Workshop Schedule

10:30 – 11:00
Oliver Ray and Bruno Golenia
A Neural Network Approach for First-Order Abductive Inference
11:00 – 11:30
Amitabha Mukerjee
Using attentive focus to discover action structures from perceptual data
11:30 – 12:00
Kun Tu and Hava Siegelmann
Text-based Reasoning with Symbolic Memory Model

Lunch break

13:45 – 15:00
Keynote Talk by Ben Goertzel
Cognitive Synergy: A Principle to Guide the Tight Integration of Heterogeneous Components in Integrative AI Systems

Coffee break

15:30 – 16:00
Sebastian Bader
Extracting Propositional Rules from Feed-forward Neural Networks by Means of Binary Decision Diagrams
16:00 – 16:30
Amitabha Mukerjee and Madan Dabeeru
Symbol emergence in design
16:30 – 16:45
Leo de Penning, Bart Kappe and Karel van den Bosch
A Neural-Symbolic System for Automated Assessment in Training Simulators: A Position Paper
16:45 – 17:45
Discussion session, chaired by Luis Lamb

Workshop Organisers

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

Programme Committee

Sebastian Bader, University of Rostock, Germany
Howard Blair, Syracuse University, NY, U.S.A.
Claudia d'Amato, University of Bari, Italy
Marco Gori, University of Siena, Italy
Barbara Hammer, TU Clausthal, Germany
Ioannis Hatzilygeroudis, University of Patras, Greece
Steffen Hölldobler, TU Dresden, Germany
Henrik Jacobsson, Google Zurich, Switzerland
Ekaterina Komendantskaya, University of St. Andrews, Scotland
Kai-Uwe Kühnberger, University of Osnabrück, Germany
Luis Lamb, Federal University of Rio Grande do Sul, Brazil
Hannes Leitgeb, University of Bristol, UK
JamesL. McClelland, Stanford University, CA, U.S.A
Anthony K. Seda, University College Cork, Ireland
Ron Sun, Rensselaer Polytechnic Institute, NY, U.S.A.
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 computation bring an opportunity to integrate well-founded symbolic artificial intelligence with robust neural computing machinery to help tackle some of these challenges.

Neural-symbolic systems combine the statistical nature of learning and the logical nature of reasoning.

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;
- Neural networks and relational learning;
- Applications in robotics, semantic web, engineering, bioinformatics, etc.

Table of Contents

Invited Keynote Talk

Cognitive Synergy: A Principle to Guide the Tight Integration of Heterogeneous Components in Integrative AI Systems

Ben Goertzel, Novamente LLC

The concept of "cognitive synergy" is introduced, as a formalization of the idea that in a cognitive system containing multiple heterogeneous learning processes, the different processes should be connected in such a way that each one can get help from the others when it "gets stuck." The role of cognitive synergy in the OpenCog integrative AI architecture is described, with examples given involving the application of OpenCog to control animated agents in virtual worlds. The potential implications of cognitive synergy for the design of neural-symbolic systems is also discussed, in the context of an in-development system called XIA-MAN that is intended to combine neural net evolution with OpenCog to control a Nao humanoid robot.

A Neural Network Approach for First-Order Abductive Inference

Oliver Ray and **Bruno Golénia**
Department of Computer Science
University of Bristol
Bristol, BS8 1UB, UK
{oray,goleniab}@cs.bris.ac.uk

Abstract

This paper presents a neural network approach for first-order abductive inference by generalising an existing method from propositional logic to the first-order case. We show how the original propositional method can be extended to enable the grounding of a first-order abductive problem; and we also show how it can be modified to allow the prioritised computation of minimal solutions. We illustrate the approach on a well-known abductive problem and explain how it can be used to perform first-order conditional query answering.

1 Introduction

Neurosymbolic research aims to combine neural inference methods with symbolic knowledge formalisms in order to better understand and exploit the cognitive functions of brain and mind. The integration of neural networks and logic programs is a major focus in this field. But, most existing work only deals with neural representations of propositional logic programs – which are not very well suited to applications with complex or incomplete information. Thus, our goal is to develop a connectionist approach for typed abductive logic programs – that are specifically intended for this purpose.

