on the percentage of participants solving an item in the 15 seconds condition [10]. The percentage of participants solving a problem varies between 0% and 96% and we divide the categories in three equal parts: easy problems are solved by 64%–96% participants (17 problems), medium by 32%–64% (43 problems) and hard problems by 0%–32% participants (58 problems).

The performance on all three categories is tested by varying two model parameters: the number of responses and the spreading threshold. The number of responses is the number of words in the search path, starting from the first word selected by the WTA. If the correct response was not among that predetermined number of responses, the problem is annotated as unsolved. As a first approximation, the number of responses can be related to the time allowed to produce the answer. The basic assumption is that with more time a participant will be able to think of more words. The second parameter is the spreading threshold ϑ_s in the semantic layer. By increasing the threshold we are discarding all word pair associations with the association strength weaker than the threshold. Lower association strengths correspond to word association pairs given by fewer individuals in the free association task [21]. Thus, increasing the thresholding reduces the number of uncommon and rare cue-target associations. Such associations, interpreted in the context of Mednick's theory on associative hierarchies [8], are more likely to be produced by highly creative individuals, as the associations of low creative individuals are characterised by stereotypy and commonality. We explore the relationship between the importance of such associations and the performance on the RAT.

Network simulation results for 117 problems and the three difficulty categories are shown in Figure 3. As expected, the performance on the RAT increases with the number of responses. Compared to the medium and hard items, all easy items are solved with fewer responses (10 responses). Approximately 20% of easy problems are solved with a single response, implying that the solution to easy RAT items is a close association between one of the problem cues and the solution. Close associates are strong word associations, in this case, the word pairs with high association strengths in the free

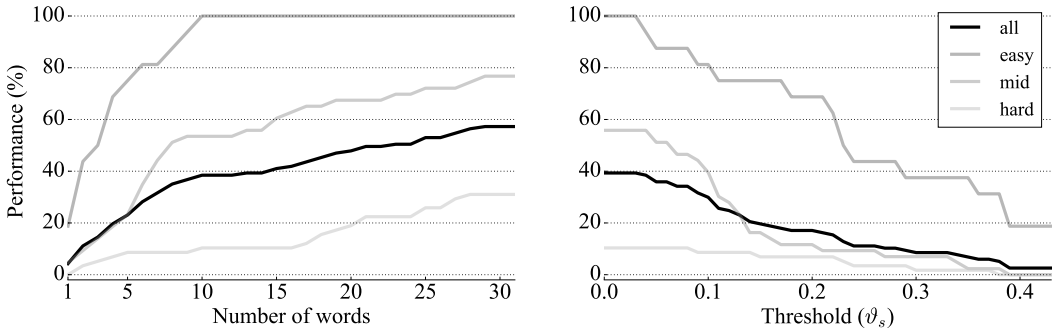


Figure 3: Performance on the easy, medium and difficult RAT items depending on the number of responses in the search process (left) and the spreading threshold (right). Increasing the threshold removes word pair associations with association strengths weaker than the threshold.

association database [21]. Such close associations emerge because many participants in the free association task have responded with the same word to a cue word. A continuous increase in the performance for medium and hard problems is observed when 15 or more responses are considered. For the purposes of current analysis, we have restricted the range of responses to an interval which could be interpreted in the context of known data. Participants instructed to report every word they consider as a solution when trying to solve a RAT problem on average produce eight words within two minutes [13], although there are large variations in the number of responses. As we are not explicitly modelling this process, we take this number as a reference. Therefore, we assume this number to be greater in the model which does not model a specific cognitive strategy that would differentiate between reported and unreported words. The average percentage of solved items for 117 tested problems for humans in 15 sec condition was 28.8%. With six responses the model yields similar performance (28.2%) on the same set of problems.

Right plot in Figure 3 shows how removing word associations has different effects on the test performance depending on the problem difficulty. The weight value of 0.4 is the 98th percentile of all non-zero association strengths in the free association database [21]. Overall, removing word associations impairs the performance on the test for all problem difficulties. However, easier and difficult problems are affected differently. Relative to the performance on the RAT without pruning of associations ($\vartheta_s = 0$), the drop in performance by 50% occurs at lower threshold values for problems of medium difficulty compared to easy problems. For easy items, 50% decrease in performance occurs when all word pairs of association strength $\vartheta_s = 0.23$ (94th percentile) or lower are removed, while for the items of medium difficulty this already occurs for the threshold value of $\vartheta_s = 0.12$ (87th percentile). This effect is also observed for smaller drops in performance when comparing the performance on the RAT problems of medium and hard difficulty with easy RAT problems, and when using a different number of words (here only shown for 15 words). This indicates that less common word pair associations are important for the better performance on the difficult RAT problems.

4 Discussion

With this work we have aimed to identify a basic set of computational mechanisms underlying the semantic processing in the RAT solving. This has been done by devising a neural network model in a way consistent with the existing understanding of human semantic processing. Semantic layer in the model implements a localist representation of lexical knowledge. Memory search in the semantic layer is realised by spreading activation and a WTA network, both cognitively realistic mechanisms. The spreading of activity has been specified in the context of associative hierarchies relevant for characterising creative abilities of an individual [8]. It is assumed to occur at the subconscious level of semantic processing [12, 22]. The WTA mechanism and the inhibitory layer responsible for selecting a single word in the vocabulary are reminiscent of attentional mechanisms mediating cognitive control attributed to the function of the frontal brain areas [4, 5]. The focus of attention is directed towards a single word which is selected among several competing alternatives. Anterior

cingulate cortex (ACC) has been shown to play an important role in attentional switches and conflict resolution in case of competing alternatives [23, 24].

The model is able to distinguish between easy and difficult RAT problems in the normative RAT data set [10]: easy items are solved within fewer response attempts and, compared to the more difficult items, are less affected by the removal of unusual word associations. This is in accordance with Mednick's theory on associative hierarchies [8] according to which creative individuals are more likely to produce less stereotypical and uncommon word associations. Thus, lower threshold values in the model might be related to the ability of an individual to consider such associations. Alternatively, and not exclusively, this could be a property of the organisation of an individual's semantic network. The semantic network constructed from a free association database [21] satisfies small-world properties with power-law degree distribution [25] important for a good performance on the RAT [19]. Therefore, semantic networks of individuals scoring lower on the RAT might lack such associations, resulting in compromised small-world network organisation. When interpreted in the context of number of words needed to produce a correct response to a RAT problem, more difficult items have a solution which is more distant from the problem cues in the search path. This would justify the difficult RAT problems as having more "remote" associations.

While the proposed model is based on theories of semantic processing and biologically realistic mechanisms, a detailed theory of different cognitive strategies in the RAT is needed to model the differences in the underlying processes. Different hemispheric contributions in semantic processing have been observed when people solve the RAT by insight and analytically [5]. It is to expect that analytic solving requires greater engagement of semantic and working memory, reasoning and other systems involved in problem solving. Therefore, different and possibly overlapping brain networks would require a more comprehensive, large-scale brain model, simulating interactions among several brain regions realising different cognitive functions. With this work, instead of capturing the processing spanning several brain regions, we have focused on a minimal set of basic neurocomputational mechanisms independently of a cognitive strategy. Future work will address the extension and expansion of the model. One advantage of a large-scale model would be a biologically realistic, distributed representation of sensory information which can better capture associative nature of word representations at the neural level. It remains to be explored how a higher level of biological realism can inform our understanding of associative knowledge and creative problem solving.

5 Conclusion

We have developed and presented a neural network model of semantic processing in the Remote Associates Test, a commonly used paradigm in the research of insight and creative problem solving. The model is based on the theory of spreading activation in semantic processing, and uses neurally realistic computations. We have shown that the model exhibits human like performance on the task, demonstrating that uncommon and less stereotypical associations are important for a good performance on difficult test problems. Finally, the Python scripts used for the processing of the free association data and the complete source code are available online at `http://github.com/ikajic/remote_associates_test`.

Acknowledgments

The authors would like to thank Terry Stewart, Vaibhav Tyagi, and Michael Klein for discussions that helped to improve this paper. They would also like to thank Haline Schendan and Giorgio Ganis for their valuable comments and insights throughout the project. This work has been supported by the Marie Curie Initial Training Network FP7-PEOPLE-2013-ITN, CogNovo, grant number 604764.

References

[1] Margaret A. Boden. *The Creative Mind: Myths and Mechanisms*. Routledge, 2 edition, November 2003.

[2] Dietrich Arne. The Cognitive Neuroscience of Creativity. *Psychonomic Bulletin & Review*, 11(6):1011–1026, 2004.

[3] E. Paul Torrance. *Guiding creative talent*. Prentice-Hall, 1962.

[4] Andreas Fink, Mathias Benedek, Roland H. Grabner, Beate Staudt, and Aljoscha C. Neubauer. Creativity meets neuroscience: Experimental tasks for the neuroscientific study of creative thinking. *Methods*, 42(1):68–76, 2007.

[5] John Kounious and Mark Beeman. The cognitive neuroscience of insight. *Annual Review of Psychology*, 65:71–93, 2014.

[6] Sebastien Helie and Ron Sun. Incubation, insight, and creative problem solving: a unified theory and a connectionist model. *Psychological Review*, 117(3):994–1024, 2010.

[7] Mark Jung-Beeman, Edward M. Bowden, Jason Haberman, Jennifer L. Frymiare, Stella Arambel-Liu, Richard Greenblatt, Paul J. Reber, and John Kounios. Neural Activity When People Solve Verbal Problems with Insight. *PLoS Biology*, 2(4):e97, 04 2004.

[8] Sarnoff A. Mednick. The associative basis of the creative process. *Psychological Review*, 69(3):220–232, 1962.

[9] Roderick W. Smith and John Kounios. Sudden insight: All-or-none processing revealed by speed–accuracy decomposition. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 22(6):1443, 1996.

[10] Edward M. Bowden and Mark Jung-Beeman. Normative data for 144 compound remote associate problems. *Behavior Research Methods, Instruments, & Computers: A Journal of the Psychonomic Society, Inc*, 35(4):634–639, 2003.

[11] Stephen G. Harkins. Mere effort as the mediator of the evaluation-performance relationship. *Journal of Personality and Social Psychology*, 91(3):436–455, 2006.

[12] Allan M. Collins and Elizabeth F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407 – 428, 1975.

[13] Kevin A. Smith, David E. Huber, and E Vul. Multiply-constrained semantic search in the remote associates test. *Cognition*, 128:64–75, 2013.

[14] Eddy J. Davelaar. Semantic search in the remote associates test. *Topics in Cognitive Science*, 7(3):494–512, 2015.

[15] David D. Bourgin, Joshua T. Abbott, Thomas L. Griffiths, Kevin A. Smith, and Edward Vul. Empirical Evidence for Markov Chain Monte Carlo in Memory Search. In *Annual Conference of the Cognitive Science Society*, 2014.

[16] Ariel Klein and Toni Badia. The Usual and the Unusual: Solving Remote Associates Test Tasks Using Simple Statistical Natural Language Processing Based on Language Use. *The Journal of Creative Behavior*, 49(1):13–37, 2015.

[17] Hannu Toivonen, Oskar Gross, Jukka M. Toivanen, and Alessandro Valitutti. On Creative Uses of Word Associations. In *Synergies of Soft Computing and Statistics for Intelligent Data Analysis*, volume 190 of *Advances in Intelligent Systems and Computing*, pages 17–24. Springer Berlin Heidelberg, 2013.

[18] Padraic Monaghan, Tom Ormerod, and Ut N. Sio. Interactive activation networks for modelling problem solving. In *Computational models of cognitive processes: Proceedings of the 13th Neural Computation and Psychology Workshop*, volume 21, pages 185–195, 2014.

[19] Yoed N. Kenett, David Anaki, and Miriam Faust. Investigating the structure of semantic networks in low and high creative persons. *Frontiers in Human Neuroscience*, 8(407), 2014.

[20] Nagendra Marupaka, Laxmi R. Iyer, and Ali A. Minai. Connectivity and thought: the influence of semantic network structure in a neurodynamical model of thinking. *Neural Networks*, 32:147–158, 2012.

[21] Douglas L. Nelson, Cathy L. McEvoy, and Thomas A. Schreiber. The University of South Florida Free Association, Rhyme, and Word Fragment Norms. *Behavior Research Methods, Instruments, & Computers*, 36(3):402–407, 2004.

[22] Ilan Yaniv and David E. Meyer. Activation and metacognition of inaccessible stored information: potential bases for incubation effects in problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13(2):187, 1987.

[23] Matthew M. Botvinick, Jonathan D. Cohen, and Cameron S. Carter. Conflict monitoring and anterior cingulate cortex: an update. *Trends in Cognitive Sciences*, 8(12):539 – 546, 2004.

[24] John G. Kerns, Jonathan D. Cohen, Angus W. MacDonald, Raymond Y. Cho, V. Andrew Stenger, and Cameron S. Carter. Anterior Cingulate Conflict Monitoring and Adjustments in Control. *Science*, 303(5660):1023–1026, 2004.

[25] Mark Steyvers and Joshua B Tenenbaum. The Large-scale structure of semantic networks: Statistical analyses and a model of semantic growth. *Cognitive Science*, 29(1):41–78, 2005.

Probability Matching via Deterministic Neural Networks

Milad Kharratzadeh
Department of Electrical & Computer Engineering
McGill University
Motreal, Canada
milad.kharratzadeh@mail.mcgill.ca

Thomas Shultz
Department of Psychology
& School of Computer Science
McGill University
Montreal, Canada
thomas.shultz@mcgill.ca

Abstract

We propose a constructive neural-network model comprised of deterministic units which estimates and represents probability distributions from observable events — a phenomenon related to the concept of probability matching. We use a form of operant learning, where the underlying probabilities are learned from positive and negative reinforcements of the inputs. Our model is psychologically plausible because, similar to humans, it learns to represent probabilities without receiving any representation of them from the external world, but rather by experiencing individual events. Also, we discuss how the estimated probabilities can be used in a setting with deterministic units to produce matching behaviour in choice. Our work is a step towards understanding the neural mechanisms underlying probability matching behavior by specifying processes at the algorithmic level.

1 Introduction

The matching law states that the rate of a response is proportional to its rate of observed reinforcement and has been applied to many problems in psychology and economics [1, 2]. A closely related empirical phenomenon is probability matching where the predictive probability of an event is matched with the underlying probability of its outcome [3]. For example, in decision theory, many experiments show that participants select alternatives proportional to their reward frequency. This means that in many scenarios, instead of maximizing their utility by always choosing the alternative with the higher chance of reward, they match the underlying probabilities of different alternatives. This is in contrast with the reward-maximizing strategy of always choosing the most probable outcome. The apparently suboptimal behaviour of probability matching is a long-standing puzzle in the study of decision making under uncertainty and has been studied extensively [4–9].

In this paper, we provide a psychologically plausible neural framework to explain probability matching at Marr's implementation level [10]. We introduce an artificial neural network framework which can be used to explain how deterministic neural networks can learn to represent probability distributions, even without receiving any direct representations of these probabilities from the external world. We offer an explanation of how the network is able to estimate and represent probabilities solely from observing the occurrence patterns of events, in the manner of probability matching. In the context of Bayesian models of cognition, such probability-matching processes could explain the origin of the prior and likelihood probability distributions that are currently assumed or constructed by modelers. Thus, in contrast to current literature that proposes probability matching as an alternative to Bayesian models [11, 12], we argue that probability matching can be seen as part of a larger Bayesian framework to learn prior and likelihood distributions which can then be used for Bayesian inference.

2 Problem Statement

We provide a neural-network framework with deterministic units capable of implementing probability matching, i.e., learning the underlying probabilities (knowledge) and making choices using those probabilities (use). We assume that a direct representation of these probabilities from the external world is not available, and the probabilities must be estimated from input instances reinforced at various frequencies. For example, for a stimulus, s, reinforced on k out of its total n presentations in the training set, probability matching yields $\widehat{P}(s) = k/n$.

Mathematically, we assume the task of learning a probability mass function $P : H \rightarrow [0, 1]$, where H is a discrete hypothesis space. The training set consists of a collection of input instances reinforced with a frequency proportional to an underlying probability function; i.e., the hypothesis h_i is paired with observations sampled from $Bernoulli\,(P(h_i))$ where 1 corresponds to a positive reinforcement and 0 corresponds to a negative reinforcement. Then, the knowledge part of probability matching reduces to estimating the actual probabilities from these 0 or 1 observations. This is in accordance with the real-world scenarios, where observations are in the form of events which can occur or not (represented by outputs of 1 and 0, respectively) and the learner does not have access to the actual probabilities of those events.

We use deterministic neural networks where each unit takes a weighted sum of inputs from some other units and, using its activation function, computes its output. These outputs are propagated through the network until the network's final outputs are computed in the last layer. We consider a neural network with a single input unit (taking h_i) and a single output unit (representing the probability). Our goal is to learn a network that outputs $P(h_i)$ when h_i is presented at the input.

In classical artificial neural networks, the target values are fixed and deterministically derived from the underlying function and the corresponding inputs. However, in the problem we consider here, we do not have access to the final, fixed targets (i.e., the actual probabilities). Instead, the training set is composed of input instances that are reinforced with various frequencies. An important question is whether a network of deterministic units can learn the underlying probability distributions from such 0, 1 observations. And if yes, how? We answer these two questions in Sections 4 and 5, respectively. Then, in Section 6, we show the learned probabilities can be used to produce matching behaviour.

3 Related Work

Our proposed scheme differs from the classical approach to neural networks in that there is no one-to-one relationship between inputs and output. Instead of being paired with one fixed output, each input is here paired with a series of 1s and 0s presented separately at the output unit. Moreover, in our framework, the actual targets (underlying probabilities) are hidden from the network and, in the training phase, the network is presented only with inputs and their probabilistically varying outputs.

The main difference of our work with the current literature (and the main novelty of this work) is the use of a population of deterministic units to learn the probabilities and producing the matching behaviour. The relationship between neural network learning and probabilistic inference has been extensively studied mainly with stochastic units that fire with particular probabilities. Boltzmann machines [13] and their various derivatives, including Deep Learning in hierarchical restricted Boltzmann machines (RBM) [14], have been proposed to learn a probability distribution over a set of inputs. There are many other papers studying probabilistic computations that can be done using similar networks (e.g., [15–18]). See [19] for a more comprehensive review.

In our model, representation of probability distributions emerges as a property of a network of deterministic units rather than having individual units with activations governed by some probability distribution. Moreover, models with stochastic units such as RBM "require a certain amount of practical experience to decide how to set the values of numerical meta-parameters" [20], which makes them neurally and psychologically implausible for modeling probability matching in the relatively autonomous learning of humans or animals. As we see later, our model implements the probability matching in a relatively autonomous, neurally–plausible fashion, by using deterministic units in a constructive learning algorithm that builds the network topology as it learns.

4 Statistical Properties

In this section, we show that the successful training of a deterministic neural network to minimize the output error for the problem defined in Section 2 results in learning the underlying probabilities.

Remember that the training set consists of hypotheses, h_i, paired with a sequence of probabilistic outputs, r_{ij}, set to either 1 (positive reinforcement) or 0 (negative reinforcement). The frequency of the reinforcement (outputs of 1) is determined by the underlying probability distribution. The structure of the network and the weights are adjusted (more details later) to minimize the sum-of-squares error:

$$\mathcal{E} = \frac{1}{2}\sum_{i,j}(o_i - r_{ij})^2, \tag{1}$$

where o_i is the network's output when h_i is presented at the input layer. We show that minimizing this error, results in learning the underlying distribution: if we present a sample input, the output of the network would be its probability of being reinforced. Note that we never present this probability explicitly to the network. This means that the network learns and represents the probability distributions from observing patterns of events.

The statistical properties of feed-forward neural networks with deterministic units have been studied as non-parametric density estimators. Denote the inputs of a network with X and the outputs with Y (both can be vectors). In a probabilistic setting, the relationship between X and Y is determined by the conditional probability $P(Y|X)$. In [21] and [22], White showed that under certain assumptions, feed-forward neural networks with a single hidden layer can consistently learn the conditional expectation function $E(Y|X)$. However, as White mentions, his analyses "do not provide more than very general guidance on how this can be done" and suggests that "such learning will be hard" [21, p. 454]. Moreover, these analyses "say nothing about how to determine adequate network complexity in any specific application with a given training set of size n" [21, p. 455]. In this section, we extend these results to a more general case with no restrictive assumptions about the structure of the network and the learning algorithm. Then, in the next section, we propose a learning algorithm that automatically determines the adequate network complexity in any specific application.

In the following, we state the theorem and our learning technique for the case where $Y \in \{0, 1\}$, since in this case $E(Y = 1|X) = P(Y = 1|X)$. Thus, learning results in representing the underlying probabilities in the output unit. The extension of the theorem and learning algorithm to more general cases is straightforward.

Theorem 1. Assume that $P : H \to \mathbb{R}$ is a probability mass function on a hypothesis space, H, and we have observations $\{(h_i, r_{ij}) \mid r_{ij} \sim \text{Bernoulli}(P(h_i)),\ h_i \in H\}$. Define the network error as the sum–of–squares error at the output:

$$\mathcal{E}_p = \frac{1}{2}\sum_{i}\sum_{j=1}^{n}(o_i - r_{ij})^2. \tag{2}$$

where o_i is the network's output when h_i is presented at the input, and r_{ij} is the probabilistic output determining whether the hypothesis h_i is reinforced ($r_{ij} = 1$) or not ($r_{ij} = 0$). Then, any learning algorithm that successfully trains the network to minimize the output sum–of–squared error yields probability matching (i.e., reproduces f in the output).

Proof. Minimizing the error, we have:

$$\nabla\mathcal{E}_p = \left(\frac{\partial\mathcal{E}_p}{\partial o_1}, \ldots, \frac{\partial\mathcal{E}_p}{\partial o_m}\right) = \left(n \cdot o_1 - \sum_{j=1}^{n} r_{1j}, \ldots, n \cdot o_m - \sum_{j=1}^{n} r_{mj}\right) = 0 \tag{3}$$

$$\Rightarrow o_i^* = \sum_{j=1}^{n}\frac{r_{ij}}{n}\ , \quad \forall i. \tag{4}$$

According to the strong law of large numbers $o_i^* \overset{a.s.}{\to} E[r_{ij}] = P(h_i), \forall h_i \in H$, where $\overset{a.s.}{\to}$ denotes almost sure convergence. Therefore, the network's output converges to the underlying probability distribution, P, at all points. □

Theorem 1 shows the important point that neural networks with deterministic units are able to asymptotically estimate an underlying probability distribution solely based on observable reinforcement rates. Unlike previous similar results in literature [21–23], Theorem 1 does not impose any constraint on the network structure, the learning algorithm, or the distribution being learned. However, an important assumption in this theorem is the successful minimization of the error by the learning algorithm. As pointed out earlier, two important questions remain to be answered: (i) how can this learning be done? and (ii) how can adequate network complexity be automatically identified for a given training set? In the next section, we address these problems and propose a learning framework to successfully minimize the output error.

5 The Learning Algorithm

The outputs in the training set, paired with input hypotheses, are 0 or 1. Our goal in probability matching is not to converge to any of these values, but to the underlying probability. To achieve that goal we use the idea of learning cessation [24]. The learning cessation method monitors learning progress in order to autonomously abandon unproductive learning. It checks the absolute difference of consecutive errors and if this value is less than a fixed threshold multiplied by the current error for a fixed number of consecutive learning phases (called patience), learning is abandoned. This technique for stopping deterministic learning of stochastic patterns does not require the psychologically unrealistic validation set of training patterns [25, 26].

Our method is presented in Algorithm 1 where we represent the whole network (units and connections) by the variable `NET`. Also, the learning algorithm we use to train our network is represented by the operator `train_one_epoch`, where an epoch is a pass through all of the training patterns. We can use any algorithm to train our network, as long as it successfully minimizes the error term in (2). Next, we present a learning algorithm that can achieve that goal.

Algorithm 1 Probability matching with neural networks and learning cessation

Input: Training Set $S_{train} = \{(h_i, r_{ij}) \mid h_i \in X\ ; r_{ij} \sim \text{Bernoulli}(P(h_i))\}$;
Cessation threshold ϵ_c; Cessation patience $patience$
Output: Learned network outputs $\{o_i\ ,\ i = 1, \ldots, m\}$
$counter \leftarrow 0, t \leftarrow 0$
while true **do**
  $(\{o_i \mid i = 1, \ldots, m\}, \texttt{NET}) \leftarrow \texttt{train\_one\_epoch}(\texttt{NET}, S_{train})$ ▷ Updating the network
  $\mathcal{E}_p(t) \leftarrow \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} (o_i - r_{ij})^2$ ▷ Computing the updated error
  if $|\mathcal{E}_p(t) - \mathcal{E}_p(t-1)| \geq \epsilon_c \cdot |\mathcal{E}_p(t)|$ **then** ▷ Checking the learning progress
    $counter \leftarrow 0$
  else
    $counter \leftarrow counter + 1$
    if $counter = patience$ **then**
      break
    end if
  end if
  $t \leftarrow t + 1$
end while

Theorem 1 proves that the minimization of the output sum–of–squared error yields probability matching. However, the unusual properties of the training set we employ (such as the probabilistic nature of input/output relations) as well as the fact that we do not specify the complexity of the underlying distribution in advance may cause problems for some neural learning algorithms. The most widely used learning algorithm for neural networks is Back Propagation, also used in [27] in the context of probability matching. In Back Propagation (BP), the output error is propagated backward and the connection weights are individually adjusted to minimize this error. Despite its many successes in cognitive modeling, we do not recommend using BP in our scheme for two important reasons. First, when using BP, the network's structure must be fixed in advance (mainly heuristically). This makes it impossible for the learning algorithm to automatically adjust network complexity to the

problem at hand [21]. Moreover, this property limits the generalizability and autonomy of BP and also, along with back-propagation of error signals, makes it psychologically implausible. Second, due to their fixed design, BP networks are not suitable for cases where the underlying distribution changes over time. For instance, if the distribution over the hypothesis space gets much more complicated over time, the initial network's complexity (i.e., number of hidden units) would fall short of the required computational power.

Instead of BP, we use a variant of the cascade correlation (CC) method called sibling-descendant cascade correlation (SDCC) which is a constructive method for learning in multi-layer artificial neural networks [28]. SDCC learns both the network's structure and the connection weights; it starts with a minimal network, then automatically trains new hidden units and adds them to the active network, one at a time. Each new unit is employed at the current or a new highest layer and is the best of several candidates at tracking current network error.

The SDCC network starts as a perceptron topology, with input units coding the example input (in our case, a single unit coding the input) and output units coding the correct response to that input (in our case, a single unit representing the probability). In a constructive fashion, deterministic units are recruited into the network one at a time as needed to reduce error. In classical CC, each new recruit is installed on its own layer, higher than previous layers. The SDCC variant is more flexible in that a recruit can be installed either on the current highest layer (as a sibling) or on its own higher layer as a descendent, depending on which location yields the higher correlation between candidate unit activation and current network error [28]. In both CC and SDCC, learning progresses in a recurring sequence of two phases – output phase and input phase. In output phase, network error at the output units is minimized by adjusting connection weights without changing the current topology. In the input phase, a new unit is recruited such that the correlation between its activation and network error is maximized. In both phases, the optimization is done by the Quickprop algorithm [29].

SDCC offers two major advantages over BP. First, it constructs the network in an autonomous fashion (i.e., a user does not have to design the topology of the network, and also the network can adapt to environmental changes). Second, its greedy learning mechanism can be orders of magnitude faster than the standard BP algorithm [30]. SDCC's relative autonomy in learning is similar to humans' developmental, autonomous learning [31]. With SDCC, our method implements psychologically realistic learning of probability distributions, without any preset topological design. The psychological and neurological validity of cascade-correlation and SDCC has been well documented in many publications [32, 33]. These algorithms have been shown to accurately simulate a wide variety of psychological phenomena in learning and psychological development. Like all useful computational models of learning, they abstract away from neurological details, many of which are still unknown. Among the principled similarities with known brain functions, SDCC exhibits distributed representation, activation modulation via integration of neural inputs, an S-shaped activation function, layered hierarchical topologies, both cascaded and direct pathways, long-term potentiation, self-organization of network topology, pruning, growth at the newer end of the network via synaptogenesis or neurogenesis, weight freezing, and no need to back-propagate error signals.

6 Generating Matching Behaviour with Deterministic Units

The term "probability matching" either refers to learning the underlying probabilities or to making choices using those probabilities. So far, we explained how a neural network can learn to estimate the probabilities. In this section, we discuss how this estimated probability can be used in a setting with deterministic units to produce matching behaviour in choice. We show that deterministic units with simple thresholding activation functions and added Gaussian noise in the input can generate probabilistic outputs similar to probability matching behaviour. Assume that we have a neuron with two inputs: the estimated probability that a response is correct, $0 \leq v \leq 1$, and a zero–mean Gaussian noise, $\epsilon \sim \mathcal{N}(0, \gamma)$. Then, given the thresholding activation function, the output will be 1 if $v + \epsilon > \tau$ and 0 if $v + \epsilon \leq \tau$ for a given threshold τ. Therefore, the probability of producing 1 at the output is:

$$P(\text{output} = 1|v, \tau, \gamma) = P(v + \epsilon > \tau) = P(\epsilon > \tau - v) = \overbrace{0.5 - 0.5 \text{ erf}\left(\frac{\tau - v}{\gamma\sqrt{2}}\right)}^{f(v)}, \qquad (5)$$

where *erf* denotes the error function: $\mathrm{erf}(x) = (2/\sqrt{\pi}) \int_0^x e^{-t^2}\, \mathrm{d}t$. It is easy to see that $f(v)$ lies between 0 and 1 and, for appropriate choices of τ and γ, we have $f(v) \simeq v$ for $0 < v < 1$ (see Fig. 1). Thus, a single thresholding unit with additive Gausian noise in the input can use the estimated probabilities to produce responses that match the response probabilities (similar to the matching behaviour of people using probabilistic knowledge to make their choices).

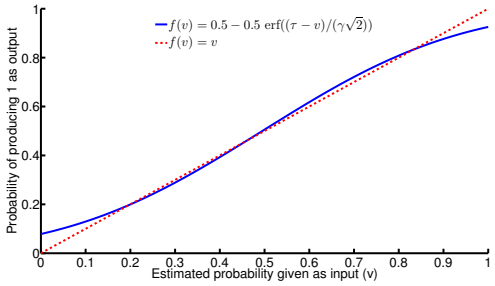
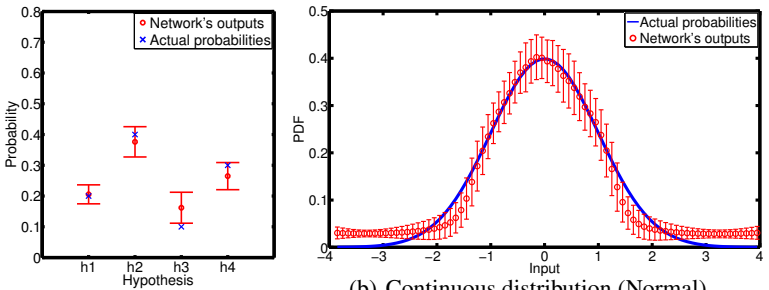


Figure 1: A deterministic unit with a thresholding activation function generating responses that match the probabilities that each response is correct ($\tau = 1, \gamma = 0.35$)

7 Simulation Study

7.1 Probability Matching

Through simulations, we show that our proposed framework is indeed capable of learning the underlying distributions. We consider two cases here, but similar results are observed for a wide range of distributions. First, we consider a case of four hypotheses with probability values $0.2, 0.4, 0.1$, and 0.3. Also, we consider a Normal probability distribution where the hypotheses correspond to small intervals on the real line from -4 to 4. For each input sample we consider 15 randomly selected instances in each training epoch. As before, these instances are positively or negatively reinforced independently and with a probability equal to the actual underlying probability of that input. We use SDCC with learning cessation to train our networks. Fig. 2, plotted as the average and standard deviation of the results for 50 networks, demonstrates that for both discrete and continuous probability distributions, the network outputs are close to the actual distribution. Although, to save space, we show the results for only two sample distributions, our experiments show that our model is able to learn a wide range of distributions including Binomial, Poisson, Gaussian, and Gamma [34]. Replication of the original probability distribution by our model is important, because, contrary to previous models, it is done without stochastic neurons and without any explicit information about the actual distribution or fitting any parameter or structure in advance. Moreover, it is solely based on observable information in the form of positive and negative reinforcements.



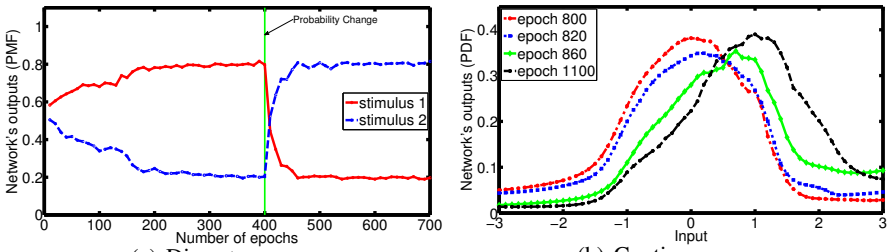
(a) Discrete distribution (b) Continuous distribution (Normal)

Figure 2: Replication of the underlying probability distribution by our SDCC model. The results (mean and standard deviation) are averaged over 50 different networks.

