

*The 18<sup>th</sup> European Conference on Artificial Intelligence*

Proceedings

**4<sup>th</sup> International Workshop on  
Neural-Symbolic Learning and Reasoning**

**NeSy'08**

Monday July 21, 2008  
Patras, Greece

*Artur S. d'Avila Garcez and Pascal Hitzler*

# Workshop Schedule

09:00 – 10:00

Keynote Talk

Kai-Uwe Kühnberger

Modeling Reasoning Mechanisms by Neural-Symbolic Learning

10:30 – 10:50

Ekaterina Komendantskaya

Unification by Error-Correction

10:55 – 11:10

Matthew Cook

The Reusable Symbol Problem

11:15 – 11:35

Claudine Brucks, Michael Hilker, Christoph Schommer, Cynthia Wagner, Ralph Weires

Symbolic Computing with Incremental Mind-maps to Manage and Mine Data Streams – Some Applications

11:40 – 12:00

Sebastian Bader, Steffen Hölldobler, Nuno C. Marques

Guiding Backprop by Inserting Rules

12:05 – 12:25

Tsvi Achler, Eyal Amir

Hybrid Classification and Symbolic-Like Manipulation Using Self-Regulatory Feedback Networks

## **Workshop Organisers**

Artur d'Avila Garcez, City University London, UK  
Pascal Hitzler, University of Karlsruhe, Germany

## **Programme Committee**

Sebastian Bader, TU Dresden, Germany  
Howard Blair, Syracuse University, U.S.A.  
Luc de Raedt, KU Leuven, Belgium  
Marco Gori, University of Siena, Italy  
Barbara Hammer, TU Clausthal, Germany  
Ioannis Hatzilygeroudis, University of Patras, Greece  
Steffen Hölldobler, TU Dresden, Germany  
Ekaterina Komendantskaya, Sophia Antipolis, France  
Kai-Uwe Kühnberger, University of Osnabrück, Germany  
Luis Lamb, Federal University of Rio Grande do Sul, Brazil  
Roberto Prevete, University of Naples, Italy  
Dan Roth, University of Illinois at Urbana-Champaign, U.S.A.  
Anthony K. Seda, University College Cork, Ireland  
Frank van der Velde, Leiden University, The Netherlands  
Gerson Zaverucha, Federal University of Rio de Janeiro, Brazil

# Preface

Artificial Intelligence researchers continue to face huge challenges in their quest to develop truly intelligent systems. The recent developments in the field of neural-symbolic integration bring an opportunity to integrate well-founded symbolic artificial intelligence with robust neural computing machinery to help tackle some of these challenges.

The Workshop on Neural-Symbolic Learning and Reasoning provides a forum for the presentation and discussion of the key topics related to neural-symbolic integration.

Topics of interest include:

- The representation of symbolic knowledge by connectionist systems;
- Learning in neural-symbolic systems;
- Extraction of symbolic knowledge from trained neural networks;
- Reasoning in neural-symbolic systems;
- Biological inspiration for neural-symbolic integration;
- Neural networks and probabilities;
- Applications in robotics, semantic web, engineering, bioinformatics, etc.

# Invited Keynote Talk

## Modeling Reasoning Mechanisms by Neural-Symbolic Learning

Kai-Uwe Kühnberger, University of Osnabrück, Germany, [kkuehnbe@uos.de](mailto:kkuehnbe@uos.de)

Currently, neural-symbolic integration covers – at least in theory – a whole bunch of types of reasoning: neural representations (and partially also neural-inspired learning approaches) exist for modeling propositional logic (programs), whole classes of manyvalued logics, modal logic, temporal logic, and epistemic logic, just to mention some important examples [2,4]. Besides these propositional variants of logical theories, also first proposals exist for approximating “infinity” with neural means, in particular, theories of first-order logic. An example is the core method intended to learn the semantics of the single-step operator  $T_P$  for first-order logic (programs) with a neural network [1]. Another example is the neural approximation of variable-free first-order logic by learning representations of arrow constructions (which represent logical expressions) in the  $\mathbf{R}^n$  using Topos constructions [3].

Although these examples show a certain success of neural-symbolic learning and reasoning research, there are several non-trivial challenges. First, there exist a variety of frameworks that seem to have rather different and seemingly incompatible foundations. Second, potential application domains where the strengths of neural-symbolic integration could be documented and its potential be shown are not really known. Third, the conceptual understanding of the cognitive relevance and the cognitive plausibility of neural-symbolic learning and reasoning should be clarified. In this talk, I will address these questions and propose some ideas for answers. I will sketch general assumptions of solutions for the neural-symbolic modeling of a variety of logical reasoning mechanisms. Then I will propose some application domains where neural-symbolic frameworks can be successfully applied. I will finish the talk with some speculations concerning cognitively relevant implications and the degree of cognitive plausibility of neural-symbolic learning and reasoning in general.

## References

- [1] S. Bader, P. Hitzler, S. Hölldobler and A. Witzel: A Fully Connectionist Model Generator for Covered First-Order Logic Programs. In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, 2007, pp. 666–671.
- [2] A. d’Avila Garcez, L. Lamb & D. Gabbay: Neural-Symbolic Cognitive Reasoning, Cognitive Technologies, Springer, 2008.

[3] H. Gust, K.-U. Kühnberger & P. Geibel: Learning Models of Predicate Logical Theories with Neural Networks based on Topos Theory. In P. Hitzler and B. Hammer (eds.): Perspectives of Neuro–Symbolic Integration, Studies in Computational Intelligence (SCI) 77, Springer, 2007, pp. 233–264.

[4] E. Komendantskaya, M. Lane & A. Seda: Connectionist Representation of Multi-Valued Logic Programs. In P. Hitzler and B. Hammer (eds.): Perspectives of Neuro–Symbolic Integration, Studies in Computational Intelligence (SCI) 77, Springer, 2007, pp. 283–313.

# Unification by Error-Correction

Ekaterina Komendantskaya<sup>1</sup>

**Abstract.** The paper formalises the famous algorithm of first-order unification by Robinson by means of the error-correction learning in neural networks. The significant achievement of this formalisation is that, for the first time, the first-order unification of two arbitrary first-order atoms is performed by finite (two-neuron) network.

## 1 Introduction

The field of neuro-symbolic integration is stimulated by the fact that logic theories are commonly recognised as deductive systems that lack such properties of human reasoning, as adaptation, learning and self-organisation. On the other hand, neural networks, introduced as a mathematical model of neurons in human brain, possess all of the mentioned abilities, and moreover, they provide parallel computations and hence can perform certain calculations faster than classical algorithms.

As a step towards integration of the two paradigms, there were built connectionist neural networks (also called neuro-symbolic networks), see [1, 8] for a very good survey. In particular, there were built neural networks [16, 17] that can simulate the work of the semantic operator  $T_P$  for logic programs. These neural networks processed classical truth values 0 and 1 assigned to clauses and clause atoms. These values were presented to the neural networks as input vectors and emitted by the neural networks as output vectors.

The connectionist neural networks of different architectures [8, 15, 7, 1] bore different advantages, but one similar feature: the unification of first-order terms, atoms, or clauses was achieved by building a separate neuron for each of the ground instances of an atom. Then all neurons were connected in a particular way that they reflected intended logical relations between the ground atoms. In many cases, e.g., in the presence of function symbols in logic programs, the number of the required ground instances can become infinite. This makes building corresponding neural networks impractical. The problem gave rise to a series of papers about possibility of approximation of potentially infinite computations by (a family of) finite neural networks; see [13, 3, 25, 2].

In this paper, I propose a different direction for the development of the connectionist neural networks. In particular, I propose to use two-neuron networks with error-correction learning to perform the first-order unification over two arbitrary first-order atoms. A simple form of error-correction learning is adapted to syntax of a first-order language in such a way that unification of two atoms is seen as a correction of one piece of data relative to the other piece of data.

The problem of establishing a way of how to perform first-order unification in finite neural networks has been tackled by many researchers over the last 30 years; [4, 21, 15, 14, 8, 26, 27, 1, 28].

The way of performing unification that I propose here is novel, fast and simple, and can be easily integrated into a number of various existing neuro-symbolic networks. The paper develops ideas which were first spelt out in [18, 19]. Here, I simplify the construction of the networks, generalise the theorem about unification by error correction and give a more subtle analysis of the new functions introduced into the neural networks. Notably, the statement and the proof of the main theorem do not depend anymore on Gödel numbers, as in [18, 19].

The structure of the paper is as follows. Section 2 outlines the classical algorithm of unification. Section 3 defines artificial Neurons and Neural Networks following [12, 13]. Section 4 describes the error-correction learning algorithm. In Section 5, I re-express several logic notions in terms of recursive functions over terms. These functions are then embedded into the network. In Section 6, I prove that the algorithm of Unification for two arbitrary first-order atoms can be simulated by a two-neuron network with error-correction function. Finally Section 7 concludes the discussion.

## 2 Unification algorithm

The algorithm of unification for first-order atoms was introduced in [24] and has been extensively used in Logic programming [22] and theorem proving.

I fix a first-order language  $\mathcal{L}$  consisting of constant symbols  $a_1, a_2, \dots$ , variables  $x_1, x_2, \dots$ , function symbols of different arities  $f_1, f_2, \dots$ , predicate symbols of different arities  $Q_1, Q_2, \dots$ , connectives  $\neg, \wedge, \vee$  and quantifiers  $\forall, \exists$ . This syntax is sufficient to define first-order language or first-order Horn clause programs, [22].

I follow the conventional definition of a term and an atomic formula. Namely, a constant symbol is a term, a variable is a term, and if  $f_i^n$  is a n-ary function symbol and  $t_1, \dots, t_n$  are terms, then  $f_i^n(t_1, \dots, t_n)$  is a term. If  $Q_i^n$  is an n-ary predicate symbol and  $t_1, \dots, t_n$  are terms, then  $Q_i^n(t_1, \dots, t_n)$  is an atomic formula, also called an atom.

Let  $S$  be a finite set of atoms. A substitution  $\theta$  is called a unifier for  $S$  if  $S\theta$  is a singleton. A unifier  $\theta$  for  $S$  is called a *most general unifier* (mgu) for  $S$  if, for each unifier  $\sigma$  of  $S$ , there exists a substitution  $\gamma$  such that  $\sigma = \theta\gamma$ . To find the *disagreement set*  $D_S$  of  $S$  locate the leftmost symbol position at which not all atoms in  $S$  have the same symbol and extract from each atom in  $S$  the term beginning at that symbol position. The set of all such terms is the disagreement set.

The unification algorithm [24, 20, 22] is described as follows.

**Unification algorithm:**

1. Put  $k = 0$  and  $\sigma_0 = \varepsilon$ .
2. If  $S\sigma_k$  is a singleton, then stop;  $\sigma_k$  is an mgu of  $S$ . Otherwise, find the disagreement set  $D_k$  of  $S\sigma_k$ .
3. If there exist a variable  $v$  and a term  $t$  in  $D_k$  such that  $v$  does not occur in  $t$ , then put  $\sigma_{k+1} = \sigma_k\{v/t\}$ , increment  $k$  and go to 2.

<sup>1</sup> INRIA Sophia Antipolis, France, email: ekaterina.komendantskaya@inria.fr

Otherwise, stop;  $S$  is not unifiable.

The *Unification Theorem* establishes that, for any finite  $S$ , if  $S$  is unifiable, then the unification algorithm terminates and gives an mgu for  $S$ . If  $S$  is not unifiable, then the unification algorithm terminates and reports this fact.

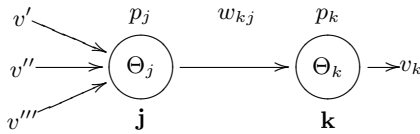
### 3 Connectionist Neural Networks

I follow the definitions of a connectionist neural network given in [16, 17], see also [7, 13, 9].

A *connectionist network* is a directed graph. A *unit*  $k$  in this graph is characterised, at time  $t$ , by its *input vector*  $(v_{i_1}(t), \dots, v_{i_n}(t))$ , its potential  $p_k(t)$ , its *threshold*  $\Theta_k$ , and its *value*  $v_k(t)$ . Note that in general, all  $v_i$ ,  $p_i$  and  $\Theta_i$ , as well as all other parameters of a neural network can be performed by different types of data, the most common of which are real numbers, rational numbers [16, 17], fuzzy (real) numbers [23], complex numbers, numbers with floating point, Gödel numbers [19], and some others, see also [12].

Units are connected via a set of directed and weighted connections. If there is a connection from unit  $j$  to unit  $k$ , then  $w_{kj}$  denotes the *weight* associated with this connection, and  $i_k(t) = w_{kj}v_j(t)$  is the *input* received by  $k$  from  $j$  at time  $t$ . The units are updated synchronously. In each update, the potential and value of a unit are computed with respect to an *activation* and an *output function* respectively. Most units considered in this paper and [16] compute their potential as the weighted sum of their inputs minus their threshold:  $p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj}v_j(t)\right) - \Theta_k$ . The units are updated synchronously, time becomes  $t + \Delta t$ , and the output value for  $k$ ,  $v_k(t + \Delta t)$  is calculated using  $p_k(t)$  by means of a given *output function*  $F$ , that is,  $v_k(t + \Delta t) = F(p_k(t))$ . For example,  $F$  can be an identity function  $\text{id}$ , or the binary threshold function  $H$ , that is,  $v_k(t + \Delta t) = H(p_k(t))$ , where  $H(p_k(t)) = 1$  if  $p_k(t) > 0$  and  $H(p_k(t)) = 0$  otherwise.

**Example 3.1** Consider two units,  $j$  and  $k$ , having thresholds  $\Theta_j$ ,  $\Theta_k$ , potentials  $p_j$ ,  $p_k$  and values  $v_j$ ,  $v_k$ . The weight of the connection between units  $j$  and  $k$  is denoted by  $w_{kj}$ . Then the following graph shows a simple neural network consisting of  $j$  and  $k$ . The neural network receives signals  $v'$ ,  $v''$ ,  $v'''$  from external sources and sends an output signal  $v_k$ .



### 4 Error-Correction Learning

Error-correction learning is one of the algorithms among the paradigms that advocate *supervised learning*; see [12, 11] for further details.

Let  $d_k(t)$  denote some *desired response* for unit  $k$  at time  $t$ . Let the corresponding value of the *actual response* be denoted by  $v_k(t)$ . The input signal  $i_k(t)$  and desired response  $d_k(t)$  for unit  $k$  constitute a particular *example* presented to the network at time  $t$ . It is assumed that this example and all other examples presented to the network are generated by an environment. It is common to define an *error signal* as the difference between the desired response  $d_k(t)$  and the actual response  $v_k(t)$  by  $e_k(t) = d_k(t) - v_k(t)$ .

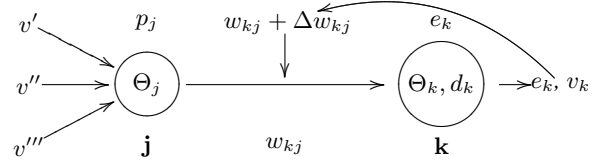
The *error-correction learning rule* is the adjustment  $\Delta w_{kj}(t)$  made to the weight  $w_{kj}$  at time  $n$  and is given by

$$\Delta w_{kj}(t) = \eta e_k(t) v_j(t),$$

where  $\eta$  is a positive constant that determines the rate of learning.

Finally, the formula  $w_{kj}(t + 1) = w_{kj}(t) + \Delta w_{kj}(t)$  is used to compute the updated value  $w_{kj}(t + 1)$  of the weight  $w_{kj}$ . I use formulae defining  $v_k$  and  $p_k$  as in Section 3.

**Example 4.1** The neural network from Example 3.1 can be transformed into an error-correction learning neural network as follows. I introduce the desired response value  $d_k$  into the unit  $k$ , and the error signal  $e_k$  computed using  $d_k$  must be sent to the connection between  $j$  and  $k$  to adjust  $w_{kj}$ .