Abductive logic programs extend normal logic programs by allowing the truth of particular literals, called *abducibles*, to be assumed subject to certain conditions, called *integrity constraints*, when deciding which instances of a query, or *goal*, succeed from some program, or *theory*. Thus, any solution to a first-order abductive problem has two parts: a set of variable bindings, called an *answer*, denoting a successful instance of the goal; together with a set of abducibles, called an *explanation*, denoting a set of auxiliary assumptions. In this way, abductive logic programming can be seen as a form of conditional query answering.

This paper presents a neural network approach for first-order abductive inference. The approach generalises an existing method of Ray & Garcez [9] from propositional logic to the first-order case. Like its predecessor, the approach is based on translating an abductive problem into a neural network such that the fixpoints of the network are in one-to-one correspondence with the solutions of the original problem. Unlike other methods for neural abduction, such as [3; 10; 2; 13; 7; 12; 8; 1], our approach has the benefits of placing no restrictions on the use of negation or recursion and being able to systematically compute all required solutions.

Our main contributions are twofold: we show how the propositional method can be extended to enable the practical grounding of a first-order abductive problem; and we also show how it can be modified to allow the prioritised computation of minimal solutions. The former extension uses techniques from the field of Answer Set Programming (ASP) [6] to reduce the number of logically redundant clauses and literals in the ground program. The latter modification ensures the hypothesis space is searched in a way that gives higher priority to solutions with fewer abducibles. We illustrate the approach and explain how it can be used to perform first-order conditional query answering.

The rest of the paper is structured as follows: Section 2 recalls the relevant background material; Section 3 presents our approach; and the paper concludes with a summary and directions for future work.

2 Background

This section recalls some basic notions of neural networks and logic programs. The definitions closely follow those of Ray & Garcez [9], except that variables are now explicitly typed and may appear in any input to an abductive problem.

(Threshold) Neural Networks: A *neural network*, or just *network*, is a graph (N, E) whose nodes N are called *neurons* and whose edges $E \subseteq N \times N$ are called *connections*. Each neuron $n \in N$ is labelled with a real number $t(n)$ called its *threshold* and each connection $(n, m) \in E$ is labelled with a real number $w(n, m)$ called its *weight*. The *state* of a network is a function s that assigns to each neuron the value 0 or 1. A neuron is said to be *active* if its value is 1 and it is said to be *inactive* if its value is 0. For each state s of the network, there is a unique successor state s' such that a neuron n is active in s' iff its threshold is exceeded by the sum of the weights on the connections coming into n from nodes which are active in s. A network is said to be *relaxed* iff all of its neurons are inactive. A *fixpoint* of the network is any state that is identical to its own successor state. If a fixpoint t is reachable from an initial state s by repeatedly computing successor states, then t is referred to as the *fixpoint of* s.

$$
\begin{aligned}
T &= \left\{ \begin{array}{l} wont\_start(C) \leftarrow flat\_battery(C) \\ wont\_start(C) \leftarrow no\_fuel(C) \\ wont\_start(C) \leftarrow spark\_plugs\_dirty(C) \\ head\_lights\_work(c1) \\ fuel\_gauge\_empty(c2) \\ spark\_plugs\_dirty(c3) \end{array} \right\} \cup \left\{ \begin{array}{l} car(c1) \\ car(c2) \\ car(c3) \end{array} \right\} \\
G &= \{\ wont\_start(C) \wedge \neg spark\_plugs\_dirty(C)\ \} \\
IC &= \left\{ \begin{array}{l} \leftarrow flat\_battery(C) \wedge head\_lights\_work(C) \\ \leftarrow no\_fuel(C) \wedge \neg fuel\_gauge\_empty(C) \wedge \neg broken\_gauge(C) \end{array} \right\} \\
A &= \left\{ \begin{array}{l} flat\_battery(C) \\ no\_fuel(C) \\ broken\_gauge(C) \end{array} \right\} \\
D &= \{\ car(C)\ \}
\end{aligned}
$$