7.2 Adapting to Changing Environments

In many naturally–occurring environments, the underlying reward patterns change over time. For example, in a Bayesian context, the likelihood of an event can change as the underlying conditions change. Because humans are able to adapt to such changes and update their internal representations of probabilities, successful models should have this property as well. We examine this property in the following example experiment. Assume we have a binary distribution where the possible outcomes have probabilities 0.2 and 0.8, and these probabilities change after 400 epochs to 0.8 and 0.2, respectively. In Fig. 3(a), we show the network's outputs for this scenario. We perform a similar simulation for the continuous case where the underlying distribution is Gaussian and we change the mean from 0 to 1 at epoch 800; the network's outputs are shown in Fig. 3(b). We observe that in both cases, the network successfully updates and matches the new probabilities.

We also observe that adapting to the changes takes less time than the initial learning. For example, in the discrete case, it takes 400 epochs to learn the initial probabilities while it takes around 70 epochs to adapt to the new probabilities. The reason is that for the initial phase, constructive learning has to grow the network until it is complex enough to represent the probability distribution. However, once the environment changes, the network has enough computational capability to quickly adapt to the environmental changes with a few internal changes (in weights and/or structure). We verify this in our experiments. For instance, in the Gaussian example, we observe that all 20 networks recruited 5 hidden units before the change and 11 of these networks recruited 1 and 9 networks recruited 2 hidden units afterwards. We know of no precise psychological evidence for this reduction in learning time, but our results serve as a prediction that could be tested with biological learners. This would seem to be an example of the beneficial effects of relevant existing knowledge on new learning.



(a) Discrete case (b) Continuous case

Figure 3: Reaction of the network to the changes in target probabilities.

8 Discussion

A mentioned before, probability matching (choosing alternatives proportionally to their reward frequency) is in contrast with the reward-maximizing strategy of always choosing the most probable outcome. There are numerous, and sometimes contradictory, attempts to explain this choice anomaly. Some suggest that probability matching is a cognitive shortcut driven by cognitive limitations [3, 4]. Others assume that matching is the outcome of misperceived randomness which leads to searching for patterns even in random sequences [5, 35]. It is shown that as long as people do not believe in the randomness of a sequence, they try to discover regularities in it to improve accuracy [6]. It is also shown that some of those who perform probability matching in random settings have a higher chance of finding a pattern when one exists [7]. In contrast to this line of work, some researchers argue that probability matching reflects a mistaken intuition and can be overridden by deliberate consideration of alternative choice strategies [8]. In [9], the authors suggest that a sequence-wide expectation regarding aggregate outcomes might be a source of the intuitive appeal of matching. It is also shown that people adopt an optimal response strategy if provided with (i) large financial incentives, (ii) meaningful and regular feedback, or (iii) extensive training [36].

Our neural-network framework is compatible with all these accounts of probability matching. Firstly, probability matching is the norm in both humans [37] and animals [38, 39]. It is clear that in these settings agents who match probabilities form an internal representation of the outcome

probabilities. Even for particular circumstances where a maximizing strategy is prominent [7,36], it is necessary to have some knowledge of the distribution to produce optimal-point responses. Having a sense of the distribution provides the flexibility to focus on the most probable point (maximizing), sample in proportion to probabilities (matching), or even generate expectations regarding aggregate outcomes (expectation generation), all of which are evident in psychology experiments.

Probabilistic models of cognition can be defined with either symbolic or continuous representations, or hybrids of both. In fact, more effective routes to understanding human intelligence can be found by combining these two traditionally opposing approaches using a statistical inference scheme over structured symbolic knowledge representations [40]. Our proposed neural interpretation of probabilistic representations helps to explore that interface in greater depth.

References

[1] R. J. Herrnstein, "Relative and absolute strength of response as a function of frequency of reinforcement," *Journal of the Experimental Analysis of Behaviour*, vol. 4, pp. 267–272, 1961.

[2] ——, *The Matching Law: Papers on Psychology and Economics*, H. Rachlin and D. Laibson, Eds. Cambridge, MA: Harvard University Press, 2000.

[3] N. Vulkan, "An economist's perspective on probability matching," *Journal of Economic Surveys*, vol. 14, no. 1, pp. 101–118, 2000.

[4] R. F. West and K. E. Stanovich, "Is probability matching smart? associations between probabilistic choices and cognitive ability," *Memory & Cognition*, vol. 31, no. 2, pp. 243–251, 2003.

[5] G. Wolford, S. E. Newman, M. B. Miller, and G. S. Wig, "Searching for patterns in random sequences." *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, vol. 58, no. 4, p. 221, 2004.

[6] J. I. Yellott Jr, "Probability learning with noncontingent success," *Journal of mathematical psychology*, vol. 6, no. 3, pp. 541–575, 1969.

[7] W. Gaissmaier and L. J. Schooler, "The smart potential behind probability matching," *Cognition*, vol. 109, no. 3, pp. 416–422, 2008.

[8] D. J. Koehler and G. James, "Probability matching in choice under uncertainty: Intuition versus deliberation," *Cognition*, vol. 113, no. 1, pp. 123–127, 2009.

[9] G. James and D. J. Koehler, "Banking on a bad bet probability matching in risky choice is linked to expectation generation," *Psychological Science*, vol. 22, no. 6, pp. 707–711, 2011.

[10] D. Marr, *Vision*. San Francisco, CA: W. H. Freeman, 1982.

[11] J. S. Bowers and C. J. Davis, "Bayesian just-so stories in psychology and neuroscience." *Psychological Bulletin*, vol. 138, no. 3, pp. 389–414, 2012.

[12] F. Eberhardt and D. Danks, "Confirmation in the cognitive sciences: The problematic case of Bayesian models," *Minds and Machines*, vol. 21, no. 3, pp. 389–410, 2011.

[13] H. Ackley, G. Hinton, and J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive Science*, pp. 147–169, 1985.

[14] G. Hinton and S. Osindero, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527 – 1554, 2006.

[15] J. Movellan and J. L. McClelland, "Learning continuous probability distributions with symmetric diffusion networks," *Cognitive Science*, vol. 17, pp. 463–496, 1993.

[16] J. L. McClelland, "Connectionist models and bayesian inference," *Rational models of cognition*, pp. 21–53, 1998.

[17] T. S. Jaakkola, L. K. Saul, and M. I. Jordan, "Fast learning by bounding likelihoods in sigmoid type belief networks," in *Advances in Neural Information Processing Systems 22*, 1996.

[18] J. L. McClelland, D. Mirman, D. J. Bolger, and P. Khaitan, "Interactive activation and mutual constraint satisfaction in perception and cognition," *Cognitive science*, vol. 38, no. 6, pp. 1139–1189, 2014.

[19] M. Kharratzadeh and T. R. Shultz, "Neural implementation of probabilistic models of cognition," *arXiv preprint arXiv:1501.03209*, 2015.

[20] G. Hinton, "A practical guide to training restricted boltzmann machines," *Momentum*, vol. 9, no. 1, p. 926, 2010.

[21] H. White, "Learning in artificial neural networks: A statistical perspective," *Neural computation*, vol. 1, no. 4, pp. 425–464, 1989.

[22] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[23] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, "Backpropagation: The basic theory," in *Backpropagation: Theory, Arcitecture, and applications*, Y. Chauvin and D. E. Rumelhart, Eds., Hillsdale, NJ, USA, 1995, pp. 1–34.

[24] T. Shultz, E. Doty, and F. Dandurand, "Knowing when to abandon unproductive learning," in *Proceedings of the 34th Annual Conference of the Cognitive Science Society*, Austin, TX: Cognitive Science Society, 2012, pp. 2327–2332.

[25] L. Prechelt, "Early stopping - but when?" in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Orr and K.-R. Muller, Eds. Berlin: Springer, 1998, vol. 1524, pp. 55–69.

[26] C. Wang, S. S. Venkatesh, and J. S. Judd, "Optimal stopping and effective machine complexity in learning," in *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1993, pp. 303–310.

[27] M. Dawson, B. Dupuis, M. Spetch, and D. Kelly, "Simple artificial neural networks that match probability and exploit and explore when confronting a multiarmed bandit," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1368–1371, 2009.

[28] S. Baluja and S. E. Fahlman, "Reducing network depth in the cascade-correlation learning architecture," Carnegie Mellon University, School of Computer Science, Tech. Rep., 1994.

[29] S. E. Fahlman, "Faster-learning variations on back-propagation: An empirical study," in *Proc. of the Connectionist Models Summer School*. Los Altos, CA: Morgan Kaufmann, 1988, pp. 38–51.

[30] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*. Loas Altos, CA: Morgan Kaufmann, 1990, pp. 524–532.

[31] T. Shultz, "A constructive neural-network approach to modeling psychological development," *Cognitive Development*, vol. 27, pp. 383–400, 2012.

[32] ——, *Computational Developmental Psychology*. Cambridge, MA: MIT Press, 2003.

[33] ——, "Computational models of developmental psychology," in *Oxford Handbook of developmental Psychology, Vol. 1: Body and mind*, P. D. Zelazo, Ed. Newyork: Oxford University Press, 2013.

[34] M. Kharratzadeh and T. Shultz, "Neural-network modelling of Bayesian learning and inference," in *Proceedings of the 35th Annual Meeting of Cognitive Science*. Austin, TX: Cognitive Science Society, 2013, pp. 2686–2691.

[35] G. Wolford, M. B. Miller, and M. Gazzaniga, "The left hemisphere's role in hypothesis formation." *The Journal of Neuroscience*, 2000.

[36] D. R. Shanks, R. J. Tunney, and J. D. McCarthy, "A re-examination of probability matching and rational choice," *Journal of Behavioral Decision Making*, vol. 15, no. 3, pp. 233–250, 2002.

[37] D. R. Wozny, U. R. Beierholm, and L. Shams, "Probability matching as a computational strategy used in perception," *PLoS computational biology*, vol. 6, no. 8, p. e1000871, 2010.

[38] K. L. Kirk and M. Bitterman, "Probability-learning by the turtle," *Science*, vol. 148, no. 3676, pp. 1484–1485, 1965.

[39] U. Greggers and R. Menzel, "Memory dynamics and foraging strategies of honeybees," *Behavioral Ecology and Sociobiology*, vol. 32, no. 1, pp. 17–29, 1993.

[40] T. L. Griffiths, C. Kemp, and J. B. Tenenbaum, "Bayesian models of cognition," 2008.

The Usefulness of Past Knowledge when Learning a New Task in Deep Neural Networks

Guglielmo Montone*
Laboratoire Psychologie de la Perception
Université Paris Descartes
75006 Paris, France
montone.guglielmo@gmail.com

J. Kevin O'Regan*
Laboratoire Psychologie de la Perception
Université Paris Descartes
75006 Paris, France
jkevin.oregan@gmail.com

Alexander V. Terekhov*
Laboratoire Psychologie de la Perception
Université Paris Descartes
75006 Paris, France
avterekhov@gmail.com

Abstract

In the current study we investigate the ability of a Deep Neural Network (DNN) to reuse, in a new task, features previously acquired in other tasks. The architecture we realized, when learning the new task, will not destroy its ability in solving the previous tasks. Such architecture was obtained by training a series of DNNs on different tasks and then merging them to form a larger DNN by adding new neurons. The resulting DNN was trained on a new task, with only the connections relative to the new neurons allowed to change. The architecture performed very well, requiring few new parameters and a smaller dataset in order to be trained efficiently and, on the new task, outperforming several DNNs trained from scratch.

1 Introduction

Deep Neural Networks (DNNs) have yielded impressive results during the last decade, in many cases outperforming other existing machine learning algorithms, bringing the dream of building intelligent machines closer to becoming a reality. But despite these impressive results, DNNs are still limited in their ability to replicate many characteristics of the human brain. The learning process in humans, for example, seems to involve the creation of abstract concepts and strategies. Such concepts and strategies can then be re-applied to new scenarios that share symbols or relations between symbols with familiar tasks. This capacity is accompanied by two other human abilities that it would be very useful to reproduce in DNNs: the ability to learn a new task from a very limited set of examples, and the ability to use a gradual and sequential learning process, where more complex tasks are presented only after the learner is familiar with simpler tasks.

The brain can be considered as a very large neural network where some areas are principally involved in one specific task or computation and others are used for multiple tasks. Two characteristics of the brain likely contribute to the development of abstraction abilities in humans. Specifically, the neural system is able to:

- store previously learned capacities,
- reuse areas used for solving previous tasks to solve a new task.

*lpp.psycho.univ-paris5.fr/feel/

One strategy for implementing such characteristics in a DNN could consist of a training algorithm that differentiates between the areas in the network that have been highly involved in solving a previous task from areas that were exploited less intensively. In this way could be possible to realize a training algorithm such that the areas involved in previous tasks would be slightly effected by the training while the areas less exploited previously would be more deeply modified.

It is likely that a network trained in such a way would be able not only to store previously learned tasks, but also to reuse some previously learned features and strategies when performing a new task. In this paper we present a step towards building such a network. To do so, we first trained several networks on different tasks. We then connected the networks together in a larger DNN, adding new groups of neurons. The resulting architecture was then trained on a new task, with only the newly introduced areas involved in the training. Our aim was to show that such an architecture is able to solve several tasks and to reuse areas trained on previous task to perform new tasks.

Many interesting attempts to implement these desired characteristics in a DNN can be found in the literature. The *pre-training* of a network is one example. In this technique, before a network is trained on the desired task, it is trained on a similar task in either a supervised or unsupervised fashion. This technique has been shown to work as a regularizer [8], and when used with the *dropout* [11] technique, has been shown to produce clear advantages compared to a network that is trained from scratch. Advantages that are particularly evident when the first and second tasks are functionally similar or when the tasks have a common input format [10]. However, training on a second task is often accompanied by *catastrophic forgetting* of the first task, with performance on the first task severely reduced by the new training.

Another technique, usually referred to as *multitask learning* [5], is also relevant to the construction of networks that benefit from learning several tasks. This technique consists of simultaneously training a DNN on several tasks. It has been recently applied in the domain of natural language processing [6, 7, 13]. Multitask learning has shown promising results. It seems that simultaneously learning on several tasks forces the network to learn more general features, which results in the network performing well even when trained on smaller datasets. However, the major disadvantage of this technique is that the network must be trained on all tasks in parallel, making it impossible to sequentially train the network with tasks of increasing difficulty. Although it has not yet been widely studied, a sequential learning strategy, similar to Bengio's *curriculum learning*, could be quite powerful in training DNN [3].

The method we propose in this paper consists of training a series of DNNs on different tasks. These networks are then connected with groups of new neurons, forming a bigger network to be trained on a task similar to the previous ones. Importantly, only connections relative to newly added neurons are learned during the new training. Using this method, the architecture does not suffer from catastrophic forgetting; and more importantly, as we will show, it can exploit the positive properties of pre-training and multitasking, namely:

- a smaller dataset is sufficient for training on a new task,
- fewer computational resources are needed in order to learn a new task.

In the next section, we illustrate the way in which we merge various DNNs into a single larger network. In section 1.2 we describe the artificially created tasks, and explain why they are particularly well suited for investigating the ability of the network to reuse previously learned features. In section 2 we give specific details about the DNNs used and the learning procedure. In section 3 we present the results. In the last section we suggest directions for further research.

1.1 Merging DNNs

Imagine that we have trained a DNN on several tasks at the same time, and that we have a learning algorithm which is able to differentiate between the areas of the network that were heavily used in any of the tasks and those that were less used. Now let us imagine training the network on a new task, and suppose that the training algorithm could be constructed so as to focus on the areas that were less used for the previous tasks. Such an architecture, we hypothesized, would be able to learn multiple tasks, even when presented in a sequential order, and would be able to *efficiently reuse previously trained areas*. The work presented here empirically confirms the previous hypothesis. The procedure used was as follows.

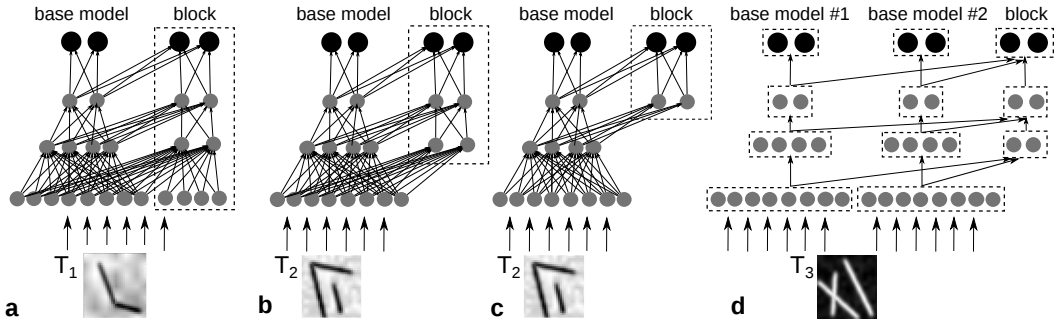


Figure 1: (a) The architecture built by adding a block neurons with three hidden layers to one base model. (b) Adding a block neurons with two hidden layers to one base model. (c) Adding a block neurons with one hidden layer to one base model. (d) Adding a block neurons to two base models. The dashed boxes indicate the layers of the two base models and the block neurons added. An arrow connecting two boxes indicates that all the neurons in the first box are connected to all the neurons in the second box.

We defined a set of tasks $T_1, \ldots, T_M$ and trained a DNN $N_1, \ldots, N_M$ on each task. The networks used were feed-forward DNNs with three hidden layers. After the first training phase, we used some of the trained networks, say $N_1, \ldots, N_m$, to build a *block architecture* that was then trained on one of the remaining tasks, say T_{m+1}. The block architecture was formed by adding a set of new neurons to the previously trained networks N_1,...,N_m (we refer to such networks as *base models* and to the added neurons as *block neurons*). The block neurons together comprised a DNN, which we connected to the base models as follows. The first hidden layer of the block neurons received the input for the task T_{m+1}. The same input was sent to all networks N_1,...,N_m. The second hidden layer was fully connected to both the first hidden layer of the block neurons and the first hidden layer of each network N_1,...,N_m. This pattern was repeated for the third hidden layer of the block neurons, which was fully connected to its own second hidden layer and to the second hidden layer of each network N_1,...,N_m. Finally, the output layer of the block neurons received the input coming from its own third layer and from the third layers of each network. Figure 1(a) illustrates this architecture when block neurons were added to only one base model. In this work, this architecture was tested along with two variations. In the first variation, figure 1(b), the block neurons did not have a first hidden layer. The second hidden layer was fully connected to the first hidden layer each network N_1,...,N_m. In the other variation, figure 1(c), the second layer of block neurons was also removed, with the only remaining hidden layer receiving input from the second layer of each network. When training on the task T_{m+1} *none of the parameters in the base model networks was allowed to change*. Training on task T_{m+1} was performed exclusively on the connections linking the base model to the layers of block neurons and on the connections between the different layers of block neurons.

Our results from the first study on this kind of architecture have been published elsewhere [16]. In the present study, the DNNs were trained using a dataset half the size of that used in the previous work. Moreover, a much larger number of architectures was tested. In particular, we constructed architectures with up to 5 base models, while reducing the number of hidden layers in the block neurons, thereby reducing the number of parameters (weights and neurons) associated solely with the block neurons. Finally, we tested the capacity of the block architecture to learn from a dataset much smaller than the one used to train the base models.

1.2 The tasks

We developed six binary classification tasks, which the networks were trained on. The tasks all involved the concepts of line and angle. We wished to show that the networks $N_1, \ldots, N_m$, when trained on such tasks, would develop features that could be reused by the block architecture to solve another task involving the same concepts.

In each task the stimuli were gray scale images, 32 x 32 pixels in size. Each image contained two to four line segments, each at least 13 pixels long (30% of the image diagonal). The segments were white on a dark random background or black on a light random background. The distance between

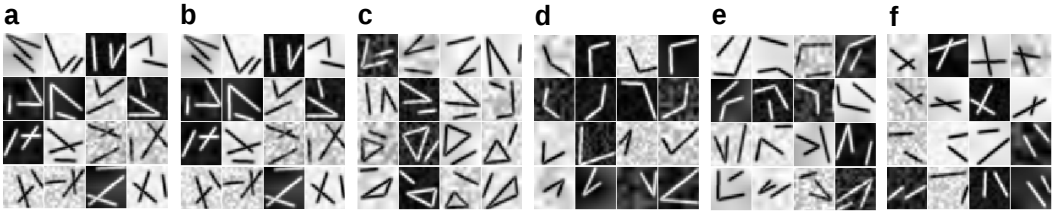


Figure 2: Examples of stimuli: (a) *ang_crs* – line segments forming an angle vs. two crossing line segments; (b) *ang_crs_ln* – same, with an additional non-crossing line segment; (c) *ang_tri_ln* – angle vs. triangle; (d) *blt_srp* – blunt angle vs. sharp angle; (e) *blt_srp_ln* – same, with a non-crossing line segment; (f) *crs_ncrs* – two crossing line segments vs. two non-crossing line segments.

the end points of each line segment and every other line segment was at least 4 pixels (10% of the image diagonal). In order to obtain anti-aliased images, the lines were first generated on a grid three times larger (96 x 96). The images were then filtered with a Gaussian filter with a sigma of 3 pixels, and down-sampled to the final dimensions. The 6 tasks were (see examples in figure 2):

ang_crs: requires classifying the images into those containing an angle (between 20° and 160°) and a pair of crossing line segments (the crossing point must lay between 20% and 80% along each segment's length).

ang_crs_ln: the same as *ang_crs*, but has an addition line segment crossing neither of the other line segments.

ang_tri_ln: distinguishes between images containing an angle (between 20° and 160°) and a triangle (with each angle between 20° and 160°); each image also contains a line segment crossing neither angle nor triangle.

blt_srp: requires classifying the images into those having blunt (between 100° and 160°) and those having sharp (between 20° and 80°) angles in them.

blt_srp_ln: the same as *blt_srp*, but has an additional line segment, crossing neither of the line segments forming the angle.

crs_ncrs: distinguishes between a pair of crossing and a pair of non-crossing lines (the crossing point must lay between 20% and 80% of each segment length).

Each stimulus was generated by randomly selecting the parameters describing each figure (length and orientation of straight segments, amplitude and orientation of the angles) and verifying that all conditions were satisfied. Four different types of random backgrounds were generated with four patterns changing with different velocities. For our experiments we generated 350,000 stimuli for each condition.

2 Network details

The networks used were feed-forward DNNs with three hidden layers. The number of hidden units in the different experiments varied, but the largest network had 200, 100, and 50 units in its first, second, and third hidden layers, respectively. We refer to these networks with the letters NN followed by the number of neurons in each layer. Thus, the name of the network just described would be NN-200-100-50.

We constructed different types of block architectures by varying the type of base models used, the number of base models used, and the number of neurons per layer in the block neurons. We refer to a block architecture obtained by connecting base models to a DNN with 100 units in the first hidden layer, 50 units in the second hidden layer, and 50 units in the third hidden layer as BA-100-50-50. We refer to an architecture with no hidden units in the first layer and 50 units in the other two hidden layers as BA-0-50-50, while an architecture with no units in the first and second hidden layers and 50 units in the third layer would be named BA-0-0-50.

All networks used a softmax logistic regression as the output layer. The activation functions of the neurons of the network were *rectified linear units* (ReLUs) [14]. Each network was trained to minimize a cost function which combines three terms: the negative log-likelihood of the prediction given the data, and two regularization terms. The first regularization term forces sparsity in the neurons activation and consists of the KL divergence between the mean activation of each neuron and a uniform probability distribution of value 0.05. The second regularization term is the squared sum of all the weights in the networks. The coefficient of the first regularization term was equal to 0.01. The weight-limiting coefficient was set to 0.0001. The weights of the k-th hidden level were initialized, according to [9], with random uniformly distributed values in the range $\pm\sqrt{6/(n_{(k-1)}+n_{(k)})}$, where $n_{(k)}$ is the number of neurons at the k-th level and $n_{(0)}$ is equal to the number of inputs. The biases were all initialized to 0.

The total dataset was split into training (330,000 samples), validation (10,000 samples), and test (10,000 samples) datasets. All of the architectures were trained on the entire training dataset using mini-batch gradient descent learning with a batch size of 20. The initial learning rate for the gradient descent was set to 0.01, and it decreased by a factor 0.985 after every epoch. We used early stopping of the training process if the error on the validation dataset did not decrease after 5 epochs. The test score corresponding to the minimal validation error is presented as the performance of the network. The initial learning rate was selected using a human-guided search. Different values of the initial learning rate were tested, with uniform steps for the logarithm of the tested values taken over the interval $[log(0.1), log(10^{-6})]$.

All code was written in python using Theano [1, 4]. Source files are available online: `https://github.com/feel-project/abstraction`

3 Results

In this section, we first report the results obtained by training a DNN on each of the previously described tasks. Then we report the results of training different block architectures on the same tasks. We present the results for block architectures with one or two base models, then the results for those with three, four, or five base models. The number of possible architectures that can be built by changing the base models, the number of block neurons and the task on which the block architecture is trained, is very large, and exploring all possibilities was not feasible. Instead, we tried to select the conditions that best allowed us to obtain an understanding of the performance of the block architectures. The performance was evaluated by computing the percentage of misclassified samples on the test dataset. Each architecture was trained five times, randomly initializing its weights. The mean performance over the five repetitions and the best and worst performance are reported in the tables.

Original networks

Prior to building block architectures, we trained a DNN on each task. The networks used were of type NN-200-100-50, with 200, 100, and 50 nodes in the first, second, and third layers, respectively. Networks of this type were used as base models for all of the block architectures. The percentages of misclassified test examples for these networks are shown in table 1 together with the results for other architectures, namely NN-30-20-10, NN-60-40-30 and NN-70-50-30. Each of these last three networks had approximately the same number of parameters (weight of the networks) of some of the block architectures, making interesting performance comparisons possible.

One and two base-model architectures

The percentages of misclassified test examples for block architectures with one and two base models are presented in the tables 2 and 3, respectively. The architectures that performed better than (or equal to) the network NN-200-100-50, which was trained from scratch, are shown in bold. In these tables, the tasks on which the block architectures were trained are listed together with the tasks on which the base models were trained (in parentheses).

Architectures with just one base model (table 2) rarely outperformed the original network, NN-200-100-50. This only happened when the task on which the block architecture was trained was similar to the one used to train the base model, namely for the conditions *ang_crs_ln* (*ang_crs*) and

Table 1: Original networks result.

condition	200-100-50	70-50-30	60-40-20	30-20-10
ang_crs	5.5 (5.4–5.9)	8.6 (8.1–9.3)	9.4 (8.9–9.8)	13.3 (12.8–15.0)
ang_crs_ln	13.6 (12.5–15.2)	17.2 (16.5–17.4)	18.3 (16.7–18.8)	21.9 (21.5–23.5)
ang_tri_ln	6.1 (5.5–6.8)	9.7 (8.8–10.1)	11.4 (10.6–14.0)	14.1 (13.4–15.2)
blt_srp	2.0 (1.8–2.3)	3.4 (3.1–3.8)	3.7 (3.4–4.2)	5.9 (5.1–6.3)
blt_srp_ln	6.5 (6.4–6.9)	10.7 (10.0–13.7)	12.5 (11.6–14.1)	17.2 (16.6–19.5)
crs_ncrs	2.8 (2.3–2.9)	3.7 (3.2–4.3)	4.5 (4.1–5.2)	5.5 (4.6–6.1)

The numbers correspond to the median (min–max) percentage of misclassified examples.

blt_srp (*blt_srp_ln*). Even when a second base model was added, *ang_crs_ln* and *blt_srp* remained the only two tasks on which the block architecture outperformed the original network, NN-200-100-50. It is interesting to compare the results of the architecture BA-100-50-50 with one base model to those of the architecture BA-0-50-50 with two base models. In this example, even though it has fewer parameters, the architecture with two base models outperforms that with one in three of the four tasks on which it was tested, namely *blt_srp*, *blt_srp_ln* and *ang_crs*. This result seem to suggests that increasing the number of base models yields better results than increasing the number of neurons in the block architecture. We decided to further study this possibility by adding even more base models.

Table 2: Adding blocks to one base model.

condition	0-50-50	0-100-50	100-50-50
ang_crs_ln (*ang_crs*)	13.8(13.7–14.3)	**13.4 (13.1–13.6)**	**13.2 (13.1 – 14.0)**
blt_srp_ln (*blt_srp*)	9.2 (8.8–9.4)	8.4(8.3–8.6)	8.1 (8.0–8.6)
ang_crs (*ang_crs_ln*)	6.0 (5.9 – 6.2)	5.8 (5.7–6.1)	6.3 (6.0–6.4)
blt_srp (*blt_srp_ln*)	**1.8(1.7–1.9)**	**1.6(1.5–1.9)**	**1.8 (1.5–2.3)**
ang_tri_ln (*ang_crs_ln*)	11.2 (11.1–11.7)	10.8 (10.2–11.1)	7.5(7.1–8.0)
blt_srp_ln (*ang_crs_ln*)	11.7 (11.3–12.3)	10.9 (10.5–11.2)	8.0 (7.6–8.7)
ang_crs_ln (*ang_tri_ln*)	17.3 (16.6–18.2)	16.6 (15.9–17.2)	14.4 (13.4–14.7)
ang_crs_ln (*blt_srp_ln*)	18.1 (17.2–18.6)	17.1 (16.2–17.3)	14.5 (14.3–14.9)

The tasks on which the block architectures were trained are listed together with the tasks on which the base models were trained (in parentheses). The architectures that performed better than (or equal to) the networks NN-200-100-50 trained from scratch are shown in bold. This is also the case for Tables 3-8.

Table 3: Adding blocks to two base models.

condition	0-50-50	0-100-50	50-50-50
ang_crs_ln (*ang_tri_ln*+*crs_ncrs*)	13.7 (13.1–14.2)	**13.2 (12.9–13.3)**	**13.1(12.7–13.8)**
ang_crs_ln (*ang_tri_ln*+*blt_srp_ln*)	14.2(13.8–14.6)	**13.6(12.9–13.8)**	**13.5(12.9–14.2)**
blt_srp (*ang_tri_ln*+*crs_ncrs*)	**1.9 (1.8–2.3)**	**1.6(1.5–1.7)**	**1.8(1.6–2.1)**
blt_srp (*ang_tri_ln*+*ang_crs_ln*)	**2.0(1.7–2.3)**	**1.5(1.4–1.8)**	**1.7(1.5–1.7)**
blt_srp (*ang_tri_ln*+*blt_srp_ln*)	**1.5 (1.3–1.7)**	**1.3 (1.3–1.6)**	**1.4 (1.2–1.7)**
blt_srp_ln (*ang_tri_ln*+*ang_crs_ln*)	7.6 (7.5–7.8)	7.3 (7.1–7.8)	6.9 (6.8–7.3)
blt_srp_ln (*ang_tri_ln*+*crs_ncrs*)	7.6 (7.5–7.8)	7.3 (7.1–7.8)	6.9 (6.8–7.3)
ang_crs (*ang_tri_ln*+*blt_srp_ln*)	6.2 (6.0–6.3)	5.7 (5.6–5.9)	5.6 (5.5–5.7)
ang_crs (*ang_tri_ln*+*ang_crs_ln*)	6.2 (6.0–6.3)	5.7 (5.6–5.9)	5.6 (5.5–5.7)

Architectures with more than two base models

In this paragraph we present the results obtained adding three, four and five base-models to two kinds of block-architectures: BA-0-50-50 and BA-0-0-50. In table 4 are presented the percentages of misclassified test examples obtained using three base-models. The results for the architecture BA-0-50-50 were better than those obtained for the architecture NN-60-40-20 that has the same number of parameters(approximately 60,000 weights), and better, for nearly all tasks, than those for the architecture NN-200-100-50. This suggest that the block architecture highly benefits from

Table 4: Adding blocks to three base-models.

condition	0-50-50	0-0-50
ang_crs (*ang_tri_ln*+*crs_ncrs*+*blt_srp*)	**5.5 (5.2–5.8)**	6.0 (5.7–6.1)
ang_crs (*ang_tri_ln*+*ang_crs_ln*+*crs_ncrs*)	**4.0 (3.9–4.1)**	**4.5 (4.2 – 4.6)**
ang_crs (*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*)	**4.6 (4.3–4.8)**	5.9 (5.6 – 6.0)
ang_crs_ln (*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*)	**11.3 (10.9–11.8)**	15.0 (14.1 – 16.3)
ang_crs_ln (*ang_tri_ln*+*crs_ncrs*+*blt_srp*)	**12.0 (11.6–12.5)**	15.2 (15.0–15.9)
blt_srp (*ang_crs*+*ang_tri_ln*+*crs_ncrs*)	**1.4 (1.4–1.6)**	2.2 (2.0 –2.3)
blt_srp (*ang_crs_ln*+*ang_tri_ln*+*crs_ncrs*)	**1.5 (1.3–1.8)**	2.4 (2.0–3.0)
blt_srp_ln (*ang_crs_ln*+*ang_tri_ln*+*crs_ncrs*)	7.0 (6.7–7.4)	10.2(9.9 –10.5)
blt_srp_ln (*ang_crs*+*ang_tri_ln*+*crs_ncrs*)	6.9 (6.7–7.4)	9.8 (9.3–10.1)

being constituted by several base models trained on different tasks. Eliminating the second hidden layer from the block architecture yielded much poorer results. For example, the architecture BA-0-0-50 outperformed NN-200-100-50 on only one task. However, the results for BA-0-0-50 improved significantly when the number of base models was increased to four (Table 5). This performance was comparable to that of BA-0-50-50 with three base models. However, at the same time, it seems that for the architecture BA-0-0-50 with four base models the choice of the base models has a big influence on the final performance. This can be observed, for example, in the task *ang_crs_ln* where using to performance drops from 11.4 to 14.1 when just one of the base model is changed.