This learning rule has been extensively used for “recognition” tasks, such as image and speech recognition.

### 5 The data type of parameters of the neural network

In order to perform the algorithm of unification in neural networks and not to depend on truth values of formulae, I need to allow the syntax of first-order formulae directly into the neural network.

Initially, Gödel numbers were used as parameters of the novel neural networks, [19]. It was inspired by the idea that some sort of numerical representation is crucial because Neural networks are numerical machines, and can process only numbers. However, from computational point of view the numerical encoding of the first-order syntax plays no crucial role in the development of the neural networks, and so I omit enumeration here. Instead, I give a more subtle analysis of the new functions that I embed into neural networks.

The significant feature of the Gödel enumeration in [19] was that the notions of the *disagreement set*, *substitution*, and *application of the computed substitutions* were formally expressed as functions over first-order atoms viewed as lists. These functions were embedded into the neural network. The appearance of new functions in the neural network architecture was natural because the neural networks used the new data type - Gödel numbers of atoms.

The algorithm of Unification from Section 2 can be reformulated functionally. Thus, we can define a function  $\ominus$  computing a disagreement set of two given atoms, or a function  $\odot$  applying substitutions, and some other functions, and then compose them into a more complex function that is able to unify terms. There exist several functional formalisations of the algorithm of unification in different functional languages, [6, 10, 29]. It can be quite hard to formalise this algorithm, as [6, 10, 29] indicate. In this paper, we will use only two simple auxiliary functions  $\odot$  and  $\ominus$ , and will not go into further details.

In the rest of the section, we give an example of how the two functions  $\odot$  and  $\ominus$  can be formalised using the language of Coq. The reader not familiar with functional languages can pass on to the next Section, simply bearing in mind that  $\ominus$  will denote the function computing a disagreement set, and  $\odot$  is a function that applies substitutions. The complete Coq file containing more details and examples can be found in [30].



When formalised functionally, the definitions of the *disagreement set*, *substitution*, and *application of the computed substitutions* defined over first-order terms and atoms bear no serious computational difference from formalisations of arithmetic functions  $+$ ,  $-$ ,  $*$  defined over integers. In fact, all of the above-mentioned operations, both logical and arithmetic, are defined recursively by computing a fixpoint.

**Example 5.1** *This is how conventional  $+$  over natural numbers is defined in Coq:*

```
Fixpoint plus (n m: nat){struct n}: nat :=
match n with
0 => m | S p => S (plus p m) end.
```

We will see in this section that logical operations can be defined analogously, by fixpoint. I will define functions that find a disagreement set, a substitution, and apply the computed substitutions using the Coq code, the same functions expressed in terms of Gödel numbers can be found in [19, 18].

We need to specify the inductive data type of first-order terms and atoms. We will define recursive functions over this data type:

```
Inductive term : Set :=
App (t1 t2: term) | Const (c: cindex)
| Var (v: vindex).
```

**Example 5.2** *For example,  $Q(x, y)$  will be denoted by  $(App (App (Const 0) x) y)$ .*

I start with the function applying substitutions:

```
Fixpoint subst (v: vindex) (t1 t2: term)
{struct t2} : term := match t2 with
| App t3 t4 =>
App (subst v t1 t3) (subst v t1 t4)
| Const _ => t2
| Var v1 => if v_eq v v1 then t1 else t2
end.
```

To define the function computing the disagreement set, one needs to inductively define the type of possible outputs of the function. Unification algorithm of Section 2 could output either “failure”, or a computed mgu, or an “empty” substitution  $\varepsilon$ :

```
Inductive dresult: Set :=
Dempty | Dfail | Dsubst (v: vindex) (t: term).
```

The function computing the disagreement set reformulated in Gödel numbers was called  $\ominus$  in [19].

**Operation  $\ominus$** , written  $(t_1 \ominus t_2)$ , or `delta (t1 t2)`:

```
Fixpoint delta (t1 t2: term)
{struct t1} : dresult :=
match t1, t2 with
| Var v1, Var v2 =>
if v_eq v1 v2 then Dempty else Dsubst v1 t2
| Var v1, _ =>
if v_is_in v1 (free_vars t2) then
Dfail else Dsubst v1 t2
| _, Var v2 =>
if v_is_in v2 (free_vars t1) then
Dfail else Dsubst v2 t1
```

```
| Const c1, Const c2 =>
if c_eq c1 c2 then Dempty else Dfail
| App t11 t12, App t21 t22 =>
match delta t11 t21 with Dempty =>
delta t12 t22 | r => r end
| _, _ => Dfail
end.
```

The function above is built using another function, `free_vars`, that performs the occur check. The definition of this function can be found in [30]; it is a simple recursive function defined by fixpoint.

The function that applies *computed* substitutions was formulated in Gödel numbers and denoted  $\odot$  in [19]. We give its Coq code here: **Operation  $\odot$** , written  $(t \odot d)$  or `apply_delta t d`:

```
Definition apply_delta t d :=
match d with Dsubst v t1 =>
subst v t1 t | _ => t end.
```

The functions  $\ominus, \odot$  will be taken as new parameters of a neural network.

## 6 Unification in Neural Networks

Neural networks constructed in this section perform unification by error-correction.

The next theorem and its proof present this novel construction. The construction is effectively based upon the error-correction learning algorithm defined in Section 4 and makes use of the operations  $\odot$  and  $\ominus$  defined in Section 5. I will also use  $\oplus$  - the conventional list concatenation that, given lists  $x$  and  $y$ , forms the list  $x \oplus y$ . The standard Coq formalisation of this function can be found in [5].

**Remark 1.** The next Theorem requires the last short remark. The item 3 in the Unification algorithm of Section 2 requires composition of computed substitutions at each iteration of the algorithm. That is, given a set  $S$  of atoms, the unification algorithm will compute  $S(\sigma_0\sigma_1\dots\sigma_n)$ , which means that the composition of substitutions  $\sigma_0\sigma_1\dots\sigma_n$  is applied to  $S$ . However, one can show that  $S(\sigma_0\sigma_1\dots\sigma_n) = (\dots((S\sigma_0)\sigma_1)\dots\sigma_n)$ . That is, one can alternatively concatenate substitutions and apply them one by one to atoms in  $S$ . We will use this fact when constructing the neural networks. The two functions - concatenation  $\oplus$  and application  $\odot$  of substitutions will be used to model  $S(\sigma_0\sigma_1\dots\sigma_n)$ .

**Theorem 1** *Let  $k$  be a neuron with the desired response value  $d_k = g_B$ , where  $g_B$  is (the encoding of) a first-order atom  $B$ , and let  $v_j = 1$  be the signal sent to  $k$  with weight  $w_{kj} = g_A$ , where  $g_A$  is (the encoding of) a first-order atom  $A$ . Let  $h$  be a unit connected with  $k$ . Then there exists an error signal function  $e_k$  and an error-correction learning rule  $\Delta w_{kj}$  such that the **unification algorithm for  $A$  and  $B$**  is performed by **error-correction learning** at unit  $k$ , and the unit  $h$  outputs (the encoding of) an mgu of  $A$  and  $B$  if an mgu exists, and it outputs 0 if no mgu of  $A$  and  $B$  exists.*

### Construction:

We construct the two neuron network as follows. The unit  $k$  is the input unit, the signal from the unit  $k$  goes to the unit  $h$ , and the unit  $h$  outputs the signal.

#### Parameters:

Set thresholds  $\Theta_k = \Theta_h = 0$ , and the initial weight  $w_{hk}(0) = 0$  of the connection between  $k$  and  $h$ . The input signal  $i_k(t) = v_j(t)w_{kj}(t) = w_{kj}(t)$ . The initial input  $i_k(0) = w_{kj}(0) = g_A$ .

Because  $v_j(t) = 1$  for all  $t$ , the standard formula that computes potential  $p_k(t) = v_j(t)w_{kj}(t) - \Theta_k$  transforms into  $p_k(t) = w_{kj}(t)$ . We put  $v_k(t) = p_k(t)$ .

The error signal is defined as follows:  $e_k(t) = d_k(t) \ominus v_k(t)$ . The initial desired response  $d_k(0) = g_B$ .

The error-correction learning rule is as defined in Section 4:  $\Delta w_{kj}(t) = v_j(t)e_k(t)$ . In our case  $v_j(t) = 1$ , at every time  $t$ , and so  $\Delta w_{kj}(t) = e_k(t)$ . The  $\Delta w_{kj}(t)$  is used to compute

$$w_{kj}(t+1) = w_{kj}(t) \odot \Delta w_{kj}(t), \text{ and } d_k(t+1) = d_k(t) \odot \Delta w_{kj}(t).$$

Connection between  $k$  and  $h$  is “trained” as follows:

$$w_{hk}(t+1) = \begin{cases} w_{hk}(t) \oplus \Delta w_{kj}(t), & \text{if } \Delta w_{kj}(t) = \text{Dsubst} \vee \text{t} \\ w_{hk}(t) \oplus 0, & \text{if } \Delta w_{kj}(t) = \text{Dempty} \\ 0, & \text{if } \Delta w_{kj}(t) = \text{Dfail}. \end{cases}$$

Reading the resulting mgu.

Compute  $p_h(t + \Delta t) = w_{hk}(t + \Delta t)$ . Put  $v_h(t + \Delta t) = p_h(t + \Delta t)$ .

If the signal  $v_h(t + \Delta t) \neq 0$  and the first and the last symbol constituting the list  $v_h(t + \Delta t)$  is 0, stop. The signal  $v_h(t + \Delta t)$  is the mgu of  $A$  and  $B$ .

If  $v_h(t + \Delta t) = 0$ , then stop. Unification failed.

**Sketch of a proof:** Item 1 of Unification algorithm (Section 2).

The network works in discrete time, and the sequence of time steps, starting with 0, corresponds to the parameter  $k$  in the Unification algorithm, that changes from 0 to 1, from 1 to 2, etc. The initial empty substitution  $\sigma_0 = \varepsilon$  corresponds to the initial weight  $w_{hk}(0) = 0$ .

Item 2 of Unification algorithm. The application  $S\sigma_0 \dots \sigma_k$  of substitution  $\sigma_0 \dots \sigma_k$  to  $S$  is performed by the function  $\odot$  that, by **Remark 1**, applies  $\sigma_k$  to  $(\dots(S\sigma_0) \dots \sigma_{k-1})$ . The check whether the disagreement set  $D_k$  for  $S\sigma_1 \dots \sigma_k$  is empty is done by  $\ominus$ . If it is empty,  $d_k(t) \ominus v_k(t) = \text{Dempty}$ . If this happens at time  $t$ ,  $e_k(t) = \Delta w_{kj}(t) = \text{Dempty}$ , and  $v_h(t + 1) = w_{hk}(t + 1) = w_{hk}(t) \oplus \Delta w_{kj}(t) = 0 \oplus e_k(0) \oplus \dots \oplus e_k(t - 1) \oplus e_k(t) = 0 \oplus e_k(0) \oplus \dots \oplus e_k(t - 1) \oplus 0$  is sent as an output from the unit  $h$ . This will be read as “Stop, the mgu is found”. In case  $D_k$  for  $S\sigma_1 \dots \sigma_k$  is not empty, the function  $d_k(t) \ominus v_k(t)$  computes the disagreement set for  $g_B(\sigma_1 \dots \sigma_k) = d_k(t)$  and  $g_A(\sigma_1 \dots \sigma_k) = v_k(t)$  in  $S\sigma_1 \dots \sigma_k$ ;  $\Delta w_{kj}(t)$  is computed, and the new iteration starts.

Item 3 of Unification algorithm. The occur check is hidden inside the function  $\ominus$  used to define the error signal  $e_k$ , thanks to the auxiliary function `free_vars` used when defining  $\ominus$ . The step “put  $\sigma_{k+1} = \sigma_k\{v/t\}$ ” is achieved by using concatenation of substitutions by function  $\oplus$ :  $w_{hk}(t + 1) = w_{hk}(t) \oplus \Delta w_{kj}(t)$ . By **Remark 1**, we concatenate the substitutions  $\sigma_0 \dots \sigma_k \sigma_{k+1}$ , and apply them to atoms in  $S$  stepwise, that is, at each iteration of the network we use  $\odot$  to apply the new computed substitution  $\sigma_{k+1}$  given by  $\Delta w_{kj}(t + 1)$  to the two atoms in  $S\sigma_1 \dots \sigma_k$  given by  $w_{kj}(t)$  and  $d_k(t)$ .

The step “increment  $k$  and go to 2” is achieved by starting a new iteration. The condition “otherwise, stop;  $S$  is not unifiable” is achieved as follows. When the substitution is not possible, or the “occur check” is not passed,  $\Delta w_{kj}(t) = d_k(t) \ominus v_k(t) = \text{Dfail}$ , and this sets  $w_{hk}(t + 1) = 0$ . But then,  $p_h(t + 1) = w_{hk}(t + 1) = 0$ . But then, the output value  $v_h(t + 1) = p_h(t + 1)$  is set to 0. And this will be read as “substitution failed”.

Note that in case when  $A = B$  and hence the mgu of  $A$  and  $B$  is  $\varepsilon$ , the neural networks will give output  $v_h(t + 2) = 0 \oplus 0$ .

Unlike the Unification algorithm of Section 2, the neural network outputs the concatenation of the substitutions computed at each iteration, and not their composition. However, given an ordered list of computed substitutions, composing them is trivial, and can be done

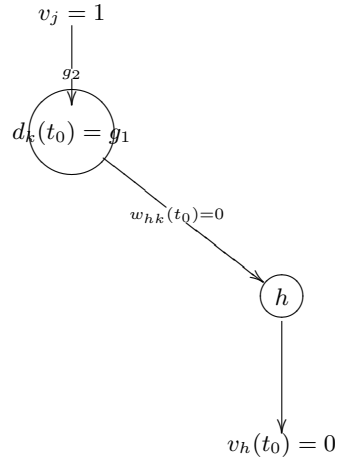
using the function  $\odot$  instead of  $\oplus$  in the networks above. The function  $\oplus$  bears an advantage that one can easily check whether the output list of substitutions ends with 0, and thus it is easy to decide whether Unification algorithm has come to an answer. In general, there is no serious obstacles for using  $\odot$  instead of  $\oplus$  in the construction above, and thus to output composition of substitutions instead of their concatenation.

The next example illustrates how the construction of Theorem 1 works.

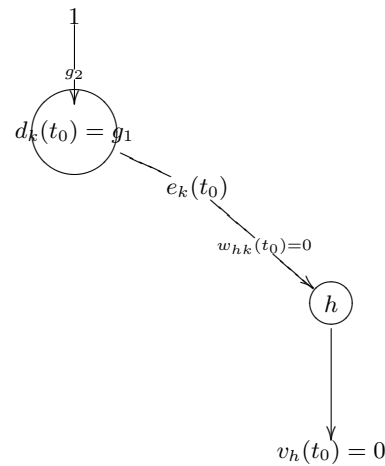
**Example 6.1** Let  $S = (Q_1(f(x_1, x_2)), Q_1(f(a_1, a_2)))$  be a set of first-order atoms. Then  $\theta = \{x_1/a_1; x_2/a_2\}$  is the mgu.

Next I show how this can be computed by neural networks. Let  $g_1$  and  $g_2$  denote the chosen encoding for  $Q_1(f(x_1, x_2))$  and  $Q_1(f(a_1, a_2))$ .