Figure 1: Abductive context for the *classic cars* problem

(Typed) Logic Programs: A *typed logic program*, or just *program*, is a set of rules of the form $H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m$ in which the H, B_i and C_j are atoms and in which any variable is associated with a unary predicate called its *type* or *domain*. The atom to the left of the arrow is called *head* of the rule, while the literals to the right comprise the *body*. The head atom H and the positive body atoms B_i are said to occur *positively* in the rule, while the negated body atoms C_j are said to occur *negatively*. A rule with no negative body literals is called a *definite clause* and written $H \leftarrow B_1 \wedge \ldots \wedge B_n$. A rule with no body literals is called a *fact* and simply written H. A rule with no head literal is called a *denial* and written $\leftarrow B_1 \wedge \ldots \wedge B_n$. If P is a program, then $\mathcal{B}_P$ (the *Herbrand base* of P) is the set of all atoms built from the predicate and function symbols in P; and $\mathcal{G}_P$ (the *ground expansion* of P) is the program comprising all *well-typed* ground instances of the clauses in P. In addition, $\mathcal{A}_p^+$ and $\mathcal{A}_P^-$ denote, respectively, the sets of ground atoms that occur positively and negatively in $\mathcal{G}_P$. A *stable model* of P is a Herbrand interpretation $I \subseteq \mathcal{B}_P$ that coincides with the least Herbrand model of the definite program P^I obtained by removing from $\mathcal{G}_P$ each rule containing a negative literal not in I, and by deleting all of the negative literals in the remaining rules.

Abductive Logic Programs: An *abductive logic program* [4] is a triple (T, IC, A) where T is a program (the *theory*), IC is a set of denials (*integrity constraints*), A is a set of facts (*abducibles*). Given a conjunction G of literals (*goals*), the task of ALP is to compute a set θ of variable substitutions (an *answer*) and a set Δ of ground abducibles (an *explanation*) such that there is a stable model of $T \cup \Delta$ (which, in the terminology of [5], is called a *generalised stable model* of T) that satisfies all of the denials in IC and all of the literals in $G\theta$. To specify an abductive problem, one must state the theory T, goal G, constraints IC, and abducibles A. If needed, the types of any variables can be explicitly given as a set of facts D (*domain declarations*) with one atom $p(X)$ for every variable X of type p. For convenience, we will refer to the collection of five inputs (T, G, IC, A, D) as a *(typed) abductive context*. A context is propositional if it contains no variables. When a context has many different solutions it is usual to prefer explanations with the fewest number of abducibles. Intuitively, this corresponds to the principle of Occam's Razor, which favours the simplest hypotheses that fit the data. In practice, this means that subset-minimal and/or cardinality-minimal explanations are usually required.

Example 2.1. *Consider the abductive context in Figure 1. The theory T describes a collection of three* classic cars*. It states that a car C wont start if its battery is flat, if its fuel tank is empty, or if its spark plugs are dirty. It also states that the headlights of the first car $c1$ are working, that the fuel gauge of the second car $c2$ is showing empty, and that the spark plugs of the final car $c3$ are dirty. The constraints IC state, firstly, that the battery of a car cannot be flat if the headlights of that car are working and, secondly, that it is impossible for a car to have no fuel if its fuel gauge is not empty and not broken. The abducibles A allow us to assume that any car has a flat battery, has no fuel, and/or has a broken fuel gauge. The goal G asks for which cars C it is possible to show that (1) the car does not start and that (2) the car does not have dirty spark plugs. The domain declarations D assert that all occurrences of the variable C represent cars.*

This problem has many solutions. There are two cardinality-minimal solutions which bind C to $c2$ after abducing $no\_fuel(c2)$ or $flat\_battery(c2)$, respectively. There is one more subset-minimal solution which binds C to $c1$ after abducing $no\_fuel(c1)$ and $broken\_gauge(c1)$. Note that, by the second constraint, $no\_fuel(c1)$ can only be abduced if $broken\_gauge(c1)$ is also abduced (at its gauge is not showing empty). Note also, by the first constraint, $flat\_battery(c1)$ cannot be abduced (as its headlights work). More than a hundred non-minimal solutions can be obtained by adding redundant abducibles to the explanations above.

As non-minimal solutions are often of little practical use, this paper will develop a neural approach for preferentially finding minimal solutions of first-order abductive problems.

3 First order Neural network abduction

The approach in this section builds on well-known methods for translating propositional logic programs into neural networks whose fixpoints correspond to stable models. In actual fact, we build upon an extension of these methods proposed by Ray and Garcez [9] for translating propositional abductive contexts into neural networks whose fixpoints correspond to generalised stable models. This is done by rewriting negative literals as abducibles and adding clauses that, when translated into a neural network, perform a systematic search through the space of abductive solutions.

In principle, one could try and solve a first-order problem by applying the method of [9] to the propositional context obtained by taking all possible well-typed ground instances of the abductive inputs. But, in practice, this naive approach is not feasible as it generates neural networks that are too large and take too long to find the minimal solutions.