Table 5: Adding blocks to four base models.

condition	0-50-50	0-0-50
ang_crs (*ang_tri_ln*+*crs_ncrs*+*blt_srp*+*blt_srp_ln*)	**4.9 (4.8–5.6)**	**5.3 (5.0–5.7)**
ang_crs (*ang_tri_ln*+*ang_crs_ln*+*crs_ncrs*+*blt_srp_ln*)	**3.6 (3.4–3.8)**	**4.3 (4.0–4.7)**
ang_crs (*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*+ *ang_crs_ln*)	**3.8 (3.5–4.0)**	**4.2 (4.0–4.4)**
ang_crs_ln (*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*+*ang_crs*)	**9.7 (9.3–10.1)**	**11.4 (11.1–12.1)**
ang_crs_ln (*ang_tri_ln*+*crs_ncrs*+*blt_srp*+*blt_srp_ln*)	**10.9 (10.6–11.1)**	14.1 (13.6–14.4)
blt_srp (*ang_crs*+*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*)	**1.1 (1.0–1.2)**	**1.3 (1.1–1.5)**
blt_srp (*ang_crs_ln*+*ang_tri_ln*+*crs_ncrs*+*ang_crs*)	**1.2 (1.0–1.4)**	**1.8 (1.7–2.0)**
blt_srp_ln (*ang_crs_ln*+*ang_tri_ln*+*crs_ncrs*+*ang_crs*)	**5.9 (5.6–6.2)**	10.2 (10.0–10.5)
blt_srp_ln (*ang_crs*+*ang_tri_ln*+*crs_ncrs*+*ang_crs_ln*)	**5.8 (5.6–6.0)**	9.9 (9.3–10.1)

In Table 6, we list the results obtained using five base models. In this case, for each task, all of the networks trained on each of the other tasks were used. Both architectures BA-0-50-50 and BA-0-0-50 outperformed the architecture NN-70-50-30, which had the same number of parameters, as well as the architecture NN-200-100-50. It is interesting to note here that the performance on the task *ang_crs_ln* (11.3) is near to the best of the performance obtained using 4 base models. This probably suggest that the architecture has been able to select among the base models the ones more useful for the new task.

Table 6: Adding blocks to five base models.

condition	0-50-50	0-0-50
ang_crs (all models used except *ang_crs*)	**3.6 (3.3–3.8)**	**4.1 (3.6–4.3)**
ang_crs_ln (all models used except *ang_crs_ln*)	**9.5 (9.3–9.9)**	**11.3 (11.0–11.8)**
blt_srp (all models used except *blt_srp*)	**1.0 (0.9–1.3)**	**1.2 (1.0–1.3)**
blt_srp_ln (all models used except *blt_srp_ln*)	**4.7 (4.4–4.9)**	**6.5 (6.2–7.0)**
crs_ncrs (all models used except *crs_ncrs*)	**1.1 (1.0–1.1)**	**1.1 (1.0–1.2)**
ang_tri_ln (all models used except *ang_tri_ln*)	**4.9 (4.6–5.0)**	7.6 (7.4–7.8)

On a smaller dataset

As mentioned in the introduction, we also wished to verify that the proposed architecture could be trained with a smaller dataset than a network trained from scratch. In this paragraph we present

the results obtained by training the same architectures presented in Tables 5 and 6 with a dataset of 200,000 examples. The architectures in Tables 5 and 6 were trained on a dataset of 350,000 examples. The percentages of misclassified examples, presented in Tables 7 8, showed that even with a much smaller training dataset, with four or five base models the architecture BA-0-50-50 outperformed the network trained from scratch, NN-200-100-50. The other architecture, BA-0-0-50, had worse results then NN-200-100-50, especially for the more complex tasks, namely *blt_srp_ln* and *ang_tri_ln*.

Table 7: Adding blocks to four base models. Dataset of 200.000 examples.

condition	0-50-50	0-0-50
ang_crs (*ang_tri_ln*+*crs_ncrs*+*blt_srp*+*blt_srp_ln*)	**5.0 (4.8–5.2)**	5.8 (5.4 – 6.3)
ang_crs (*ang_tri_ln*+*ang_crs_ln*+*crs_ncrs*+*blt_srp_ln*)	**4.3 (4.0–4.5)**	**4.6 (4.0 – 5.0)**
ang_crs (*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*+ *ang_crs_ln*)	**4.3 (4.1–4.8)**	**4.7 (4.3–5.5)**
ang_crs_ln (*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*+*ang_crs*)	**10.7 (10.4–11.3)**	**12.0 (11.5–12.4)**
ang_crs_ln (*ang_tri_ln*+*crs_ncrs*+*blt_srp*+*blt_srp_ln*)	**12.4 (12.0–12.6)**	15.1 (14.6–15.5)
blt_srp (*ang_crs*+*ang_tri_ln*+*crs_ncrs*+*blt_srp_ln*)	**1.2 (1.1–1.4)**	**1.4 (1.3–1.5)**
blt_srp (*ang_crs_ln*+*ang_tri_ln*+*crs_ncrs*+*ang_crs*)	**1.8 (1.7–2.0)**	2.1 (1.7 – 2.4)
blt_srp_ln (*ang_crs_ln*+*ang_tri_ln*+*crs_ncrs*+*ang_crs*)	**6.4 (6.3–6.6)**	9.7(9.2–10.6)
blt_srp_ln (*ang_crs*+*ang_tri_ln*+*crs_ncrs*+*ang_crs_ln*)	**6.5 (6.3–6.8)**	9.8(9.4–10.3)

Table 8: Adding blocks to five base models. Dataset of 200.000 examples.

condition	0-50-50	0-0-50
ang_crs (all models used except *ang_crs*)	**4.3 (4.0–4.7)**	**4.4 (3.9–4.7)**
ang_crs_ln (all models used except *ang_crs_ln*)	**10.6 (10.4 – 10.8)**	**11.7 (11.2 – 12.1)**
blt_srp (all models used except *blt_srp*)	**1.2 (0.9 – 1.9)**	**1.4 (1.1 – 1.8)**
blt_srp_ln (all models used except *blt_srp_ln*)	**5.6 (5.2 – 5.9)**	7.2 (6.8 – 8.0)
crs_ncrs (all models used except *crs_ncrs*)	**1.2 (1.0–1.3)**	**1.2 (1.0–1.3)**
ang_tri_ln (all models used except *ang_tri_ln*)	**5.8 (5.7–6.0)**	8.6 (8.3–9.0)

4 Conclusions

DNNs are known to have some abstraction abilities [15]. These abstractions, however, are very limited compared to those typical of humans, where different concepts recall each other through a vast network of relationships. One possible aspect of the creation of such powerful abstractions could be that humans are continuously engaged in different tasks which are each solved with a limited set of resources that must be shaped in order to optimize performance on the largest number of tasks.

In this paper we built a DNN by training several DNNs on different tasks and then merging them with a group of added neurons. The resulting architecture was capable of performing several tasks. Moreover, it was very efficiently trained, using a dataset smaller than that used for the original DNNs and adding only a small number of new neurons. Our results are consistent with previous experimental observations that training deep architectures is easier when cues to the function that intermediate levels should compute are provided ([2, 17]) and when training is not performed on the whole network at the same time [12].

Our study can be considered a first step toward the construction of DNN architectures which are able to use areas trained on previous tasks when learning a new task. This type of procedure should focus on training areas of the network that have not previously been used, while only slightly modifying areas already trained on previous tasks. Such an architecture would maintain the positive characteristics described for our architectures, and would be capable of sequential learning.

Acknowledgments

This work was funded by the European Research Council (FP 7 Program) ERC Advanced Grant "FEEL" to KO'R

References

[1] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[2] Yoshua Bengio. Evolving culture versus local minima. In *Growing Adaptive Machines*, pages 109–138. Springer, 2014.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[5] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[6] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[7] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoffrey Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at microsoft. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8604–8608. IEEE, 2013.

[8] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[10] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[11] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[12] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.

[13] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. *Proc. NAACL, May 2015*.

[14] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *arXiv preprint arXiv:1412.1897*, 2014.

[16] Alexander V Terekhov, Guglielmo Montone, and J Kevin ORegan. Knowledge transfer in deep block-modular neural networks. In *Biomimetic and Biohybrid Systems*, pages 268–279. Springer, 2015.

[17] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

Symbol Grounding in Multimodal Sequences using Recurrent Neural Networks

Federico Raue
University of Kaiserslautern, Germany
DFKI, Germany
federico.raue@dfki.de

Wonmin Byeon
University of Kaiserslautern, Germany
DFKI, Germany
wonmin.byeon@dfki.de

Thomas M. Breuel
University of Kaiserslautern, Germany
tmb@cs.uni-kl.de

Marcus Liwicki
University of Kaiserslautern, Germany
liwicki@cs.uni-kl.de

Abstract

The problem of how infants learn to associate visual inputs, speech, and internal symbolic representation has long been of interest in Psychology, Neuroscience, and Artificial Intelligence. A priori, both visual inputs and auditory inputs are complex analog signals with a large amount of noise and context, and lacking of any segmentation information. In this paper, we address a simple form of this problem: the association of one visual input and one auditory input with each other. We show that the presented model learns both segmentation, recognition and symbolic representation under two simple assumptions: (1) that a symbolic representation exists, and (2) that two different inputs represent the same symbolic structure. Our approach uses two Long Short-Term Memory (LSTM) networks for multimodal sequence learning and recovers the internal symbolic space using an EM-style algorithm. We compared our model against LSTM in three different multimodal datasets: digit, letter and word recognition. The performance of our model reached similar results to LSTM.

1 Introduction

Our brain has an important skill that is to assign semantic concepts to their sensory input signals, such as, visual, auditory. In other words, the sensory inputs can be considered as meaningless physical information and the semantic concepts are linked to their physical features. This scenario can be seen as *Symbol Grounding Problem (SGP)* [1].

Infants in their development ground the semantic concepts to their sensory inputs. For example, several cognitive researchers found a relation between the vocabulary acquisition (audio) and object recognition (visual) [2, 3]. Recently, Asano *et al.* [4] recorded the infant brain activity using three Electroencephalogram (EEG) measures. They found that infants are sensitive to the correspondence between visual stimulus and their sound-symbolic match or mismatch. Furthermore, the lack of one of these components affects the learning behavior, i.e., deafness or blindness [5, 6].

Several models have been proposed for grounding concepts in multimodal scenarios. Yu and Ballard [7] developed a multimodal learning algorithm that maximize the probabilities between spoken words and the visual perception using EM approach. Nakamura *et al.* [8] developed a different approach based on a latent Dirichlet allocation (LDA) for multimodal concepts. They used not only visual and audio information but also haptic information for grounding the concept.

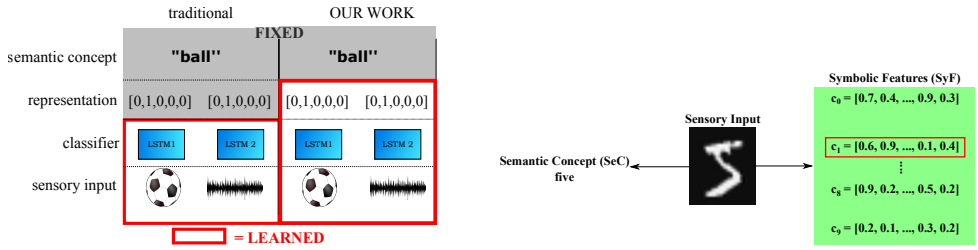


Figure 1: Examples of several components in this work. Figure 1a shows the relation between the traditional approach and our approach for multimodal association. It can be seen that proposed scenario learns the representation of the semantic concept, whereas that relation is fixed in the traditional scenario (red box). Figure 1b illustrates the relation among a *semantic concept*, a visual sensory input and a set of *symbolic features*. In this scenario, there are ten possible options ($\boldsymbol{c_0}$, ..., $\boldsymbol{c_9}$) that can be assigned to the concept 'five' in order to be represented in the network. In this example, the semantic concept 'five' is represented by the symbolic feature $\boldsymbol{c_1}$.

Previous work has focused only on segmented inputs. However, recent results in Recurrent Neural Network, mainly Long Short-Term Memory (LSTM), has been successfully applied to scenarios where the input is unsegmented, e.g., OCR and speech recognition. In this paper, we are proposing an alternative solution that exploits those benefits. Furthermore, we address a simplified version of the multimodal symbol grounding: the association of one visual input and auditory input between each other. Our model uses two parallel LSTM networks that segments, classifies and finds the agreement between two multimodal signals of the same semantic sequence. For example, the visual signal is a text line with digit '2 4 5' and the audio signal is 'two four five'. We want to indicate that our model is trained with less information because the semantic concept and its representation is learned during training. Figure 1a shows the learned components in the traditional scenario and this work. In the traditional scenario, the relation between the semantic concepts and their representation is fixed, whereas that relation in our model is trainable. Moreover, we want to point out that LSTM outputs are used as symbolic features. Figure 1b shows the relation between a semantic concept ($\boldsymbol{SeC}$), a visual sensory input and a set of symbolic features ($\boldsymbol{SyF}$). This relation from now on is called *symbolic structure*. This work is based on Raue *et al.*[9]. In their work, the model was applied to a mono-modal parallel sequence case. In more detail, they learn the association between two text lines, i.e., only visual information. In this work, we explore the model in a more complex scenario where the training is applied on multimodal sequences. Thus, the alignment and the agreement between two modalities are not as smooth as the monomodal scenario.

This paper is organized as follows. Section 2 explains LSTM network as a background information. In Section 3, we describe our model that uses two parallel LSTMs in combination with an EM-based algorithm in order to learn segmentation, classification and symbolic representations. Section 4 explains our experimental setup. Section 5 reports the performance of our model; and, a comparison between our model and a single LSTM network.

2 Background: Long Short-Term Memory (LSTM) networks

LSTM was introduced to solve the vanishing gradient in recurrent neural networks [10, 11]. In more detail, the output of the network represents the class probability at each time step. The architecture has already been applied to learn unsegmented inputs using an extra layer called Connectionist Temporal Classification (CTC) for speech recognition [12] and OCR [13]. CTC adds an extra class (called *blank class (b)*) to the target sequence for learning the monotonic alignment between two sequences. In that case, the alignment is accomplished by learning to insert the blank class at appropriate positions. As a result, LSTM learns the classification and the segmentation. CTC was motivated by the forward-backward algorithm for training Hidden Markov Models (HMM) [14]. In addition, a decoding mechanism extracted the labeled classes from LSTM outputs. Please review the original paper for more details about LSTM and CTC [12].

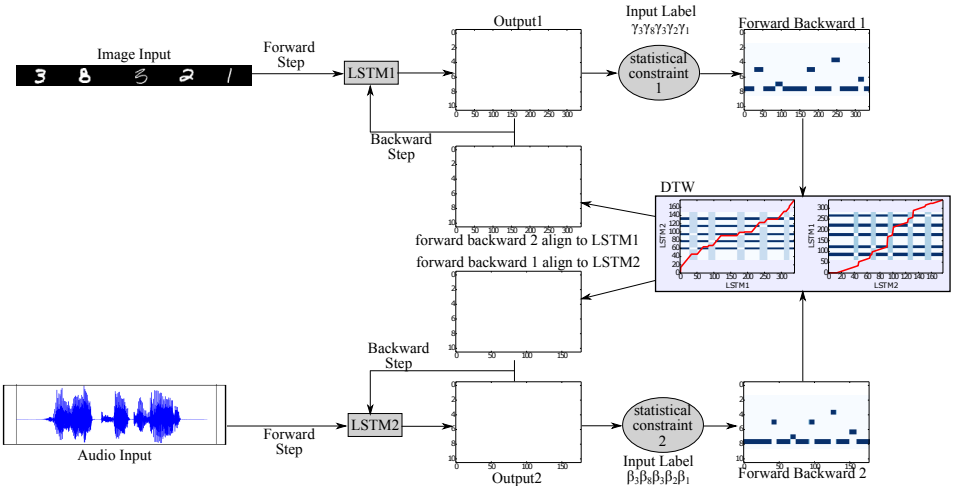


Figure 2: Overview of symbolic association framework. The statistical constraints (γ and β) guide each LSTM to the internal representation (symbolic feature) for each semantic concept. Also, the monotonic behavior is exploited by DTW. In this manner, LSTM1 output is used as target for LSTM2, and vice versa.

3 Multimodal Symbolic Association

As we mentioned in Section 1, the goal of our model is to learn the agreement in a multimodal symbolic sequences. In this case, the term 'agreement' is referred to the output classification of both LSTMs are the same (regardless of the modalities). In other words, both LSTMs learn the segmentation, the classification and the symbolic structure in a simplified multimodal scenario.

More formally, we define the multimodal symbolic association problem in the following manner. A multimodal dataset is defined by $\mathcal{M} = \{(\boldsymbol{x}_{a,t_1}; \boldsymbol{x}_{v,t_2}; \boldsymbol{s}_{1,\dots,n}) | \boldsymbol{x}_{a,t_1} \in \boldsymbol{X}_a, \boldsymbol{x}_{v,t_2} \in \boldsymbol{X}_v, \boldsymbol{s}_{1,\dots,n} \in \boldsymbol{SeC}\}$. $\boldsymbol{X}_a$ and $\boldsymbol{X}_v$ are sets of audio and visual sequences, respectively. The length of both sequences can be different. $\boldsymbol{s}_{1,\dots,n}$ define the semantic concept sequence of size n that is represented by two modalities ($\boldsymbol{X}_a$ and $\boldsymbol{X}_v$). As mentioned, the goal is to learn the same symbolic structure that is represented by both modalities. In more detail, each semantic concept is grounded by a similar symbolic feature in both modalities. Also, all semantic concepts are represented by different symbolic features.

In this work, we are proposing a framework that combines two LSTMs for learning a unified symbolic association between two modalities. The intuition behind this idea is to convert from a multimodal input feature space to a common output class space, where two modalities can be associated. Thus, LSTM outputs have the same size. Also, we introduce an EM-training rule based on two constraints: (1) a symbolic representation exists, and (2) two different inputs represent the same symbolic structure. Figure 2 shows a general view of our framework.

In more detail, our model works in the following manner. First, the sequences $\boldsymbol{x}_{a,t_1}$ and $\boldsymbol{x}_{v,t_2}$ are passed to each LSTM ($LSTM_1$, $LSTM_2$). Then, LSTM outputs ($\boldsymbol{z}_{a,t_1}, \boldsymbol{z}_{v,t_2}$) and the semantic concept sequence ($s_1, \dots, s_n$) are feeding to the statistical constraint (γ and β). We want to indicate the LSTM output are used as symbolic feature (*SyF*). This component selects the most likely relation between semantic concepts and the symbolic features (Section 3.1). As a result, this relation provides information in order to apply the forward-backward algorithm for training (cf. Section 2). Previous steps are independently applied to each LSTM. As we mentioned before, the goal of our model is to learn a unified symbolic structure. With this in mind, the next step in our framework is to align both outputs from the forward-backward algorithm. Our model exploits the monotonic behavior and both sequences are aligned by Dynamic Time Warping (Section 3.2). The aligned output of one LSTM is used as a target of the other LSTM, and vice versa.

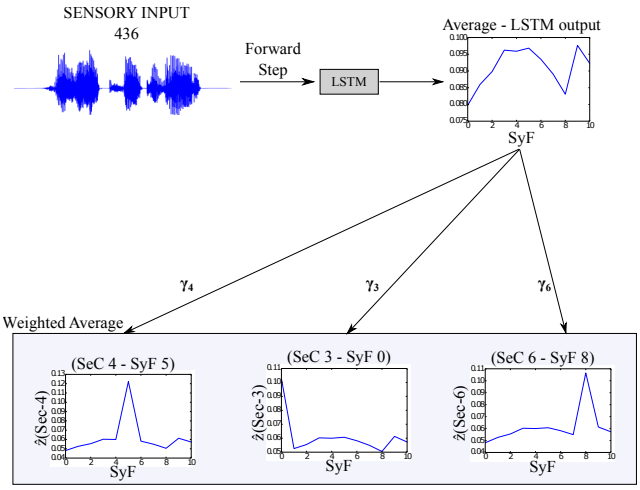


Figure 3: Example of the statistical constraint. The semantic weights $(\boldsymbol{\gamma_4}, \boldsymbol{\gamma_3}, \boldsymbol{\gamma_6})$ modify the average output of LSTM. It can be seen that only one symbolic feature spikes among all for each semantic concept.

3.1 Statistical Constraint

The goal of this component is to learn the structure between the *semantic concepts (SeC)* and the *symbolic feature (SyF)* that are represented by the output of LSTM networks. Our proposed training rule is based on EM algorithm [15]. With this in mind, we define a set of weighted concepts $(\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_c)$ where $c \in SeC$ and each $\boldsymbol{\gamma}_c$ is represented by a vector $\boldsymbol{\gamma}_c = [\gamma_{c,0}, \ldots, \gamma_{c,k}]$ where k is the size of the LSTM output. As a result, the relation can be retrieved by a *winner-take-all* rule. Figure 3 shows an example of the statistical constraint.

The *E-Step* finds the structure between *SeC* and *SyF*. First, we construct the matrix $\hat{\boldsymbol{Z}}$ which is defined by

$$\hat{\boldsymbol{Z}} = \left[\hat{\boldsymbol{z}}(\boldsymbol{1}), \ldots, \hat{\boldsymbol{z}}(\boldsymbol{c})\right]; \;\; c \in \boldsymbol{SeC} \tag{1}$$

$$\hat{\boldsymbol{z}}(\boldsymbol{c}) = \frac{1}{T}\sum_{t=1}^{T}(\boldsymbol{z_t})^{\boldsymbol{\gamma(c)}}; \;\; c \in \boldsymbol{SeC} \tag{2}$$

where $\boldsymbol{z_t}$ is a column vector that represents LSTM output[1] at time t, $t \in [1, \ldots, T]$ is the timesteps. The column vector $\hat{\boldsymbol{z}}(\boldsymbol{c})$ is the weighted average of LSTM output. Next, we convert from matrix $\hat{\boldsymbol{Z}}$ to matrix $\boldsymbol{Z}^*$. A row-column elimination is applied in order to find the symbolic structure for the training. The maximum element (i, j) in $\hat{\boldsymbol{Z}}$ is set to 1 and the elements at the same row i and column j are set to 0 except the element (i, j). This procedure is repeated $|SeC|$ times. As a result, only one symbolic representation is selected for each semantic concept.

The *M-Step* updates the set of weighted concepts given the current symbolic structure ($\hat{\boldsymbol{Z}}$). In this case, we are assuming a uniform distribution of semantic concepts. We define the following cost function

$$\boldsymbol{cost}(\boldsymbol{c}) = \left(\hat{\boldsymbol{z}}(\boldsymbol{c}) - \frac{1}{|\boldsymbol{SeC}|}\boldsymbol{z}^*(\boldsymbol{c})\right)^2; \;\; c \in \boldsymbol{SeC} \tag{3}$$

where $\boldsymbol{z}^*(\boldsymbol{c})$ is a column-vector of matrix $\boldsymbol{z}^*$. The update of $\boldsymbol{\gamma(c)}$ is accomplished by applying gradient descent

$$\boldsymbol{\gamma(c)} = \boldsymbol{\gamma(c)} - \alpha * \boldsymbol{\nabla_\gamma cost(c)}; \;\; c \in \boldsymbol{SeC} \tag{4}$$

where α is the learning rate and $\boldsymbol{\nabla cost(c)}$ is the derivatives of the cost function with respect to $\boldsymbol{\gamma(c)}$.

[1]For explanation purposes, the index that represents the modality is dropped, i.e, $\boldsymbol{z}_t \equiv \boldsymbol{z}_{a,t_1}$

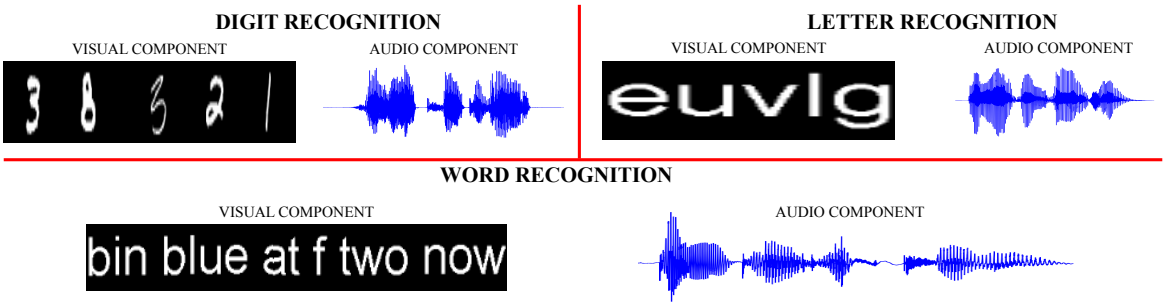


Figure 4: Several examples of the generated multimodal datasets.

After the symbolic structure is learned, the semantic concept is grounded to the symbolic feature, and vice versa. As a result, the semantic concept can be retrieved from the symbolic feature by the maximum element of the following equation:

$$c^* = arg\ max_c\ z_{k^*}^{\gamma_{c,k^*}} \tag{5}$$

where k^* is the class decoded from LSTM outputs[2], $\gamma_{(c,k^*)}$ is the value at position k^* in the column vector $\boldsymbol{\gamma(c)}$.

3.2 Dynamic Time Warping (DTW)

The goal of the second component of our modified learning rule is to align the output of both networks. In other words, the alignment is a mapping function between both networks. Thus, the output of one network can be converted as an approximated output to the other network. This mapping is important for calculating the $error$ for updating the weights in the backpropagation step. We apply Dynamic Time Warping (DTW) [16] because of its monotonic behavior scenario. For this purpose, a distance matrix is calculated between each timestep of the forward-backward algorithm from both networks. Equation 6 shows the standard constrains of the path in DTW.

$$DTW[i,j] = dist[i,j] + min \begin{cases} DTW[i-1,j-1] \\ DTW[i-1,j] \\ DTW[i,j-1] \end{cases} \tag{6}$$

where $dist[i,j]$ is the distance between the timestep i of *LSTM1* and the timestep j of *LSTM2*.

4 Experimental Design

4.1 Datasets

We generated three multimodal datasets for the following sequence classification scenarios: handwritten digit recognition, printed letter recognition, and word recognition. Each dataset has two components: visual and audio. The visual component is a text line (bitmap) and the audio component is a speech (wav file). Both components represent the same semantic sequence. For example, the semantic sequence '3 8 3 2 1' is represented by a bitmap with those digits and an audio file with 'three eight three two one'. Figure 4 shows several examples of the multimodal datasets.

Digit Recognition The first dataset was generated based on a combination between MNIST [17] and Festival Toolkit [18]. This dataset has ten semantic concepts. Sequences were randomly generated between 3 and 8 digits. The visual component was generated using MNIST dataset. MNIST has already a training set and a testing set. Thus, we kept the same division for creating our raining set and testing set. Each selected digit was attached before and after a random blank background (between 3 and 10 columns). All the selected digits were horizontally stacked. For the audio component, the audio file was generated given the sequences obtained from the visual component and

[2]cf. Section 2

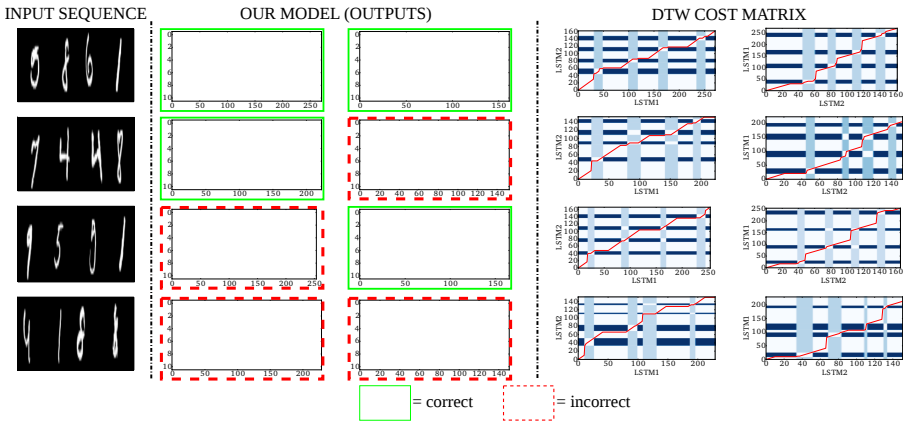


Figure 5: Several examples of the DTW cost matrix. The audio component of the sequences was omitted. The cost matrix (right) shows the path (red line) pass through nine regions. These regions represent the blank class and the semantic concepts.

selected between four artificial voices. As a result, the training set has 50,000 sequences and testing set has 15,000 sequences.

Letter Recognition The second dataset was generated following a similar procedure as the previous dataset. This dataset has 27 semantic concepts. We generated text lines of letters as the visual component. The length was randomly selected between 3 and 8 lower characters. The audio component was generated similar to the first dataset using Festival Toolkit. In contrast to MNIST, this dataset does not have an explicit division for the training set and the testing set. Thus, we decided to generate a slightly bigger dataset of 60,000 sequences.

Word Recognition The last multimodal dataset is generated based on the audio of the GRID audio-visual sentence corpus [19]. This dataset has 52 semantic concepts. The audio has a fixed sequence length of eight semantic concepts. Also, the audio component is composed by 34 talkers, 18 were males and 16 were females. We generated the text lines of each sequence semantic sequence. The size of this dataset is 34,000 sequences.

4.2 Input Features and LSTM Setup

The visual component was used raw-pixel values between 0.0 and 1.0. The audio component was converted to Mel-Frequency Cepstral Coefficient (MFCC) using HTK toolkit[3]. The following parameters were selected for extracting MFCC: a Fourier-transform-based filter-bank with 40 coefficients (plus energy) distributed on a mel-scale, including their first and second temporal derivatives. As a result, the size of the vector was 123. Also, the audio component was normalized to zero mean and unit variance. The training set of the audio component was normalized to zero mean and unit variance.

As a baseline, each component was evaluated using LSTM with CTC layer in order to test the performance of our model. The following parameters were selected for the visual component. The memory size is 20 for the first two datasets and 40 for the last dataset, the learning rate of the network is 1e-5 and the momentum is 0.9. The parameters for the audio component are similar but the memory size is 100. The statistical constraint were initialized with 1.0 and the learning rate was set to 0.01 for both networks.

5 Results and Discussion

In this paper, the performance of the presented model and the standard LSTM were compared. We want to point out that our goal is not to outperform the standard LSTM, but to know if the

[3]http://htk.eng.cam.ac.uk

Table 1: Label Error Rate (%) between the standard LSTM and our model. We want to point out that our goal is not to outperform the standard LSTM.

METHOD		DIGITS	LETTERS	WORDS
STANDARD LSTM	VISUAL	3.42 ± 0.84	0.09 ± 0.05	0.45 ± 0.68
	AUDIO	0.08 ± 0.06	1.06 ± 0.14	3.68 ± 0.27
OUR MODEL	VISUAL	2.69 ± 0.55	0.35 ± 0.33	0.51 ± 0.84
	AUDIO	0.15 ± 0.08	1.24 ± 0.50	3.77 ± 0.40

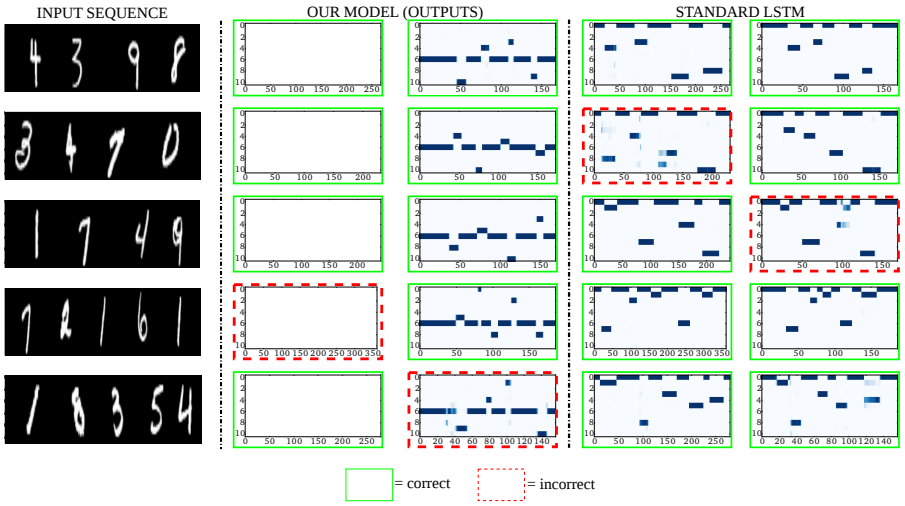


Figure 6: Symbolic Structure between our model and LSTM. The audio component was omitted. Both networks converge to the structure *(SyF, SeC): (0, 2), (1, 5), (2, 6), (3, 9), (4, 3), (5, 7), (6, blank-class), (7, 0), (8, 1), (9, 8), (10, 4)*. It is noted that both models shows similar behavior related to the symbolic structure. LSTM uses a pre-defined structure before the training, whereas the presented model learns the structure during training

performance of our model was in good range. Note that our model has less information than LSTM networks. We randomly selected 10,000 sequences and 3,000 sequences as a training set and testing set (respectively). This random selection was repeated ten times. In the word recognition dataset, we randomly selected 50% male voices and 50% female voices for each training and testing set. We are reporting *Label Error Rate* (LER), which is defined by

$$LER = \frac{1}{|Z|} \sum_{(x,y) \in Z} \frac{ED(x,y)}{|y|} \tag{7}$$

where ED is the edit distance between the classification of the network x and the correct output classification y and Z is the size of the dataset. Table 1 shows that our model reaches a similar performance to the standard LSTM. In more detail, Fig. 5 shows several examples of the output classification of our model. The first row shows a correct classification of both LSTMs. In this case, both structures of the semantic concepts and the symbolic features are the same. It can be seen that the semantic concept '5861' is represented by the symbolic feature '2867' (dark blue in column 2-3) in both LSTMs. In addition, DTW cost matrix shows an example of the alignment between the both LSTMs. We mentioned in Section 2 that CTC layer adds an extra class. Consequently, our sequence example is converted to 'b5b8b6b1b' (nine elements). The DTW cost matrix shows nine regions that the DTW path (red line) crossed. In other words, the alignment happened in the same semantic concept. Furthermore, the alignment still follows the same behavior, even if one or both output classification are wrong.