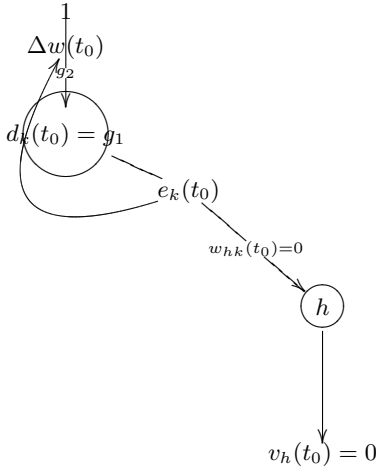
The neural network I build will consist of two units,  $k$  and  $h$ . External signal  $v_j(t_0) = 1$  is sent to the unit  $k$ . I use the numbers  $g_1$  and  $g_2$  as parameters of the neural network, as follows:  $w_{kj}(t_0) = g_2$ ; and hence  $i_k(t_0) = v_j w_{kj} = g_2$ . The desired response  $d_k(t_0)$  is set to  $g_1$ . See the next diagram.



At time  $t_0$ , this neural network computes  $e_k(t_0) = d_k(t_0) \ominus v_k(t_0)$  - the disagreement set  $\{x, a\}$ :



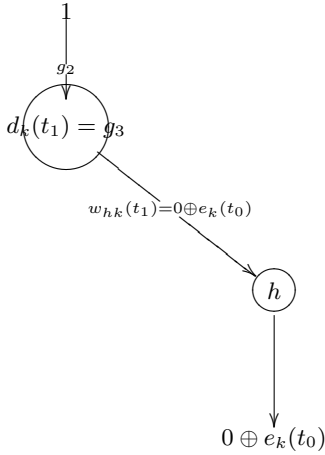
The error signal  $e_k(t_0)$  will then be used to change the weight of the outer connection  $w_{kj}$  to  $k$  and some other parameters; and  $\Delta w(t_0)$  is computed for this purpose as follows:  $\Delta w(t_0) = e_k(t_0)$ . And this is shown on the next diagram.



At time  $t_1$ , the input weight  $w_{kj}$  is amended using  $e_k(t_0)$ , and the desired response value  $d_k(t_1)$  is “trained”, too:  $w_{kj}(t_1) = w_{kj}(t_0) \odot \Delta w_{kj}(t_0)$  and  $d_k(t_1) = d_k(t_0) \odot \Delta w_{kj}(t_0)$ . At this stage the computed substitution  $e_k(t_0) = \Delta w_{kj}(t_0)$  is applied to the numbers  $g_1$  and  $g_2$  of the atoms  $Q_1(f(x_1, x_2))$ ,  $Q_1(f(a_1, a_2))$  that we are unifying.

The weight between  $k$  and  $h$  is amended at this stage:  $w_{hk}(t_1) = w_{hk}(t_0) \oplus \Delta w_{kj}(t_0) = 0 \oplus e_k(t_0)$ .

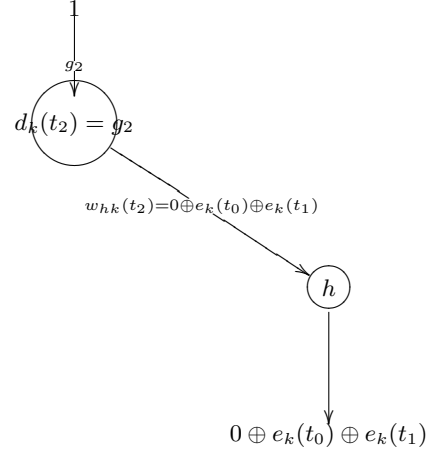
As a result, at time  $t_1$  all the parameters are updated as follows: input signal  $v_j(t_1)w_{kj}(t_1) = g_2$  is the encoding of  $Q_1(f(a_1, a_2))$ ; the desired response  $d_k(t_1) = g_3$  is  $Q_1(f(a_1, x_2))$ . And this is shown on the next diagram:



Note that on the diagram above, the unit  $h$  emits its first non-zero output signal, that is, the number of substitution  $e_k(t_0)$ .

Because the parameters  $w_{kj}(t_1) = g_2$  and  $d_k(t_1) = g_3$  are not equal yet and the list  $v_h$  does not end with 0, the same iteration as above starts again. And at times  $t_1 - t_2$ , the number  $e_k(t_1)$  of a new substitution  $\{x_2/a_2\}$  is computed and applied, as follows:

$e_k(t_1) = g_2 \ominus g_3$ ; the input signal  $v_j(t_2)w_{kj}(t_2) = w_{kj}(t_1) \odot e_k(t_1) = g_2$  is the encoding of  $Q_1(f(a_1, a_2))$ ;  
 $d_k(t_2) = d_k(t_1) \odot e_k(t_1) = g_3 \ominus g_2$  is  $Q_1(f(a_1, a_2))$ ;



At time  $t_2$ , new iteration will be initialised. But, because  $d_k(t_2) = w_{kj}(t_2) = g_2$ , the error signal  $e_k(t_2) = \text{Dempty}$  and the error-correction learning rule will compute  $\Delta w_{kj}(t_2) = \text{Dempty}$ . And then,  $w_{hk}(t_3) = 0 \oplus e_k(t_0) \oplus e_k(t_1) \oplus 0$ .

Note that the neuron  $h$  finally emits the signal that contains both substitutions computed by the network. The fact that the last symbol of the list  $0 \oplus e_k(t_0) \oplus e_k(t_1) \oplus 0$  is 0, tells the outer reader that the unification algorithm finished its computations.

## 7 CONCLUSIONS

The main conclusions to be made from the construction of Theorem 1 are as follows:

- First-order atoms are embedded directly into a neural network. That is, I allow not only binary threshold units (or binary truth values 0 and 1) as in traditional neuro-symbolic networks [16, 17, 1], but also units that can receive and send a code of first-order formulae.
- Numerical encoding of first-order atoms by means of neural network signals and parameters forced us to introduce new functions,  $\ominus$  and  $\odot$  that work over these encodings.
- Resulting neural network is finite and gives deterministic results.
- The error-correction learning recognised in Neurocomputing is used to perform the algorithm of unification.
- Unification algorithm is performed as an adaptive process, which corrects one piece of data relatively to the other piece of data.

### Discussion

The main result of this paper can raise the following questions.

*Does Theorem 1 really define a connectionist neural network?*

The graph defined in Theorem 1 complies with the general definition of a neural network of Section 3. Indeed, it is a directed graph, with usual parameters such as thresholds, weights, with potentials and values computed using the same formulae as in Section 3. The main new feature of these novel neural networks is the new data type of inputs and outputs, that is the type of first-order terms.

*Does the network really learn?*

The network follows the same scheme of error-correction learning as conventional error-correction learning neural networks, [12]. That is, we introduce the desired response value  $d_k$  in the neuron  $k$ , which is repeatedly compared with the output value  $v_k$  of the neuron  $k$ . Through the series of computations, the network amends the weight  $w_{kj}$  in such a way that the difference between  $d_k$  and  $v_k$  diminishes.

This agrees with the conventional error-correction learning rule that “trains” the weight of the given connection, and minimises the difference between the desired response and the output value. The main novelty is that the error signal is defined using  $\ominus$ , - the function computing the difference between terms, as opposed to using conventional “-” in Section 4. The error-correction learning rule is defined using  $\odot$ , - the function applying substitutions, as opposed to using conventional “+” in Section 4.

*Can we use these neural networks for massively parallel computations?*

There is no obstacles for composing the networks of Theorem 1. One can unify arbitrary many sets  $S_1, S_2, S_n$ , using composition of  $n$  networks from Theorem 1 working in parallel.

*What is the significance of these networks?*

As was shown in [18, 19], the networks of Theorem 1 can be conveniently used to formalise the algorithm of SLD-resolution for first-order logic programs. This construction can be further developed and refined in order to obtain the first neuro-symbolic theorem prover. The main advantages of the networks simulating SLD resolution [18, 19] as opposed to those simulating semantic operators [17, 8, 15, 7, 1] are their finiteness, possibility to extend to higher-order terms and the use of unification and variable substitutions, as opposed to working with ground instances of atoms and their truth values in case of semantic operators. This opens new horizons for implementing a goal-oriented proof search in neural networks.

#### Further work

The construction we have given here can be extended to higher-order terms and atoms. This should be relatively easy, because we do not depend on ground instances anymore.

Another direction for research would be to embed the neurons performing unification into the existing neuro-symbolic networks, such as those described in [7, 9].

The networks we present here can be useful for further development of neuro-symbolic networks formalising inductive logic and inductive reasoning.

Finally, we envisage to complete in the near future the full Coq formalisation of Theorem 1.

## ACKNOWLEDGEMENTS

I would like to thank Laurent Théry for Coq formalisations of functions  $\ominus$ ,  $\odot$  ([30]), and for stimulating discussions of the unification algorithm in Coq.

I thank the anonymous referees for their useful comments and suggestions.

## REFERENCES

- [1] S. Bader and P. Hitzler, ‘Dimensions of neural symbolic integration - a structural survey’, in *We will show them: Essays in honour of Dov Gabbay*, ed., S. Artemov, volume 1, 167–194, King’s College, London, (2005).
- [2] S. Bader, P. Hitzler, S. Hölldobler, and A. Witzel, ‘A fully connectionist model generator for covered first-order logic programs’, in *Proceedings of the 20th International Conference On Artificial Intelligence IJCAI-07*, Hyderabad, India, (2007).
- [3] S. Bader, P. Hitzler, and A. Witzel, ‘Integrating first-order logic programs and connectionist systems — a constructive approach’, in *Proceedings of the IJCAI-05 Workshop on Neural-Symbolic Learning and Reasoning, NeSy’05*, eds., A. S. d’Avila Garcez, J. Elman, and P. Hitzler, Edinburgh, UK, (2005).
- [4] J.A. Barnden, ‘On short term information processing in connectionist theories’, *Cognition and Brain Theory*, **7**, 25–59, (1984).
- [5] Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development, Coq’Art: the Calculus of Constructions*, Springer-Verlag, 2004.
- [6] A. Bove, *General Recursion in Type Theory*, Ph.D. dissertation, Department of Computing Science, Chalmers University of Technology, 2002.
- [7] A. d’Avila Garcez, K. B. Broda, and D. M. Gabbay, *Neural-Symbolic Learning Systems: Foundations and Applications*, Springer-Verlag, 2002.
- [8] H. W. Güsgen and S. Hölldobler, ‘Connectionist inference systems’, in *Parallelization in Inference Systems*, eds., B. Fronhöfer and G. Wrightson, Springer, LNAI 590, (1992).
- [9] B. Hammer and P. Hitzler, *Perspectives of Neural-Symbolic Integration*, Studies in Computational Intelligence, Springer Verlag, 2007.
- [10] J. Harrison, *Introduction to Logic and Automated Theorem Proving*, 2004.
- [11] S. Haykin, *Neural Networks. A Comprehensive Foundation*, Macmillan College Publishing Company, 1994.
- [12] R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, 1990.
- [13] P. Hitzler, S. Hölldobler, and A. K. Seda, ‘Logic programs and connectionist networks’, *Journal of Applied Logic*, **2(3)**, 245–272, (2004).
- [14] S. Hölldobler and F. Kurfess, ‘CHCL – A connectionist inference system’, in *Parallelization in Inference Systems*, eds., B. Fronhöfer and G. Wrightson, 318 – 342, Springer, LNAI 590, (1992).
- [15] S. Hölldobler, ‘A structured connectionist unification algorithm’, in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 587–593, (1990).
- [16] S. Hölldobler and Y. Kalinke, ‘Towards a massively parallel computational model for logic programming’, in *Proceedings of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pp. 68–77. ECCAI, (1994).
- [17] S. Hölldobler, Y. Kalinke, and H. P. Storr, ‘Approximating the semantics of logic programs by recurrent neural networks’, *Applied Intelligence*, **11**, 45–58, (1999).
- [18] E. Komendantskaya, ‘First-order deduction in neural networks’, *Submitted to the Journal of Information and Computation Postproceedings volume of LATA’07*, 26 pages, (2007).
- [19] E. Komendantskaya, *Learning and Deduction in Neural Networks and Logic*, Ph.D. dissertation, Department of Mathematics, University College Cork, Ireland, 2007.
- [20] R. A. Kowalski, ‘Predicate logic as a programming language’, in *Information Processing 74*, pp. 569–574, Stockholm, North Holland, (1974).
- [21] T. E. Lange and M. G. Dyer, ‘High-level inferencing in a connectionist network’, *Connection Science*, **1**, 181 – 217, (1989).
- [22] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 2nd edn., 1987.
- [23] D. Nauck, F. Klawonn, R. Kruse, and F.Klawonn, *Foundations of Neuro-Fuzzy Systems*, John Wiley and Sons Inc., NY, 1997.
- [24] J.A. Robinson, ‘A machine-oriented logic based on resolution principle’, *Journal of ACM*, **12(1)**, 23–41, (1965).
- [25] A. K. Seda, ‘On the integration of connectionist and logic-based systems’, in *Proceedings of MFCSIT2004, Trinity College Dublin, July, 2004*, eds., T. Hurley, M. Mac an Airchinnigh, M. Schellekens, A. K. Seda, and G. Strong, volume 161 of *Electronic Notes in Theoretical Computer Science*, pp. 109–130. Elsevier, (2005).
- [26] L. Shastri and V. Ajjanagadde, ‘An optimally efficient limited inference system’, in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 563–570, (1990).
- [27] L. Shastri and V. Ajjanagadde, ‘From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony’, *Behavioural and Brain Sciences*, **16(3)**, 417–494, (1993).
- [28] P. Smolensky, ‘On the proper treatment of connectionism’, *Behavioral and Brain Sciences*, **11**, 1–74, (1988).
- [29] L. Théry. Formalisation of unification algorithm, 2008.
- [30] L. Théry. Functions for neural unification: Coq code, 2008. [www.cs.ucc.ie/~ek1/Neuron.v](http://www.cs.ucc.ie/~ek1/Neuron.v).

# The Reusable Symbol Problem

A position paper for NeSy'08

Matthew Cook  
ETH Zürich

**Abstract.** Examining the major differences between how traditional programs compute and our current understanding of how brains compute, I see only one key gap in our ability to carry out logical reasoning within neural networks using standard methods. I refer to this gap as *the reusable symbol problem*: How can neural systems use multiply-instantiatable symbols to represent arbitrary objects? This problem is fundamental and cannot be readily decomposed into simpler problems. Solving this problem would solve many individual problems such as the problem of representing relations between objects represented as neural activation patterns, the problem of implementing grammars in neural networks, and the well-known binding problem [3] for neural models. In this paper I discuss the use of reusable symbols and I give a concrete simple canonical example of the reusable symbol problem.

## Introduction

It is perfectly possible to train a neural network with logical data. However, the learned logical system typically has a fixed structure, on the order of the size of the neural network, and rules must be learned at each position in the structure where they are to be used. The network is unable to re-apply abstract logical rules at multiple locations in the structure.

Some systems expressly designed for logical reasoning have been enhanced with probabilistic capabilities, giving them many of the benefits of neural systems (e.g. [2]), but at the top level they remain rigidly structured, missing neural advantages at the highest level.

Solving the reusable symbol problem defined here would solve several of the integration challenge problems of [1]. For example, logical statements (including ones with quantifiers) can be processed just as in formal logic, using axioms and axiom schemas as the “larger patterns” (discussed below) into which reusable symbols are placed, with the sequence of statements in a proof creating a coherent structure much like the tiling in the problem discussed below. This approach is not such a new idea (note the title and date of [4]), but so far it has not been successful, simply because the reusable symbol problem has not yet been solved.

In this paper, first I will examine in detail what symbols are and how they are useful, then I will briefly dispel the illusion that pointers are the key to the power of traditional computation (as compared with neural network approaches), and finally I will give a clear instance of the reusable symbol problem.

## Symbols: a mechanism for encapsulation and reuse

The symbolic processing that arises in neural networks occurs among a fixed set of symbols with fixed relationships to each other. The fixed

set of symbols is not so worrisome, as people also tend to be content with existing symbols when manipulating information. But the fixed relationships are a more of a problem. The reason we are happy to use the symbols  $A$  and  $B$  over and over again is because we can easily remap the relationships between them. If we are told that two  $A$ 's must always be followed by a  $B$ , we can immediately understand and apply this new constraint on old symbols. Perhaps surprisingly, this is even easier to understand than using dedicated symbols for this constraint, e.g. two  $\mathcal{A}$ 's must always be followed by a  $\mathcal{B}$ .