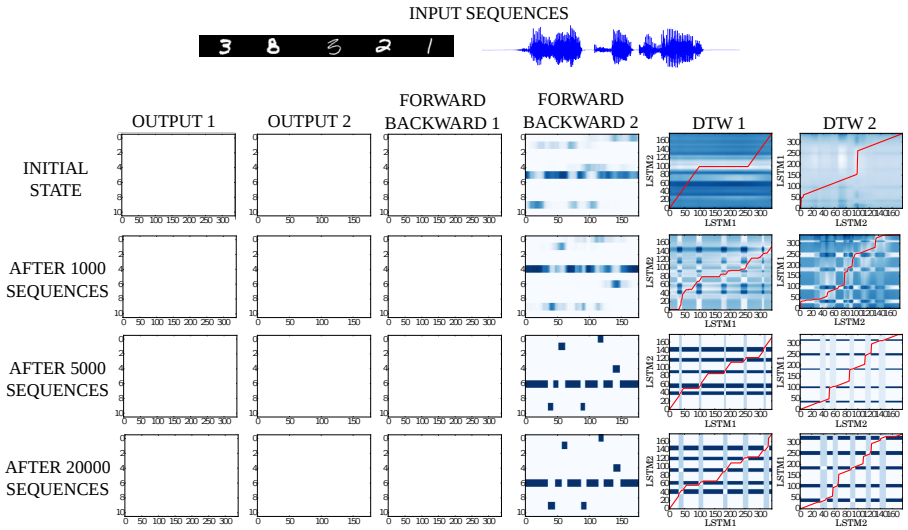


Figure 7: Steps of the training rule. In the beginning, the output of the networks is sparse and they point to align first the blank-class (first three rows). The forward_backward algorithm shows high values (dark blue) where the blank-class appears. After the blank-class is aligned, the remaining symbolic features are slowly converging to the same representation. The last row shows that both outputs classify the multimodal sequence with similar symbolic features. The DTW cost matrix shows the alignment (red line) between the symbolic features. The alignment has two cases: blank-class to blank-class and semantic concepts to semantic concepts.

Figure 6 shows examples of the symbolic structure. It can be noted that the presented model has a similar behavior as the standard LSTM. For example, our model also learns the blank class for segmenting the semantic concepts. The difference is mainly in the symbolic features for each semantic concept. It is observed that the standard LSTMs used a pre-defined structure between the semantic concepts and the symbolic features. For example, the semantic concept '1' is represented by the symbolic feature '1', the same happened with the rest of semantic concepts. In the other hand, our model learns the structure for each LSTM and both LSTMs converge to a common symbolic structure.

Figure 7 shows the behavior of our model during training. In the beginning, the output of the networks has sparse values and the DTW cost matrices do not have clear regions as in Figure 6. After 10,000 sequences, both networks align first to blank class. DTW cost matrices start showing some initial regions of alignment. After 50,000 sequences, the blank class changes because the structure of the semantic concepts and the symbolic features are not stable. After 20,000 sequences, both networks converge to a common a structure and the DTW cost matrices show a clear DTW path similar to Figure 6.

6 Conclusions

This paper has demonstrated that learning symbolic representations of unsegmented sensory inputs is possible with a minimum of assumptions, namely that symbolic representations exist, that two inputs represent the same symbolic content and that classes follow prior distribution. One limitation of our model is the constraint to one-dimension. However, there are many applications in this context, e.g., combining eye tracking system with audio. We will validate our findings with more realistic scenarios, i.e., unknown semantic concepts, aligning a two-dimensional image and a one-dimensional speech, handling missing semantic concepts in one component or both components of the sequence. Finally, it can be seen that this scenario is simple but assigning semantic meanings to symbols is important for language development and remains as an open problem [20, 21, 22].

References

[1] S. Harnad, "The symbol grounding problem," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1, pp. 335–346, 1990.

[2] M. T. Balaban and S. R. Waxman, "Do words facilitate object categorization in 9-month-old infants?" *Journal of experimental child psychology*, vol. 64, no. 1, pp. 3–26, Jan. 1997.

[3] L. Gershkoff-Stowe and L. B. Smith, "Shape and the first hundred nouns." *Child development*, vol. 75, no. 4, pp. 1098–114, 2004.

[4] M. Asano, M. Imai, S. Kita, K. Kitajo, H. Okada, and G. Thierry, "Sound symbolism scaffolds language development in preverbal infants," *cortex*, vol. 63, pp. 196–205, 2015.

[5] E. S. Andersen, A. Dunlea, and L. Kekelis, "The impact of input: language acquisition in the visually impaired," *First Language*, vol. 13, no. 37, pp. 23–49, Jan. 1993.

[6] P. E. Spencer, "Looking without listening: is audition a prerequisite for normal development of visual attention during infancy?" *Journal of deaf studies and deaf education*, vol. 5, no. 4, pp. 291–302, Jan. 2000.

[7] C. Yu and D. H. Ballard, "A multimodal learning interface for grounding spoken language in sensory perceptions," *ACM Transactions on Applied Perception (TAP)*, vol. 1, no. 1, pp. 57–80, 2004.

[8] T. Nakamura, T. Araki, T. Nagai, and N. Iwahashi, "Grounding of word meanings in latent dirichlet allocation-based multimodal concepts," *Advanced Robotics*, vol. 25, no. 17, pp. 2189–2206, 2011.

[9] F. Raue, W. Byeon, T. Breuel, and M. Liwicki, "Parallel Sequence Classification using Recurrent Neural Networks and Alignment," in *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*.

[10] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735—-1780, 1997.

[11] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, Apr. 1998.

[12] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification," in *Proceedings of the 23rd international conference on Machine learning - ICML '06*. New York, New York, USA: ACM Press, 2006, pp. 369–376.

[13] T. Breuel, A. Ul-Hasan, M. Al-Azawi, and F. Shafait, "High-performance ocr for printed english and fraktur using lstm networks," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, Aug 2013, pp. 683–687.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[15] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society.*, vol. 39, no. 1, pp. 1–38, 1977.

[16] D. J. Berndt and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," pp. 359–370, 1994.

[17] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits."

[18] P. Taylor, A. W. Black, and R. Caley, "The architecture of the festival speech synthesis system," 1998.

[19] M. Cooke, J. Barker, S. Cunningham, and X. Shao, "An audio-visual corpus for speech perception and automatic speech recognition," *The Journal of the Acoustical Society of America*, vol. 120, no. 5, pp. 2421–2424, 2006.

[20] C. J. Needham, P. E. Santos, D. R. Magee, V. Devin, D. C. Hogg, and A. G. Cohn, "Protocols from perceptual observations," *Artificial Intelligence*, vol. 167, no. 1, pp. 103–136, 2005.

[21] L. Steels, "The symbol grounding problem has been solved, so whats next ?" *Symbols, Embodiment and Meaning. Oxford University Press, Oxford, UK*, no. 2005, pp. 223–244, 2008.

[22] S. Coradeschi, A. Loutfi, and B. Wrede, "A short review of symbol grounding in robotic and intelligent systems," *KI-Künstliche Intelligenz*, vol. 27, no. 2, pp. 129–136, 2013.

Extracting Interpretable Models from Matrix Factorization Models

Ivan Sanchez Carmona
Department of Computer Science
University College London
i.sanchezcarmona@cs.ucl.ac.uk

Sebastian Riedel
Department of Computer Science
University College London
s.riedel@cs.ucl.ac.uk

Abstract

Matrix factorization models have been successfully used in many real-world tasks, such as knowledge base completion and recommendation systems. However, explaining the causes that elicit a particular prediction by a manual inspection of its latent representations is a difficult task. In this paper we try to overcome this problem by exploring descriptive model classes in their ability to faithfully approximate the behavior of a pre-trained matrix factorization model. Crucially, our choice of descriptive model will allow us to provide an interpretable structured proof for each prediction of the original model. We compare the descriptive models in these two scopes: Fidelity and interpretability. We find that Bayesian network trees, a class of models that has not been considered for this purpose before, capture the matrix factorization model faithfully while providing multi-step explanations of predictions.

1 Introduction

A highly desirable property of a predictive system is the ability to provide an interpretable explanation of a particular prediction. Matrix factorization (MF) models, due to their advantages (high accuracy and scalability), have been used in real-world tasks such as recommendation systems [1], and knowledge base population [2]. These models are a type of latent variable model (LVM) where a set of latent vectors is learned from a matrix of relational data. However, they are deficient in providing an explanation of the relations among observed variables that elicit a particular prediction. This lack of interpretability prevents them from both providing a support for the predictions and from analysing errors. These two properties would not only benefit NLP tasks, but also decision-support tasks, such as credit-risk evaluation [3], and medical diagnosis.

One solution to this problem is to focus on designing more interpretable models from the onset such as [4], but this often means trading off accuracy and scalability. Another option is to use visualization methods [5], where high-dimensional vectors are projected into a two-dimensional space. Such methods are useful for model inspection, but it is unclear how they can provide fine-grained multi-step explanations of a prediction in the way that, say, rule-based systems can. We believe this granularity is important to spot higher level problems in the model that go beyond "one vector is too close to another."

Here we investigate an alternative option: Learn an interpretable descriptive model that *mimics* the behavior of the original LVM model as close as possible. While we still use the original LVM to make predictions, we use the descriptive model to explain these predictions. Our goal then is to find descriptive models for (so-called) *donor* LVMs that *faithfully* capture the idiosyncrasy of the donor LVM (they respond similarly to same inputs), while their anatomy remains *interpretable*.

Our starting point is work by [6] on extracting decision trees (DT) from neural networks. We aim to apply a variant of their method to the MF model of [2] to get DTs that can be used to explain

extracted relations. We identify two core problems with this approach: *1*) we can capture some of the decision boundaries of the MF model, but we cannot capture its *ranking* behaviour which is most relevant for applying the model in practice; *2*) the [2] model is a full *joint* model over a *universal schema* of both surface form relations such as *X-a-professor-at-Y* and Freebase relations such as *employee(X,Y)*; representing each of these thousands of relations with an own decision tree both makes the model harder to interpret, and less faithful as far as model structure is concerned.

To overcome the problems above we propose to use a class of descriptive models that have not been used before for this purpose: tree-structured Bayesian networks (BN) [7]. Such BNs are interpretable, as their sparse connectivity explains the most important correlations. As joint probabilistic models they are able to capture both the joint nature of the donor MF model, and its ranking behaviour. But in contrast to more complex probabilistic models, they are much easier to train.

We *quantitatively* compare the learned BN trees to two baselines: A variant of [6] and a method to extract logical rules from the MF model. We compare *fidelity* of the descriptive models by measuring how well their rankings match the ranking of the original MF model on test data. Fidelity is a crucial metric for a descriptive model: It may be very interpretable, but without high fidelity it explains the wrong behaviour.

We *qualitatively* compare our approach to the baselines by contrasting the explanations of two wrong predictions of the MF model. In the BN tree these explanations are represented as subgraphs that span from observed nodes to the node predicted (Figure 2).

2 Background

We describe the predictive model (MF) and the descriptive models (BN tree, logic rules, decision trees).

2.1 Matrix Factorization

Objectives in a MF formulation are a) to learn a low-rank matrix $X_{m\times n} = UV'$ that reconstructs a given matrix of relational data $Y_{m\times n}$, where $U_{m\times k}$ and $V_{n\times k}$ are latent factors, and b) to predict confidence values for unobserved cells (matrix completion). We use a variant of the MF model in [2], where each row of the matrix Y corresponds to a pair of entities (e.g. $(London, England)$), and each column corresponds to either a surface form or a Freebase relation (e.g. $capitalOf$). The matrix X is learned by minimizing a logistic loss function over data. Each latent vector U_p is a distributed representation of a pair of entities. Similarly, each vector V'_r represents a relation. The latent vectors of pair p and relation r define the prediction $x_{r,p} \in [0, 1]$ of the fact that relation r holds for pair p as $x_{r,p} = sigmoid(U_p V'_r)$.

An alternative view of a matrix factorization model is as a one-hidden-layer neural network [8], where the activation function is linear in the hidden neurons and non-linear (sigmoid) in the output layer. The latent factors U, V correspond to the parameters of the network, i.e., the weights in the hidden and output layers. The model can be re-written as $\mathbf{y} = sigmoid(UV'\mathbf{x})$, where $\mathbf{x}$ is a one-hot input vector and $sigmoid$ is a component-wise function.

2.2 Logic Rules

We choose implication rules as a baseline due to their comprehensibility and to the maturity of rule extraction algorithms [9]. The rules learned are of the form $\forall x, y : A(x, y) \Rightarrow B(x, y)$, where *A, B* are predicates. All rules are range restricted (arguments occur in both body and head predicates). A rule, as the simplest building block in an explanation, accounts for the cause of predicting the realization of a fact. For example, the rule *professorAt*$(x, y) \Rightarrow$ *employeeAt*(x, y) would explain the cause of predicting the fact that x is an *employeeAt* y by observing the realization of the body of the rule as a true fact.

2.3 Decision Trees

A decision tree [10] is a hierarchical classification model. Each internal node corresponds to an input variable. The root node is the highest in the hierarchy (it splits first the input space). Leaf

nodes represent the class decision. We choose DTs as a baseline due to their main properties: a) interpretability (a path in the tree can be seen as a conjuctive logic rule), and b) suitability for estimating class probabilities (maximum likelihood estimation in each leaf).

2.4 Bayesian Networks

Decision trees can capture, to some extent, the probabilistic nature of the MF model, but they do not define a joint model across the complete relational schema. A set of logical rules can provide a fuller picture, and a notion of a proof that covers reasoning in various parts of the schema, but they cannot simulate the ranking behaviour of the MF model well.

We think that Bayesian networks can overcome both issues. A BN is the set of conditional independencies holding for a set of random variables in the form of a directed acyclic graph. A node corresponds to a random variable. A directed edge between two nodes, $x_i \rightarrow x_j$, represents a local influence which defines a conditional probability distribution (CPD): $p(x_j|x_i)$ (a parameter of the BN). A BN encodes a factorized probability distribution over the variables of the form $p(x_1, ..., x_n) = \prod_i p_i(x_i|\textit{Parents}(x_i))$, where *Parents*$(x_i)$ is the set of nodes that have an outgoing edge to x_i, (e.g. Figure 2).

3 Related Work

Learning descriptive models from complex donor models such as neural networks [11, 12, 13, 14] and support vector machines [15] has been previously considered. They extract both a set of logic rules and decision trees with the objective of behavior explanation. Nevertheless, the methods used are not suitable for the donor model we propose due to structural constraints. Moreover, to the best of our knowledge, this is the first time a joint probability model (Bayesian network) is compared with a classifier (decision trees) with the purpose of prediction explanation for a matrix factorization model.

4 Learning Interpretable Models

We treat the MF model as both a joint probability model and a classifier due to the nature of the descriptive models. To learn a BN tree and a set of logic rules we take each row of the MF model as a training instance and each colum as a variable. To learn decision trees we take input vectors from the training data used for training the original MF model and attach them with class labels from the MF predictions, i.e., the MF model re-labels input instances.

4.1 Extracting Logic Rules

We used mutual information as a support measure for rule acceptance. A rule is accepted if the strength of dependency between two predicates (columns in the MF model) surpasses a threshold m (we manually selected m=0.1 based on predictive performance). The directionality of each rule is then determined by its confidence, *if* $p(B|A) > p(A|B)$*then* $A \Rightarrow B$ *else* $B \Rightarrow A$. We opted for an information-theoretic based measure due to its property of assigning monotonically increasing values to more statistically dependent variables. We refrained from using an inductive logic programming approach due to its requirement of negative examples [16]. In order to obtain a test prediction we applied a transitive closure over a set of observed facts (i.e., we apply modus ponens).

4.2 Extracting Decision Trees

We used the R package *rpart* [17] in order to extract decision trees. Learning a decision tree corresponds to recursively adding nodes to its hierarchy, partitioning the input space. Node selection is performed by minimizing a cost function that measures the number of instances correctly classified after partitioning. In each leaf of the tree maximum likelihood estimation is performed in order to compute the probability of the class variable given the input variables in the path from the root node to the leaf. Once the tree has been learned, classifying a test instance corresponds to traversing the tree along the path that matches the observed facts until a leaf node is reached.

4.3 Extracting Bayesian Networks

Learning the structure of a BN is NP-hard [18]. This means that we can either resort to approximate algorithms, or restrict the model class. In preliminary experiments we found it extremely difficult to learn useful, interpretable BNs with approximate learning schemes primarily due to the scale of the data. Therefore, we constrained the structure of the BN to be a tree.

Learning the structure of such BNs reduces to finding a maximum spanning tree with respect to mutual information between variables (columns in the MF model). This problem can be solved optimally in $O(N^2)$ using Prim's algorithm, where N is the number of variables. Parameter estimation is performed by smoothed maximum likelihood estimation (we share parameters accross training instances).

Besides being easy to learn, a BN tree is also easy to interpret in that each variable can have at most one parent, and the complete model is described using only N edges. In addition, inference in BN trees is linear in N, which makes it easy to evaluate the fidelity of the model by computing its predictions on test data.

5 Experiments

We use the predictions from the MF model as training data for learning the descriptive models: We predict a confidence value for each cell in the MF model and threshold them at *t*=0.5 (we tried for different t in [0,1]). Training datasets are of size 4111 variables by 39864 instances. We obtained 4572 logic rules, a BN tree with 4111 nodes, and 19 decision trees (we chose 19 target variables) with average depth of 5 nodes. We evaluated the descriptive models on the test set of [2]. We performed two types of model analyses: Fidelity and interpretability.

Fidelity Figure 1a shows the 11-point average precision curves of the descriptive models with respect to the predictions of the MF donor model. We see that the logic rules learned are not a faithful representation of the predictive model: The ranked list of facts produced by the logic rules poorly matches the predictions of the MF model. This might be due to their deterministic nature: Since their response is in a binary domain they are not able to provide a confidence value in [0,1]. This makes it difficult to capture the ranking behavior of the MF model.

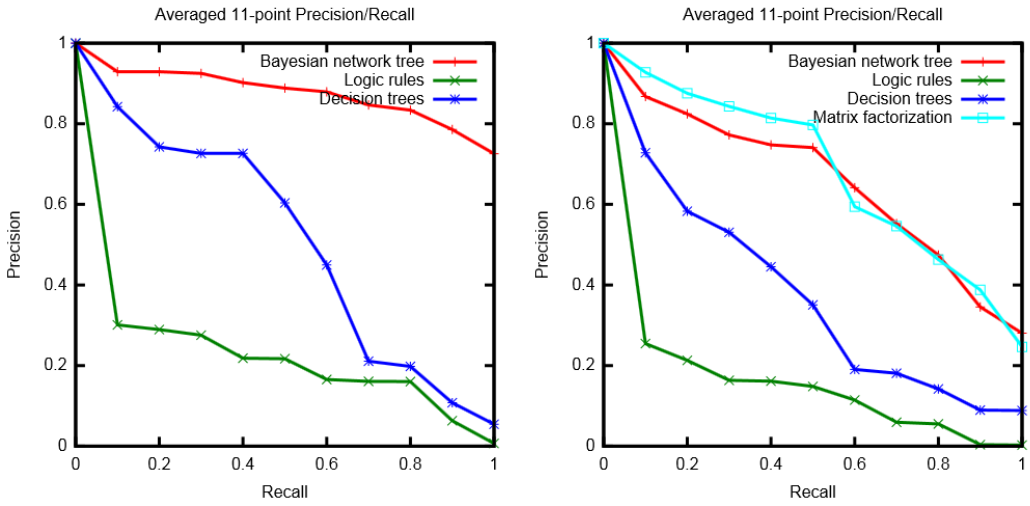
The decision trees provide more sensible confidence scores and hence rankings. This is reflected in better average precision curves. The BN model outperforms the other models substantially. We believe that this is a consequence of its probabilistic formulation, and the ability to capture the joint nature of the MF model better.

We also compute generalization performance of the descriptive models on a gold test sample and compare against generalization performance of the MF donor model. Figure 1b shows how low fidelity models (logic rules, decision trees) generalize poorly whereas high fidelity models (Bayesian network tree) have a generalization performance comparable to the original donor model. This confirms that the descriptive models approximates the MF model well.

Interpretability We show two examples of *explanations* for wrong predictions as produced by the descriptive models. Figure 2a shows possible causes for the MF model predicting the wrong fact $arenaStadium(PhiladelphiaEagles, Canton)$ as a true fact with confidence of 0.885: The observed node, $playAt$, influences the next nodes in the trajectory towards the target node $arenaStadium$. A clear error of the MF model is indicated by the connection in the BN: $beatAt \rightarrow arenaStadium$ (a team x beating another team at arena y does not necessarily entails that y is its home stadium).

Following the above example, the explanation from the decision tree learned for the relation $arenaStadium$ is the rule: *if* $playAt = 1$ *then* $p(arenaStadium = 1) = 1.0$. We think that the interpretability of the DT and the BN are quite comparable in this case. By contrast, the logical system does not even predict this fact as it did not recover the $playAt(x, y) \Rightarrow arenaStadium(x, y)$ implication.

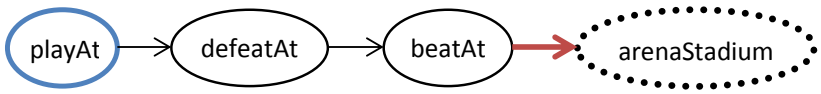
Figure 2b shows the explanation for the MF model wrongly predicting as a true fact: $reviewMovie(DanielKahneman, Nobel)$ (Kahneman is a Nobel price winner, not a reviewer for



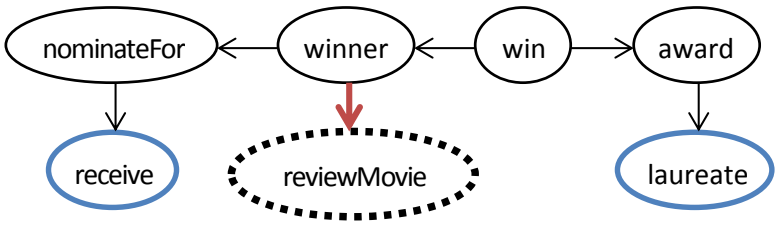
(a) Fidelity of the descriptive models to the matrix factorization model.

(b) Generalization of the descriptive models and the MF model on gold test data.

Figure 1: Fidelity and generalization of the descriptive models.



(a) variable predicted: $arenaStadium$



(b) variable predicted: $reviewMovie$

Figure 2: Two snippets from the BN tree: Influences from observed to predicted variables as an explanation of the causes that elicited a wrong prediction by the MF model. Bold arrows: wrong influences, bold nodes: observed variables.

a movie called $Nobel$). Given that $reviewMovie$ is not one of the 19 target variables no decision tree was learned for it, so no explanation from this model can be sought. On the other hand, in the set of logic rules none of the observed variables appeared, meaning that their statistical dependence with respect to other variables was low.

6 Conclusion

The problem of finding interpretable descriptive models for latent variable models has been discussed before. But we believe it is time to revisit it due to their recent successes and the increasing complexity of the tasks they address. In this work we looked at matrix factorization models for knowledge base population, a more complex task than the classification problems considered in ex-

Table 1: CPDs for Figure 2a.

A:parent→ B:child	$p(B=1\|A=1)$	$p(B=0\|A=0)$
$playAt \rightarrow defeatAt$	0.8651	0.9978
$defeatAt \rightarrow beatAt$	0.8435	0.9999
$beatAt \rightarrow arenaStadium$	0.8186	0.9989

isting literature. We proposed Bayesian network trees as a descriptive model and compared to two baselines: logic rules and decision trees. We found that BN trees provide a very competitive combination of fidelity and interpretability outperforming the baselines. We believe this model is prone to be used for analysing the latent variable model model by spotting wrong edges in its structure. In the future we like to investigate further representations, develop better ways to quantitatively evaluate the *utility* of a descriptive model, and apply the approach to other LVM models in NLP (such as as the matrix factorization formulation of the word2vec model).

Acknowledgments

The first author is sponsored by CONACYT. The second author is sponsored in part by the Paul Allen Foundation through an Allen Distinguished Investigator grant and in part by a Marie Curie Career Integration Fellowship. Thanks to Ivan Meza Ruiz for helpful discussions and to the anonymous reviewers who provided insightful comments.

References

[1] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, 2009.

[2] Sebastian Riedel, Limin Yao, Benjamin M. Marlin, and Andrew McCallum. Relation extraction with matrix factorization and universal schemas. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, June 2013.

[3] Bart Baesens, Rudy Setiono, Christophe Mues, and Jan Vanthienen. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management science*, 49(3):312–329, 2003.

[4] Benjamin Letham, Cynthia Rudin, Tyler H McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. 2013.

[5] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *The Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[6] Mark W Craven and Jude W Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems (NIPS-8)*, pages 24–30, 1996.

[7] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[8] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *Proc. 3rd ICLR*, 2015.

[9] Jiawei Han, Micheline Kamber, and Jian Pei. Data mining: Concepts and techniques, 3rd edition. 2011.

[10] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[11] Jens Lehmann, Sebastian Bader, and Pascal Hitzler. Extracting reduced logic programs from artificial neural networks. In *IJCAI Workshop on Neural-Symbolic Learning and Reasoning*, 2005.

[12] Sebastian Thrun. Extracting rules from artificial neural networks with distributed representations. *Advances in neural information processing systems*, pages 505–512, 1995.

[13] Hyeoncheol Kim, Tae-Sun Yoon, Yiying Zhang, Anupam Dikshit, and Su-Shing Chen. Predictability of rules in hiv-1 protease cleavage site analysis. In *Computational Science (ICCS)*, pages 830–837. 2006.

[14] AS d'Avila Garcez, Krysia Broda, and Dov M Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125(1):155–207, 2001.

[15] Nahla Barakat and Andrew P Bradley. Rule extraction from support vector machines: a review. *Neurocomputing*, 74(1):178–190, 2010.

[16] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422. International World Wide Web Conferences Steering Committee, 2013.

[17] Terry M Therneau, Beth Atkinson, Brian Ripley, et al. rpart: Recursive partitioning. *R package version*, 3:1–46, 2010.

[18] David Maxwell Chickering. Learning bayesian networks is np-complete. In *Learning from data*, pages 121–130. Springer, 1996.

Early Detection of Combustion Instability by Neural-Symbolic Analysis on Hi-Speed Video

Soumalya Sarkar
United Technology Research Center
East Hartford, CT 06118
sms388@gmail.com

Kin Gwn Lore
Mechanical Engineering, Iowa State University
Ames, IA 50011
kglore@iastate.edu

Soumik Sarkar
Mechanical Engineering, Iowa State University
Ames, IA 50011
soumiks@iastate.edu

Abstract

This paper proposes a neural-symbolic framework for analyzing a large volume of sequential hi-speed images of combustion flame for early detection of instability that is extremely critical for engine health monitoring and prognostics. The proposed hierarchical approach involves extracting low-dimensional semantic features from images using deep Convolutional Neural Networks (CNN) followed by capturing the temporal evolution of the extracted features using Symbolic Time Series Analysis (STSA). Furthermore, the semantic nature of the CNN features enables expert-guided data exploration that can lead to better understanding of the underlying physics. Extensive experimental data have been collected in a swirl-stabilized dump combustor at various operating conditions for validation.

1 Introduction

Recent advancements in deep learning shows that neural approaches are excellent at low-level feature extraction from raw data, automated learning and discriminative tasks. However, such models still may not be suited as much for logical reasoning, interpretation and domain knowledge incorporation. On the other hand, symbolic approaches can potentially alleviate such issues as they are shown to be effective in high-level reasoning and capturing sequence of actions. Therefore, a hybrid neural-symbolic [1] learning architecture has the potential to execute high-level reasoning tasks using the symbolic part based on the automated features extracted by the neural segment.

In this paper, we propose a neural-symbolic anomaly detection framework for the crucial physical process of combustion where a pure black box model is unacceptable in order to enable domain interpretation and better understanding of the underlying complex physics. Combustion instability, that reduces the efficiency and longevity of a gas-turbine engine, is considered a significant anomaly characterized by high-amplitude flame oscillations at discrete frequencies. These frequencies typically represent the natural acoustic modes of the combustor. Combustion instability arises from a positive coupling between the heat release rate oscillations and the pressure oscillations, provided this driving force is higher than the damping present in the system. Coherent structures are fluid mechanical structures associated with coherent phase of vorticity, high levels of vorticity among other definitions [2]. These structures, whose generation mechanisms vary system wise, cause large scale

velocity oscillations and overall flame shape oscillations by curling and stretching. These structures can be caused to shed/generated at the duct acoustic modes when the forcing (pressure) amplitudes are high. The interesting case of the natural shedding frequency of these structures, causing acoustic oscillations, has been observed by Chakravarthy et al. [3]. There is a lot of recent research interest on detection and correlation of these coherent structures to heat release rate and unsteady pressure. The popular methods resorted for detection of coherent structures are proper orthogonal decomposition (POD) [4] (similar to principal component analysis [5]) and dynamic mode decomposition (DMD) [6], which use tools from spectral theory to derive spatial coherent structure modes.

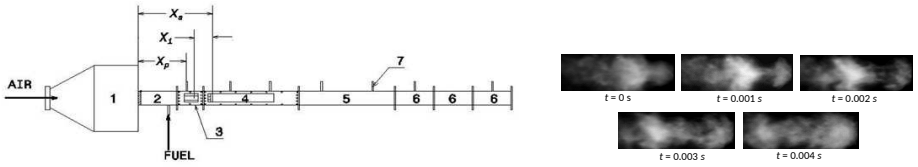
Although it is known that abundant presence of coherent structure indicates instability, it is quite difficult to visually characterize such structures. Furthermore, it becomes particularly difficult to identify precursors of instability due to the lack of physical understanding of the coherent structures. In this paper, we show that a deep CNN [7] based feature extractor can learn meaningful patterns from unstable flame images that can be argued as coherent structures. Then a symbolic model can capture the temporal dynamics of appearance of such patterns as a flame makes transition from stable to unstable states which results in an early detection of instability. Specifically, we use a recently reported Symbolic time series analysis (STSA) [8], a fast probabilistic graphical modeling approach. Among many other applications such as fault detection in gas turbine engines [9], STSA has been recently applied on pressure and chemiluminescence time series for early detection of Lean-blow out [10] and thermo-acoustic instability [11]. Note, a fully neural temporal model (e.g., deep RNN) would not be preferable as it is important to understand specific transitions among various coherent structures. Major contributions of the paper are delineated below.

- A novel data-driven framework, with CNN at lower layer and STSA at upper layer, is proposed for early detection of thermo-acoustic instability from hi-speed videos.
- In the above framework, the CNN layers extract meaningful shape-features to represent the coherent structures of varied sizes and orientations in the flame images. This phenomenon enables STSA at the temporal modeling layer to capture all the fast time scale precursors before attaining persistent instability.
- The proposed theory and the associated algorithms have been experimentally validated for transition data at multiple operating conditions in a swirl-stabilized combustor by characterizing the stable and unstable states of combustion.
- Training and testing of the proposed framework have been performed on different operating conditions (e.g., air flow rate, fuel flow rate, and air-fuel premixing level) of the combustion process to test the transferability of the approach. Performance of the proposed framework ('CNN+STSA') have been evaluated by comparison with that of a framework, where CNN is replaced by another extensively used dimensionality reduction tool, principal component analysis (PCA) [5].

2 Problem Setup and Experiments

To collect training data for learning the coherent structures, thermo-acoustic instability was induced in a laboratory-scale combustor with a 30 mm swirler (60 degree vane angles with geometric swirl number of 1.28). Figure 1 (a) shows the setup and a detail description can be found in [12]. In the combustor, 4 different instability conditions are induced: 3 seconds of hi-speed videos (i.e., 9000 frames) were captured at 45 lpm (liters per minute) FFR (fuel flow rate) and 900 lpm AFR (air flow rate) and at 28 lpm FFR and 600 lpm AFR for both levels of premixing. Figure 1 (b) presents sequences of images of dimension 100×237 pixels for unstable ($AFR = 900lpm$, $FFR = 45lpm$ and full premixing) state. The flame inlet is on the right side of each image and the flame flows downstream to the left. As the combustion is unstable, figure 1 (b) shows formation of mushroom-shaped vortex (coherent structure) at $t = 0, 0.001s$ and the shedding of that towards downstream from $t = 0.002s$ to $t = 0.004s$. For testing the proposed architecture, 5 transition videos of 7 seconds length were collected where stable combustion progressively becomes unstable via intermittancy phenomenon (fast switching between stability and instability as a precursor to persistent instability) by reducing FFR or increasing AFR. The transition protocols are as follows (all units are

lpm): (i) AFR = 500 and FFR = 40 to 28, (ii) AFR = 500 and FFR = 40 to 30, (iii) FFR = 40 and AFR = 500 to 600, (iv) AFR = 600 and FFR = 50 to 35, (v) FFR = 50 and AFR = 700 to 800. These data sets are mentioned as 500_{40to38}, 500_{40to30}, $40_{500to600}$, 600_{50to35} and $50_{700to800}$ respectively throughout the rest of this paper.