This sort of symbol reuse matches with our experience of programming, but contrasts sharply with the kind of symbolic processing that appears in neural networks (even in modern network architectures such as graphical model based designs). In these networks, we typically do not have any notion of reuse of symbols.

We are so used to symbolic reasoning that it is worth reminding ourselves what kinds of computational primitives are implicit when one uses symbols. The most basic fact about a symbol is that it represents something. That is, there are two objects: the symbol, and the object it represents. The symbol could be a letter representing a mathematical variable, or a street sign representing a particular rule of the road, or a name representing a variable in some computer code, or a word representing an idea, or any of many other possibilities. The symbol itself is typically relatively small, while the represented object can be quite complicated. The symbol “stands for” the represented object, meaning that wherever the symbol appears, we understand that the represented object is essentially there (although this substitution may be hard to imagine in cases such as a variable name in code, or a particular bitmap representing a letter of the alphabet, or other cases where the represented object does not have any other practical form in which it could appear). This link from symbol to represented object is one-way: the represented object is not tied to the symbol, and could equally well be represented by any other available symbol.

Importantly, a symbol can appear an arbitrary number of times. Once we know how to recognize the symbol and what the symbol stands for, we are ready to use the symbol. Using the symbol means that the symbol can appear in some larger pattern which provides a context for this instantiation of the represented object. This larger pattern may be visual, or may be textual, or grammatical, or it may simply be a fixed relationship between a fixed number of objects. For example, the larger pattern could be “\_\_ comes between \_\_ and \_\_”, and this larger pattern might get filled with the symbols A, B, and C, with A representing noon, B representing morning, and C representing evening. The symbol carries the meaning of the represented object into the larger pattern, meaning that the represented object has some kind of structure, and the larger pattern indicates some kind of

structure between the elements of the pattern, and the symbol represents the presence of the represented object's structure within the structure of the larger pattern. Typically there exist constraints on the represented object imposed either by the larger pattern itself or by the larger pattern in conjunction with other represented objects that also symbolically appear in the larger pattern. For example, in the above example, if A represents noon and B represents morning, then C is fairly strongly constrained to also represent a time, perhaps a time in the afternoon or evening.

In language or imagery we are used to reasoning along lines like "this can go here, that can go there," meaning that certain symbols can fit into the larger pattern in certain places, subject to the constraint that the represented object should fit well into the larger pattern. Even when putting the pieces together of anything conceptual or abstract, we appear to operate in terms of building up coherent larger patterns of represented objects. In this construction process, why do I say we are placing symbols? Why not skip the symbols and simply place subpatterns into larger patterns? One reason is that if we reuse an object, then in the result we know that the two instances are exactly the same object, without having to inspect the subpatterns (object instances) to see if all their details match. Another reason is that pattern parts have their own structure of subparts, each part of which again has its own structure of subparts, and so on, and it seems unreasonable to assume that this unbounded detail is copied into each instance of pattern use, or even fully present in a single use. Symbols break this cycle of substructure, allowing there to be parts whose details can be recalled only if needed. Symbols also obviate the need to copy larger structures, limiting copying to duplication of symbols.

Symbols encapsulate their represented object, allowing multiple uses of the object in larger patterns. This fundamental operation is generally lacking in neural network models of information processing.

## Pointers and copying

One of the immediately noticeable differences between neural networks and computer programs is that programming languages have pointers. For example, every data structure is found using a pointer. Indeed, much of the symbolic manipulation carried out by computers is done by using pointers as symbols for the objects they point to. As with symbols, it is worth reminding ourselves how it is that pointers are useful.

In computer memory a pointer is typically not replaced by what it points to. Rather, the use of a pointer is to allow the CPU to find (copy into the ALU) parts of the pointed-to object. Once an object part has been copied to the ALU, data processing operations can be performed, perhaps calculating another pointer. Finally, if part of an object needs to change, that part is copied back into memory.

In the electronic hardware, data is represented by bits, which are in turn represented by voltage levels, which are capable of existing on (and importantly, traveling along) any set of wires. Indeed, we often think of information processing in terms of data moving around.

Pointers are simply what allow this copying to occur. The pointers themselves do not provide any great functionality – after all, neural networks don't have trouble finding information even without flexible pointers. Rather, it is the movability and copyability of the chosen data format that lies behind our current approach to electronic computing. (And it is our symbolic approach to reasoning that led us to choose such a data format in the first place, having the properties needed by symbols: copyability, and usability as the "address" of addressable memory, which is a one-way associative memory, just what

is needed for symbols.)

After decades of neuroscience electrophysiology experiments, there is no evidence that the brain uses such copying operations (although it cannot be ruled out with certainty, and there is not universal agreement). Instead, in the brain, to the extent that we understand the signals we find, they appear to have meaning based on their location (which neuron), rather than based on a spatial or temporal pattern which would have the same meaning even if coming from a completely different set of cells. This makes it unclear how symbolic reasoning is performed in the brain.

## The open problem

Here I propose a concrete canonical instance of the reusable symbol problem. Solving this instance would clearly illuminate how this problem could be solved in general.

The goal is to design a neural-style architecture (I will leave this undefined) which permits activations corresponding to solutions of a Wang tiling problem [4]. Wang tiles are square tiles (not to be rotated or flipped) with a color on each edge. A tiling must use tiles from the finite set of available reusable tiles to cover a grid. Two tiles may be used at adjacent locations in the tiling only if they have the same color on the edge where they meet.

The overall architecture of the neural network should consist of two parts: a workspace, and a set of allowed tiles. Each of these may be of a fixed size in a given network, although it should be clear how to expand the network to allow a larger workspace or a larger set of allowed tiles. The workspace should consist of some form of a grid of positions where tiles may go. The set of allowed tiles should be implemented in such a way so that if one wants to change the definition of a tile, then this change can be made in one place, within the "set of allowed tiles" portion of the architecture.

This problem is easily solved by a Markov random field if every location in the workspace knows about the set of possible tiles and their color constraints. The problem with this solution is that every location in the workspace must be independently trained to learn which tiles can go next to which other tiles. In other words, it violates the constraint that a tile definition should only appear in one place.

The challenge is to solve the problem using reusable symbols (whose implementation I also leave undefined). The workspace should be fillable with symbols which represent tiles from the set of allowed tiles, but this should only be stable when done in ways that satisfy the matching edge color constraints.

If this problem could be solved in a neurally plausible way, especially in a way that allows the set of allowed tiles to be learnable from example tilings, then this would represent a significant advance in our understanding of how neural systems can perform symbolic processing.

## REFERENCES

- [1] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler, 'The integration of connectionism and first-order knowledge representation and reasoning as a challenge for artificial intelligence', *Journal of Information*, 9(1), (2006).
- [2] Kathryn Laskey and da Costa Paulo, 'Of starships and klingons: Bayesian logic for the 23rd century', in *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pp. 346–353. AUA Press, (2005).
- [3] Christoph von der Malsburg, 'The correlation theory of brain function', Technical Report 81-2, Max Planck Institute for Biophysical Chemistry, (1981).
- [4] Hao Wang, 'Proving theorems by pattern recognition II', *Bell System Technical Journal*, 40, 1–42, (1961).

# Symbolic Computing with Incremental Mind-maps to Manage and Mine Data Streams - Some Applications

Claudine Brucks and Michael Hilker and Christoph Schommer  
and Cynthia Wagner and Ralph Weires<sup>1</sup>

**Abstract.** In our understanding, a mind-map is an adaptive engine that basically works incrementally on the fundament of existing transactional streams. Generally, mind-maps consist of symbolic cells that are connected with each other and that become either stronger or weaker depending on the transactional stream. Based on the underlying biologic principle, these symbolic cells and their connections as well may adaptively survive or die, forming different cell agglomerates of arbitrary size. In this work, we intend to prove mind-maps' eligibility following diverse application scenarios, for example being an underlying management system to represent normal and abnormal traffic behaviour in computer networks, supporting the detection of the user behaviour within search engines, or being a hidden communication layer for natural language interaction.

## 1 ABOUT MIND-MAPS

Transactional streams are to be understood as an endless flow of data that is lost once it is read. Furthermore, the data can be classified into categories, for example sentences or paragraphs inside a text document. These categories form the transaction, having items inside, for example words or paraphrases. An example for a transactional stream may be the reading of a book, where the text is read exactly once but lost if it is pronounced. The management, and moreover, the analysis of transactional streams is often problematic for several reasons. One of them is that data streams are potentially infinite or at least their end is not known until it is actually reached. Storing the whole data stream is therefore not an option, and the analysis cannot rely on traditional mining techniques that require the whole dataset to be available or that need random access or multiple passes over the data. Currently a lot of research focuses on the processing of such streams of different kind. Some of the typical techniques used with data streams are sliding windows, incremental approaches, or synopses of the data. Surveys of current methods and issues can be found in [4], [7], [9], [18].

With the discussion around mind-maps, we argue for its eligibility by demonstrating its applicability on a couple of algorithmic ideas. In our understanding, mind-maps are to be seen as adaptive and incremental knowledge structures, which live from depending on the occurrence of an input stream. A first approach in stream data analysis with mind-maps had been done in the processing of transactional streams with the creation of mini-networks. These base on transactional data ([19]), the mini-network consists of simple

symbolic cells that share a weight value and that represent an individual item in a transaction. The symbolic cells are interconnected with other cells that occur in the transaction as well. In a subsequent step, the mini-network becomes integrated to the mind-map itself, where those cells become merged with those in the mind-map in case that they are identical (merge). Using the fundamental principle of adaptation and Hebbian Learning, the mind-map can be seen as a living engine as it is initially empty but grows over time. Since the states of connections and cells change over time the cells may die or revive as well. Focusing on the skeleton of the mind-map, a retrieve yields on delivering the strongest cell connections.

Although mind-maps often refer to such a structure to process fluid signals like text streams, mind-maps often claim to have its associative nature as the fundamental principle. This is true, however, we believe that a temporal character of such systems may be accepted as well to manage temporal stimulation that comes in. Secondly, mind-maps can be seen both from a verificative and an explorative perception: talking about mind-map software mostly refers to a top-down directed production of logically connected entities, for example work-flows or coherent cogitations ([5], [8], [10]), whereas the word of explorative more related to a learning or discovering process of such logical structures. In this respect, the following applications refers to the second class of mind-maps and present some algorithmic ideas for the temporal processing of text streams to map contextual information. A simple implementation of mind-maps is the system ANIMA. It refers to a mind-map model of incremental and adaptive nature and allows to manage associations between symbolic cells while having a transaction stream as input. The aim of ANIMA is the efficient processing and management of transactions over time, to present related patterns inside a stream [19].

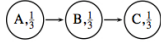
## 2 APPLICATIONS

### 2.1 TARGETING NETWORK PROTECTION

Network-based anomaly detection [11] [21] refers to a system-based understanding of what the structure and the behaviour of network traffic is, and in this respect to identify abnormal situations. Here, the mind-map model ANIMA-AR is implemented to represent network traffic events while having network packets including header and content as one transaction, is one promising application. It fragments such packet transactions into meaningful symbolic cells within the mind-map and connects the packet cells. Additionally, connection values are established relating to the corresponding frequencies, respectively. Due to the architecture, we may rate the usual network traffic by a lower connection weight - although the frequency is high - but may rate an abnormal/unusual network behaviour by a higher

<sup>1</sup> University of Luxembourg, Campus Kirchberg, Dept. of Computer Science and Communication (CSC). MINE Research Group @ ILIAS Laboratory. Address: 6, Rue Coudenhove Kalergi, 1359 Luxembourg, Luxembourg. Email: { name . surname } @ uni.lu

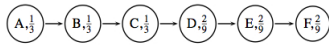
value - although it appears more seldom. We identify the abnormal traffic as it is rated significantly and therefore tolerant against temporal connection updates between the symbolic cells. The following sequence of pictures gives an example on how the mind-map is used; it refers to the management of bad signatures as described in [11], taking several insertion rules into account. First, the incoming signature ABC is considered and assigned to symbolic cells, being equal-weighted.



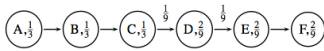
Then, if a new signature CDEF is added to the mini-network, and the weights are adapted. At each step, substrings are considered and evaluated as follows: given a sub-string, then if the sum of all activation states ...

- ... is exactly 1, then a virus alert takes place.
- ... is exactly 0, then no virus alert takes place.
- ... is between 0 and 1, an alert takes place with a probability value.
- ... increases 1, then it is not considered.

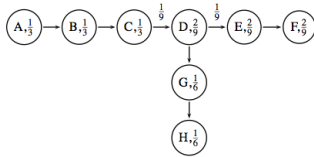
For example, ABC forces a virus alert as it is infected for 100%, whereas only H is unlikely to be infected. To continue the example, as C is already known, this value stays constantly.



The following photographs of the mind-map refers to the situation when the signatures CDE



and CDGH are being inserted. With this, the probability values for each signatures is clearly available through the whole life-time of the mind-map. More information can be found in [11].



The mind-map model ANIMA-AR is implemented to detect well-known viruses. The signature of viruses is stored in a graph-like structure; virus signatures are managed and stored, incoming packets - to identify intrusions - evaluated. The scanning speed and the required storage space outperforms current approaches and emerges out of the compression of the signature database. ANIMA-AR is theoretically analysed showing that viruses and similarities are detected. Simulations substantiate the theoretical analysis and show the low false-positive rate tolerating the normal system. In addition, ANIMA-AR is able to automatically detect similar viruses as small mutations or new variants.

## 2.2 MANAGING IMPLICIT FEEDBACK

The consideration of implicit feedback in the field of information retrieval and the automatic collection of information about the user's

behaviour [15] [24] is an application of interest. Without an explicitly request of information, we intend to gain some information about what really interesting is. The aim is to use the mind-map as an adaptive storage for such kind of information and consequently, for the enhancement of user-based research requests. We therefore monitor the user's behaviour in interaction with a search engine, keeping an eye on queries and their results, links that the user follows, and diverse time-related information, for example how long he or she stays on web sites. Search sessions like this are considered as single transactions for the mind-map. And, building up such a network provides information about typical queries, results, and measures, especially regarding the relevance of search results, namely towards an enhancement of further queries.

An example might be a giving of additional hints or altering the results themselves. One concrete step is a re-ranking of search results, which may help to place those results further on the top of the list that are probably of greater importance according to the given query. Following this, the strengthens of the mind-map are mainly due to the ability to cope with transactional streaming data. We concern with information about search sessions of users, which can easily be broken down into transactions. Moreover, the mind-map is able to store only the most important aspects of the information without the need of storing all feedback data. This helps to keep the network at a reasonable size. Furthermore, the dynamic nature of this mind-map fits quite well to the purpose of this approach: if there is a change of the user's feedback over time, then this trend will be reflected in the mind-map as well. Figure 1 shows an architectural snapshot of the mind-map, where we use three different types of cells:

- The query terms are single terms that are observed in user queries.
- Queries that form the transactional input to the mind-map.
- The resulting list of documents that have been provided by the underlying search engine for one or more queries.

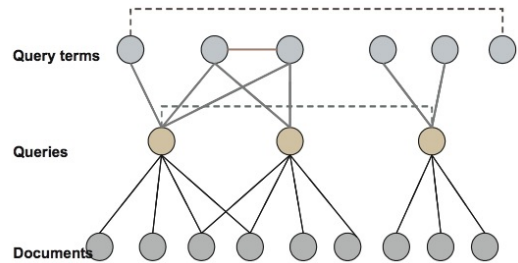


Figure 1. Architecture of the mind-map [24]

There may exist different connections between these nodes, which might be weighted indicating the strength of the relationship. These are for example connections that indicate relationships between different query terms, connections between queries and query terms, and connections between queries and documents. More information can be found in [24] [25].

## 2.3 TARGETING DBLP

Bibliographical databases such as Citeseer, Google Scholar, and DBLP serve as a bibliographic source with lots of information concerning a publication. This compounds the names



of the authors, the publication title, the conference, and many other attributes. A bibliography database is accessible online, where all entries share an electronic index to articles, journals, magazines, etc. containing citations and abstracts. Understanding the bibliographic database as a digital collection that is intelligently managed and that supports a search for and the retrieve of bibliographic information to defined queries is a convenient procedure in demanding information right in time. Regularly, the retrieve bases on a collection of queries that consists of keywords in the publication title or the keyword list. For example with DBLP, the querying using a keyword plagiarism leads to an answer set of almost 70 articles, and a search refinement with detection and pattern to 34 and 2 bibliographic entries, respectively. Accordingly, the two referenced publications pledge close; and the names of the authors overlap:

- 1 NamOh Kang, Sang-Yong Han: Document Copy Detection System Based on Plagiarism Patterns. CICLing 2006: 571-574.
- 2 NamOh Kang, Alexander F. Gelbukh, Sang-Yong Han: PPChecker: Plagiarism Pattern Checker in Document Copy Detection. TSD 2006: 661-667.

With this, we reference to a graph structure representing the association of the three authors Han, Kang and Gelbukh. This is similar to [12] who introduces mini-networks. The double edges signalise a double connection as the single (and parallel) edge between Han to Gelbukh and Kang to Gelbukh refers to a one-way association. Whereas the meaning of a double connection is unique while having two publications, the double entries to Gelbukh seem to be ambiguous: on the one side, it refers to one common publication with both individually, on the other to two single publications with one of the other authors. However, the graph is node-oriented in a way that it simply represents the situation as it is: Gelbukh has one common publication with both of them, and here, it plays no role if this is a common one or not.

For static databases, the discovery of associative patterns has been an area of extensive research, and multiple approaches and solutions to the static problem have been presented in the past. A major problem in these approaches is the combinatorial explosion of the search space and the research has therefore mainly focused on reducing this space. Since mind-maps are targeted to data streams, it can not possibly make use of the methods developed for static databases, since these algorithms require multiple passes over the data to calculate frequencies of associated items. This is indeed not acceptable when dealing with data streams, because data streams are potentially infinite, or at the very least their end cannot be foreseen. In this respect, the idea of searching for temporal patterns in a bibliographic database like DBLP while taking the time as the core medium leads to a couple of interesting questions, for example

- In general, may we discover scientific communities? While observing the visualisation of associative relationships between authors, we might ask if such dependencies generally form a community, and secondly, how strong these communities may be. Furthermore, if dependencies of of communities exist, are these temporal or visiting, recurring, or constant as it has been mentioned in section 1?
- Do there exist diverse trends in publishing? For example, the occurrence of a common publication may be the initiator for a fruitful collaboration (which is proven by following publications on the same or a different research topic)

With this, we may perform mind-mapping over a period of time, for example moving a corresponding window temporarily over time.

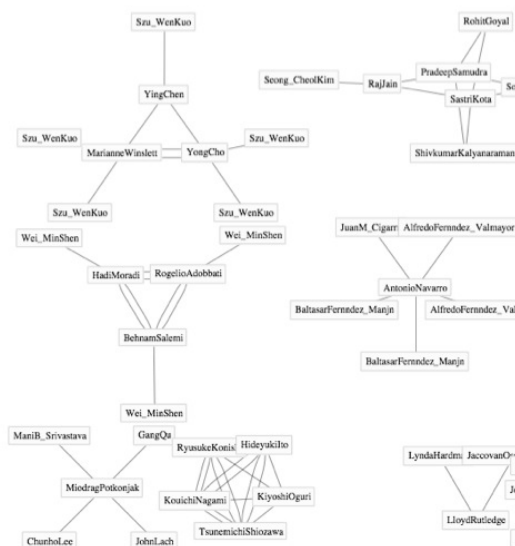


Figure 2. Temporary mind-map (DBLP, year of 1993), consisting of associated author nodes.

## 2.4 SEMANTIC NET-LEARNING

The mind-map model WYWI stands for a simple communication paradigm that focus on natural language communication. The model uses a mind-map to manage words and their relationship to others associatively. A sentence is read incrementally and threatened as a transaction with concepts and roles. Currently, only adjectives, nouns, and verbs are considered as worth, they are extracted and put into the mind-map as a semantic structure. Adjectives are considered as sub-concepts of nouns. For example,

```
#S (CONCEPT :NAME MAN :CAT N :FATHER (ROOT)
:CHILDREN (YOUNG) :ROLES (READ) :ACT 0.9577)
#S (ROLE :NAME READ :CAT V :CONNECTION ((MAN
BOOK 0.9577)) :ACT 0.9577)
#S (ROLE :NAME SEE :CAT V :CONNECTION ((LION
PETER 0.8253)) :ACT 0.9014)
```

Here, the word READ acts as relationship (role) connects both MAN and BOOK, sharing a connection and activation value of 0.9577, respectively. Other roles exist, but are unrelated. Each time-step, the activation values are decreased unless the role or the associated concepts remain inactive. As for the ANIMA mind-map model, the underlying net structure is refreshed after each transaction; it adapts to the given stimulation. However, WYWI does not use any extern source (for example thesaurus) but actively interacts with the user in case a word is unknown. This is to understand the morphologic basis or the lexical category. WYWI also establishes a semantic representation through micro-features: here, the user must rate a word semantically against given (predefined) entities. This produces a k-dimensional space, having k as the number of independent micro-features, for example

```
...
MAN is ((1.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0 ...))
LION is ((0.0 0.0 0.0 0.0 0.7 0.0 0.4 ...))
...
```

The communication in WYWI is performed as a conversation. For a sentence like The yellow lion sees Peter, the word

YELLOW becomes a sub-concept of a word LION, whereas LION and PETER are connected by SEE. Additionally, SEE is becomes higher activated. Additionally, all concepts are represented in the k-space. As a consequence, transactional (dialogue) stream can be processed and queries be sent at each time.

```

...
WYWI: (ENTER THE NEXT WORD OR CLOSE BY '-')
USER: Peter
WYWI: COULD YOU PLEASE GIVE SOME EXPLANATION,
      LIKE THE MORPHOLOGIC BASIS?
USER: Peter
WYWI: AND WHAT IS THE LEXICAL BASIS OF peter?
USER: person
WYWI: I NEED SOME SEMANTIC INFORMATION ABOUT
      peter. HAVE A LOOK AND RATE peter FROM
      0 to 1:
      TALL?
USER: 0.9
WYWI: And with HEAVY?
...

```

An extended version of the semantic net-learning from text streams is to define a temporal mind-map for a certain actor of a text: where the text stream is consolidated over time by a semantic graph structure. In this work, an important issue for building an author-related mind-map is the resolution of linguistic anaphors.

So far, the major idea is to process each sentence incrementally so that a general and pre-defined scheme structure of subject - verb - object is instantiated every time ([14]). At the moment, we only concern with nouns, verbs, and adjectives, where other linguistic categories are disregarded. In this respect, verbs act as a role between the concepts (noun, object). Each concept may have sub-concepts that correspond to attributes, for example the sentence the old man likes the green juicy grassland is translated into

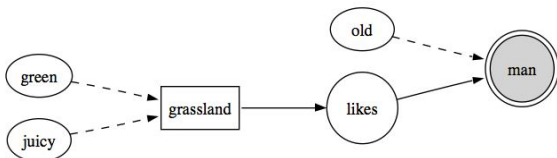


Figure 3. A semantic graph having man as the centric concept. Attributes are adjunct by dashed lines, whereas roles are connected as a circle.

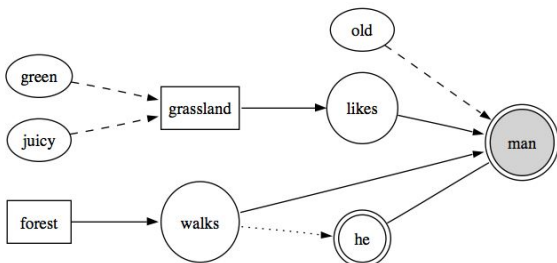


Figure 4. The anaphor he is recognized as to be related to man, whereas the two sentence structures are merged by the concept man.

where man and grassland represent main concepts and green, juicy, and old sub-concepts. The process of relating identical concepts together is accomplished by a matching of identical words and a resolution of linguistic anaphors ([13], [17]). The following Figure 4 shows the semantic structure after having read the sentence He walks into the forest.

So far, the incremental processing of texts can be stopped at any moment. The semantic structure is a mind-map with concepts and relationships of consistent states. Beside focusing on the elaboration of methods to realize pronominal anaphora and co-reference in text streams, we will assign weight values to concepts and roles in order to prove their importance for an actor.

The idea of content zoning is to refer to a segmentation of a text document into semantic zones. As indicated in [3] and moreover as firstly discussed in [22] with Argumentative Zoning, the basic idea here is to structure texts on the basis on pre-defined categories. An example might be the following text, having the actors Harry, Hedwig, and owl:

*“Harry got up off the floor, stretched, and moved across to his desk. Hedwig made no movement as she began to flick through newspapers, throwing them into the rubbish pile one by one. The owl was asleep or else faking; she was angry with Harry about the limited amount of time she was allowed out of her cage at the moment. As her neared the bottom of the pile of newspapers. Harry slowed down, searching for one particular issue that he knew had arrived shortly after he had returned to Privet Drive for the summer, he remembered that there had been a small mention on the front about the resignation of Charity Burbage, the Muggle Studies teacher at Hogwarts.”*

an then be zoned to

- Harry <ACTOR=HARRY> got up off the floor, stretched, and moved across to his (→his ⇒ HARRY) desk. </ACTOR=HARRY>
- Hedwig <ACTOR=HEDWIG> made no movement as she (→she = HEDWIG) began to flick through newspapers, throwing them (→them = newspapers) into the rubbish pile one by one.
- The owl <ACTOR=OWL> was asleep or else faking; she (→she = OWL) was angry with Harry (→Harry = HARRY) about the limited amount of time she (→she = OWL) was allowed out of her (→her ⇒ OWL) cage at the moment. As her (→her ⇒ OWL) neared the bottom of the pile of newspapers, </ACTOR=OWL>.
- Harry<ACTOR=HARRY> slowed down, searching for one particular issue that he (→he = HARRY) knew had arrived shortly after he (→he = HARRY) had returned to Privet Drive for the summer, he (→he = HARRY) remembered that there had been a small mention on the front about the resignation of Charity Burbage, the Muggle Studies teacher at Hogwarts. At last he (→he = HARRY) found it.</ACTOR=HARRY>

where linguistic anaphors are solved depending on the current actor, the gender, and/or between different candidates that have already occurred in the text. Knowing that the owl has the name Hedwig, and moreover, having a hierarchy on site, the zoning can be updated even more. An additional analysis can be performed to gain further information about the separate zones. Rather simple to extract information could be statistics about the size and layout of zones, but a more sophisticated analysis of their text content is possible. The latter can lead to an extraction of the semantic content and purpose of a zone. Such kind of information can be used for various purposes, such as comparing documents to each other (regarding their analysed zone structure and content) by simply referring to information about the zones as zone variables ([3]).

We apply content zoning to text streams in order to establish a semantic mind-map, while using a sliding window of user-specified

length; text is buffered in, immediately zoned and analysed. Intermediate statistical results are managed to produce a user specific summary on a given subject. The definition of zones is insufficient to an effective zoning as zones only indicate a position of an information in the text, but do less give information about the content. A solution might be the introduction of zone variables, which describe the content of the input stream and are the core parameters for the summary generation. In this respect, we concern with two categories of zones, namely *document independent* and *document dependent* zones, for example the position of the zone in the text stream, its length or the most occurring word (without stopwords). During the zoning process, nearly all sentences are attributed to zones. Useless sentences and sentences which cannot be attributed to a zone are skipped or can be regrouped a user-defined zone. After different steps as anaphor resolution for example, the values of the variables are used for statistical evaluations to generate a summary on a user specific subject. During the process of buffering and processing the text streams, there is an option of real-time evaluation, so that changing values are immediately visible to the user. At each instant, the user has the possibility to call a summary on a previously defined subject, as for example an actor of a fairy tale. But, also less sophisticated results as the top most occurring words or collocations can be called at each moment.

### 3 CONCLUSIONS

A mind-map is an adaptive engine that basically works incrementally on the fundament of transactional streams. Following our model, mind-maps consist of symbolic neural cells that are connected with each other and that become either stronger or weaker depending on the transactional stream: based on the underlying biologic principle, these symbolic cells and their connections as well may adaptively survive or die, forming different cell agglomerates of arbitrary size.

With that, mind-maps may be applicable for the management of trust as well: every human shares an own attitude about others, for example *Person R* has an attitude to *Person S* and vice versa. Both are probably different from each other and different to *Person T*'s view to *Person R* or *Person S*. Furthermore, one might get the conclusion if *Person R* trusts *Person S* but not vice versa. With this approach, we may follow a proposition by [16] who suggests a model on human conversations. The attitude of someone's mind is modelled as a self-organising mind-map. Every person has a model of his/her view of the world and models for other people whom he/she has interacted with. All views including a self view and views to other persons (others) will be modified through conversations between people. We therefore intend to introduce an engine to find regularities between human's objects and to model trust based on this. The creation of an artificial mind-map, where a textual data stream is read and represented in an associative dynamic network. Each incoming stream is decomposed to its items, for example a text stream may be decomposed to words.

### ACKNOWLEDGEMENTS

This work has been done at the MINE research group at the Laboratory for Intelligent and Adaptive Systems across several research projects funded by the University of Luxembourg and the Ministry of Higher Education.