(a) (b)

Figure 1: (a) Schematics of the experimental apparatus. 1 - settling chamber, 2 - inlet duct, 3 - inlet optical access module (IOAM), 4 - test section, 5 & 6 - big and small extension ducts, 7 - pressure transducers, X_s - swirler location, X_p - transducer port location, X_i - fuel injection location, (b) Visible coherent structure in greyscale images at 900 lpm AFR and full premixing for 45 lpm FFR

3 Neural symbolic dynamics

This section describes the proposed architecture for early detection of thermo-acoustic instability in a combustor via analyzing a sequence of hi-speed images. Figure 2 presents the schematics of the framework where a deep CNN is stacked with symbolic time series analysis (STSA). In the training phase, images (or a segment of the images) from unstable state for various operating conditions are used as the input to the CNN.

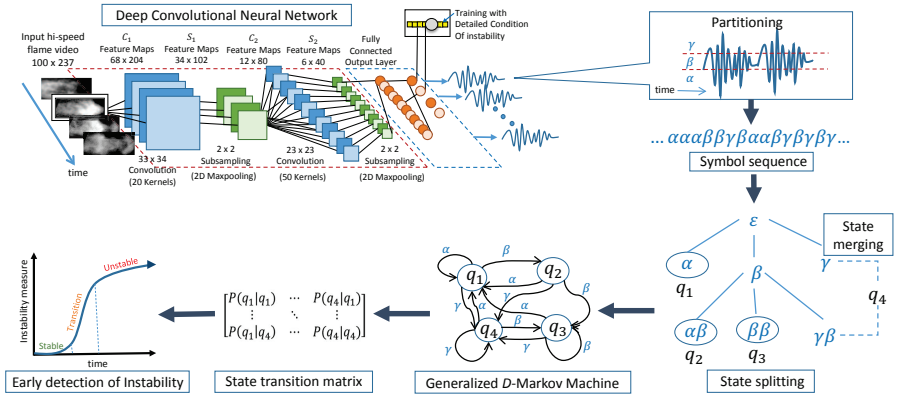


Figure 2: Neural-Symbolic Dynamics Architecture

While testing, sigmoid outputs from the fully connected layer can be utilized as a symbol sequence to facilitate in capturing the temporal evolution of coherent structures in the flame, thereby serving as a precursor in the early detection of unstable combustion flames. In STSA module, the time-series is symbolized via partitioning the signal space and a symbol sequence is created as shown in the figure 2. A generalized D-Markov machine is constructed from the symbol sequence via state splitting and state merging [13, 14], which models the transition from one state to another as state transition matrix. State transition matrix is the extracted feature which represents the sequence of images, essentially capturing the temporal evolution of coherent structures in the flame. Deep CNN and STSA structures are explained in the sequel.

3.1 Deep Convolutional Neural Network

The recent success of the deep learning architecture can be largely attributed to the strong emphasis on modeling multiple levels of abstractions (from low-level features to higher-order representations, i.e., features of features) from data. For example, in a typical image processing application while low-level features can be partial edges and corners, high-level features may be a combination of

edges and corners to form part of an image [7]. Among various deep learning techniques, Convolutional Neural Network (CNN) [15] is an attractive option for extracting pertinent features from images in a hierarchical manner for detection, classification, and prediction. For the purpose of the study, CNN remains a suitable choice as it preserves the local structures in an image at various scales. Hence, it is capable to extract local coherent structures of various sizes in a flame image. CNNs are also easier to train while achieving a comparable (and often better) performance despite the fact that it has fewer parameters relative to other fully connected networks with the same number of hidden layers.

In CNNs, data is represented by multiple feature maps in each hidden layer as shown in the figure 2. Feature maps are obtained by convolving the input image by multiple filters in the corresponding hidden layer. To further reduce the dimension of the data, these feature maps typically undergo non-linear downsampling with a 2×2 or 3×3 maxpooling. Maxpooling essentially partitions the input image into sets of non-overlapping rectangles and takes the maximum value for each partition as the output. After maxpooling, multiple dimension-reduced vector representations of the input is acquired and the process is repeated in the next layer to learn a higher representation of the data. At the final pooling layer, resultant outputs are linked with the fully connected layer where sigmoid outputs from the hidden units are post-processed by a softmax function in order to predict the class that possesses the highest joint probability given the input data. This way, coherent structures in the unstable flame can be learned at different operating condition.

3.2 Symbolic Time Series Analysis (STSA)

STSA [16] deals with discretization of dynamical systems in both space and time. The notion of STSA has led to the development of a (nonlinear) data-driven feature extraction tool for dynamical systems. Rao et al. [17] and Bahrampour et al. [18] have shown that the performance of this PFSA-based tool as a feature extractor for statistical pattern recognition is comparable (and often superior) to that of other existing techniques (e.g., Bayesian filters, Artificial Neural Networks, and Principal Component Analysis [5]). The trajectory of the dynamical system is partitioned into finitely many mutually exclusive and exhaustive cells for symbolization, where each cell corresponds to a single symbol belonging to a (finite) alphabet Σ. There are different types of partitioning tools, such as maximum entropy partitioning (MEP), uniform partitioning (UP) [19] and maximally bijective partitioning [20]. This paper has adopted MEP for symbolization of time series, which maximizes the entropy of the generated symbols by putting (approximately) equal number of data points in each partition cell. The next step is to construct probabilistic finite state automata (PFSA) from the symbol strings to encode the embedded statistical information. PFSA is a 4-tuple $K = (\Sigma, Q, \delta, \pi)$ which consists of a finite set of states (Q) interconnected by transitions [21], where each transition corresponds to a symbol in the finite alphabet (Σ). At each step, the automaton moves from one state to another (including self loops) via transition maps ($\delta : Q \times \Sigma \rightarrow Q$) according to probability morph function ($\tilde{\pi} : Q \times \Sigma \rightarrow [0, 1]$), and thus generates a corresponding block of symbols so that the probability distributions over the set of all possible strings defined over the alphabet are represented in the space of PFSA.

3.2.1 Generalized D-Markov Machine [10]

D-Markov machine is a model of probabilistic languages based on the algebraic structure of PFSA. In D-Markov machines, the future symbol is causally dependent on the (most recently generated) finite set of (at most) D symbols, where D is a positive integer. The underlying FSA in the PFSA of D-Markov machines are deterministic. The complexity of a D-Markov machine is reflected by the entropy rate which also represents its overall capability of prediction. A D-Markov machine and its entropy rate are formally defined as:

Definition 3.1 *(D-Markov) A D-Markov machine is a statistically stationary stochastic process $S = \cdots s_{-1}s_0s_1 \cdots$ (modeled by a PFSA in which each state is represented by a finite history of at most D symbols), where the probability of occurrence of a new symbol depends only on the last D symbols, i.e.,*

$$P[s_n \mid \cdots s_{n-D} \cdots s_{n-1}] = P[s_n \mid s_{n-D} \cdots s_{n-1}] \tag{1}$$

D is called the depth. Q is the finite set of states with cardinality $|Q| \leq |\Sigma|^D$, i.e., the states are represented by equivalence classes of symbol strings of maximum length D, where each symbol belongs to the alphabet Σ. $\delta : Q \times \Sigma \rightarrow Q$ is the state transition function that satisfies the following condition: if $|Q| = |\Sigma|^D$, then there exist $\alpha, \beta \in \Sigma$ and $x \in \Sigma^\star$ such that $\delta(\alpha x, \beta) = x\beta$ and $\alpha x, x\beta \in Q$.

Definition 3.2 *(D-Markov Entropy Rate) The D-Markov entropy rate of a PFSA (Σ, Q, δ, π) is defined in terms of the conditional entropy as:*

$$H(\Sigma|Q) \triangleq \sum_{q \in Q} P(q) H(\Sigma|q) = -\sum_{q \in Q} \sum_{\sigma \in \Sigma} P(q) P(\sigma|q) \log P(\sigma|q)$$

where $P(q)$ is the probability of a PFSA state $q \in Q$ and $P(\sigma|q)$ is the conditional probability of a symbol $\sigma \in \Sigma$ given that a PFSA state $q \in Q$ is observed.

3.2.2 Construction of a D-Markov Machine [13]

The underlying procedure for construction of a D-Markov machine from a symbol sequence consists of two major steps: *state splitting* and *state merging* [13, 14]. In general, state splitting increases the number of states to achieve more precision in representing the information content of the dynamical system. State merging reduces the number of states in the D-Markov machine by merging those states that have similar statistical behavior. Thus, a combination of state splitting and state merging leads to the final form of the generalized D-Markov machine as described below.

State Splitting: The number of states of a D-Markov machine of depth D is bounded above by $|\Sigma|^D$, where $|\Sigma|$ is the cardinality of the alphabet Σ. As this relation is exponential in nature, the number of states rapidly increases as D is increased. However, from the perspective of modeling a symbol sequence, some states may be more important than others in terms of their embedded information contents. Therefore, it is advantageous to have a set of states that correspond to symbol blocks of different lengths. This is accomplished by starting off with the simplest set of states (i.e., $Q = \Sigma$ for $D = 1$) and subsequently splitting the current state that results in the largest decrease of the D-Markov entropy rate. The process of splitting a state $q \in Q$ is executed by replacing the symbol block q by its *branches* as described by the set $\{\sigma q : \sigma \in \Sigma\}$ of words. Maximum reduction of the entropy rate is the governing criterion for selecting the state to split. In addition, the generated set of states must satisfy the self-consistency criterion, which only permits a unique transition to emanate from a state for a given symbol. If $\delta(q, \sigma)$ is not unique for each $\sigma \in \Sigma$, then the state q is split further. The process of state splitting is terminated by either the threshold parameter η_{spl} on the rate of decrease of entropy rate or a maximal number of states N_{max}. For construction of PFSA, each element $\pi(\sigma, q)$ of the morph matrix Π is estimated by frequency counting as the ratio of the number of times, $N(q\sigma)$, the state q is followed (i.e., suffixed) by the symbol σ and the number of times, $N(q)$, the state q occurs; the details are available in [13]. The estimated morph matrix $\widehat{\Pi}$ and the stationary state probability vector $\widehat{P}(q)$ are obtained as:

$$\hat{\pi}(q, \sigma) \triangleq \frac{1 + N(q\sigma)}{|\Sigma| + N(q)} \; \forall \sigma \in \Sigma \; \forall q \in Q; \widehat{P}(q) \triangleq \frac{1 + N(q)}{|Q| + \sum_{q' \in Q} N(q')} \; \forall q \in Q \tag{2}$$

where $\sum_{\sigma \in \Sigma} \hat{\pi}(\sigma, q) = 1 \; \forall q \in Q$. Then, the D-Markov entropy rate (see Definition 3.2) is computed as:

$$H(\Sigma|Q) = -\sum_{q \in Q} \sum_{\sigma \in \Sigma} P(q) P(\sigma|q) \log P(\sigma|q) \approx -\sum_{q \in Q} \sum_{\sigma \in \Sigma} \widehat{P}(q) \hat{\pi}(q, \sigma) \log \hat{\pi}(q, \sigma)$$

State Merging: While merging the states, this algorithm aims to mitigate this risk of degraded precision via a stopping rule that is constructed by specifying an acceptable threshold η_{mrg} on the distance $\Psi(\cdot, \cdot)$ between the merged PFSA and the PFSA generated from the original time series. The distance metric $\Psi(\cdot, \cdot)$ between two PFSAs $K_1 = (\Sigma, Q_1, \delta_1, \pi_1)$ and $K_2 = (\Sigma, Q_2, \delta_2, \pi_2)$ is as follows:

$$\Psi(K_1, K_2) \triangleq \lim_{n \rightarrow \infty} \sum_{j=1}^{n} \frac{\left\| P_1(\Sigma^j) - P_2(\Sigma^j) \right\|_{\ell_1}}{2^{j+1}} \tag{3}$$

where $P_1(\Sigma^j)$ and $P_2(\Sigma^j)$ are the steady state probability vectors of generating words of length j from the PFSA K_1 and K_2, respectively, i.e., $P_1(\Sigma^j) \triangleq [P(w)]_{w\in\Sigma^j}$ for K_1 and $P_2(\Sigma^j) \triangleq [P(w)]_{w\in\Sigma^j}$ for K_2. States that behave similarly (i.e., have similar morph probabilities) have a higher priority for merging. The similarity of two states, $q, q' \in Q$, is measured in terms of the respective morph functions of future symbol generation as the distance between the two rows of the estimated morph matrix $\widehat{\Pi}$ corresponding to the states q and q'. The ℓ_1-norm has been adopted to be the distance function as seen below.

$$\mathcal{M}(q, q') \triangleq \|\hat{\pi}(q, \cdot) - \hat{\pi}(q', \cdot)\|_{\ell_1} = \sum_{\sigma\in\Sigma} |\hat{\pi}(q, \sigma) - \hat{\pi}(q', \sigma)|$$

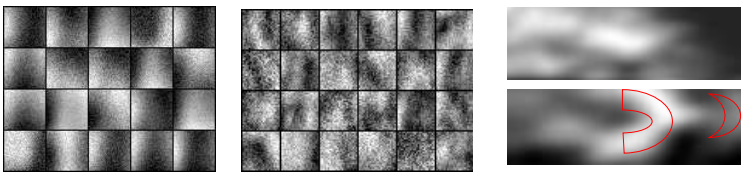
Hence, the two closest states (i.e., the pair of states $q, q' \in Q$ having the smallest value of $\mathcal{M}(q, q')$) are merged using the merging algorithm explained in [13]. The merging algorithm updates the morph matrix and transition function in such a way that does not permit any ambiguity of nondeterminism [8]. Subsequently, distance $\Psi(\cdot, \cdot)$ of the merged PFSA from the initial symbol string is evaluated. If $\Psi < \eta_{mrg}$ where η_{mrg} is a specified merging threshold, then the machine structure is retained and the states next on the priority list are merged. On the other hand, if $\Psi \geq \eta_{mrg}$, then the process of merging the given pair of states is aborted and another pair of states with the next smallest value of $\mathcal{M}(q, q')$ is selected for merging. This procedure is terminated if no such pair of states exist, for which $\Psi < \eta_{mrg}$.

4 Results and Discussions

This section discusses the results that are obtained when the proposed framework is applied on the experimental data of hi-speed video for early detection of thermo-acoustic instability.

4.1 CNN training

The network is trained using flame images with 4 different unstable combustion conditions mentioned in the section 2. The data consists of 24,000 examples for training and 12,000 examples for cross-validation. In the first convolutional layer, 20 filters of size 33×34 pixels (px) reduce the input image of dimension 100×237 pixels to feature maps of 68×204. Next, the feature maps are downsampled with a 2×2 max-pooling, resulting in pooled maps of 34×102 px. Each of these maps undergoes another convolutional layer with 50 filters of 23×23 px which produces feature maps of 12×80 px (before 2×2 max-pooling), and 6×40 pooled maps after max-pooling. All generated maps are connected to the fully connected layer of 100 hidden units followed by 10 output units where the sigmoid activations are extracted. Training is performed with a batch size of 20 and learning rate of 0.1. Convolution is done without any padding with a stride-size of 1. Visualization of few filters at first and second convolutional layer is shown in figure 3 (a), (b). Second layer visualization shows that it captures fragments of the flame coherent structures. Figure 3 (c) presents couple of the feature maps of a stable frame (top) and an unstable frame (bottom) after convolving with first layer filter. Red outline at the bottom exhibits how the mushroom-shaped coherent structure is highlighted on the unstable frame feature map.



(a) (b) (c)

Figure 3: Filter visualization at convolutional layer (a) one and (b) two. (b) shows fragmented representations of coherent structures that are visible in unstable flame. (c) Feature maps of a stable frame (top) and an unstable frame (bottom) after applying first convolutional layer filter. Red outline on the unstable flame visualization shows how the mushroom-shaped coherent structure is highlighted

4.2 STSA-based Instability measure

Once the CNN is trained on the sets of unstable data, every frame of the transition data sets(i.e., 500_{40to38}, 500_{40to30}, $40_{500to600}$, 600_{50to35} and $50_{700to800}$ as mentioned in section 2) are fed to

the CNN. Each sigmoid activation unit out of ten at the last fully connected layer generates a time series for one transition data set. For capturing the fast change in a transition data, a window of 0.5 seconds (1500 frames) is traversed over the hi-speed video with an overlap of 80% to keep the response speed at 10 Hz, which is necessary for real-time combustion instability control. The time window output of a sigmoid activation unit is symbolized by maximum entropy partitioning (MEP) with an alphabet size of $|\Sigma| = 3$. Considering the first window to be reference stable state, a generalized D-Markov machine is constructed by state splitting with $N_{max} = 10$ and state merging with $\eta_{mrg} = 0.05$. N_{max} is chosen as 10 because window length is not enough to learn a large state machine. For the alphabet $\{1, 2, 3\}$, the set of states after state splitting is $\{11, 21, 31, 2, 113, 213, 313, 23, 133, 233, 333\}$ and state merging leads to $\{11, 21, 31, 2, \{113, 313\}, 213, \{23, 133\}, \{233, 333\}\}$ for one of the sigmoid activation outputs in the transition video 600_{50to35}. State probability vector, arising from D-Markov machine at each time window, is the feature capturing the extent of instability which is transmitted through the corresponding sigmoid hidden unit. Instability measure of a time window is defined as the l_2 norm distance from the reference stable time window.

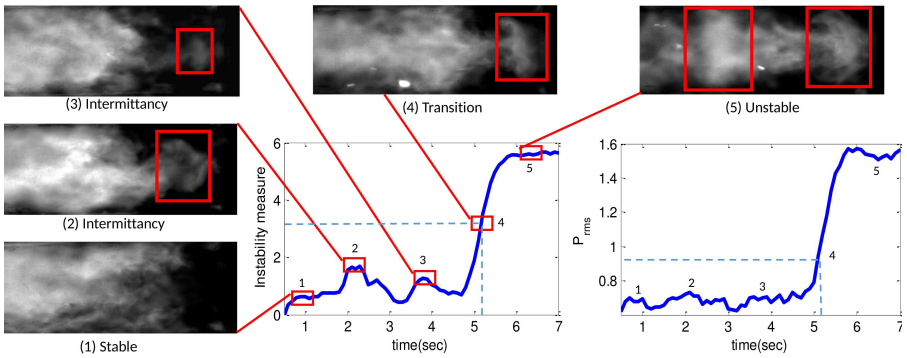


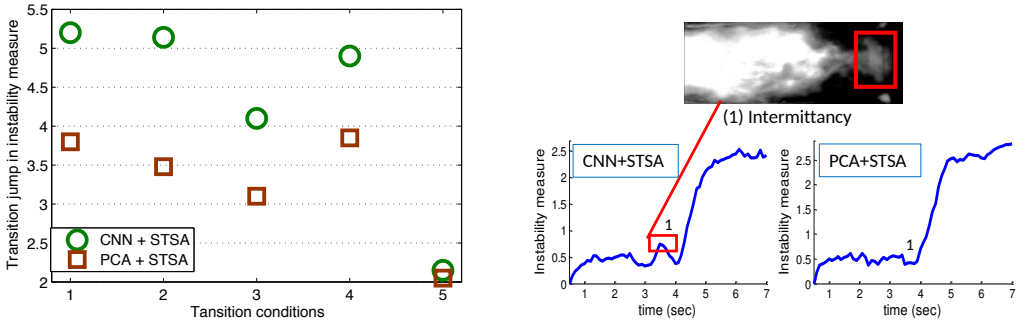
Figure 4: Variation of the proposed instability measure with time for the transition video named 600_{50to35}. Multiple regions on the measure curve denote different combustion states such as stable, temporary intermittancy (a significant precursor to persistent instability) and unstable . They are corresponded to varied coherent structures (bounded by red box) that are detected by the 'CNN+STSA' framework. On the right, rms variation of the pressure is shown as it is one of the most commonly used instability measures. Progression of P_{rms} can not detect the aforementioned precursors.

Figure 4 shows an aggregated progression (summation of individual instability measure obtained from each sigmoid activation unit) of instability measure for 600_{50to35}. The rms curve of the pressure on right of the figure 4 gives a rough idea about the ground truth regarding stability. Two fold advantages of the proposed instability measure over P_{rms} are as follows: (i) intermittancy phenomenon (region 2 and 3 on figure 4) is captured by this measure because it can detect variable-size mildly-illuminated mushroom-shaped coherent structure (bounded by red box in the figure 4) in the 'CNN+STSA' framework whereas P_{rms} ignores these important precursors to instability and (ii) region 4 of the figure 4 shows that the proposed measure rises faster towards instability. Other transition data sets also exhibit similar nature regarding this measure. Hence, the proposed measure performs better in early detection of instability than other commonly used measures such as P_{rms}.

4.3 Comparison with 'PCA+STSA'

To compare with the proposed approach, Principal Component Analysis (PCA) [5], a well-known dimensionality reduction technique is used as a replacement of CNN module. Figure 5 (a) shows that the transition (stable to unstable) increment of aggregated instability measure for 'CNN+STSA' is larger than that for 'PCA+STSA' in all transition data. This will result in more precise instability control in real time. The condition $50_{700to800}$ is observed in figure 5 (b) as the transition jump for both frameworks are very close. A close observation of the instability measure variation reveals that 'CNN+STSA' can detect an intermittancy precursor (region 1 at figure 5 (b)) although the coherent structure formation is not very prominent. However, 'PCA+STSA' misses this precursor before

arriving at the inception of persistent instability. A probable rationale behind this observation is that, while PCA is averaging the image vector based on just maximum spatial variance, CNN is learning semantic features based on the coherent structures of varied illumination, size and orientation seen during unstable combustion.



(a) (b)

Figure 5: (a) Comparison of sudden change in instability measure when instablity sets in for different transition conditions which are 1. 500_{40to38}, 2. 500_{40to30}, 3. $40_{500to600}$, 4. 600_{50to35} and 5. $50_{700to800}$. The jump is larger for 'CNN+STSA' than 'PCA+STSA'. (b) Variation of instability measure for both 'CNN+STSA' than 'PCA+STSA' at transition condition $50_{700to800}$. The measure arising from 'CNN+STSA' can detect the intermittancy precursor whereas it is mostly ignored by 'PCA+STSA'. A frame with an intermittancy coherent structure in a red box is shown on the top.

5 Conclusions and future work

The paper proposes a framework that synergistically combines the recently introduced concepts of CNN and STSA for early detection of thermo-acoustic instability in gas turbine engines. Extensive set of experiments have been conducted on a swirl-stabilized combustor for validation of the proposed method. Sequences of hi-speed greyscale images are fed into a multi-layered CNN to model the fluctuating coherent structures in the flame, which are dominant during unstable combustion. Fragments of coherent structures are observed in the CNN filter visualization. Therefore, an ensemble of time series data is constructed from sequence of images based on the sigmoid activation probability vectors of last hidden layer at the CNN. Then, STSA is applied on the time series that is generated from an image sequence and 'CNN+STSA' is found to exhibit larger change in instability measure while transition to instability than 'PCA+STSA'. The proposed framework detects all the intermittent precursors for different transition protocols, which is the most significant step towards detecting the onset of instability early enough for mitigation. In summary, while CNN captures the semantic features (i.e., coherent structures) of the combustion flames at varied illuminations, sizes and orientations, STSA models the temporal fluctuation of those features at a reduced dimension.

One of the primary advantages of the proposed semantic dimensionality reduction (as opposed to abstract dimensionality reduction, e.g., using PCA) would be seamless involvement of domain experts into the data analytics framework for expert-guided data exploration activities. Developing novel use-cases in this neural-symbol context will be a key future work. Some other near-term research tasks are: (i) dynamically tracking multiple coherent structures in the flame to characterize the extent of instability, (ii) multi-dimensional partitioning for direct usage of the last sigmoid layer and (iii) learning CNN and STSA together.

Acknowledgment

Authors sincerely acknowledge the extensive data collection performed by Vikram Ramanan and Dr. Satyanarayanan Chakravarthy at Indian Institute of Technology Madras (IITM), Chennai. Authors also gratefully acknowledge the support of NVIDIA Corporation with the donation of the GeForce GTX TITAN Black GPU used for this research.

References

[1] A. Garcez, T. R. Besold, L. de Raedt, P. Foeldiak, P. Hitzler, T. Icard, K. Kuehnberger, L. C. Lamb, R. Miikkulainen, and D. L. Silver. Neural-symbolic learning and reasoning: Contributions and challenges. *Proceedings of the AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches, Stanford*, March 2015.

[2] A. K. M. F. Hussain. Coherent structures - reality and myth. *Physics of Fluids*, 26(10):2816–2850, 1983.

[3] S. R. Chakravarthy, O. J. Shreenivasan, B. Bhm, A. Dreizler, and J. Janicka. Experimental characterization of onset of acoustic instability in a nonpremixed half-dump combustor. *Journal of the Acoustical Society of America*, 122:120127, 2007.

[4] G Berkooz, P Holmes, and J L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25(1):539–575, 1993.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA, 2006.

[6] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[7] K. Kavukcuoglu, Y. L. Sermanet, P. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierachies for visual recognition. In *NIPS*, 2010.

[8] A. Ray. Symbolic dynamic analysis of complex systems for anomaly detection. *Signal Processing*, 84(7):1115–1130, July 2004.

[9] S. Sarkar, K. Mukherjee, S. Sarkar, and A. Ray. Symbolic dynamic analysis of transient time series for fault detection in gas turbine engines. *Journal of Dynamic Systems, Measurement, and Control*, 135(1):014506, 2013.

[10] S. Sarkar, A. Ray, A. Mukhopadhyay, R. R. Chaudhari, and S. Sen. Early detection of lean blow out (lbo) via generalized d-markov machine construction. In *American Control Conference (ACC), 2014*, pages 3041–3046. IEEE, 2014.

[11] V. Ramanan, S. R. Chakravarthy, S. Sarkar, and A. Ray. Investigation of combustion instability in a swirl-stabilized combustor using symbolic time series analysis. In *Proc. ASME Gas Turbine India Conference, GTIndia 2014, New Delhi, India*, pages 1–6, December 2014.

[12] S. Sarkar, K.G. Lore, S. Sarkar, V. Ramanan, S.R. Chakravarthy, S. Phoha, and A. Ray. Early detection of combustion instability from hi-speed flame images via deep learning and symbolic time series analysis. In *Annual Conference of The Prognostics and Health Management*, pages pre–prints. PHM, 2015.

[13] K. Mukherjee and A. Ray. State splitting and state merging in probabilistic finite state automata for signal representation and analysis. *Signal Processing*, 104:105–119, November 2014.

[14] S. Sarkar, A. Ray, A. Mukhopadhyay, and S. Sen. Dynamic data-driven prediction of lean blowout in a swirl-stabilized combustor. *International Journal of Spray and Combustion Dynamics*, 7(3):in–press, 2015.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] C. Daw, C. Finney, and E. Tracy. A review of symbolic analysis of experimental data. *Review of Scientific Instruments*, 74(2):915–930, 2003.

[17] C. Rao, A. Ray, S. Sarkar, and M. Yasar. Review and comparative evaluation of symbolic dynamic filtering for detection of anomaly patterns. *Signal, Image and Video Processing*, 3(2):101–114, 2009.

[18] S. Bahrampour, A. Ray, S. Sarkar, T. Damarla, and N.M. Nasrabadi. Performance comparison of feature extraction algorithms for target detection and classification. *Pattern Recogntion Letters*, 34(16):2126–2134, December 2013.

[19] V. Rajagopalan and A. Ray. Symbolic time series analysis via wavelet-based partitioning. *Signal Processing*, 86(11):3309–3320, November 2006.

[20] S. Sarkar, A. Srivastav, and M. Shashanka. Maximally bijective discretization for data-driven modeling of complex systems. In *Proceedings of American Control Conference, Washington, D.C.*, 2013.

[21] M. Sipser. *Introduction to the Theory of Computation, 3rd ed.* Cengage Publishing, Boston, MA, USA, 2013.

Predicting Embedded Syntactic Structures from Natural Language Sentences with Neural Network Approaches

Gregory Senay
Panasonic Silicon Valley Lab
Cupertino, CA 95014
gregory.senay@us.panasonic.com

Fabio Massimo Zanzotto
University of Rome Tor Vergata
Viale del Politecnico, 1, 00133 Rome, Italy
fabiomassimo.zanzotto@gmail.com

Lorenzo Ferrone
University of Rome Tor Vergata
Viale del Politecnico, 1, 00133 Rome, Italy
lorenzo.ferrone@gmail.com

Luca Rigazio
Panasonic Silicon Valley Lab
Cupertino, CA 95014
luca.rigazio@us.panasonic.com

Abstract

Syntactic parsing is a key component of natural language understanding and, traditionally, has a symbolic output. Recently, a new approach for predicting syntactic structures from sentences has emerged: directly producing small and expressive vectors that embed in syntactic structures. In this approach, parsing produces distributed representations. In this paper, we advance the frontier of these novel predictors by using the learning capabilities of neural networks. We propose two approaches for predicting the embedded syntactic structures. The first approach is based on a multi-layer perceptron to learn how to map vectors representing sentences into embedded syntactic structures. The second approach exploits recurrent neural networks with long short-term memory (LSTM-RNN-DRP) to directly map sentences to these embedded structures. We show that both approaches successfully exploit word information to learn syntactic predictors and achieve a significant performance advantage over previous methods. Results on the Penn Treebank corpus are promising. With the LSTM-RNN-DRP, we improve the previous state-of-the-art method by 8.68%.

1 Introduction

Syntactic structure is a key component for natural language understanding [8], with several studies showing that syntactic information helps in modeling meaning [21, 14, 25]. Consequently, a very active area in natural language processing is building predictors of *symbolic* syntactic structures from sentences; such predictors, called parsers, are commonly implemented as complex recursive or iterative functions. Even when learned from data, the recursive/iterative nature of parsers is generally not changed since learning is confined to a probability estimation of context-free rules [9, 6] or learning of local discriminative predictor ([23, 26]).

Despite the effort in building explicit syntactic structures, they are rarely used in that form for semantic tasks such as question answering [28], recognizing textual entailment [13], semantic textual similarity [1]. These tasks are generally solved by learning classifiers or regressors. Hence, syntactic structures are unfolded to obtain syntactic-rich feature vectors [14], used within convolution kernel functions [17], or guiding the application of recursive neural networks [25]. Syntactic structures are first discovered by parsers, then, unfolded by "semantic learners" in explicit or implicit syntactic feature vectors.

Distributed syntactic trees [30] have offered a singular opportunity to redraw the path between sentences and feature vectors used within learners of semantic tasks. These distributed syntactic trees embed syntactic trees in small vectors. Hence, a possibility is to learn functions to map sentences in distributed syntactic trees [29]. These functions have been called *distributed representation parsers* (DRPs) [29]. However, these distributed representation parsers suffer from major limitations because, due to data sparsity, these functions can only transform part-of-speech tag sequences in syntactic trees without the lexical information.

In this paper, we propose two novel approaches based on neural networks for building predictors of distributed syntactic structures. The first model is based on a multi-layer perceptron (MLP) which learns how to map sentences, transformed into vectors to distributed syntactic representations. The second model is based on a recurrent neural network (RNN) with long short-term memory (LSTM) which learns to directly map sentences to distributed trees. Both models show the ability to positively exploit words in learning these predictors and significantly outperform previous models [29].

The paper is organized as follows: Section 2 describes the background by reporting on the distributed syntactic trees and the idea of distributed representation parsers; Section 3 introduces our two novel approaches for distributed representation parsing: the model based on a multi-layer perceptron (MLP-DRP) and the model based on long short-term memory (LSTM-RNN-DRP); Section 4 reports on the experiments and the results. Finally, section 5 draws conclusions.

2 Background

2.1 Distributed Syntactic Trees: Embedding Syntactic Trees in Small Vectors

Embedding syntactic trees in small vectors [30] is a key idea which changes how syntactic information is used in learning. Stemming from the recently revitalized research field of Distributed Representations (DR) [18, 24, 4, 12, 25], distributed syntactic trees [30] have shown that it is possible to use small vectors for representing the syntactic information. In fact, feature spaces of subtrees underlying tree kernels [10] are fully embedded by these distributed syntactic trees.

We want to give an intuitive idea how this embedding works. To explain this idea, we need to start from the definition of tree kernels [10] used in kernel machines. In these kernels, trees T are seen as collections of subtrees $S(T)$ and a kernel $TK(T_1, T_2)$ between two trees performs a weighted count of common subtrees, that is:

$$TK(T_1, T_2) = \sum_{\tau_i \in S(T_1), \tau_j \in S(T_2)} \omega_{\tau_i} \omega_{\tau_j} \delta(\tau_i, \tau_j)$$

where ω_{τ_i} and ω_{τ_j} are the weights for subtrees τ_i and τ_j and $\delta(\tau_i, \tau_j)$ is the Kronecker's delta between subtrees. Hence, $\delta(\tau_i, \tau_j) = 1$ if $\tau_i = \tau_j$ else $\delta(\tau_i, \tau_j) = 0$. Distributed trees, in some sense, pack sets $S(T_{s_1})$ in small vectors. In the illustration of Figure 1, this idea is conveyed by packing images of subtrees in a small space, that is, the box under $DT(T_{s_1})$. By rotating and coloring subtrees, the picture in the box under $DT(T_{s_1})$ still allows us to recognize these subtrees. Consequently, it is possible to count how many subtrees are similar by comparing the picture in the box under $DT(T_{s_1})$ with the one under $DT(T_{s_2})$. We visually show that it is possible to pack subtrees in small boxes, hence, it should be possible to pack this information in small vectors.

The formal definition of these embeddings, called distributed syntactic trees $DT(T)$, is the following:

$$DT(T) = \sum_{\tau_i \in S(T)} \omega_i \vec{\tau}_i = \sum_{\tau_i \in S(T)} \omega_i \mathrm{dt}(\tau_i)$$

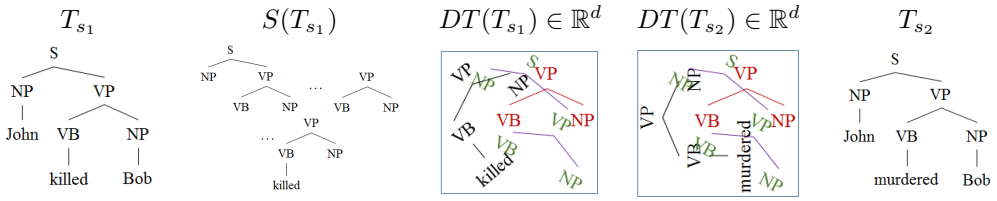


Figure 1: Distributed tree idea

where $S(T)$ is the set of the subtrees τ_i of T, $dt(\tau_i) = \vec{\tau}_i$ is a vector in $\mathbb{R}^d$ corresponding to the subtree τ_i, and ω_i is the weight assigned to that subtree. These vectors are obtained compositionally using vectors for node labels and shuffled circular convolution $\otimes$ as a basic composition function. For example, the last subtree of $S(T_{s_1})$ in Figure 1 has the following vector:

$$dt(T_1) = (\vec{S} \otimes (\vec{NP} \otimes \vec{John}) \otimes (\vec{VP} \otimes (\vec{VB} \otimes \vec{killed}) \otimes (\vec{VP} \otimes \vec{Bob})))$$

Vectors $dt(\tau_i)$ have the following property:

$$\delta(\tau_i, \tau_j) - \epsilon < |dt(\tau_i) \cdot dt(\tau_j)| < \delta(\tau_i, \tau_j) + \epsilon \tag{1}$$

with a high probability. Therefore, given two trees T_1 and T_2, the dot product between the two related, distributed trees approximates the tree kernel between trees $TK(T_1, T_2)$, that is:

$$DT(T_1) \cdot DT(T_2) = \sum_{\tau_i \in S(T_1), \tau_j \in S(T_2)} \omega_{\tau_i} \omega_{\tau_j} dt(\tau_i) \cdot dt(\tau_j) \approx TK(T_1, T_2)$$

with a given degree of approximation [30]. Hence, distributed syntactic trees allow us to encode syntactic trees in small vectors.

2.2 Distributed Representation Parsers

Building on the idea of encoding syntactic trees in small vectors [30], distributed representation parsers (DRPs) [29] have been introduced to predict these vectors directly from sentences. DRPs map sentence s to predicted distributed syntactic trees $DRP(s)$ (Figure 2), and represent the expected distributed syntactic trees $DT(T_s)$. In Figure 2, $DRP(s_1)$ is blurred to show that it is a predicted version of the correct distributed syntactic tree, $DT(T_{s_1})$. The DRP function is generally divided in two blocks: a sentence encoder SE and a transducer P, which is the actual "parser" as it reconstructs distributed syntactic subtrees. In contrast, the sentence encoder SE maps sentences into a distributed representation. For example, the vector $SE(s_1)$ represents s_1 in Figure 2 and contains subsequences of part-of-speech tags.

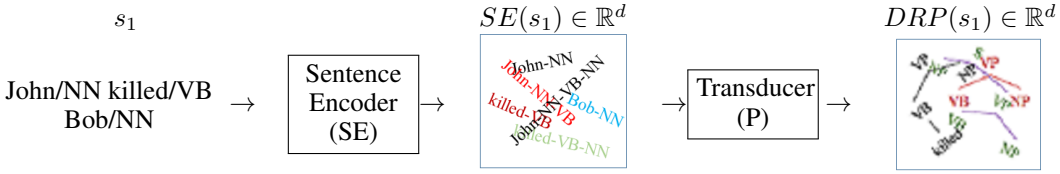


Figure 2: Visualization of the distributed representation parsing

Formally, a DRP is a function $DRP : \mathcal{X} \rightarrow R^d$ that maps sentences into $\mathcal{X}$ to distributed trees in R^d. The sentence encoder $SE : \mathcal{X} \rightarrow R^d$ maps sentences into $\mathcal{X}$ to distributed representation of sentence sequences defined as follows:

$$SE(s) = \sum_{seq_i \in SUB(s)} \vec{seq}_i$$

where $SUB(s)$ is a set of all relevant subsequences of s, and $\vec{seq}_i$ are nearly orthonormal vectors representing given sequences seq_i. Also, vectors seq_i are nearly orthonormal (c.f., Equation 1 applied to sequences instead of subtrees) and are obtained composing vectors for individual elements in sequences. For example, the vector for the subsequence $seq_1 =$ *John-NN-VB* is:

$$\vec{seq}_1 = \vec{John} \otimes \vec{NN} \otimes \vec{VB}$$

The transducer $P : R^d \rightarrow R^d$ is instead a function that maps distributed vectors representing sentences to distributed trees. In [29], P has been implemented as a square matrix trained with a partial least square estimate.

3 Predicting Distributed Syntactic Trees

Distributed representation parsing establishes a different setting for structured learning where a multi-layer perceptron (MLP) can help. In this novel setting, MLP are designed to learn functions that map sentences s or distributed sentences $SE(s)$ to low dimensional vectors embedding syntactic trees $DRP(s)$.

We thus explored two models: (1) a model based on a multi-layer perceptron to learn to transducers P that maps distributed sentences $SE(s)$ to distributed trees $DRP(s)$; (2) a model based on a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) which learns how to map sentences s as word sequences to distributed trees $DRP(s)$.

3.1 From Distributed Sentences to Distributed Trees with Multi-Layer Perceptrons

Our first model is based on a multi-layer perceptron (MLP) to realize the transducer $P_{MLP} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ (see Figure 2), which maps distributed sentences $SE(s)$ to distributed structures $DRP(s)$. The overall distributed representation parser based on the multi-layer perceptron is referred to as ***MLP-DRP***. To define our ***MLP-DRP*** model, we need to specify: (1) the input and the expected output of P_{MLP}; (2) the topology of the MLP.

We defined two classes of input and output for the transducer P_{MLP}: an *unlexicalized model* (UL) and a *lexicalized model* (L). In the *unlexicalized* model, input distributed sentences and output distributed trees do not contain words. Distributed sentences encode only sequences of part-of speech tags. We experimented with $SEQ_{UL}(s)$ containing sequences of part-of-speech tags up to 3. For example, $SEQ_{UL}(s_1) = \{$NN,NN-VB,NN-VB-NN,VB,VB-NN,NN$\}$ (see Figure 2). Similarly, distributed trees encode syntactic subtrees without words, for example, *(VP (VB NN))* . On the other hand, in the lexicalized model, input distributed sentences and output distributed trees are lexicalized. The lexicalized version of distributed sentences was obtained by concatenating previous part-of-speech sequences with their first words. For example, $Seq_{UL}(s_1) = \{$John-NN,John-NN-VB,John-NN-VB-NN,killed-VB,killed-VB-NN,Bob-NN$\}$. Distributed trees encode all the subtrees, including those with words.

Then, we setup a multi-layer perceptron that maps $x = SE(s)$ to $y' = DRP(s)$ and its expected output is $y = DT(T_s)$. The layer 0 of the network has the activation:

$$a^{(0)} = \sigma(W^{(0)}x + b^{(0)})$$

We selected a sigmoid function as the activation function σ:

$$\sigma(z) = \frac{1}{1 + exp(-z)}.$$

All intermediate $n - 2$ layers of the network have the following activation :

$$\forall n \in [1; N-2] : a^{(n)} = \sigma(W^{(n-1)}a^{(n-1)} + b^{(n-1)})$$

The final reconstructed layer, with output y', is done with a linear function:

$$y'(x) = W^{(N-1)}a^{(N-1)} + b^{(N-1)}$$

We learn the network weights by using the following cost function:

$$J(W, b; x, y', y) = 1 - \frac{y \cdot y'}{||y||\ ||y'||},$$

that evaluates the cosine similarity between y and y'.

Learning unlexicalized and lexicalized MLP-DRPs is feasible even if the two settings hide different challenges. The unlexicalized MLP-DRP exposes network learned with less information to encode. However, the model cannot exploit the important information on words. In contrast, the lexicalized MLP-DRP can exploit words but it has to encode more information. Experiments with the two settings are reported in Section 4.

3.2 From Word Sequences to Distributed Trees with Long Short Term Memory

Our second model is more ambitious: it is an end-to-end predictor of distributed syntactic trees $DRP(s)$ from sentences s. We based our approach on recurrent neural networks (RNN) since RNNs have already proven their efficiency to learn complex sequence-to-sequence mapping in speech recognition [16] and in handwriting [15]. Moreover, RNNs have been also successfully used to learn mapping functions between word sequences through sentence embedding vectors [7, 2].

Our end-to-end predictor of distributed syntactic structures is built on the recurrent neural network model with long-short term memory (LSTM-RNN) [20] to overcome the vanishing gradient problem. However, to increase computational efficiency, in this model the activation of the output gate of each cell does not depend on its memory state.

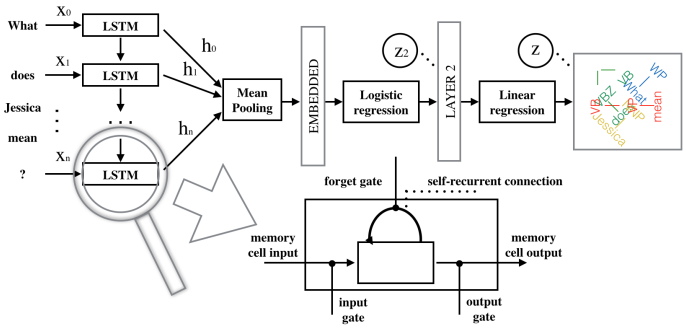


Figure 3: Structure of our LSTM-RNN-DRP encoder and a detail of the LSTM neuron

The resulting distributed representation parser LSTM-RNN-DRP is then defined as follows: Input sentences s are seen as word sequences. To each word in these sequences, we assigned a unit base vector $x_t \in \mathbb{R}^L$ where L is the size of the lexicon. x_t is 1 in the t-th component representing the word and 0 otherwise. Words are encoded with 4 matrices $W_i, W_c, W_f, W_o \in \mathbb{R}^{m \times L}$. Hence, m is the size of word encoding vectors. The LSTM cells are defined as follows: x_t is an input word to the memory cell layer at time t. i_t is the input gate define by:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{2}$$

where σ is a sigmoid. $\tilde{C}_t$ is the candidate values of the states of the memory cells:

$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c). \tag{3}$$

f_t is the activation of the memory cell's forget gates:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f). \tag{4}$$

Given i_t, f_t and $\tilde{C}_t$, C_t memory cells are computed with:

$$C_t = i_t \star \tilde{C}_t + f_t \star C_{t-1}, \tag{5}$$

where $\star$ is the element-wise product. Given the state of the memory cells, we compute the output gate with:

$$\begin{aligned} o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_1) \\ h_t &= o_t \star tanh(C_t) \end{aligned} \tag{6}$$

The non recurrent part of this model is achieved by an average pooling of the sequence representation $h_0, h_1, ..., h_n$, the 4 matrix W_* are concatenated into a single one: W, the U_* weight matrix into U

and the bias b_* into b (see Figure 3). Then, a pre-nonlinear function is computed with W, U and b, following by a linear function:

$$\begin{aligned} z_2 &= \sigma(Wx_t + U_{t-1} + b) \\ z &= W_2 z_2 + b_2 \end{aligned} \tag{7}$$

Finally, the cost function of this model is the cosine similarity between the reconstructed output z and $DT(T_s)$.

4 Experiments

This section explores whether our approaches can improve existing models for learning distributed representation parsers (DRPs). Similarly to [29], we experimented with the classical setting of learning parsers adapted to the novel task of learning DRPs.

In these experiments, all trainings are done with a maximum number of epochs of 5000. If a better result is not found on the validation set after a patience of 30 epochs, we stop the training. All deep learning experiments are done with the *Theano* toolkit [5, 3]. The dimension of the embedded vector after the mean pooling is fixed to 1024 and the second layer size is fixed to 2048. There dimensions are fixed empirically.

4.1 Experimental set-up

The experiment is based on the revised evaluation model for parsers adapted to the task of learning distributed representation parsers [29]. Here we use the **Penn Treebank** corpus for learning and predicting the embedded syntactic structures. The distributed version of the Penn Treebank contains distributed sentences $SE(s)$ along related oracle distributed syntactic trees $DT(T_s)$ for all the sections of the Penn Treebank. Distributed syntactic trees are provided for three different λ values: 0, 0.2 and 0.4. As in tree kernels, λ governs weights ω_{τ_i} of subtrees τ_i. For each λ, there are two versions of the data sets: an un-lexicalized version (UL), where sentences and syntactic trees are considered without words, and a lexicalized version (L), where words are considered. Because the LSTM-RNN-DRP approach is based on word sequence, only the lexicalized results are reported. As for parsing, the datasets from the Wall Street Journal (WSJ) section are divided in: sections 20-21 with 39,832 distributed syntactic trees for training, section 23 with 2,416 distributed syntactic trees for testing and section 24 with 1,346 distributed syntactic trees for parameter estimation.

The evaluation measure is the cosine similarity $cos(DRP(s), DT(T_s))$ between predicted distributed syntactic trees $DRP(s)$ and distributed syntactic trees $DT(T_s)$ of the distributed Penn Treebank, computed for each sentence in the testing and averaged on all the sentences.

We compared our novel models with respect to the model in [29], ZD-DRP (the baseline), and we respect the chain of building distributed syntactic representations that involve a symbolic parser SP, that is, $DSP(s) = DT(SP(s))$. In line with[29], as symbolic parser SP, we used the Bikel's parser.

4.2 Results and discussion

The question we want to answer with these experiments is whether MLP-DRP and LSTM-RNN-DRP can produce better predictors of distributed syntactic trees from sentences. To compare with previous results, we experimented with the distributed Penn Treebank set.

We experimented with d=4096 as the size of the space for representing distributed syntactic trees. We compared with a previous approach, that is ZD-DRP [29] and with the upper-bound of the distributed symbolic parser DSP.

Our novel predictors of distributed syntactic trees outperform previous models for all the values of the parameters (see Table 1). Moreover, our MLP-DRP captures better structural information than the previous model ZD-DRP. In fact, when λ is augmented, the difference in performance between our MLP-DRP and ZD-DRP increases. With higher λ, larger structures have higher weights. Hence, our model captures these larger structures better than the baseline system. In addition, our model is definitely closer to the distributed symbolic parser DSP in the case of unlexicalized trees. This is promising, as the DSP is using lexical information whereas our MLP-DRP does not.

Table 1: Predicting distributed trees on the Distributed Penn Treebank (section 23): average cosine similarity between predicted and oracle distributed syntactic trees. ZD-DRP is a previous baseline model, MLP-DRP is our model and DSP is a the Bikel's parser with a distributed tree function.

	unlexicalized trees			*lexicalized trees*		
Model	$\lambda = 0$	$\lambda = 0.2$	$\lambda = 0.4$	$\lambda = 0$	$\lambda = 0.2$	$\lambda = 0.4$
ZD-DRP (baseline)	0.8276	0.7552	0.6506	0.7192	0.6406	0.0646
MLP-DRP	**0.8358**	**0.7863**	**0.7038**	**0.7280**	0.6740	0.4960
LSTM-RNN-DRP	-	-	-	0.7162	**0.7274**	**0.5207**
DSP	0.8157	0.7815	0.7123	0.9073	0.8564	0.6459

Our second approach LSTM-RNN-DRP, based on the word sequence, outperforms the other approaches for lexicalized setup. Results show a high improvement compared to the baseline (+8.68% absolute with $\lambda = 0.2$) and it shows this model can represent lexical information better than MLP-DRP under the same conditions.

Finally, our new models reduce the gap in performances with the DSP on the lexicalized trees by dramatically improving over previous models on $\lambda = 0.4$. The increase in performance of our approaches with respect to ZD-DRP is extremely important as it confirms that MLP-DRP and LSTM-RNN-DRP can encode words better.

5 Conclusion

This paper explores two novel methods to merge symbolic and distributed approaches. Predicting distributed syntactic structures is possible and our models show that neural networks can definitely play an important role in this novel emerging task. Our predictor based on a Multi-Layer Perceptron and Long-Short Term Memory Recurrent Neural Network outperformed previous models. This last method, RNN-LSTM-DRP is able, other than the word level, to predict the syntactic information from the sentence. This is a step forward to use these predictors that may change the way syntactic information is learned.

Future research should focus on exploring the promising capability of encoding words shown by recurrent neural networks with long-short term memory. But we think a combinaison of both our approaches can also increase the quality of our predictor due to the fact that each approach encode different information of the tree. This should lead a better predictor of distributed syntactic structures.

References

[1] E. Agirre, D. Cer, M. Diab, A. Gonzalez-Agirre, and W. Guo. *sem 2013 shared task: Semantic textual similarity. In **SEM*, pages 32–43, USA, 2013. ACL.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

[3] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *arXiv:1211.5590*, 2012.

[4] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1), 2009.

[5] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of SciPy* , 2010.

[6] E. Charniak. A maximum-entropy-inspired parser. In *Proc. of the 1st NAACL*, 2000.

[7] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.

[8] N. Chomsky. *Aspect of Syntax Theory*. MIT Press, Cambridge, Massachussetts, 1957.

[9] M. Collins. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4), 2003.

[10] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *NIPS*, 2001.

[11] Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of ACL02*. 2002.

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12, 2011.

[13] I. Dagan, D. Roth, M. Sammons, and F.M. Zanzotto. *Recognizing Textual Entailment: Models and Applications*. Synthesis Lectures on HLT. Morgan&Claypool Publishers, 2013.

[14] Daniel Gildea and Daniel Jurafsky. Automatic Labeling of Semantic Roles. *Comp. Ling.*, 28(3), 2002.

[15] A. Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[16] A. Graves, A. R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, IEEE, 2013.

[17] D. Haussler. Convolution kernels on discrete structures. Tech.Rep., Univ. of California at S. Cruz, 1999.

[18] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, MA., 1986.

[19] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[20] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

[21] B. MacCartney, T. Grenager, M. -C. de Marneffe, D. Cer, and C. D. Manning. Learning to recognize features of valid textual entailments. In *Proceedings of NAACL*, New York City, USA, 2006.

[22] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330, 1993.

[23] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Nat. Lang. Eng.*, 13(2), 2007.

[24] T. A. Plate. *Distributed Representations and Nested Compositional Structure*. PhD thesis, 1994.

[25] R. Socher, E. H. Huang, J. Pennington, A. Y. Ng, and C. D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*. 2011.

[26] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of ICML*, 2011.

[27] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, 2013.

[28] Ellen M. Voorhees. The trec question answering track. *Nat. Lang. Eng.*, 7(4):361–378, 2001.

[29] F.M. Zanzotto and L. Dell'Arciprete. Transducing sentences to syntactic feature vectors: an alternative way to "parse"? In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, 2013.

[30] F.M. Zanzotto and L. Dell'Arciprete. Distributed tree kernels. In *Proceedings of ICML*, 2012.

Building Memory with Concept Learning Capabilities from Large-scale Knowledge Base

Jiaxin Shi[⋆] **Jun Zhu**[†]
Department of Computer Science
Tsinghua University
Beijing, 100084
[⋆]ishijiaxin@126.com [†]dcszj@mail.tsinghua.edu.cn

Abstract

We present a new perspective on neural knowledge base (KB) embeddings, from which we build a framework that can model symbolic knowledge in the KB together with its learning process. We show that this framework well regularizes previous neural KB embedding model for superior performance in reasoning tasks, while having the capabilities of dealing with unseen entities, that is, to learn their embeddings from natural language descriptions, which is very like human's behavior of learning semantic concepts.

1 Introduction

Recent years have seen great advances in neural networks and their applications in modeling images and natural languages. With deep neural networks, people are able to achieve superior performance in various machine learning tasks [1, 2, 3, 4]. One of those is relational learning, which aims at modeling relational data such as user-item relations in recommendation systems, social networks and knowledge base, etc. In this paper we mainly focus on knowledge base.

Generally a knowledge base (KB) consists of triplets (or facts) like (e_1, r, e_2), where e_1 and e_2 denote the left entity and the right entity, and r denotes the relation between them. Previous works on neural KB embeddings model entities and relations with distributed representation, i.e., vectors [5] or matrices [6], and learn them from the KB. These prove to be scalable approaches for relational learning. Experiments also show that neural embedding models obtain state-of-art performance on reasoning tasks like link prediction. Section 2 will cover more related work.

Although such methods on neural modeling of KB have shown promising results on reasoning tasks, they have limitations of only addressing known entities that appear in the training set and do not generalize well to settings where we have unseen entities. Because they do not know embedding representations of new entities, they cannot establish relations with them. On the other hand, the capability of KB to learn new concepts as entities, or more specifically, to learn what a certain name used by human means, is obviously highly useful, particularly in a KB-based dialog system. We observe that during conversations human does this task by first asking for explanation and then establishing knowledge about the concept from other peoples' natural language descriptions. This inspired our framework of modeling human's cognitive process of learning concepts during conversations, i.e., the process from natural language description to a concept in memory.[1] We use a neural embedding model [5] to model the memory of concepts. When given description text of a new concept, our framework directly transforms it into an entity embedding, which captures semantic information about this concept. The entity embedding can be stored and later used for other

[1]Concept learning in cognitive science usually refers to the cognitive process where people grow abstract generalizations from several example objects [7]. We use concept learning here to represent a different behavior.

semantic tasks. Details of our framework are described in Section 3. We will show efficiency of this framework in modeling entity relationships, which involve both natural language understanding and reasoning.

Our perspective on modeling symbolic knowledge with its learning process has two main advantages. First, it enables us to incorporate natural language descriptions to augment the modeling of relational data, which fits human's behavior of learning concepts during conversations well. Second, we also utilize the large number of symbolic facts in knowledge base as labeled information to guide the semantic modeling of natural language. The novel perspective together with framework are the key contributions of this work.

2 Related work

Statistical relational learning has long been an important topic in machine learning. Traditional methods such as Markov logic networks [8] often suffer from scalability issues due to intractable inference. Following the success of low rank models [9] in collaborative filtering, tensor factorization [10, 11] was proposed as a more general form to deal with multi-relational learning (i.e., multiple kinds of relations exist between two entities). Another perspective is to regard elements in factorized tensors as probabilistic latent features of entities. This leads to methods that apply nonparametric Bayesian inference to learn latent features [12, 13, 14] for link prediction. Also, attempts have been made to address the interpretability of latent feature based models under the framework of Bayesian clustering [15]. More recently, with the noticeable achievements of neural embedding models like word vectors [16] in natural language processing area, various neural embedding models [6, 17, 5, 4, 18] for relational data have been proposed as strong competitors in both scalability and predictability for reasoning tasks.

All these methods above model relational data under the latent-feature assumption, which is a common perspective in machine learning to gain high performance in prediction tasks. However, these models leave all latent features to be learnt from data, which suffers from substantial increments of model complexity when applying to large-scale knowledge bases. For example, [10] can be seen as having a feature vector for each entity in factorized tensors, while [6] also represents entities in separate vectors, or embeddings, thus the number of parameters scales linearly with the number of entities. A large number of parameters in these models often increases the risk of overfitting, but few of these works have proposed effective regularization techniques to address it. On the other hand, when applying these models to real world tasks (e.g., knowledge base completion), most of them have a shared limitation that entities unseen in training set cannot be dealt with, that is, they can only complete relations between known entities, which is far from what human's ability of learning new concepts can achieve. From this perspective, we develop a general framework that is capable of modeling symbolic knowledge together with its learning process, as detailed in Section 3.

3 The framework

Our framework consists of two parts. The first part is a memory storage of embedding representations. We use it to model the large-scale symbolic knowledge in the KB, which can be thought as memory of concepts. The other part is a concept learning module, which accepts natural language descriptions of concepts as the input, and then transforms them into entity embeddings in the same space of the memory storage. In this paper we use translating embedding model from [5] as our memory storage and use neural networks for the concept learning module.

3.1 Translating embedding model as memory storage

We first describe translating embedding (TransE) model [5], which we use as the memory storage of concepts. In TransE, relationships are represented as translations in the embedding space. Suppose we have a set of N true facts $D = \{(e_1, r, e_2)\}_N$ as the training set. If a fact (e_1, r, e_2) is true, then TransE requires $e_1 + r$ to be close to e_2. Formally, we define the set of entity vectors as E, the set of relation vectors as R, where $R, E \subset \mathbb{R}^n$, $e_1, e_2 \in E$, $r \in R$. Let d be some distance measure, which is either the L_1 or L_2 norm. TransE minimizes a margin loss between the score of true facts

in the training set and randomly made facts, which serve as negative samples:

$$\mathcal{L}(D) = \sum_{(e_1,r,e_2)\in D} \sum_{(e_1',r',e_2')\in D'_{(e_1,r,e_2)}} \max(0, \gamma + d(e_1 + r, e_2) - d(e_1' + r', e_2')), \tag{1}$$

where $D'_{(e_1,r,e_2)} = \{(e_1', r, e_2) : e_1' \in E\} \cup \{(e_1, r, e_2') : e_2' \in E\}$, and γ is the margin. Note that this loss favors lower distance between translated left entities and right entities for training facts than for random generated facts in D'. The model is optimized by stochastic gradient descent with mini-batch. Besides, TransE forces the L_2 norms of entity embeddings to be 1, which is essential for SGD to perform well according to [5], because it prevents the training process from trivially minimizing loss by increasing entity embedding norms.

There are advantages of using embeddings instead of symbolic representations for cognitive tasks. For example, it's kind of easier for us to figure out that a person who is a violinist can play violin than to tell his father's name. However, in symbolic representations like knowledge base, the former fact `<A, play, violin>` can only be deduced by reasoning process through facts `<A, has profession, violinist>` and `<violinist, play, violin>`, which is a two-step procedure, while the latter result can be acquired in one step through the fact `<A, has father, B>`. If we look at how TransE embeddings do this task, we can figure out that A plays violin by finding nearest neighbors of `A`'s embedding + `play`'s embedding, which costs at most the same amount of time as finding out who A's father is. This claim is supported by findings in cognitive science that the general properties of concepts (e.g., `<A, play, violin>`) are more strongly bound to an object than its more specific properties (e.g., `<A, has father, B>`) [19].

3.2 Concept learning module

As mentioned earlier, the concept learning module accepts natural language descriptions of concepts as the input, and outputs corresponding entity embeddings. As this requires natural language understanding with knowledge in the KB transferred into the module, neural networks can be good candidates for this task. We explore two kinds of neural network architectures for the concept learning module, including multi-layer perceptrons (MLP) and convolutional neural networks (CNN).

For MLP, we use one hidden layer with 500 neurons and RELU activations. Because MLP is fully-connected, we cannot afford the computational cost when the input length is too long. For large scale datasets, the vocabulary size is often as big as millions, which means that bag-of-words features cannot be used. Here, we use bag-of-n-grams features as inputs (there are at most $26^3 = 17576$ kinds of 3-grams in pure English text). Given a word, for example $word$, we first add starting and ending marks to it like #$word$#, and then break it into 3-grams (#wo, wor, ord, rd#). Suppose we have V kinds of 3-grams in our training set. For an input description, we count the numbers of all kinds of 3-grams in this text, which form a V-dimensional feature vector x. To control scale of the input per dimension, we use $log(1 + x)$ instead of x as input features. Then we feed this vector into the MLP, with the output to be the corresponding entity embedding under this description.

Since MLP with bag-of-n-grams features loses information of the word order, it has very little sense of the semantics. Even at the word level, it fails to identify words with similar meanings. From this point of view, we further explore the convolutional architecture, i.e. CNN together with word vector features. Let $s = w_1 w_2 ... w_k$ be the paragraph of a concept description and let $v(w_i) \in \mathbb{R}^d$ be the vector representation for word w_i. During experiments in this paper, we set $d = 50$ and initialize $v(w_i)$ with w_i's word vector pretrained from large scale corpus, using methods in [16]. Let A^s be the input matrix for s, which is defined by:

$$A^s_{:,i} = v(w_i), \tag{2}$$

where $A^s_{:,i}$ denotes the ith column of matrix A^s. For the feature maps at the lth layer $F^{(l)} \in \mathbb{R}^{c\times n\times m}$, where c is the number of channels, we add the convolutional layer like:

$$F^{(l+1)}_{i,:,:} = \sum_{j=1}^{c} F^{(l)}_{j,:,:} * K^{(l)}_{i,j,:,:}, \tag{3}$$

where $K^{(l)}$ denotes all convolution kernels at the lth layer, which forms an order-4 tensor (output channels, input channels, y axis, x axis). When modeling natural language, which is in a sequence

Table 1: CNN layers

Layer	Type	Description
1	convolution	kernel: $64 \times 50 \times 1$, stride: 1
2	convolution	kernel: $64 \times 1 \times 3$, stride: 1
3	max-pooling	pooling size: 1×2, stride: 2
4	convolution	kernel: $128 \times 1 \times 3$, stride: 1
5	convolution	kernel: $128 \times 1 \times 3$, stride: 1
6	max-pooling	pooling size: 1×2, stride: 2
7	convolution	kernel: $256 \times 1 \times 3$, stride: 1
8	max-pooling	pooling size: 1×2, stride: 2
9	convolution	kernel: $512 \times 1 \times 3$, stride: 1
10	max-pooling	pooling size: 1×2, stride: 2
11	dense	size: 500
12	output layer	normalization layer

form, we choose $K^{(l)}$ to have the same size in the y axis as feature maps $F^{(l)}$. So for the first layer that has the input size $1 \times D \times L$, we use kernel size $D \times 1$ in the last two axes, where D is the dimension of word vectors. After the first layer, the last two axes of feature maps in each layer remain to be vectors. We list all layers we use in Table 1, where kernels are described by output channels $\times$ y axis $\times$ x axis.

Note that we use neural networks (either MLP or CNN) to output the entity embeddings, while according to Section 3.1, the embedding model requires the L_2-norms of entity embeddings to be 1. This leads to a special normalization layer (the 12th layer in Table 1) designed for our purpose. Given the output of the second last layer $x \in \mathbb{R}^n$, we define the last layer as:

$$e_k = \frac{w_{k,:}^T x + b_k}{[\sum_{k'=1}^{n} (w_{k',:}^T x + b_{k'})^2]^{1/2}} \tag{4}$$

e is the output embedding. It's easy to show that $\|e\|_2 = 1$. Throughout our experiments, we found that this trick plays an essential role in making joint training of the whole framework work. We will describe the training process in Section 3.3.

3.3 Training

We jointly train our embedding model and concept learning module together by stochastic gradient descent with mini-batch and Nesterov momentum [20], using the loss defined by equation 1, where the entity embeddings are given by outputs of the concept learning module. When doing SGD with mini-batch, We back-propagate the error gradients into the neural network, and for CNN, finally into word vectors. The relation embeddings are also updated with SGD, and we re-normalize them in each iteration to make their L_2-norms stay 1.

4 Experiments

4.1 Datasets

Since no public datasets satisfy our need, we have built two new datasets to test our method and make them public for research use. The first dataset is based on FB15k released by [5]. We dump natural language descriptions of all entities in FB15k from Freebase [21], which are stored under relation `/common/topic/description`. We refer to this dataset as FB15k-desc[2]. The other dataset is also from Freebase, while we make it much larger. In fact, we include all entities that have descriptions in Freebase and remove triplets with relations in a filter set. Most relations in the filter set are schema relations like `/type/object/key`. This dataset has more than 4M entities, for which we call it FB4M-desc[3]. Statistics of the two datasets are presented in Table 2.

[2]FB15k-desc: Available at http://ml.cs.tsinghua.edu.cn/~jiaxin/fb15k_desc.tar.gz

[3]FB4M-desc: Available at http://ml.cs.tsinghua.edu.cn/~jiaxin/fb4m_desc.tar.gz

Table 2: Statistics of the datasets.

Dataset	Entities	Relations	Descriptions		Triplets (Facts)		
			Vocabulary	Length	Train	Validation	Test
FB15k-desc	14951	1345	58954	$\leqslant$435	483142	50000	59071
FB4M-desc	4629345	2651	1925116	$\leqslant$617	16805830	3021749	3023268

Table 3: Link prediction results on FB15k-desc.