### REFERENCES

- [1] D. J. Abadi, D. Carney, U. Centitemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonkik: Aurora: a new model and architecture for data stream management. *The VLDB Journal*. 2003.
- [2] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom: STREAM - The Stanford Data Stream Management System, Stanford University, 2004.
- [3] C. Brucks, M. Hilker, C. Schommer, C. Wagner, and R. Weires: Semi-automated Content Zoning of Spam Emails. *Lecture Notes on Business Information Processing* (Springer). 2008. (to appear)
- [4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom: Models and issues in data stream systems. In *Proceedings of PODS*, 2002.
- [5] K. Chen-Chuan Chang, H. Garcia-Molina: Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Sources. *SIGMOD Conference 1999*: 335-346.
- [6] Digital Bibliography & Library Project, Computer Science Bibliography, University of Trier, <http://dblp.uni-trier.de/>.
- [7] P. Domingos, and G. Hulten: Catching up with the data. Research issues in mining data streams. In *Workshop on Research Issues in Data Mining and Knowledge Discovery*, Santa Barbara, CA, USA, May 2001.
- [8] M. J. Eppler: A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing. *Information Visualization* 5(3): 202-210 (2006).
- [9] L. Golab, and M. T. Ozsu: Issues in data stream management. *SIGMOD Record*, 32(2):514, June 2003.
- [10] L. M. Haas: Review - Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Sources. *ACM SIGMOD Digital Review*.
- [11] M. Hilker, and C. Schommer: Description of bad-signatures for network intrusion detection. In *AISW-NetSec 2006 during ACSW 2006, CRPIT*, volume 54, Hobart, Australia, 2006.
- [12] C. Kemke. *Connectionist Parsing with Dynamic Neural Networks: Technical Report*, Computing Research Laboratory, New Mexico State University, 2001.
- [13] S. Lappin and H. J. Leass: An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4):535-561, 1994.
- [14] J. Leskovec, N. Milic-Frayling, and M. Grobelnik: Impact of linguistic analysis on the semantic graph coverage and learning of document extracts. In M. M. Veloso and S. Kambhampati, editors, *AAAI*, pages 1069-1074. AAAI Press / The MIT Press, 2005.
- [15] L. Liu, C. Pu, and W. Tang: Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE Knowledge and Data Engineering*, 1999.
- [16] C. Miller: *Modeling Trust in Human Conversation*. 2005. Master Thesis, Massachusetts Institute of Technology, Dept. of Computer Science and Engineering.
- [17] R. Mitkov: Robust pronoun resolution with limited knowledge. In *COLING-ACL*, pages 869-875, 1998.
- [18] S. Muthukrishnan: Data streams - Algorithms and applications. In *Proceedings of the 14<sup>th</sup> annual ACM-SIAM symposium on discrete algorithms*, 2003.
- [19] C. Schommer: Incremental Discovery of Association Rules with Dynamic Neural Cells. *Proceedings of the Workshop on Symbolic Networks. ECAI 2004*, Valencia, Spain, 2004.
- [20] C. Schommer, and B. Schroeder: ANIMA - Associate Memories for Categorical data Streams. *Proceedings of the 3<sup>rd</sup> International Conference on Computer Science and its Applications (ICCSA-2005)*. San Diego, USA.
- [21] M. Sullivan: A Stream Database Manager for Network Traffic Analysis, Bell Communications Research, Morristown, NJ 07960, 22nd VLDB Conference, Mumbai, India, 1996.
- [22] S. Teufel: *Argumentative Zoning: Information Extraction from Scientific Text*. 1999. Phd Thesis, University of Edinburgh, England.
- [23] Ö. Uzuner, R. Davis, B. Katz: Using Empirical Methods for Evaluating Expression and Content Similarity. MIT, Computer Science and artificial intelligence Laboratory, 200 Technology Sq. NE43-825
- [24] R. Weires. B. Schroeder, and M. Hilker. Dynamic association networks in information management. In *Proceedings of the 4<sup>th</sup> International Conference on Machine Learning and Data Analysis (MLDA)*. 2007.
- [25] R. Weires, C. Schommer, and S. Kaufmann: SEREBIF - Search Engine Result Enhancement by Implicit Feedback. *4<sup>th</sup> Intl Conference on Web Information Systems and Technologies (WebIst 2008)*. Poster Presentation. Funchal, Madeira. 2008.

# Guiding Backprop by Inserting Rules

Sebastian Bader<sup>1</sup> and Steffen Hölldobler<sup>2</sup> and Nuno C. Marques<sup>3</sup>

**Abstract.** We report on an experiment where we inserted symbolic rules into a neural network during the training process. This was done to guide the learning and to help escape local minima. The rules are constructed by analysing the errors made by the network after training. This process can be repeated, which allows to improve the network performance again and again. We propose a general framework and provide a proof of concept of the usefulness of our approach.

## 1 Introduction and Motivation

Rule insertion prior to training can lead to faster convergence and to better results (see e.g. [9, 4]). Here, we will investigate whether we can do this repeatedly during the training process.

Artificial neural networks are a very powerful tool to learn from examples in high dimensional and noisy data. Standard feed-forward networks together with back-propagation of errors have been successfully applied in many domains. However, in most domains, at least some background knowledge in form of symbolic rules is available. And this knowledge can not be used easily. As mentioned above, there have been studies in which this knowledge is embedded prior to the training. But there is no way to guide the learning process using this rules.

The following observations have triggered this research:

- Very general rules, i.e., those that cover many training samples, are quickly acquired by back-propagation. And hence, embedding those rules does not help too much.
- Very specific rules that are embedded prior to the training will very likely be overwritten by newly learned rules.
- We can analyse the errors made by the network to obtain correcting rules.

In the area of natural language processing, the combination of rule based and machine learning based methods seems to be very promising. Recently, we showed that we can accuracy by inserting symbolic rules into an artificial neural network [7]. We believe that we can even improve even further if we embed rules repeatedly during the training. In this paper, we will describe how to do this and why we believe that this seems to be a good idea.

We propose a general method to guide the training of artificial neural networks. Our approach is based on the repeated embedding of rules during the training. Here, we will focus on the description of an experiment to verify our claims. This paper is only meant to be a case study and to serve as a proof of concept. And shall be the starting point for a bigger investigation. We are currently preparing larger experiments in the area of natural language processing.

<sup>1</sup> International Center for Computational Logic (ICCL), Technische Universität Dresden, Germany, email: sebastian.bader@inf.tu-dresden.de

<sup>2</sup> ICCL, email: sh@iccl.tu-dresden.de

<sup>3</sup> Centria, DI-FCT-UNL, Lisbon, Portugal, nmm@di.fct.unl.pt. Work developed while the author was a visiting the ICCL.

## 2 Preliminaries

We assume the reader to be familiar with basic concepts of neural networks and the training by back-propagation as e.g., described in [8] and implemented in [10]. Here, we will only use simple three layer fully connected feed-forward neural networks applying the tanh-function in hidden and output layer. We will train them using back-propagation without enhancements like e.g., momentum and fixed learning parameters.

The rules we will use here, are simple propositional if-then rules. Each rule consists of multiple preconditions which have to be satisfied simultaneously, and a single atomic consequence. Below we will study a classification task for tic-tac-toe boards, hence the consequence will be the class and the preconditions will be (partial) board descriptions.

## 3 A General Method for Guiding Backprop

Our approach, is based on the repeated modification of the network during the training. After a number of training cycles, the errors made by the network are analysed. This analysis should yield a rule that can be used to correct some of the errors. We can now embed the rule into the network and continue with the next epoch. This process is depicted in Figure 1. As we will see below, this will help the network to escape local minima during the training. To summarise, our approach works as follows:

1. Initialise the network.
2. Repeat until some stopping condition is satisfied:
  - (a) Train the network for a given number of cycles.
  - (b) Analyse the errors of the network to obtain correcting rules.
  - (c) Embed the rule(s) into the network.

In the sequel, we will show that this works using a simple classification task. This is only meant to be a proof of concept, and not to show the superiority of this method in general.

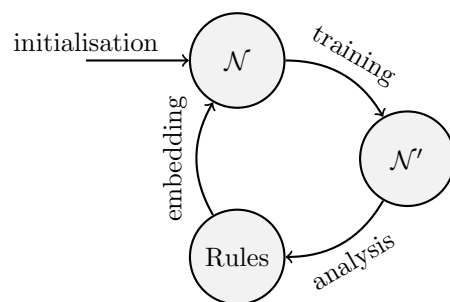


Figure 1. The rule-insertion cycle.

## 4 The "Tic-Tac-Toe" Classification Task

We used the "Tic-Tac-Toe Endgame Data Set" of the UCI Machine Learning Repository [1] for our experiments described below. The data set contains all 958 possible board configurations that can be reached while playing Tic-Tac-Toe starting with player X. Each configuration contains a description of the board and is classified as win or no-win for player X. A player wins the game, if he manages to place 3 of his pieces in a line. The board contains  $3 \times 3$  cells and each cell can be marked by either x or o, or can be blank (b). Possible samples are  $[x, b, o, o, x, o, x, b, x]^+$ ,  $[x, x, o, o, o, x, x, o, x]^+$  and  $[x, o, x, o, o, b, x, o, x]^+$ . The corresponding boards are shown in Figure 2 from left to right. The goal of the task is to decide whether a given board is a win-situation for player X or not.

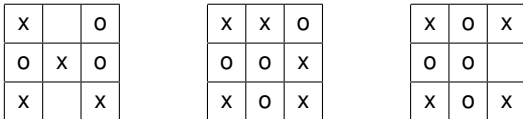


Figure 2. Three board configurations, showing a win for player X, a draw and a win for player O, i.e., this are one positive and two negative examples.

### 4.1 A Network to Classify Board States

We used a simple 3-layer feed-forward network with tanh as activation function in all units. It contains  $9 \times 3 = 27$  units in the input layer, that is three for each of the nine position on the board, corresponding to the three possible states. For each of this triples we used an 1-out-of-3 activation schema. If a cell contains for example an "x" the x-unit for this cell will be activated by setting the to output 1, while the o and b-units are set to -1. Initially, the network contains one hidden units and one output unit. A board state is fed into the network by activating the appropriate input units. This activation pattern is propagated through the network and the networks decision can be read from the output unit. If its activation is greater than 0 the network evaluates the board as a win for player X, and as a no-win otherwise. Figure 3 shows the network used in our experiments. We will now continue by detailing the general steps of our method as introduced in Section 3.

**Initialising the Network** We initialised the network with a single hidden units and all connections as well as the bias of the units were randomised to values between  $-0.2$  and  $0.2$ .

**Training the Network** The network was trained using standard back-propagation in the SNN Simulator [10]. We used simple back-propagation without any additions. The learning rate was set to  $\eta = 0.1$ . During each epoch we presented all samples 100 times to the network.

**Analysis of the Errors to Obtain Correcting Rules** After the training, we presented all samples again to the network and compared the networks output to the desired output. All samples where the difference between the computed and the desired output was greater than 0.5 were collected and grouped into positive and negative samples. Depending on whether there have been more errors on positive or on negative samples, we continued with the bigger set. From this set, we constructed a template that is as accurate as possible and at the same time as general as possible. This was done by repeatedly selecting a

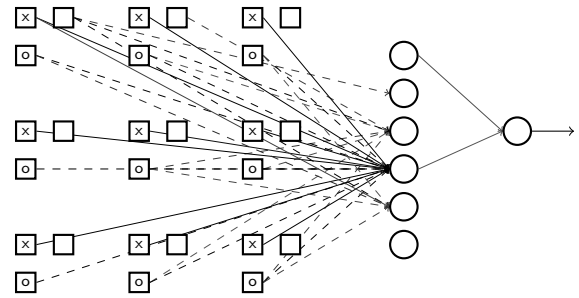


Figure 3. A 3-layer fully connected feed-forward network with 27 input (rectangular), 6 hidden (cyclic) and a single output (cyclic) unit to classify Tic-Tac-Toe boards. The 27 input units are arranged like the board itself. Positive connections are shown as solid lines, while negative ones dashed. The width corresponds to the strength and small connections are omitted.

cell and a value such that most erroneous samples agree on this. This is a variant of the "Learn one rule"-algorithm as used in sequential covering algorithms, but considers only one class [8].

The following rule was constructed from 99 wrongly classified sample:  $[b_{13}, b_3, o_{23}, x_{43}, x_{34}, x_{70}, o_7, b_1, b_1] \mapsto +$ . The subscripts show the percentage of the 99 samples which agree on the value. E.g., 13% of those samples had an empty upper left corner and 34% an x in the centre. We constructed the rule template by ignoring all entries with a support  $< 20\%$ . This resulted in the following template  $[?, ?, o, x, x, x, ?, ?, ?] \mapsto +$ , which actually covers 38 of the samples.

**Embedding Rules into the Network** The rules constructed above, are embedded into the network following the general ideas of the *Core Method* as specified in [5], [4] and [3]. A new hidden unit is inserted into the network and the connections and the bias are set up such that this unit becomes active if and only if the input of the network coincides with the rules precondition.

Let us use the rule  $[?, ?, o, x, x, x, ?, ?, ?] \mapsto +$  to exemplify the idea. For all entries different from  $?$ , a connection with weight  $\omega$  from the input unit corresponding to the value and connections with weight  $-\omega$  from the other two units are created. E.g., the x-unit for the central cell is connected to the new hidden unit with weight  $\omega$ , while the corresponding o and b-units are connected with weight  $-\omega$ . All connections from input units for cells marked  $?$  in the template have been initialised to 0.0. And the connection to the output unit has been set to  $\omega$  for positive rules and to  $-\omega$  otherwise. The result is shown in Figure 4. Finally, all connections have been slightly disturbed by adding some small random noise from  $[-0.05, 0.05]$ . In the experiments described below we used different values for  $\omega$ .

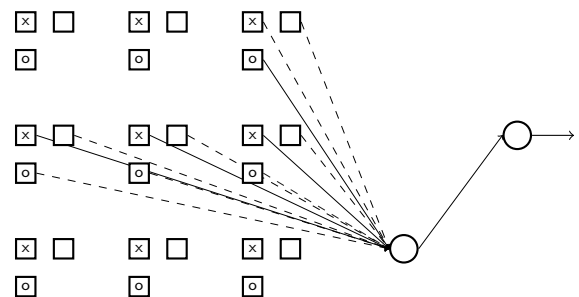


Figure 4. The embedding of the rule  $[?, ?, o, x, x, x, ?, ?, ?] \mapsto +$ . Connections are depicted as described in Figure 3.

## 4.2 Experimental Evaluation

We performed several experiments to evaluate our method, varying the parameter  $\omega$ . We used  $\omega \in \{0.0, 0.5, 1.0, 2.0, 5.0\}$ . By setting it to 0.0, we can emulate the simple addition of a free hidden unit during the training, i.e., the new unit will be initialised randomly. Therefore, this case should be the baseline for our experiments. Small values would only push the network slightly into the direction of a rule, while a value of 5.0 would strongly enforce the rule. In particular, if a rule was embedded with  $\omega = 5.0$ , then as soon as the corresponding hidden unit became active its output will very likely dominate the sum of the outputs of the other hidden unit. This is due to the fact that its activation of approximately 1.0 is propagated through a connection with a weight of 5.0 or  $-5.0$  to the output unit. Therefore, the output unit will usually simply follow the rule. Consequently, as a rule of thumb, only rules where one is absolutely certain should be embedded with very large  $\omega$ .

Figure 5 shows the result of the experiment. It shows the mean squared error over the time for the different settings of  $\omega$ . For each value, we repeated the experiment 50 times and showed the averages. It also shows a zoom into the lower right corner, i.e., into the final cycles. Here, not only the averages, but also the standard deviation is shown. There are a few points to notice in those plots, which will be discussed in more detail below:

- For  $\omega = 0$  the network stops to improve at some point, even if units are added.
- For  $\omega > 0$  the network outperforms the network where  $\omega$  was set to 0 in the later cycles.
- For  $\omega = 5$  the network did not learn very well.

Due to local minima in the error surface, back-propagation can get stuck in non-optimal solutions. Usually this is the case if there are only errors on few samples, because back-propagation follows in principle the majority vote. This also happens in our experiments. For  $\omega = 0$ , the network fails to improve further and remains at the 0.01-level. Our method, proposed here, can help to solve this problem by inserting a unit which covers those few samples. Thus, we allow back-propagation to jump to better solutions.

The bad performance for  $\omega = 5$ , is due to the above mentioned fact about big values for  $\omega$  and the fact that our rules are not necessarily correct. It happened several times in our experiments that the same (only partially correct) rule was embedded again and again. E.g., the rule  $[?, ?, x, ?, ?, ?, x, ?, ?] \mapsto +$  was embedded, because most of the positive samples on which there were errors agreed on those values. But at the same time there are 42 negative samples with those values. As the network blindly followed the rule, those negative samples are wrongly classified afterwards. During the training phase, the network basically unlearned the rule but could not improve on the original wrong positive samples. And thus, the same bad rule was embedded again and again. This problem could be solved using e.g., tabu-lists preventing the re-insertion. But we believe that better rule-generation should be used, which construct only valid rules.

There is another finding in our experiments which is worth to be mentioned. E.g., the rule discussed in Section 4.1, covers only 38 samples, but the number of errors of the network on the training data went down from 99 to 26, i.e., inserting the rule apparently helped to better classify 73 samples. We think this is due to the better starting point provided by the insertion of the rule, that helps the network to correct errors more efficiently. This effect will be subject to further studies because it seems to improve the performance to a level which cannot be achieved by symbolic or connectionist learning alone.

## 5 Conclusions and Further Work

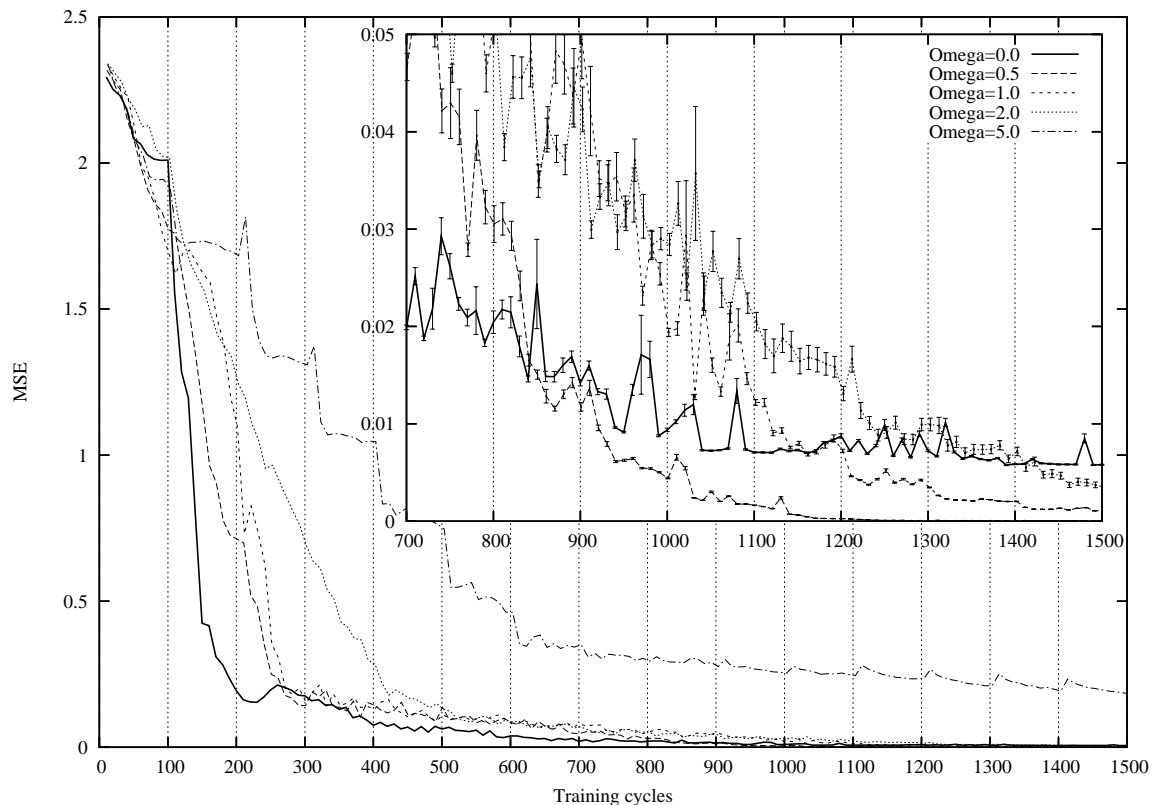
We presented a first experiment to support our claim that we can positively influence the training process of a feed-forward neural network by incorporating knowledge in the form of rules into the training process. Thus, we make a first attempt to solve the challenge-problem number 4 mentioned in [2], i.e., the improvement of established learning algorithms by using symbolic rules. Here, the rules were obtained from wrongly classified examples after an epoch of training the network using back-propagation.

One of the main problems was to find good rules. So far, we applied ID3 and C4.5 techniques for rule learning on all available data prior to the training and embedded the rules. On the other hand, we found that artificial neural networks with proper encoding learnt such (very general) rules easily. The main problems are usually related with infrequent data patterns. Therefore, we believe that the analysis of the errors as proposed here is better than an a-priori generation of rules. But the problem of finding good rules remains. In this paper, we used a very naive approach to construct the correcting rules, but more powerful methods like ID3 or others could improve the performance even more [8]. But the method of choice will probably depend a lot on the application domain. And it is actually the point where background knowledge about the domain can be incorporated into the training process. Alternatively, error-analysis and rule creation could be done by a human expert. One should observe that this expert is only required once the network has learnt most of the rules. I.e., the human intervention is done only for the difficult cases and thus the expert does not need to explicitly state the simple rules.

In this work, we did not use separate training and validation sets, because we wanted to study the improvement during learning. But we observed in other experiments, that rule insertion can also lead to better generalisation. But this is again very much dependant on the quality of the rules. If the rules generalise, their embedding will also lead to a better generalisation of the neural network. We will study this problem in our follow-up work on natural language processing in more detail.

We have also found that rule insertion improves results beyond the knowledge directly expressed in rules. Indeed, similarly to work done when trying to improve neural networks by pruning irrelevant weights, rule insertion can also, indirectly help in the task of overriding irrelevant information. We believe this relation should be made clear in further work. Indeed the relevance of magnitude based pruning methods is well known in methods such as the ones related with optimal brain damage [6]. In this sense, we believe that there should be some resemblance between these methods. Therefore, we will investigate the relations with other pruning and growing techniques in the future. This will be done empirically by comparing the performance of different methods, but the general methodology of the core method may even serve as a basis for a theoretical comparison between the different approaches.

We understand that this paper presents only a starting point. But we believe that the methodology presented here can be developed into a full fledged training paradigm that allows to incorporate domain-dependent background knowledge in a concise way. We are currently applying our algorithm to larger data-sets. As pointed out in the beginning, natural language processing problems seem to provide a good challenge where neuro-symbolic integration should be advantageous. As already reported in [7], we can improve the performance of a connectionist Part-of-Speech tagger by inserting using prior to the training. Next, we will study whether we can further improve it by guiding the training using error-correcting rules as proposed here.



**Figure 5.** The development of the mean squared error over time for different values of  $\omega$  (averaged over 50 runs). The vertical lines show the timepoints where rules have been inserted (every 100 cycles). The zoom in the upper right corner shows also the standard deviation over all 50 runs.

## ACKNOWLEDGEMENTS

We would like to thank the referees for their comments which indeed helped to improve the paper.

## REFERENCES

- [1] A. Asuncion and D. J. Newman. UCI machine learning repository, 2007.
- [2] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler, ‘The integration of connectionism and first-order knowledge representation and reasoning as a challenge for artificial intelligence’, *Journal of Information*, **9**(1), 7–20, (January 2006).
- [3] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler, ‘Connectionist model generation: A first-order approach’, *Neurocomputing*, (2008). accepted, in press.
- [4] Artur S. d’Avila Garcez, Krysia B. Broda, and Dov M. Gabbay, *Neural-Symbolic Learning Systems — Foundations and Applications*, Perspectives in Neural Computing, Springer, Berlin, 2002.
- [5] Steffen Hölldobler and Yvonne Kalinke, ‘Towards a massively parallel computational model for logic programming’, in *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pp. 68–77. ECCAI, (1994).
- [6] Y. LeCun, J. Denker, S. Solla, R. E. Howard, and L. D. Jackel, ‘Optimal brain damage’, in *Advances in Neural Information Processing Systems II*, ed., D. S. Touretzky, San Mateo, CA, (1990). Morgan Kaufman.
- [7] Nuno C. Marques, Sebastian Bader, Vitor Rocio, and Steffen Hölldobler, ‘Neuro-symbolic word tagging’, in *New Trends in Artificial Intelligence*, ed., José Machado José Neves, Manuel Filipe Santos, pp. 779–790. APPIA - Associação Portuguesa para a Inteligência Artificial, (12 2007).
- [8] Tom M. Mitchell, *Machine Learning*, McGraw-Hill, March 1997.
- [9] Geoffrey G. Towell and Jude W. Shavlik, ‘Knowledge-based artificial neural networks’, *Artificial Intelligence*, **70**(1–2), 119–165, (1994).
- [10] Andreas Zell, ‘SNNS. stuttgart neural network simulator, user manual, version 2.1’, Technical report, Stuttgart, (1992).

# Hybrid Classification and Symbolic-Like Manipulation Using Self-Regulatory Feedback Networks

Tsvi Achler<sup>1</sup> and Eyal Amir

**Abstract.** We propose a hybrid model based on self-regulatory feedback. It is a connectionist network, which can be composed in a modular fashion. It can be composed of simple constructs that can be combined together to interact logically without requiring a-priori declaration of ‘interaction-type’ connections. It is primarily a classifier but resolves binding and is pliable to certain symbolic manipulations. We show that 1) compositions can simply be combined to create a hybrid-recognition/symbolic network. 2) Demonstrate how these compositions perform binding logic. 3) Lastly, show how representations can be manipulated in a symbolic-like fashion. These properties are an integral part of intelligent inference and such networks provide a new direction for future research.

## INTRODUCTION

Artificial Intelligence researchers continue to face huge challenges in their quest to develop truly intelligent systems. Neural-symbolic integration may bring an opportunity to integrate well-founded symbolic artificial intelligence. Hybrid models may inherit the best of both approaches.

We propose a hybrid model based on self-regulatory feedback. It is a connectionist network, but is modular. It can be composed of simple constructs that can be combined together to interact logically without requiring a-priori declaration of ‘interaction-type’ connections. It is primarily a classifier but resolves binding and is pliable to certain symbolic manipulations.

Self-Regulatory Feedback Networks (RFN) are unique because 1) they only require connections between inputs and outputs (simple connectivity that is resistant to a combinatorial explosion) and 2) they do not require conventional connection weights (maintain flexibility). They function in a recursive fashion [1-4].

This type of network is better suited for symbolic manipulation than conventional connection-weight models and represents its own flavor of network symbolism.

Section 1 introduces the theory and equations of this network. Section 2 & 3 shows how compositions can simply be combined to create a hybrid-recognition/symbolic network. Section 3 shows how these compositions perform binding logic. Section 4 we show how representations can be manipulated in a symbolic-like fashion.

## 1. Model

RFNs use top-down regulatory feedback to modify input activation. The modified input activity is then re-distributed to the network. This is repeated iteratively to dynamically determine stimuli relevance. In this manner top-down regulatory feedback determines the relevance of inputs to an output node.

This model does not assume calculations are based on predetermined connection weights. All input features are connected equally to associated output nodes. Thus each representation is equally connected to all of its parts. Since connection weights are not relevant, only qualitative relations between feature membership to classes need to be determined during setup; e.g.  $y_1 \in \{x_1, x_3, \dots\}$ ,  $y_2 \in \{x_2, x_3, \dots\}$ . These represent symbolic-like interconnections.

Regulatory feedback has been proposed as an alternate model of lateral inhibition [5]. Regulatory feedback was separately developed as a model for classification [1, 2], for binding and distributed processing [3], and as a multiclass classifier that can process multiple stimuli simultaneously [4]. This work focuses on the flexibility within this model to manipulate representations in a symbolic-like manner.

### 1.1 Model Function

Using our approach features weights are not assigned. Instead, a feature’s information content is dynamically evaluated under different contexts. RFN inhibits *ambiguous* inputs during classification. An input is *ambiguous* if multiple candidate representations using that input are active. Thus, RFNs are more flexible towards combinations of priors since no weighted relations between features or nodes are defined a-priori. This distinction is crucial – traditional Neural Network (NN) notions of a feature being *uninformative* are associated statically between representation and its input feature(s). NN feature extraction methods will assign reduced weights to features that are generally uninformative as determined by the training set.

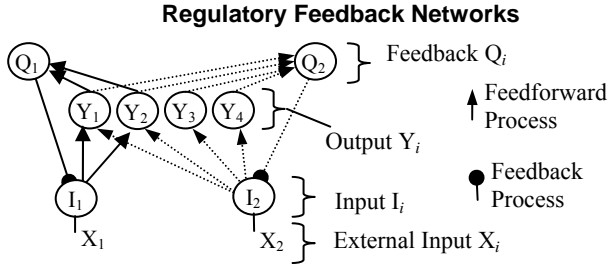
### 1.2 Model Schematic

RFN is unique due to the tight association between input features and outputs representations. This is implemented by a triad of interconnections between an input, the output it supports and feedback from that output (Figure 1). Every input has a corresponding feedback ‘Q’, which samples the output processes that the input cell activates. The feedback modulates the input.

---

<sup>1</sup> University of Illinois at Urbana Champaign, Urbana, IL 61801 USA (217-244-7118; fax: 217-265-6591; e-mail: achler@uiuc.edu).





**Figure 1: Regulatory Feedback Schematic.** Each external input  $X_i$  is modified by feedback  $Q_i$  to produce a feedback-updated input  $I_i$ . Each feedforward connection has an associated feedback connection. For example, if  $I_1$  projects to  $Y_1$  &  $Y_2$ , then  $Q_1$  must receive projections from  $Y_1$  &  $Y_2$  and feedback to the input  $I_1$ . Similarly if  $I_2$  projects to  $Y_1, Y_2, Y_3,$  &  $Y_4$ , then  $Q_2$  receives projections from  $Y_1, Y_2, Y_3,$  &  $Y_4$  and projects to  $I_2$ . Since the connections between  $I$  &  $Q$  are symmetric, thus can be drawn using bidirectional connections (fig 2).

Every output must project to the feedback  $Q$  processes that correspond to its inputs. For example if an output process receives inputs from  $I_1$  and  $I_2$  it must project to  $Q_1$  and  $Q_2$ . If it receives inputs from  $I_1$ , it only needs to project to  $Q_1$ .

This creates a situation where an output cell can only receive full activation from an input if that input's  $Q$  is low. The  $Q$  is low if the sum of activation of the outputs that use that input is low. Thus, if representations that share the input are very active, no cell will receive full activation from that input. If outputs share inputs, they inhibit each other at their common inputs, forcing the outcome of competition to rely on other non-overlapping inputs. The more cells have overlapping inputs, the more competition exists between them. The less overlap between two output cells, the less competition, more independent from each other the output cells can be.

The networks dynamically test recognition of representations using 1) regulatory feedback to the individual inputs of representations 2) modifying the next input state based on the input's use 3) re-evaluating representations based on new activity. Steps 1-3 are continuously cycled through. This requires a tight association between inputs and outputs and feedback processes.

RFN is flexible because it doesn't a-priori define which input is ambiguous. Which input is ambiguous depends on which representation(s) are active which in turn depends on which stimuli and task are being evaluated.

### 1.3 Equations

This section introduces general nonlinear equations governing RFN. Borrowing nomenclature from engineering control theory, this type of inhibitory feedback is negative feedback, in other words stabilizing or regulatory feedback.

For any output cell  $Y$  denoted by index  $a$ , let  $N_a$  denote the input connections to cell  $Y_a$ . For any input cell  $I$  denoted by index  $b$ , let  $M_b$  denote the feedback connections to input cell  $I_b$ . The feedback to input  $I_b$  is defined as  $Q_b$ .  $Q_b$ , is a function of the sum of activity from all cells  $Y_j$  that receive activation from that input:

$$Q_b = \sum_{j \in M_b} Y_j(t) \quad (1)$$

Input  $I_b$  is regulated based on  $Q_b$ , which is determined by the activity of all the cells that project to the input, and driven by  $X_b$  which is the raw input value.

$$I_b = \frac{X_b}{Q_b} \quad (2)$$

The activity of  $Y_a$  is a product of its previous activity and the input cells that project to it. This property can arise biologically from NMDA channels that are found within neuron membranes. These channels are mediated by previous neuron activity. If a self-multiplicative (delay) term is not included in equation 3, the network can immediately change values and the feedback will not be a function of previous activity. Thus the equations are designed so the output cells are proportional to their input activity, inversely proportional to their  $Q$  feedback and also depend on their previous activity [1-4].  $n_a$  represents the number of processes in set  $N_a$ .

$$Y_a(t + \Delta t) = \frac{Y_a(t)}{n_a} \sum_{i \in N_a} I_i \quad (3)$$

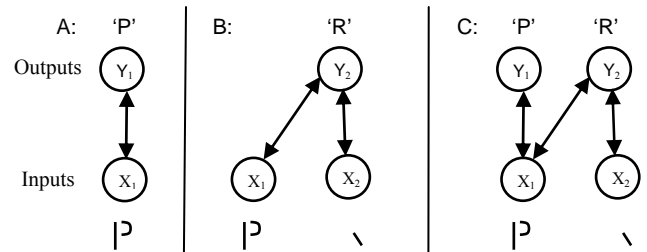
$$= \frac{Y_a(t)}{n_a} \sum_{i \in N_a} \frac{X_i}{Q_i} = \frac{Y_a(t)}{n_a} \sum_{i \in N_a} \left( \frac{X_i}{\sum_{j \in M_i} Y_j(t)} \right) \quad (4)$$

### 1.4 Simple Connectivity

Feedback networks do not require a vast number of connections; the number of connections required for competition is a function of the number of inputs the cell uses. Addition of a new cell to the network requires only that it forms symmetrical connections about its inputs and not directly connect with the other output cells. Thus the number of connections of a specific cell in feedback competition is independent of the size or composition of the classification network, allowing large and complex feedback networks to be combinatorially and biologically practical.