Model	Mean rank			Hits@10 (%)		
	Left	Right	Avg	Left	Right	Avg
TransE[5]	-	-	243	-	-	34.9
Ours	252	176	**214**	34.3	41.1	**37.7**

Note that the scale is not the only difference between these two datasets. They also differ in splitting criteria. FB15k-desc follows FB15k's original partition of training, validation and test sets, in which all entities in validation and test sets are already seen in the training set. FB4M-desc goes the contrary way, as it is designed to test the concept learning ability of our framework. All facts in validation and test sets include an entity on one side that are not seen in the training set. So when evaluated on FB4M-desc, a good embedding for a new concept can only rely on information from the natural language description and knowledge transferred in the concept learning module.

4.2 Link prediction

We first describe the task of link prediction. Given a relation and an entity on one side, the task is to predict the entity on the other side. This is a natural reasoning procedure which happens in our thoughts all the time. Following previous work [5], we use below evaluation protocol for this task. For each test triplet (e_1, r, e_2), e_1 is removed and replaced by all the other entities in the training set in turn. The neural embedding model should give scores for these corrupted triplets. The rank of the correct entity is stored. We then report the mean of predicted ranks on the test set as the left mean rank. This procedure is repeated by corrupting e_2 and then we get the right mean rank. The proportion of correct entities ranked in the top 10 is another index, which we refer to as hits@10.

We test our link prediction performance on FB15k-desc and report it in Table 3. The type of concept learning module we use here is CNN. Note that all the triplets in training, validation and test sets of FB15k-desc are the same as FB15k, so we list TransE's results on FB15k in the same table. Compared to TransE which cannot make use of information in descriptions, our model performs much better, in terms of both mean rank and hits@10. As stated in Section 4.1, all entities in the test set of FB15k are contained in the training set, which, together with the results, shows that our framework well regularizes the embedding model by forcing embeddings to reflect information from natural language descriptions. We demonstrate the concept learning capability in the next subsection.

4.3 Concept learning capabilities

It has been shown in Section 4.2 that our framework well regularizes the neural embedding model for memory storage. Next we use FB4M-desc to evaluate the capability of our framework on learning new concepts and performing reasoning based on learnt embeddings. We report the link prediction performance on FB4M-desc in Table 4. Note that the test set contains millions of triples, which is very time-consuming in the ranking-based evaluation. So we randomly sample 1k, 10k and 80k triplets from the test set to report the evaluation statistics. We can see that CNN consistently outperforms MLP in terms of both mean rank and hits@10. All the triplets in the test set of FB4M-desc include an entity unseen in the training set on one side, requiring the model to understand natural language descriptions and to do reasoning based on it. As far as we know, no traditional knowledge base embedding model can compete with us on this task, which again claims the novelty of our framework.

Table 4: Link prediction results (of unseen entities) on FB4M-desc.

Model	**Mean rank**			**Hits@10 (%)**		
	1k samples	10k samples	80k samples	1k samples	10k samples	80k samples
MLP	62657	62914	64570	13.2	13.95	14.06
CNN	**50164**	**54033**	**54536**	**14.8**	**14.29**	**14.52**

Table 5: Concept learning examples by our method on FB4M-desc.

Left entity	**Description**	**Hit@10 facts (partial)**	
		Rank	Relation, right entity
Lily Burana	Lily Burana is an American writer whose publications include the memoir I Love a Man in Uniform: A Memoir of Love, War, and Other Battles, the novel Try and Strip ...	0	`/people/person/profession, writer`
		1	`/people/person/profession, author`
		0	`/people/person/gender, female`
		0	`/people/person/nationality, the United States`
Ajeyo	Ajeyo is a 2014 Assamese language drama film directed by Jahnu Barua ... Ajeyo depicts the struggles of an honest, ideal revolutionary youth Gajen Keot who fought against the social evils in rural Assam during the freedom movement in India. The film won the Best Feature Film in Assamese award in the 61st National Film Awards ...	0	`/film/film/country, India`
		7	`/film/film/film_festivals, Mumbai Film Festival`
		7	`/film/film/genre, Drama`
		9	`/film/film/language, Assamese`
4272 Entsuji	4272 Entsuji is a main-belt asteroid discovered on March 12, 1977 by Hiroki Kosai and Kiichiro Hurukawa at Kiso Observatory.	9	`/astronomy/astronomical_discovery/discoverer, Kiichir Furukawa`
		0	`/astronomy/celestial_object/category, Asteroids`
		2	`/astronomy/star_system_body/star_system, Solar System`
		4	`/astronomy/asteroid/member_of_asteroid_group, Asteroid belt`
		0	`/astronomy/orbital_relationship/orbits, Sun`

Finally, we show some examples in Table 5 to illustrate our framework's capability of learning concepts from natural language descriptions. From the first example, we can see that our framework is able to infer <`Lily Burana, has profession, author`> from the sentence "Lily Burana is an American writer." To do this kind of reasoning requires a correct understanding of the original sentence and knowledge that writer and author are synonyms. In the third example, with limited information in the description, the framework hits correct facts almost purely based on its knowledge of astronomy, demonstrating the robustness of our approach.

5 Conclusions and future work

We present a novel perspective on knowledge base embeddings, which enables us to build a framework with concept learning capabilities from large-scale KB based on previous neural embedding models. We evaluate our framework on two newly constructed datasets from Freebase, and the results show that our framework well regularizes the neural embedding model to give superior performance, while has the ability to learn new concepts and use the newly learnt embeddings to deal with semantic tasks (e.g., reasoning).

Future work may include consistently improving performance of learnt concept embeddings on large-scale datasets like FB4M-desc. For applications, we think this framework is very promising in solving problems of unknown entities in KB-powered dialog systems. The dialog system can ask users for description when meeting an unknown entity, which is a natural behavior even for human during conversations.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[3] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[4] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, pages 926–934, 2013.

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, pages 2787–2795, 2013.

[6] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Conference on Artificial Intelligence*, number EPFL-CONF-192344, 2011.

[7] Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.

[8] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[9] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[10] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 809–816, 2011.

[11] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280. ACM, 2012.

[12] Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, volume 3, page 5, 2006.

[13] Kurt Miller, Michael I Jordan, and Thomas L Griffiths. Nonparametric latent feature models for link prediction. In *Advances in neural information processing systems*, pages 1276–1284, 2009.

[14] Jun Zhu. Max-margin nonparametric latent feature models for link prediction. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 719–726, 2012.

[15] Ilya Sutskever, Joshua B Tenenbaum, and Ruslan R Salakhutdinov. Modelling relational data using bayesian clustered tensor factorization. In *Advances in neural information processing systems*, pages 1821–1828, 2009.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[17] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[18] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph and text jointly embedding. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1591–1601, 2014.

[19] James L McClelland and Timothy T Rogers. The parallel distributed processing approach to semantic cognition. *Nature Reviews Neuroscience*, 4(4):310–322, 2003.

[20] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.

[21] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

" "

Fractal grammars which recover from perturbations

Whitney Tabor (whitney.tabor@uconn.edu)
Department of Psychology, University of Connecticut
Storrs, CT 06269-1020 USA

Abstract

Neural symbolic integration may be a natural phenomenon of dynamical systems. Attractors—subsets of a state space to which a dynamical system returns when perturbed—are a broadly relevant dynamical systems phenomenon. The mathematical theory has mainly focused on autonomous dynamical systems (i.e., not driven by an environment) of the form $f : X \to X$ (where $x(t+1) = f(x(t))$ [iterated map] or $\frac{dx}{dt} = f(x)$ [differential equation]), and discovered a rich inventory of attractors, including stable fixed points, limit cycles, and chaos. Here, I focus on the iterated map case and consider certain nonautonomous dynamical systems characterized by a finite set of functions $f_1, f_2, ..., f_k : X \to X$ and a language on alphabet $\Sigma = \{1, \ldots, k\}$ of one-sided infinite strings which applies the functions in particular orders starting from a specified initial state $x(0)$ in X. I extend the definition of attractor by considering cases where the system returns to an invariant proper subset when perturbed in the environment of the language. The news of this paper is that there is a class of nonautonomous dynamical systems that have attractors for mirror recursion languages, a type of language arguably central to natural language syntax.

Keywords: dynamical systems theory, attractors, asymptotic stability, grammar, context free languages, mirror recursion languages; neural-symbolic integration

1 Introduction

This paper approaches neural symbolic integration by interpreting certain dynamical systems, which can be implemented in neural networks, as symbol processors. It uses insights from the classical theory of computation (Chomsky Hierarchy) to explore and categorize the behavior of these models, thus helping to relate them to previous, symbolically oriented work in cognitive science, especially that on natural language syntax. It begins with a brief review of methods for symbol processing using discrete-update recurrent neural networks, making a transition in the process, from a perspective in terms of neural information processing to a perspective in terms of dynamical systems theory. This leads to the question of stability—here, by "stability", I mean the ability of a network processing a complex language to get back on track if something throws it off. The paper proposes a precise definition of stability, suitable to complex language processing, and shows that at least one interesting class of dynamical systems for language processing possesses this property. The paper concludes with some remarks on implications for sentence processing research, dynamical systems research, and the project of neural-symbolic integration.

1.1 Elman Net and Subsequent Work

Elman (1991) found that a discrete update recurrent neural network (the "Simple Recurrent Network") trained on sequences of symbols encoded as indexical bit vectors learned to keep track of English-like center-embedding dependencies, suggesting that the network might be able to model

the phrase-structure foundation that many linguists posit for natural language (Chomsky, 1957; Gazdar, 1981). This work indicates a path to neural-symbolic integration, but only suggestively because the structure of the model's computation was only observed impressionistically. A series of related projects (Bodén & Wiles, 2000; Pollack, 1987; Rodriguez, 2001; Siegelmann & Sontag, 1991; Siegelmann, 1999; Tabor, 2000; Wiles & Elman, 1995) ask how networks of recurrently connected sigmoidal units can precisely process infinite state languages of various kinds. Indeed, the range of possible computations in networks is great, including all of the Chomsky Hierarchy (Siegelmann & Sontag, 1991). These projects all refer to a common principle: the network can use a fractal subset of its state space to implement a stack or tape memory. A fractal is a set that is made up of smaller replicas of itself (Mandelbrot, 1977). A key insight of all these projects is that the spatial recursive structure of a fractal can be used to keep track of the temporal recursive structure of a complex language.

Recurrent neural networks are instances of the type of feedback systems standardly studied in dynamical systems theory (the mathematical theory concerned with systems characterized in terms of how they change). Dynamical systems seem to have a precise relationship to grammars: (a) "symbolic dynamics", a method of treating a dynamical system on a connected space as discrete symbol processor has been very useful in characterizing chaos (e.g., Devaney, 1989), an important dynamical phenomenon; (b) an indexed context free language gives the topology of a key trajectory of the logistic map, a rich and much-studied dynamical system (Crutchfield & Young, 1990); (c) dynamical systems construed as symbol processors in various ways, exhibit a rich range of computational types, including finite languages, finite state languages, context free languages, queue-based languages, Turing Machines, and non-computable languages (Moore, 1998; Siegelmann, 1999; Tabor, 2009). In all the cases involving computable infinite state languages, a fractal subset of the state space supports recursive temporal computation.

Generally, when an iterated map dynamical system corresponds to a particular discrete-update neural network for recursive processing, the state space of the dynamical system corresponds to the recurrently-connected activation space of the network; the control parameters of the dynamical system correspond to the weights of the network; the parameterization of the dynamics associated with a particular symbol corresponds to the activation of a particular unit corresponding to the symbol, which has first-order or second-order connections to the recurrently connected units, thus specifying the update behavior of those units; the branches of the fractal are typically associated with different classes of symbols; the dynamical system may have an associated (finite) partition of the state space which specifies which maps can be applied from which states; correspondingly, the network may have one or more classifier layers outside the recurrent dynamics which map the recurrent state space to next-symbol options. (Bodén & Wiles, 2002; Rodriguez, 2001; Siegelmann, 1999; Tabor, 2000, 2003, 2011, e.g.,)

1.2 An issue: stability

The results just discussed point to the rich computational capability of neural networks and related dynamical systems. However, this computational capability is not very helpful if it is unstable—that is if disturbance of the system through noise or other forces not tied to the computation easily cause the system to stop computing successfully on future inputs. There are at least two senses of stability that one might be concerned with: (i) stability with respect to changes in the state variables—e.g., perturbation of the activation states of the neurons in a neural network, and (ii) stability with respect to changes in the parameter variables—e.g., perturbation of the weights of a neural network. When small parameter perturbations do not change the topology of the dynamics, the system is said to exhibit *structural stability*. Here we focus on state variable stability, noting that structural stability is an additional property of interest which deserves further investigation (see Conclusions). If a system is a stable language processor, then it should be able to recover from forces that temporarily knock it off the track of grammatical processing. Evidence from the sentence processing literature suggests that when people are disturbed in the course of processing a sentence (e.g., by a garden path event, or a confusion due to interference) they exhibit a disturbance which lasts beyond the word that causes the disturbance ("spill-over effects"), but typically only for a few words, with evidence for resolution of the disturbance occurring at the end of the sentence ("sentence wrap-up effects"). This suggests that human minds processing language have "asymptotic stability"—as processing progresses following a disturbance, the system returns to normal processing.

Motivated by these observations, we establish, in the next section, a definition of stability for symbol-driven dynamical automata.

2 Back in Kansas Stability

In classical formal grammar theory, languages are sets of finite-length strings. In the present work, we consider languages to be sets of one-sided infinite length strings. This simplifies the formalism. We can assign each language on the Chomsky Hierarchy a place in this framework by considering, for language L, the set of all one-sided infinite concatenations of strings from L, which we denote L^∞, thinking of this as a model of all the sentences that a speaker hears and/or produces in succession in their lifetime.

Def. An *iterated function system* (Barnsley, [1988]1993; Hutchinson, 1981) is a (finite) set of functions $IFS = \{f_1, \ldots, f_k\}$ that map from a space X to itself. We assume that X is a complete metric space with metric d.

Def. A *one-sided-infinite-string language*, L, is a set of one-sided infinite strings drawn from a finite alphabet, $\Sigma = \{1, \ldots, k\}$. We assume that for every member j of Σ, there is some string of L which contains k.

For $x \in X$, and $S = \sigma_1\sigma_2 \ldots \sigma_N$ a finite string of symbols drawn from Σ, we use the notation $IFS_S(x)$ to denote the function $f_{\sigma_1}(f_{\sigma_2}(\ldots(f_{\sigma_N}(x))\ldots))$.

Def. Consider a point $x(0)$ in X. The *labeling*, $Lab_{x(0)}$, of IFS driven by L is a function from points in X to the power set of Σ, such that $j \in Lab_{x(0)}(x)$ iff there is a finite initial substring, S, of some string in L such that $IFS_S(x(0)) = x$ and the string formed by adding j to the end of S is also an initial substring of some string in L.

Def. $Lab_{x(0)}$ is said to *process a symbol* j at a point x iff $j \in Lab_{x(0)}(x)$ and the system moves from x to $f_j(x)$. A labelling $Lab_{x(0)}$ under IFS is said to *process a string* S iff starting at $x(0)$, every symbol of S is processed in sequence. The *language of* $Lab_{x(0)}$ under IFS is the set of one-sided infinite strings processed by $Lab_{x(0)}$ under IFS. We say that an $IFS - L$ systems is *on* $Lab_{x(0)}$ when it visits a point x if all continuations of the current string under L are processed by $Lab_{x(0)}$.

Def. Consider $x \in X$ and $j \in \Sigma$. The *distance* $d((x, j), Lab_{x(0)})$ from the ordered pair (x, j) to the labeling, $Lab_{x(0)}$, is $inf\{d(x, y)) : y \in X$ and $j \in Lab_{x(0)}(y)\}$.

In other words, assuming a standard (infimum-based) definition of point-set distance, the distance of a point-symbol ordered pair, (x, j), to a labeling is the distance from x to the set of points at which j can come next under the labelling.

Def. A sequence of ordered point-symbol pairs $(x(n), \sigma_n)$ for $n = \{0, 1, 2, \ldots\}$ is said to *converge* to a labeling $Lab_{x(0)}$ if, for every ϵ, there exists N such that $M > N$ implies $d((x(M), \sigma_M), Lab_{x(0)}) < \epsilon$.

Def. Consider an IFS on a complete metric space X with functions $\Sigma = \{1, \ldots, k\}$ and a one-sided infinite string language L on Σ. For initial state, $x(0)$, consider the labeling $Lab_{x(0)}$ induced by L. Consider the sequence of point-symbol pairs $PSP_S = \{(x(n), \sigma_n), n = 0, 1, \ldots\}$ induced by a member, $S = \sigma_1\sigma_2 \ldots$, of L (i.e., $x((n + 1)) = f_{\sigma_n}(x(n))$ for all n). If, when the system is perturbed within a radius δ of $x(n)$ for any $(x(n), \sigma_n) \in PSP_S$ and then driven henceforth by the remaining symbols of S, it converges to $Lab_{x(0)}$, then it is said to exhibit *Back in Kansas Stability from* $x(0)$ *when driven by* S. If there is a δ such that all futures from $x(0)$ converge when perturbed once within δ, then the system is said to exhibit *Back In Kansas stability from* $x(0)$. We say, in such a case, that points of $Lab_{x(0)}$ with nonempty labeling, along with their labels, are an $attractor$ of the IFS-L.

The idea of Back in Kansas Stability is that the system is expected to recover its rhythm, not under the influence of some external signal, but rather simply through exposure to sufficient material from the familiar language. We adopt this approach because, as noted above, studies of sentence processing provide evidence that recovery from disturbance in sentence processing often takes place across multiple words and seems to involve a convergence process.

3 Lack of Back in Kansas Stability in existing fractal models

We next offer some demonstrations that existing fractal computers lack Back in Kansas Stability. As noted above, Moore (1998) describes a one-dimensional dynamical automaton that implements a stack memory for recognizing context free languages. For a stack alphabet of k symbols, Moore's system uses a Cantor-set with $k-1$ gaps between the fractal branches. Pushing and popping are accomplished by

$$\begin{array}{rcl} push_i(x) & = & \alpha x + (1-\alpha)\frac{i}{k} \\ pop_i(x) & = & push_i{}^{-1}(x) \end{array} \tag{1}$$

where $1 \leq i \leq k$ is the symbol being pushed or popped and $0 < \alpha < \frac{1}{2k+1}$ is a constant, and $x(0) = 1$. This system has the property that grammatical processing is restricted to the interval [0, 1], but an erroneous stack operation (e.g., $pop_j(push_i(x))$ where $j \neq i$) results in $|x| > 1$ (Moore's system uses this property to detect ungrammatical transitions). Because push and pop are inverses across the entire state space and grammatical processing (to empty stack) implements a corresponding pop for every push, the displacement created by any error persists no matter how many grammatical sentences follow the error (hence the system does not have Back in Kansas Stability). For example, if one implements S $\rightarrow$ ϵ, S $\rightarrow$ 1 S 2, S $\rightarrow$ 3 S 4 with a two-symbol stack alphabet, choosing $\alpha = \frac{1}{7} < \frac{1}{2 \cdot 2+1}$, then the once-erroneous sequence $1\underline{3}121212\ldots$ results in an endless cycle between -2 and 0.1429—that is, the system never returns to $Lab_{x(0)}$, and, the magnitude of the distance between the state and $Lab_{x(0)}$ endlessly visits a positive constant.

Similarly, Tabor (2000) describes a fractal grammar which processes the non-finite-state context free language, S $\rightarrow$ A B C D, A $\rightarrow$ a (S), B $\rightarrow$ b (S), C $\rightarrow$ c (S), D $\rightarrow$ d (S). The model's state space is R^2 with initial state $(1/2, 1/2)$ and the IFS is given by

$$\begin{array}{rcl} f_a(\mathbf{x}) & = & \frac{\mathbf{x}}{2} + \binom{1/2}{0} \\ f_b(\mathbf{x}) & = & \mathbf{x} - \binom{1/2}{0} \\ f_c(\mathbf{x}) & = & \mathbf{x} + \binom{0}{1/2} \\ f_d(\mathbf{x}) & = & 2\left(\mathbf{x} - \binom{0}{1/2}\right) \end{array} \tag{2}$$

The points with nonempty labels in $Lab_{x(0)}$ form a "Sierpinski Gasket", a kind of two-dimensional Cantor Set. In this system, as in Moore's, all the functions are affine, and the map $f_d \circ f_c \circ f_b \circ f_a(x)$ is identity across the whole state space. Since the language always completes every $abcd$ sequence that is begun, this system, like Moore's, repeatedly revisits the magnitude of any displacement from $Lab_{x(0)}$, independently of the value of $x(0)$.

These systems have a form of stability generally recognized in dynamical systems theory—there is a bound on how far the system travels away from the invariant set of interest—but they lack the asymptotic stability that seems to characterize human behavior. The lack of asymptotic stability is related to the fact that transitions from empty stack to empty stack implement identity across the state space. But identity is only required for grammatical processing, which, in these cases, is restricted to a proper subset of the state space. It will not disturb grammatical processing to modify the map in locations away from this manifold. In the next section, we show, for an important class of languages, a simple way of modifying the off-manifold maps, so that the system gets back onto the manifold when it has been knocked off, provided it receives exposure to enough grammatical material following perturbation.

4 The Simplest Context Free Language, 1^n2^n

We begin with the simplest context free language, 1^n2^n, and then generalize to all mirror recursion languages, $L_{mirror\text{-}\kappa}$, of the form, $S \rightarrow \epsilon, S \rightarrow \sigma_{11}\ S\ \sigma_{12}, S \rightarrow \sigma_{21}\ S\ \sigma_{22}, \ldots, S \rightarrow \sigma_{\kappa 1}\ S\ \sigma_{\kappa 2}$. Mirror recursion seems to capture the gist of center-embedding structures in languages around the

world. Corballis (2007) argues that mirror recursion is the crucial recursive behavior that distinguishes human languages from animal languages. Although it is true that no human language exhibits center embedding easily beyond one center-embedded clause, the degradation, as one tests successively deeper levels of embedding appears to be graceful (Lewis, 1996), and Christiansen & Chater (1999) argue that a Simple Recurrent Network captures this quality of the human language case well. Moreover, Rodriguez (2001) and Tabor et al. (2013) provide evidence that Simple Recurrent Networks trained on center-embedded material are approximating fractal grammars. These observations motivate focusing on the mirror recursion case in the effort to understand how fractal grammars can be stable.

Let $L_{1^n2^n}$ be the one-sided-infinite-string language, $\{1^n2^n\}^\infty$. Let $X = [-2, 2]$ and let $IFS_{1\text{-}d}$ be given by

$$\begin{array}{rcrcrcl} f_1(x) & = & \frac{x}{2} + 1 & & & & \\ f_2(x) & = & 2x + 2 & & & x & < & -1 \\ & & 0 & -1 & \leq & x & < & 1 \\ & & -2x + 2 & 1 & \leq & x & & \end{array} \tag{3}$$

Note that the 2-map is not affine but approximately quadratic. Figure 1 shows the IFS along with Lab_0 and illustrates recovery from a perturbation. In fact, this system always recovers from perturbation.

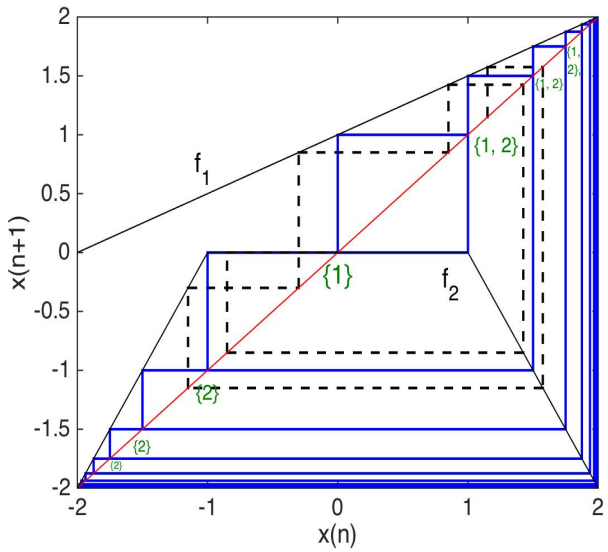


Figure 1: Cobweb diagram of IFS_{1-d} driven by $L_{1^n2^n}$ from $x(0) = 0$. The numbers in curly brackets show some of the nonempty state labels. All points in the complement of $A = \{\ldots, -\frac{7}{4}, -\frac{3}{2}, -1, 0, 1, \frac{3}{2}, \frac{7}{4}, \ldots\}$ have empty label. The dotted line shows a perturbed trajectory which recovers. After processing a single 1 (x(1) = 1), the IFS was perturbed to 1.15. Subsequently it encountered the completion of the perturbed sentence ("... 1 2 2"), followed by a single complete sentence ("1 1 2 2"), by which point it was at 0 and back on the attractor.

Thm 1. The system $IFS_{1\text{-}d}$-$L_{1^n2^n}$ processes $L_{1^n2^n}$ from $x(0) = 0$ and is Back in Kansas stable from that point.

The proof (see Appendix 1) first demonstrates that the language $L_{1^n2^n}$ is processed by $IFS_{1\text{-}d}$ from $x(0) = 0$. Then it shows that, starting from any point in the state space, if the system receives the tail of any string from $\{1^n2^n\}^\infty$, it will be back on Lab_0 within two sentences ("finite-time" convergence). This is a much stronger outcome than is required for Back in Kansas stability. The rapid convergence stems from fact that the horizontal section of the 2-map resets the system when it lands between -1 and 1. We have adopted a broad definition here, allowing also non-finite-time convergence, because simulation experiments which we will not discuss further here suggest that

similar convergence behavior occurs in systems with a single maximum, rather than a plateau, except that the convergence takes infinite time.

5 General Mirror Recursion

For 2κ symbols, let $x(0)$ be the origin in $R^{\kappa+1}$. Define $IFS_{(\kappa+1)\text{-}d}$ for $\kappa = 1, 2, \ldots, i \in \{1, \ldots, \kappa\}$ by

$$-\infty < x_1 < -1 \qquad\qquad -1 \le x_1 < 1 \qquad\qquad 1 \le x_1 < \infty$$

$$f_i(\mathbf{x}) = \begin{pmatrix} x_1/2+1 \\ x_2/2 \\ \ldots \\ x_i/2 \\ x_{i+1}/2-1 \\ x_{i+2}/2 \\ \ldots \\ x_(\kappa+1)/2 \end{pmatrix} \qquad f_i(\mathbf{x}) = \begin{pmatrix} x_1/2+1 \\ 0 \\ \ldots \\ 0 \\ -1 \\ 0 \\ \ldots \\ 0 \end{pmatrix} \qquad f_i(\mathbf{x}) = \begin{pmatrix} x_1/2+1 \\ x_2/2 \\ \ldots \\ x_i/2 \\ x_{i+1}/2-1 \\ x_{i+2}/2 \\ \ldots \\ x_(\kappa+1)/2 \end{pmatrix}$$

$$f_{2i}(\mathbf{x}) = \begin{pmatrix} 2x_1+2 \\ 2x_2 \\ \ldots \\ 2x_i \\ 2x_{i+1}+2 \\ 2x_{i+2} \\ \ldots \\ 2x_(\kappa+1) \end{pmatrix} \qquad f_{2i}(\mathbf{x}) = \mathbf{0} \qquad f_{2i}(\mathbf{x}) = \begin{pmatrix} -2x_1+2 \\ 2x_2 \\ \ldots \\ 2x_i \\ 2x_{i+1}+2 \\ 2x_{i+2} \\ \ldots \\ 2x_(\kappa+1) \end{pmatrix}$$

Here, the -1 in the state change from the middle region for $f_i(x)$ is on dimension $i+1$.

Thm 2. Each system in the class $\{IFS_{(\kappa+1)\text{-}d}\text{-}L_{Mirror\text{-}\kappa}\}$ for $\kappa \in \{1, 2, \ldots\}$ is Back in Kansas stable from $x(0) = \mathcal{O}$, the origin in $R^{\kappa+1}$.

Appendix 2 sketches the proof of Theorem 2.

6 Conclusions

This paper has defined Back in Kansas Stability for nonautonomous dynamical systems, consisting of functions $f_1, \ldots, f_k : X \to X$ on a complete metric space, driven by a language L on $\Sigma = \{1, 2, \ldots, k\}$. The definition was motivated by the fact that people who undergo a disturbance when processing complex natural language structures seem to recover naturally during the course of processing additional words. This idea makes a prediction that many other parsing models—those that tie parsing strictly to the information content of received words—do not make: recovery from a garden path might be helped by words following the disambiguating word even if these words provide no additional structural information; some evidence in support of this idea comes from certain types of length effects in sentence processing where long disambiguating regions are easier than short ones—see Tabor & Hutchins (2004).

Another motivation came from the fact that stability is an important organizing principle of dynamical systems broadly, so when working with neural network dynamical systems, it is desirable to characterize their stability. The mirror recursion finding prompts a more specific observation: in the case of iterated maps, the well-known attractors of autonomous dynamical systems are distinguished by their cardinalities—a fixed point is a single point, a limit cycle contains a finite number of points, a chaotic attractor has uncountably many points. The case of mirror recursion with Back in Kansas Stability fills in this picture by identifying countably infinite attractive sets. It may be informative to ask what conditions support countably infinite attractors and why they are not prominent in the study of autonomous dynamical systems.

6.1 Future Work

We noted above that a dynamical system is considered structurally stable at a point in its parameter space if small changes in the parameter values do not affect the topology of the system. Structural stability seems, in one way, desirable for neural networks, when one is interested in learning: the generally widely successful approach of gradient descent learning is likely to do badly in a context where small parameter changes radically alter the system structure. Results reported in Tabor (2009) for fractal grammars suggest, possibly unfortunately, that these types of systems are not structurally stable in the vicinity of points where they model complex languages. Nevertheless, an interesting alternative perspective may be worth considering: human languages seem to change structurally in a continuous way (see discussion in Tabor et al. (2013)), at least when they change historically, and possibly also when they are learned. It may be useful to invoke, in place of topological equivalence, a notion of structural continuity—two dynamical systems are considered structurally proximal if differences in their topology take a long time to detect. If there is structural continuity, gradient based learning may still be feasible.

What light does this work shed on the challenge of neural symbolic integration? Broadly, it suggests studying neural symbolic integration by studying dynamical systems more generally. Specifically, the results on Back in Kansas Stable point to the fact that not all dynamical computers are stable in this sense; it also raises the question whether all computations can be stably implemented. Stability is very likely a property of human language systems since they have to tolerate a great deal of informational and physical buffeting. It may therefore be useful for the field of neural symbolic integration to identify, both from a dynamics point of view and a language point of view, what systems can be stable.

7 Appendix 1: Proof of Thm. 1

We first show that Lab_0 under IFS_{1-d} processes $L_{1^n2^n}$ (Part A). Then we show that the system is Back in Kansas Stable from $x(0) = 0$ (Part B).

Part A

By definition, every string of L is processed by Lab_0 under IFS_{1-d}. Regarding strings of Lab_0 under IFS_{1-d}, note that 1's only ever occur if $x \geq 0$. If the system starts at 0 and experiences a sequence of the form 1^j2^i where $0 < i < j$, then $x < 0$. Therefore, substrings of the form 21^j2^i1, $0 \leq i < j$ are not processed by Lab_0. If the system starts at 0 and experiences a sequence of the form 1^j2^j where $0 \leq j$, then the system is at $x = 0$, where a 2 never occurs (because, under L, balanced 1s and 2s are always followed by a 1 and 0 is never reached except via balanced 1s and 2s). Therefore substrings of the form 21^j2^i1 where $i > j > 0$ never occur. Therefore the strings processed by Lab_0 under IFS_{1-d} are all and only the strings of $L_{1^n2^n}$.

Part B.

We wish to show that under perturbation up to radius r followed by grammatical continuation to infinity, IFS returns to $Lab_{x(0)}$ in the limit. In fact, in this case, under *any* perturbation, followed by at most one complete string of the form IFS reinhabits $Lab_{x(0)}$.

First, it is useful to define some terminology. We refer to the string in which the perturbation occurs as the *perturbation string*. We refer to the interval $h = [-1, 1]$, where the function b has a fixed value, as the *flat interval*. Note that when h is in the flat interval, $2^k(h) = 0$ for any $k \geq 0$. Note also that if the system is in a state $h_p < 0$ and it processes a string of the form 1^n2^n, then it will arrive at 0 when it processes the last 2 (this follows from the facts (i) that, at every symbol, $1^n2^n(h_p)$ is sandwiched between $1^n2^n(h_0)$ and 0 and (ii) $1^n2^n(h_0) = 0$.) Whenever the state of the system is thus sandwiched, we say that the system is *ahead* with respect to grammatical processing.

Turning now to the proof of B, there are two cases:

(i) The perturbation occurs at some time before the last 1 in 1^j2^j for some $j \geq 0$.

Within this case, it is useful to consider two subcases:

(i-1) The perturbation decreases h. In this case, during processing of the perturbation string, the system is ahead, so it arrives at 0 by the last 2 and is therefore on $Lab_{x(0)}$.

(i-2) The perturbation increases h. In this case, the system is in [-2, -1] at the end of the string. Therefore, the system is ahead during the processing of the subsequent string. Consequently, it reaches $Lab_{x(0)}$ by the end of the post-perturbation string.

(ii) The perturbation occurs after the last 1 and before the last 2 in 1^j2^j for some $j \geq 0$.