## 2. Simple Modular Composition

Networks can be created in a symbolic/modular fashion. Suppose a node to represent the patterns associated the letter P and another node represents the patterns associated the letter R. R shares some patterns with P. We can intuitively combine these nodes into one network, by ignoring this overlap and just connect the network. This defines a functioning network without formally learning how nodes R and P should interact.



**Figure 2 (A-C):** Modular nodes  $Y_1$  and  $Y_2$  (A & B respectively) can be simply combined to form a combined network (C). Since  $I$  &  $Q$  are symmetric, the network can be drawn using bidirectional connections.

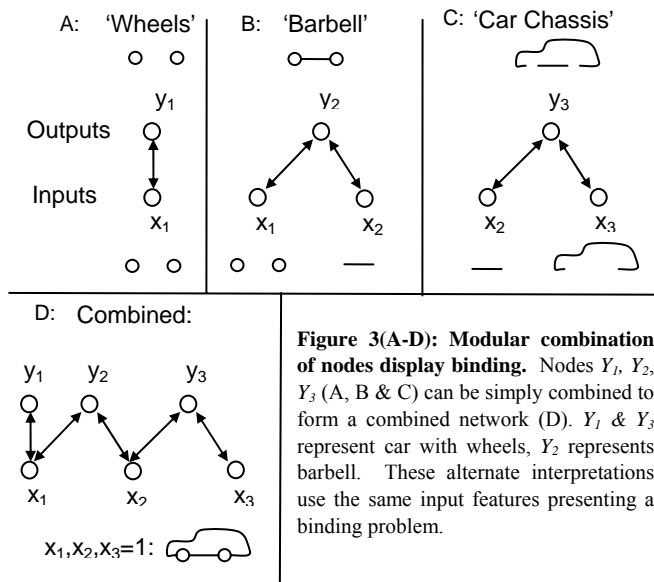
The two nodes are connected such that the inputs of  $Y_1$  (input pattern 'P') completely overlaps with a larger  $Y_2$ . But node  $Y_2$  also receives an independent input pattern 'R'. The solutions are presented as (*input values*)  $\rightarrow$  (*output vectors*) in the pattern  $(X_1, X_2) \rightarrow (Y_1, Y_2)$ . The steady state solution for example 1 is  $(X_1, X_2) \rightarrow (X_1 - X_2, X_2)$ . Substituting our input values we get  $(1, 1) \rightarrow (0, 1)$ ,  $(1, 0) \rightarrow (1, 0)$ . Given only input pattern  $X_1$  ('P') the smaller node wins the competition for representation. Given both input patterns ('P' & 'R') the larger node wins the competition for

representation. Though we used binary inputs, the solution is defined for any positive real  $X$  input values (Achler 2007). The mathematical equations and their derivation can be found in the Appendix.

Descriptively,  $Y_1$  has one input connection, thus by definition its fixed connection weight is one. Its maximal activity is 1 when all of its inputs are 1.  $Y_2$  has two input connections so its fixed input connection weights are one-half. When both inputs ('P' & '\') are 1 the activity of  $Y_2$  sums to 1. Note these weights are predetermined by the network. Connections are permanent and never adjusted. Input  $I_1$  projects to both  $Y_1$  &  $Y_2$ , thus receives inhibitory feedback from both  $Y_1$  &  $Y_2$ . Input  $I_2$  projects only to  $Y_2$  so it receives inhibitory feedback from  $Y_2$ . The most encompassing representation will predominate without any special mechanism to adjust the weighting scheme. Thus, if inputs 'P' & '\' are active  $Y_2$  wins. This occurs because when both inputs are active,  $Y_1$  must compete for all of its inputs with  $Y_2$ , however  $Y_2$  only needs to compete for half of its inputs (the input shared with  $Y_1$ ) and it gets the other half 'free'. This allows  $Y_2$  to build up more activity and in doing so inhibit  $Y_1$ .

Thus smaller representation completely encompassed by a larger representation becomes inhibited when the inputs of the larger one are present. The smaller representation is unlikely given features specific only to the large representation. It demonstrates that RFN determines negative associations (' $Y_1$ ' is unlikely given feature ' $X_B$ ') even though they are not directly encoded. In order to encode such negative associations using conventional methods, they would have to be 'hard-wired' into the network. With NN, each possible set of stimuli combinations would have to be trained. With direct Competition each possible negative association would have to be explicitly connected.

### 3 Binding with Simple Modular Composition



**Figure 3(A-D): Modular combination of nodes display binding.** Nodes  $Y_1$ ,  $Y_2$ ,  $Y_3$  (A, B & C) can be simply combined to form a combined network (D).  $Y_1$  &  $Y_3$  represent car with wheels,  $Y_2$  represents barbell. These alternate interpretations use the same input features presenting a binding problem.

We simultaneously evaluate the criteria of three representations. In e.g. 2, expanded from e.g. 1, three cells partially overlap. As in e.g. 1,  $Y_1$  competes for its single input with  $Y_2$ . However, now  $Y_2$  competes for its other input with  $Y_3$  and  $Y_3$  competes for only one of its inputs.

The steady state solution is  $(X_1, X_2, X_3) \rightarrow (Y_1 = X_1 - X_2 + X_3, Y_2 = X_2 - X_3, Y_3 = X_3)$ .

If  $X_2 \leq X_3$  then  $Y_2 = 0$  and the equations become  $(X_1, 0, \frac{X_2 + X_3}{2})$ .

If  $X_3 = 0$  the solution becomes that of e.g. 1:  $(X_1, X_2, 0) \rightarrow (X_1 - X_2, X_2, 0)$ .



The results are:  
 $(1, 0, 0) \rightarrow (1, 0, 0)$ ;  
 $(1, 1, 0) \rightarrow (0, 1, 0)$ ;  
 $(1, 1, 1) \rightarrow (1, 0, 1)$ .

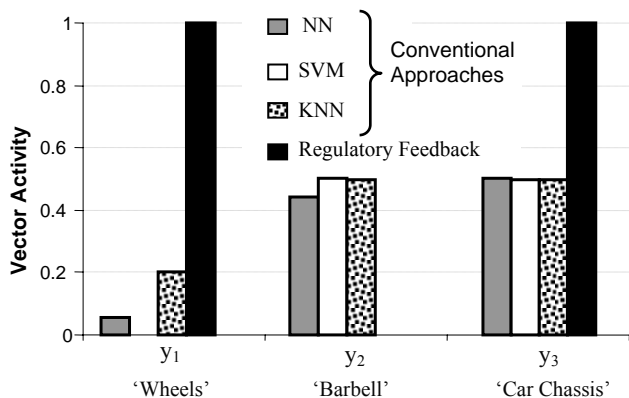
Derivations can be found in the appendix.

Thus if input  $X_1$  is active,  $Y_1$  wins. If inputs  $X_1$  and  $X_2$  are active and  $Y_2$  wins for the same reasons this occurs in e.g. 1. However, if inputs  $X_1$ ,  $X_2$  and  $X_3$  are active then  $Y_1$  and  $Y_3$  win. The network as a whole chooses the cell or cells that best represent the input pattern with the least amount of competitive overlap.

In e.g. 2,  $Y_2$  must compete with all of its inputs:  $X_1$  with  $Y_1$ ,  $X_2$  with  $Y_3$ .  $Y_3$  only competes for half of its inputs (input  $X_2$ ) getting input  $X_3$  'free'. Since  $Y_2$  is not getting its other input  $X_1$  'free' it is at a competitive disadvantage to  $Y_3$ . Together  $Y_1$  and  $Y_3$ , mutually benefit from each other and force  $Y_2$  out of competition. Competitive information travels indirectly 'through' the representations. Given active inputs  $X_1$  and  $X_2 = 1$ , the activity state of  $Y_1$  is determined by input  $X_3$  through  $Y_3$ . If input  $X_3$  is 0 then  $Y_1$  becomes inactive. If input  $X_3$  is 1,  $Y_1$  becomes active. However,  $Y_3$  does not even share input  $X_1$  with  $Y_1$ .

#### 3.1 Binding

Choosing  $Y_2$  given (1,1,1)  is equivalent to choosing the irrelevant features for binding. If the inputs represent spatially invariant features where feature  $X_1$  represents circles,  $X_3$  represents the body shape and feature  $X_2$  represents a horizontal bar.  $Y_1$  is assigned to represent wheels and thus when it is active, feature  $X_1$  is interpreted as wheels.  $Y_2$  represents a barbell  composed of a bar adjacent to two round weights (features  $X_1$  and  $X_2$ ). Note: even though  $Y_2$  includes circles (feature  $X_1$ ), they do not represent wheels ( $Y_1$ ), they represent barbell weights. Thus if  $Y_2$  is active feature  $X_1$  is interpreted as part of the barbell.  $Y_3$  represents a car body without wheels (features  $X_2$  and  $X_3$ ), where feature  $X_2$  is interpreted as part of the chassis. Now given an image of a car with all features simultaneously ( $X_1$ ,  $X_2$  and  $X_3$ ), choosing the barbell ( $Y_2$ ) even though technically a correct representation, is equivalent to a binding error within the wrong context in light of all of the inputs. Most classifiers if not trained otherwise are as likely to choose barbell or car chassis (see figure 4). In that case the complete picture is not analyzed in terms of the best fit given all of the information present. Similar to case 1, the most encompassing representations mutually predominate without any special mechanism to adjust the weighting scheme. Thus the networks are able to evaluate and bind representations in a sensible manner for these triple cell combinations.



**Figure 4: Presentation of this binding problem to conventional classifier approaches.** The Neural Networks (NN) and Support Vector Machines (SVM) were trained on  $y_1, y_2, y_3$  based on  $x_1, x_2$  and  $x_3$ , using the WEKA classifier tool. K-nearest neighbors was determined by the closest neighbor, 1-N.

Optimized NN and SVM learning is performed using the most recent version of the *Waikato Environment for Knowledge Analysis* (WEKA) package currently available [6].

#### 4. Symbolic-like Manipulation

The behavior of the network can be changed by forcing the values of the output nodes. The value of a node can be artificially increased or decreased. For example, forcing a representation to have a zero value is equivalent to eliminating it from the network. Artificially activating a representation gives it priority over other nodes and can force its representation to override inherent binding processes.

We repeat example 2 but can ask the question: can a barbell shape be found in any form? We introduce a small bias to  $Y_2$  (representing barbell) according to the equation modified from example 2:

$$Y_2(t+dt) = \frac{Y_2(t)}{2} \left( \frac{X_1}{Y_1(t)+Y_2(t)} + \frac{X_2}{Y_2(t)+Y_3(t)} \right) + b$$

Choosing a bias  $b$  of 0.2 and activating all inputs, such as car, the network results are:  $(1, 1, 1) \rightarrow (0.02, 0.98, 0.71)$ . The network now overrides its inherent properties and responds to whether inputs matching  $Y_2$  are present. Each representation can be manipulated in a similar manner. This is a form of symbolic manipulation closely tied to recognition. This demonstrates how a symbolic-like manipulation can manipulate a classification scenario.

#### CONCLUSION

Further symbolic manipulation may be possible by changing internal node properties. The equations currently sum inputs ( $I$ 's) in a linear fashion. However they can be summed using a sigmoid or other function. The activation function can be designed so that when any of a node's inputs are matched, the node becomes active. Conversely it can be designed so that when only one input is active that node is active. Initial results show that a sigmoid function gives the node an 'AND'-like function. The location of the sigmoid slope can be manipulated determining node behavior between 'AND'-like nodes to more 'OR'-like nodes. Since RFNs

are nonlinear, at this point, we can simulate general cases, but not provide analytical axioms. This is left for future work.

Using a very simple set-up process, RFN allows modular combination of nodes. It is robust in multiple scenarios and computationally economical because it does not require many variables. It only requires simple combinatorially-plausible connectivity between inputs and outputs. RFN offers a flexible and dynamic approach to intelligent applications. The network can also be manipulated to perform search tasks by biasing output representations. These properties demonstrate that such networks can be an integral part of intelligent inference and provide a new direction for future research.

#### APPENDIX

*Example 1.* RFN equations are:

$$y_1(t+dt) = \frac{y_1(t)x_1}{y_1(t)+y_2(t)}, \quad y_2(t+dt) = \frac{y_2(t)}{2} \left( \frac{x_1}{y_1(t)+y_2(t)} + \frac{x_2}{y_2(t)} \right)$$

The network solution at steady state is derived by setting  $y_1(t+dt)=y_1(t)$  and  $y_2(t+dt)=y_2(t)$  and solving these equations. The solutions are  $y_1 = x_1 - x_2$  and  $y_2 = x_2$ . If  $x_1 \leq x_2$  then  $y_1 = 0$  and the equation for  $y_2$  becomes:  $y_2 = \frac{x_1 + x_2}{2}$ .

*Example 2.* Equation  $y_1(t+dt)$  remains the same as example 1.  $y_2(t+dt)$  and  $y_3(t+dt)$  become:

$$y_2(t+dt) = \frac{y_2(t)}{2} \left( \frac{x_1}{y_1(t)+y_2(t)} + \frac{x_2}{y_2(t)+y_3(t)} \right)$$

$$y_3(t+dt) = \frac{y_3(t)}{2} \left( \frac{x_2}{y_2(t)+y_3(t)} + \frac{x_3}{y_3(t)} \right)$$

Solving for steady state by setting  $y_1(t+dt)=y_1(t)$ ,  $y_2(t+dt)=y_2(t)$ , and  $y_3(t+dt)=y_3(t)$ , we get  $y_1=x_1-x_2+x_3$ ,  $y_2=x_2-x_3$ ,  $y_3=x_3$ . If  $x_3=0$  the solution becomes that of e.g. 1:  $y_1=x_1-x_2$  and  $y_2=x_2$ . If  $x_2 \leq x_3$  then  $y_2=0$  and the equations become  $y_1 = x_1$  and  $y_3 = \frac{x_2 + x_3}{2}$ .

#### ACKNOWLEDGEMENTS

We would like to thank Cyrus Omar and anonymous reviewers for helpful suggestions. This work was supported by the U.S. National Geospatial Agency Grant HM1582-06--BAA-0001.

#### REFERENCES

- [1] Achler T, (2002) "Input Shunt Networks, Neurocomputing", 44-46c: 249-255.
- [2] Achler T, (2007) "Object classification with recurrent feedback neural networks", Proc. SPIE, Evolutionary and Bio-inspired Computation: Theory and Applications, Vol. 6563.
- [3] Achler, T., Amir, E. (2008) "Input Feedback Networks: Classification and Inference Based on Network Structure", Artificial General Intelligence Proceedings V1: 15-26.
- [4] Achler, T., C. Omar, Amir, E. (2008). "Shedding Weights: More With Less." Neural Networks IJCNN Proceedings, In Press.
- [5] Reggia, J. A., C. L. Dautrechy, et al. (1992). "A Competitive Distribution-Theory of Neocortical Dynamics." Neural Computation 4(3): 287-317.
- [6] Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005. Software available at <http://www.cs.waikato.ac.nz/ml/weka/> (version 3.5.6)