Again, we consider two cases:

(ii-1) The perturbation decreases h. In this case, the system is still in [-2, -1] at the end of the string and the future states follow the pattern of (i-2) above.

(ii-2) The perturbation increases h. If, after the increase, $h < -1$, then the remaining 2's bring the system to the flat interval and the case is like (i-1) above. If, after the increase, $-1 \leq h \leq 1$, then the remaining 2's keep the system at 0 and the system is on the attractor at the start of the next string. If, after the increase, $h > 1$, then the system is in [-2, -1] at the end of the string and the future follows (i-2) above.

Thus, in all cases, the system returns to $Lab_{x(0)}$ at least by the end of the post-perturbation string. ■

8 Appendix 2: Sketch of Proof of Thm. 2

The proof of this theorem builds on the proof of Theorem 1. The system behaves on the first dimension as if the string were 1^n2^n with all the push symbols invoking 1 and all the pop symbols invoking 2, following the same dynamics as in the 1-dimensional case above. Whenever the system is in the flat region of the pop maps and in the second half of a sentence, dimension 1 becomes 0 and stays there until the end of the sentence. Then, the start of the new sentence resets all the dimensions to the appropriate values so the effect of perturbation on any dimension is removed. By Thm. 1, this will always happen within two sentences of a perturbation.

Furthermore, when the system, initialized to $x(0) = \mathbf{0}$, is unperturbed, $f_{2i}(\mathbf{x}) \circ f_i(\mathbf{x})$ implements identity for visited points, $\mathbf{x}$ and every stack state corresponds to a distinct point in X because the push maps on X form a just-touching fractal (Barnsley, [1988]1993) so the unperturbed system in $k+1$ dimensions processes $L_{Mirror-k}$. ■

Acknowledgments

Thanks to Harry Dankowicz and Garrett Smith for comments on an earlier version of this work. We gratefully acknowledge support from NSF PAC grant 1059662 and NSF INSPIRE 1246920.

References

Barnsley, M. ([1988]1993). *Fractals everywhere, 2nd ed.* Boston: Academic Press.

Bodén, M., & Wiles, J. (2000). Context-free and context sensitive dynamics in recurrent neural networks. *Connection Science*, *12*(3), 197–210.

Bodén, M., & Wiles, J. (2002). On learning context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, *13*(2), 491–493.

Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton and Co.

Christiansen, M. H., & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, *23*, 157–205.

Corballis, M. C. (2007). Recursion, language, and starlings. *Cognitive Science*, *31*, 697–704.

Crutchfield, J. P., & Young, K. (1990). Computation at the onset of chaos. In W. H. Zurek (Ed.), *Complexity, entropy, and the physics of information* (pp. 223–70). Redwood City, California: Addison-Wesley.

Devaney, R. L. (1989). *An introduction to chaotic dynamical systems, 2nd ed.* Redwood City, CA: Addison-Wesley.

Elman, J. L. (1991). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, *7*, 195–225.

Gazdar, G. (1981). On syntactic categories. *Philosophical Transactions (Series B) of the Royal Society*, *295*, 267-83.

Hutchinson, J. E. (1981). Fractals and self similarity. *Indiana University Mathematics Journal*, *30*(5), 713-747.

Lewis, R. (1996). Interference in short-term memory: The magical number two (or three) in sentence processing. *Journal of Psycholinguistic Research*, *25*(1).

Mandelbrot, B. (1977). *Fractals, form, chance, and dimension*. San Francisco: Freeman.

Moore, C. (1998). Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, *201*, 99–136.

Pollack, J. (1987). *On connectionist models of natural language processing.* (Unpublished doctoral dissertation, University of Illinois.)

Rodriguez, P. (2001). Simple recurrent networks learn context-free and context-sensitive languages by counting. *Neural Computation*, *13*(9), 2093-2118.

Siegelmann, H. T. (1999). *Neural networks and analog computation: Beyond the turing limit*. Boston: Birkhäuser.

Siegelmann, H. T., & Sontag, E. D. (1991). Turing computability with neural nets. *Applied Mathematics Letters*, *4*(6), 77-80.

Tabor, W. (2000). Fractal encoding of context-free grammars in connectionist networks. *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, *17*(1), 41-56.

Tabor, W. (2002). The value of symbolic computation. *Ecological Psychology*, *14*(1/2), 21–52.

Tabor, W. (2003). Learning exponential state growth languages by hill climbing. *IEEE Transactions on Neural Networks*, *14*(2), 444-446.

Tabor, W. (2009). A dynamical systems perspective on the relationship between symbolic and non-symbolic computation. *Cognitive Neurodynamics*, *3*(4), 415-427.

Tabor, W. (2011). Recursion and recursion-like structure in ensembles of neural elements. In H. Sayama, A. Minai, D. Braha, & Y. Bar-Yam (Eds.), *Unifying themes in complex systems. proceedings of the viii international conference on complex systems* (p. 1494-1508). Cambridge, MA: New England Complex Systems Institute. (http//necsi.edu/events/iccs2011/proceedings.html)

Tabor, W., Cho, P. W., & Szkudlarek, E. (2013). Fractal analysis illuminates the form of connectionist structural gradualness. *Topics in Cognitive Science*, *5*, 634–667.

Tabor, W., & Hutchins, S. (2004). Evidence for self-organized sentence processing: Digging in effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *30*(2), 431-450.

Wiles, J., & Elman, J. (1995). Landscapes in recurrent networks. In J. D. Moore & J. F. Lehman (Eds.), *Proceedings of the 17th annual cognitive science conference.* Lawrence Erlbaum Associates.

Analogy Making and Logical Inference on Images using Cellular Automata based Hyperdimensional Computing

Ozgur Yilmaz*
Department of Computer Engineering
Turgut Ozal University
Ankara Turkey
ozyilmaz@turgutozal.edu.tr

Abstract

In this paper, we introduce a framework of reservoir computing that is capable of both connectionist machine intelligence and symbolic computation. Cellular automaton is used as the reservoir of dynamical systems. A cellular automaton is a very sparsely connected network with logical nodes and nonlinear/logical connection functions, hence the proposed system corresponds to a binary valued and nonlinear neuro-symbolic architecture. Input is randomly projected onto the initial conditions of automaton cells and nonlinear computation is performed on the input via application of a rule in the automaton for a period of time. The evolution of the automaton creates a space-time volume of the automaton state space, and it is used as the reservoir. In addition to being used as the feature representation for pattern recognition, binary reservoir vectors can be combined using Boolean operations as in hyperdimensional computing, paving a direct way symbolic processing. To demonstrate the capability of the proposed system, we make analogies directly on image data by asking 'What is the Automobile of Air'?, and make logical inference using rules by asking 'Which object is the largest?'

1 Introduction

We have introduced a holistic intelligence framework capable of simultaneous pattern recognition Yilmaz (2015b) and symbolic computation Yilmaz (2015a,c). Cellular automaton is the main computational block that holds a distributed representation of high order input attribute statistics as in neural architectures (Figure 1 b). The proposed architecture is a cross fertilization of cellular automata, reservoir computing and hyperdimensional computing frameworks (Figure 1 a). The cellular automata (CA) computation can be viewed as a feedforward network with logical nodes and connections, as shown in Figure 1 c. In this paper, we analyze the symbolic computation capability of the system on making analogies and rule based logical inferences, directly on the image data. The results show that (Figure 2), binary vector representation of images derived through CA evolution provide very precise analogies and accurate rule based inference, even though very small number of examples are provided. In the next subsection we review cellular automata [1], then introduce relevant neuro-symbolic computation studies. Finally, we state our contribution.

*Web: ozguryilmazresearch.net

[1]The literature review is narrowed down in this paper due to space considerations. Please visit our published papers to get a wider view of our architecture among existing reservoir and hyperdimensional computing approaches.

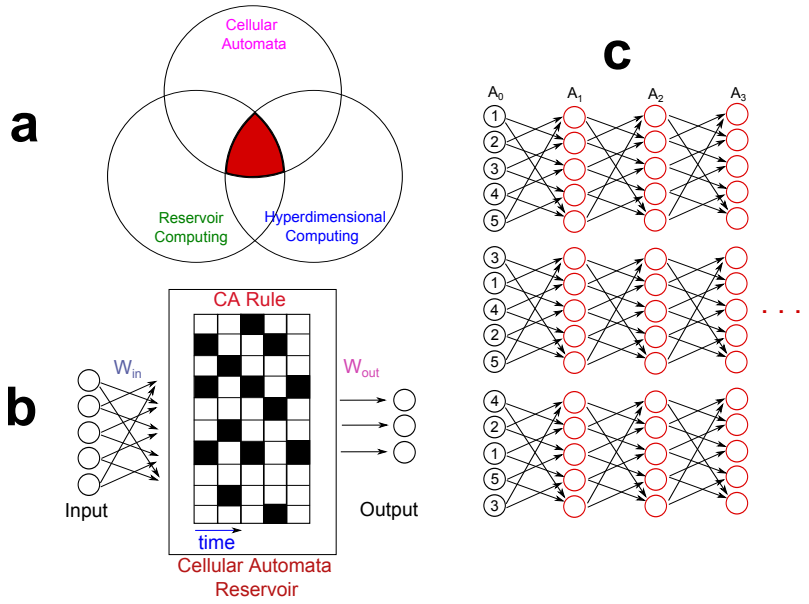


Figure 1: **a.** Our work is a cross fertilization of cellular automata, reservoir computing and hyperdimensional computing frameworks. **b.** In cellular automata reservoir, data is projected onto cellular automaton instead of a real valued node as in classical neural networks. **c.** The network formed by the cellular automaton feature space of rule 90. It can be viewed as a time unrolled feedforward network, however the connections are not all-to-all between layers due to partitioning of different permutations, given as separate rows. And the connections are not algebraic but logical, i.e. XOR operation. See Yilmaz (2015b) for details.

1.1 Cellular Automata

Cellular automaton is a discrete computational model consisting of a regular grid of cells, each in one of a finite number of states (Figure 1 c). The state of an individual cell evolves in time according to a fixed rule, depending on the current state and the state of its neighbors. The information presented as the initial states of a grid of cells is processed in the state transitions of cellular automaton and computation is typically very local. Essentially, a cellular automaton is a very sparsely connected network with logical nodes and nonlinear/logical connection functions (Figure 1 c). Some of the cellular automata rules are proven to be computationally universal, capable of simulating a Turing machine (Cook, 2004).

The rules of cellular automata are classified according to their behavior: attractor, oscillating, chaotic, and edge of chaos (Wolfram, 2002). Turing complete rules are generally associated with the last class (rule 110, Conway game of life). Lyapunov exponent of a cellular automaton can be computed and it is shown to be a good indicator of the computational power of the automata (Baetens & De Baets, 2010). A spectrum of Lyapunov exponent values can be achieved using different cellular automata rules. Therefore, a dynamical system with specific memory capacity (i.e. Lyapunov exponent value) can be constructed by using a corresponding cellular automaton. The time evolution of the cellular automata has very rich computational representation Mitchell et al. (1996), especially for the edge of chaos dynamics. The proposed algorithm in this paper exploits the entire time evolution of the CA and uses the states as the reservoir LukošEvičIus & Jaeger (2009); Maass et al. (2002) of nonlinear computation.

1.2 Symbolic Computation on Neural Representations

Uniting the expressive power of mathematical logic and pattern recognition capability of distributed representations (eg. neural networks) has been an open question for decades although several successful theories have been proposed (Garcez et al., 2012; Bader et al., 2008; Marcus, 2003; Miikkulainen et al., 2006; Besold et al., 2014; Pollack, 1990). The difficulty arises due to the very

different mathematical nature of logical reasoning and dynamical systems theory. Along with many other researchers, we conjecture that combining connectionist and symbolic processing requires commonalizing the representation of data and knowledge.

Along the same vein, (Kanerva, 2009) introduced hyperdimensional computing that utilizes high-dimensional random binary vectors for representing objects, predicates and rules for symbolic manipulation and inference. The general family of the methods is called 'reduced representations' or 'vector symbolic architectures', and detailed introductions can be found in (Plate, 2003; Levy & Gayler, 2008). In this approach, high dimensionality and randomness enable binding and grouping operations that are essential for one shot learning, analogy-making, hierarchical concept building and rule based logical inference. Most recently (Gallant & Okaywe, 2013) introduced random matrices to this context and extended the binding and quoting operations. The two basic mathematical tools of reduced representations are vector addition and XOR.

In this paper, we borrow these tools of hyperdimensional computing framework, and build a semantically more meaningful representation by removing the randomness and replacing it with the cellular automata computation. This provides not only a more expressive symbolic computation architecture, but also enables pattern recognition capabilities otherwise not possible with random vectors.

1.3 Contributions

We provide a low computational complexity method Yilmaz (2015b) for recurrent computation using cellular automata based hyperdimensional computing. It is shown that the framework has a great potential for symbolic processing such that the cellular automata feature space can directly be combined by Boolean operations as in hyperdimensional computing, hence they can represent concepts and form a hierarchy of semantic interpretations. We demonstrate this capability by making analogies directly on images and infer relationships using logical rules. In the next section, we give the details of the algorithm, and then provide results experiments that demonstrate our contributions.

2 Cellular Automata Feature Expansion

In our reservoir computing method, data are passed on a cellular automaton instead of an echo state network and the nonlinear dynamics of cellular automaton provide the necessary projection of the input data onto an expressive and discriminative space. Compared to classical neuron-based reservoir computing, the reservoir design is trivial: cellular automaton rule selection. Utilization of edge of chaos automaton rules ensures Turing-complete computation in the reservoir, which is not guaranteed in classical reservoir computing approaches.

The reservoir computing system receives the input data. First, the encoding stage translates the input into the initial states of a 1D elementary cellular automaton. For binary input data, each feature dimension can randomly be mapped onto the cells of the cellular automaton. For this type of mapping, the size of the CA should follow the input data's feature dimension. After encoding, suppose that the cellular automaton is initialized with the vector ${A_0}^{P_1}$, in which P_1 corresponds to a random permutation of raw input data. Then, cellular automata evolution is computed using a prespecified rule, Z (1D elementary CA rule), for a fixed period of iterations (I):

$$
\begin{aligned}
{A_1}^{P_1} &= Z({A_0}^{P_1}),\\
{A_2}^{P_1} &= Z({A_1}^{P_1}),\\
&\dots\\
{A_I}^{P_1} &= Z({A_{I-1}}^{P_1}).
\end{aligned}
$$

The evolution of the cellular automaton is recorded such that, at each time step a snapshot of the whole states in the cellular automaton is vectorized and concatenated. Therfore, we concatenate the evolution of cellular automata to obtain a reservoir for a single permutation:

$$A^{P_1} = [{A_0}^{P_1}; {A_1}^{P_1}; {A_2}^{P_1}; \dots {A_I}^{P_1}]$$

It is experimentally observed that multiple random permutation mappings significantly improve accuracy. There are R number of different random mappings, i.e., separate CA reservoirs, and they

are combined into a large reservoir feature vector:

$$\mathbf{A^R} = [A^{P_1}; A^{P_2}; A^{P_3}; ...A^{P_R}].$$

The computation in CA takes place when cell activities due to nonzero initial values (i.e., input) mix and interact. Both prolonged evolution duration (large I) and existence of different random mappings (large R) increase the probability of long-range interactions, hence improve computational power and enhance representation.

3 Symbolic Processing and Non-Random Hyperdimensional Computing

Hyperdimensional computing uses random very large-sized binary vectors to represent objects, concepts, and predicates. Then, appropriate binding and grouping operations are used to manipulate the vectors for hierarchical concept building, analogy-making, learning from a single example, etc., that are hallmarks of symbolic computation. The large size of the vector provides a vast space of random vectors, two of which are always nearly orthogonal. Yet, the code is robust against a distortion in the vector due to noise or imperfection in storage because after distortion it will still stay closer to the original vector than the other random vectors.

The grouping operation is normalized vector summation, and it enables forming sets of objects/concepts. Suppose we want to group two binary vectors, V_1 and V_2. We compute their element-wise sums, and the resultant vector contains 0, 1 and 2 entries. We normalize the vector by accepting the 0 entries as they are, transforming 2 entries into 1. Note that, these are consistent within the two initial vectors. Then, the inconsistent entries are randomly decided: 1 entries as transformed into 0 or 1. Many vectors can be combined iteratively or in a batch to form a grouped representation of the bundle. The resultant vector is similar to all the elements of the vector due to the fact that consistent entries are untouched. The elements of the set can be recovered from the reduced representation by probing with the closest item in the memory, and consecutive subtraction. Grouping is essential for defining 'a part of', 'contains' relationships. $+$ symbol will be used for normalized summation in the following arguments.

There are two binding operations: bitwise XOR (circled plus symbol, $\oplus$) and permutation [2]. Binding operation maps (randomizes) the vector to a completely different space, while preserving the distances between two vectors. As stated in Kanerva (2009), "...when a set of points is mapped by multiplying with the same vector, the distances are maintained, it is like moving a constellation of points bodily into a different (and indifferent) part of the space while maintaining the relations (distances) between them. Such mappings could play a role in high-level cognitive functions such as analogy and the grammatical use of language where the relations between objects is more important than the objects themselves.

A few representative examples to demonstrate the expressive power of hyperdimensional computing:

1. We can represent pairs of objects via multiplication. $O_{A,B} = A \oplus B$ where A and B are two object vectors.

2. A triplet is a relationship between two objects, defined by a predicate. This can similarly be formed by $T_{A,P,B} = A \oplus P \oplus B$. These types of triplet relationships are very successfully utilized for information extraction in large knowledge bases Dong et al. (2014).

3. A composite object can be built by binding with attribute representation and summation. For a composite object C,

$$C = X \oplus A_1 + Y \oplus A_2 + Z \oplus A_3,$$

where A_1, A_2 and A_3 are vectors for attributes and X, Y and Z are the values of the attributes for a specific composite object.

4. A value of an attribute for a composite object can be substituted by multiplication. Suppose we have assignment $X \oplus A_1$, then we can substitute A_1 with B_1 by, $(X \oplus A_1) \oplus (A_1 \oplus B_1) = X \oplus B_1$. It is equivalent to say that A_1 and B_1 are analogous. This property is essential for analogy making.

[2] Please see Kanerva (2009) for the details of permutation operation, as a way of doing multiplication.

5. We can define rules of inference by binding and summation operations. Suppose we have a rule stating that "If x is the mother of y and y is the father of z, then x is the grandmother of z" [3]. Define atomic relationships:

$$M_{xy} = M_1 \oplus X + M_2 \oplus Y,$$
$$F_{yz} = F_1 \oplus Y + M_2 \oplus Z,$$
$$G_{xz} = G_1 \oplus X + G_2 \oplus Z,$$

then the rule is,

$$R_{xyz} = G_{xz} \oplus (M_{xy} + F_{yz}).$$

Given the knowledge base, "Anna is the mother of Bill" and "Bill is the father of Cid", we can infer grandmother relationship by applying the rule R_{xyz}:

$$M_{ab} = M_1 \oplus A + M_2 \oplus B,$$
$$F_{bc} = F_1 \oplus B + M_2 \oplus C,$$
$$G^{'}_{ac} = R_{xyz} \oplus (M_{ab} + F_{bc}),$$

where vector $G^{'}_{ac}$ is expected to be very similar to G_{ac}, which says "Anna is the grandmother of Cid". Please note that the if-then rules represented by hyperdimensional computing can only be if-and-only-if logical statements because operations used to represent the rules are symmetric.

Without losing the expressive power of classical hyperdimensional computing, we are introducing cellular automata to the framework. In our approach, we use binary cellular automata reservoir vector as the representation of objects and predicates instead of random vectors to be used for symbolic computation. There are two major advantages of this approach over random binary vector generation:

1. Reservoir vector enables connectionist pattern recognition and statistical machine learning (as demonstrated in Yilmaz (2015b)) while random vectors are mainly tailored for symbolic computation.

2. The composition and modification of objects can be achieved in a semantically more meaningful way. The semantic similarity of the two data instances can be preserved in the reservoir hyperdimensional vector representation, but there is no straightforward mechanism for this in the classical hyperdimensional computing framework.

4 Experiments on Analogy Making

In order to demonstrate the power of enabled logical operation, we will use analogy making. Analogy making is crucial for generalization of what is already learned. We tested the capability of our symbolic system using images. The example given here follows "What is the Dollar of Mexico?" in Kanerva (2009). However in the original example, sensory data (i.e. image) is not considered because there is no straightforward way to introduce sensory data into the hyperdimensional computing framework. The benefit of using non-random binary vectors is obvious in this context.

We have previously shown that binarization of the hidden layer activities of a feedforward network is not very detrimental for classification purposes Yilmaz et al. (2015). For an image, the binary representation of the first hidden layer activities holds an indicator for the existence of Gabor like corner and edge features. In order to test the symbolic computation performance of CA features on binarized hidden layer activities, we use CIFAR 10 dataset Krizhevsky & Hinton (2009). We used the first 500 training/test images and obtained single layer hidden neuron representation using the algorithm in Coates et al. (2011) (200 number of different receptive fields, receptive fields size of 6 pixels). The neural activities are binarized according to a threshold and, on average, 22 percent of the neurons fired with the selected threshold. After binarization of neural activities, CA features can be computed on the binary representation as explained in section 2. We formed a separate concept vector for each class (total 10 classes, 50 examples for each class) using binary neural representation of CIFAR training data and vector addition defined in Snaider (2012). These are the basis class concepts extracted from the visual database.

[3] The example is adapted from Kanerva (2009).

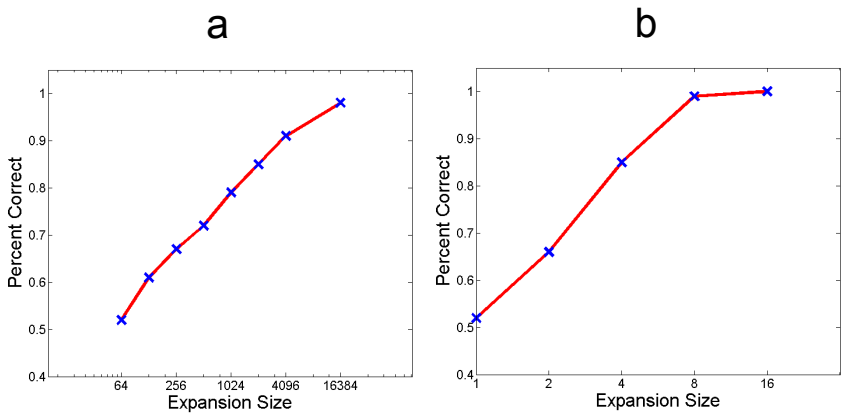


Figure 2: **a.** Analogy making experiment results. The feature expansion (defined as the product $R \times I$) due to cellular automata evolution is given in the x axis (log scale) and the percent correct is given in the y axis. **b.** Rule based logical inference experiment results.

We formed two new concepts called *Land* and *Air*:

$$Land = Animal \oplus Horse + Vehicle \oplus Automobile,$$
$$Air = Animal \oplus Bird + Vehicle \oplus Airplane.$$

In these two concepts, CA features of Horse and Bird images are used to bind with the Animal filler, and CA features of Automobile and Airplane images are used to bind with the Vehicle filler [4]. Animal and Vehicle fields are represented by two random vectors [5], those with the same size as the CA features. Multiplication is performed by xor ($\oplus$) operation and vector summation is again identical to Snaider (2012). The products, Land and Air are also CA feature vectors, and they represent the merged concept of observed animals and vehicles in Land and Air respectively. We can ask the analogical question "What is the Automobile of Air?", AoA in short. The answer can simply be given by this equality (inference):

$$AoA = Automobile \oplus Land \oplus Air.$$

AoA is a CA feature vector and expected to be very similar to Airplane concept vector. We tested the analogical accuracy using unseen Automobile test images (50 in total), computing their CA feature vectors followed by AoA inference, then finding the closest concept class vector to AoA vector (max inner product). It is expected to be the Airplane class. The result of this experiment is given in Figure 2 a [6] for various R and I combinations. The multiplication of the two defines the amount of feature expansion due to cellular automata state space. The analogy on CA features is 98 percent accurate (for both R and I equals 128), whereas if the binary hidden layer activity is used instead of CA features (corresponds to R and I equal to 1), the analogy is only 21 percent accurate. This result clearly demonstrates the benefit of CA feature expansion for symbolic computation.

The devised analogy implicitly assumes that Automobile concept is already encoded in the concept of Land. What if we ask "What is the Truck of Air?"? Even though Truck images are not used in building the Land concept, due to the similarity of Truck and Automobile concepts, we might still get good analogies. The results on this second order analogy is contrasted in Table 1. Automobile and Horse (i.e. "What is the Horse of Air?", the answer should be Bird.) are first order analogies and they result in comparably superior performance as expected, but second order analogy is much higher than chance level (i.e., 10 percent).

Please note that these analogies are performed strictly on the sensory data, i.e., images. Given an image, the system is able to retrieve a set of relevant images that are linked through a logical statement.

[4] There are 50 training images for each class. CA rule 110 is used for evolution. And mean of 20 Monte Carlo simulations is given to account for randomness in experiments

[5] Also 22 percent non-zero elements

[6] These are extended results for our previous publication Yilmaz (2015a). We were unable to test for large R and I values due to hardware limitations.

Automobile	**Horse**	Truck
79	68	52

Table 1: Analogy-making experiment results on CIFAR 10 dataset (subset). R and I are both 32. The accuracy of the analogy is given for first (given in bold) and second order analogies. See text for details.

A very small number of training data is used, yet we can infer conceptual relationships between images surprisingly accurately. However, again it should be emphasized that this is only a single experiment with a very limited analogical scope, and more experiments are needed to understand the limits of the proposed architecture.

It is possible to build much more complicated concepts using hierarchies. For example, Land and Air are types of environments and can be used as fillers in Environment field. Ontologies are helpful to narrow down the set of required concepts for attaining a satisfactory description of the world. Other modalities such as text data are also of great interest, (see Mikolov et al. (2013); Pennington et al. (2014) for state-of-the-art studies), as well as information fusion on multiple modalities (e.g., Image and text).

5 Experiments on Rule Based Inference

In order to test proposed architecture's capability for logical inference, we define a rule and form a knowledge base on image data. Then we make an inference on the knowledge base by applying the rule. The inference may or may not be right, hence the symbolic system is not completely sound. [7] For demonstration of logical inference on our system, we define size relationships among different objects using the following rule set.
Object in image X is larger than object in image Y:

$$L_{xy} = L_1 \oplus X + L_2 \oplus Y.$$

Object in image X is smaller than object in image Z:

$$S_{xz} = S_1 \oplus X + S_2 \oplus Z.$$

And finally, we state the largest object is in image Z:

$$T_z = T_1 \oplus Z.$$

Then the rule is stated as 'If object in X is larger than object in Y and smaller than object in Z, largest object is in Z'. The rule vector is computed as the manipulation of object vectors using hyperdimensional computing framework:

$$R_{xyz} = T_z \oplus (L_{xy} + S_{xz}).$$

Our knowledge base is again formed on the images of CIFAR 10 dataset. First, we use Truck, Automobile, and Airplane images (50 each) to compute X, Y and Z concept vectors (CA rule 110); then we obtain the rule vector R_{xyz} as explained above utilizing the concept vectors. In a completely different set of test image triplet, we make use of single Truck, Automobile, and Airplane images and compute their vector representation; a, b and c respectively. Knowledge base is created on the object vectors in three test images:

$$L_{ab} = L_1 \oplus a + L_2 \oplus b,$$
$$S_{ac} = F_1 \oplus a + M_2 \oplus c,$$

as 'object in image a is larger than object in image b, and object in image a is smaller than object in image c'. Can we infer the largest object? When we apply the rule vector on existing knowledge base, we get an estimate for the vector representation of the largest object:

$$T_{est} = R_{xyz} \oplus (L_{ab} + S_{ac}).$$

We compute the Hamming distance of T_{est} to existing object vectors (i.e. a, b and c), then it is possible to decide on the estimated largest object, i.e. closest vector to T_{est} which should be vector

[7] The completeness of the system requires a proof and it is a future work.

c. The average accuracy of 50 different test image triplets are shown in Figure 2 b. The chance level is 33 percent and it is observed that the binary neural representation (i.e., both R and I is equal to 1) is around 50 percent accurate, whereas cellular automata state space provides a 100 percent inference accuracy for a relatively small reservoir size. Please note that, similar to analogy making experiments logical inference is performed directly on image data and we can make object size inference using a very small number of example images (50).

6 Discussion

Along with the pattern recognition capabilities of cellular automata based reservoir computing Yilmaz (2015b), hyperdimensional computing framework enables symbolic processing. Due to the binary categorical indicator nature of the representation, the rules that make up the knowledge base and feature representation of the data that make up the statistical model live on the same space, which is essential for combining connectionist and symbolic capabilities. It is possible to make analogies, form hierarchies of concepts, and apply logical rules on the reservoir feature vectors [8].

To illustrate the logical query, we have shown the capability of the system to make analogies on image data. We asked the question "What is the Automobile of Air?" after building Land and Air concepts based on the images of Horse, Automobile (Land), Bird and Airplane (Air). The correct answer is Airplane and the system infers this relationship with 98 percent accuracy, with only 50 training images per class. Additionally we have tested the performance of our architecture on rule based logical inference on images. We defined an object size related rule on image data, provided a knowledge base and inferred the largest object strictly using the image features.

Neural network data embeddings (eg. Kiros et al. (2014); Mikolov et al. (2013)) are an alternative to our approach, in which representation suitable for logical manipulation is learned from the data using gradient descent. Although these promising approaches are showing state-of-the-art results, they are bound to suffer from the dilemma of 'no free lunch' because the representation is data-specific. The other extreme is random embeddings adopted in hyperdimensional computing and reduced vector representation approaches. Although randomness maximizes the orthogonality of vectors and optimizes the effective usage of the space, it does not allow statistical machine learning or semantically meaningful modifications on existing vectors. Our approach lies in the middle: it does not create random vectors, thus can manipulate existing vectors and use machine learning, but do not learn the representation from the data therefore it is less prone to overfitting as well as to the dilemma of 'no free lunch'. Moreover, cellular automata reservoir is orders of magnitude faster than neural network counterparts Yilmaz (2015b).

7 Acknowledgments

This research is supported by The Scientific and Technological Research Council of Turkey (TUBİTAK) Career Grant, No: 114E554.

References

Bader, S., Hitzler, P., & Hölldobler, S. (2008). Connectionist model generation: A first-order approach. *Neurocomputing*, *71*, 2420–2432.

Baetens, J. M., & De Baets, B. (2010). Phenomenological study of irregular cellular automata based on lyapunov exponents and jacobians. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *20*, 033112.

Besold, T. R., Garcez, A. d., Kühnberger, K.-U., & Stewart, T. C. (2014). Neural-symbolic networks for cognitive capacities. *Biologically Inspired Cognitive Architectures*, (pp. iii–iv).

Coates, A., Ng, A. Y., & Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics* (pp. 215–223).

[8] Linear CA rules, such as rule 90 allow superposition of initial conditions. This property provides a symbolic system with much more powerful expressive capability Yilmaz (2015a).

Cook, M. (2004). Universality in elementary cellular automata. *Complex Systems*, *15*, 1–40.

Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., & Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 601–610). ACM.

Gallant, S. I., & Okaywe, T. W. (2013). Representing objects, relations, and sequences. *Neural computation*, *25*, 2038–2078.

Garcez, A. S. d., Broda, K., & Gabbay, D. M. (2012). *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media.

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, *1*, 139–159.

Kiros, R., Salakhutdinov, R., & Zemel, R. S. (2014). Unifying visual-semantic embeddings with multimodal neural language models. *arXiv preprint arXiv:1411.2539*, .

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, .

Levy, S. D., & Gayler, R. (2008). Vector symbolic architectures: A new building material for artificial general intelligence. In *Proceedings of the 2008 conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference* (pp. 414–418). IOS Press.

LukošEvičIus, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*, 127–149.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, *14*, 2531–2560.

Marcus, G. F. (2003). *The algebraic mind: Integrating connectionism and cognitive science*. MIT press.

Miikkulainen, R., Bednar, J. A., Choe, Y., & Sirosh, J. (2006). *Computational maps in the visual cortex*. Springer Science & Business Media.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems* (pp. 3111–3119).

Mitchell, M. et al. (1996). Computation in cellular automata: A selected review. *Nonstandard Computation*, (pp. 95–140).

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, *12*, 1532–1543.

Plate, T. A. (2003). Holographic reduced representation: Distributed representation for cognitive structures, .

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, *46*, 77–105.

Snaider, J. (2012). Integer sparse distributed memory and modular composite representation, .

Wolfram, S. (2002). *A new kind of science* volume 5. Wolfram media Champaign.

Yilmaz, O. (2015a). Symbolic Computation using Cellular Automata based Hyperdimensional Computing. *Neural Computation*, .

Yilmaz, O. (2015b). Machine Learning using Cellular Automata based Feature Expansion and Reservoir Computing. *Journal of Cellular Automata*, .

Yilmaz, O. (2015c). Connectionist-Symbolic Machine Intelligence using Cellular Automata based Reservoir-Hyperdimensional Computing. *arXiv preprint arXiv:1503.00851*, .

Yilmaz, O., Ozsarac, I., Gunay, O., & Ozkan, H. (2015). Cognitively inspired real-time vision core. *Technical Report*, . Available at `http://ozguryilmazresearch.net/Publications/NeuralNetworkVisionCore_YilmazEtAl2015.pdf`.