To address these limitations, we developed an improved method for more effectively computing the minimal solutions of first-order problems. The new approach exploits an ASP system called LPARSE [11] to allow the efficient grounding of a first-order program; and it also ensures the prioritised computation of minimal solutions by modifying the search strategy of the original method.

Given a typed first-order abductive context, the new method has six main steps: First it introduces new predicates into the language to represent abducibles and negations. Then it translates the typed abductive context into a normal propositional abductive context using LPARSE together with some appropriate pre- and post-processing. Next it rewrites any negative literals in the theory as abducibles subject to some simple integrity constraints that preserve their logical meaning. After that, it adds some clauses that allow the resulting network to compute abductive solutions. At this point, it translates the resulting context into a neural network. Finally it allows the network to compute the abductive solutions in order of minimality. These steps are now described in turn.

STEP 0: Extending the Language

The logical transformations used in our method require the introduction of predicates to represent abducibles and negations. For every predicate p in a given abductive context, we assume two new predicates, denoted $p^\dagger$ and p^*, which represent the *assumption* of p and the *negation* of p, respectively.

STEP 1: Obtaining a Propositional Context

Our approach for translating a typed first-order abductive context into a normal propositional abductive context uses a system called LPARSE which, as described in [11], is practical tool for grounding typed logic programs. Compared to a naive propositionalisation approach, LPARSE further simplifies the ground program by removing literals that are known to be true from the body of a clause and by removing clauses whose bodies are known to be false from the program. Since LPARSE is not specifically designed to work with abductive logic programs, some pre- and post-processing is needed to make the approach work. Thus we have three sub-steps:

• **Pre-processing:** We first turn an abductive context into a logically equivalent program by temporarily employing negative cycles to represent abducibles. This is needed to prevent LPARSE from treating abducibles as undefined domain predicates, which would otherwise be removed from the ground program. Then domain declarations of the form $\#domain\ p(V)$ are added for each variable V of type p in D (which causes LPARSE to add the atom $p(V)$ into the body of every clause containing V). Finally, the goal G is turned into a clause by using a new propositional atom $goal$ as the head. This is carried out by the function ζ below.

Definition 3.1 (ζ)**.** *Let $X = (T, G, IC, A, D)$ be a typed abductive context and let G', A' and D' be as follows*

$$G' = \{goal \leftarrow L_1 \wedge \ldots \wedge L_n \mid L_1 \wedge \ldots \wedge L_n = G\},$$

$$A' = \left\{ \begin{array}{l} a \leftarrow \neg a^* \\ a^* \leftarrow \neg a \end{array} \;\middle|\; a \in A \right\}$$

$$D' = \{\#domain\ p(V) \mid p(V) \in D\}$$

Then $Y = \zeta(X)$ is the logic program $T \cup G' \cup IC \cup A' \cup D'$.

• **Grounding:** The resulting program is now grounded and simplified by LPARSE, as indicated by the function χ below.

Definition 3.2 (χ)**.** *Let Y be a logic program (possibly containing domain declarations). Then $Z = \chi(Y)$ is the ground logic program obtained by running* LPARSE *on Y.*

• **Post-processing:** After running LPARSE, any surviving ground instances of the negative cycles introduced by the pre-processing phase are converted back into explicit abducibles. This is done by replacing each cycle of the form $a \leftarrow \neg a^*$ and $a^* \leftarrow \neg a$ with a bridging clause $a \leftarrow a^\dagger$ and making the ground atom $a^\dagger$ abducible. It is interesting to note that this re-conversion is not strictly necessary as any negations will be replaced by abducibles and integrity constraints in the next step. But, it is more efficient to exploit the fact that each cycle represents a single abducible in order to avoid the unnecessary introduction of one additional abducible and two additional integrity constraints. The bridging clauses allow to distinguish instances of an atom whose truth is abduced from those whose truth is implied. In addition, any resulting ground instances of the goal clause are added to the theory T_1 and their head atom $goal$ becomes the new propositional goal. This process is performed by the function μ below.

Definition 3.3 (μ)**.** *Let Z be any ground logic program and define*

$$\begin{array}{rcl} Z_{IC} & = & \{\leftarrow L_1 \wedge \ldots \wedge L_n \in Z\} \\ Z_G & = & \{goal \leftarrow L_1 \wedge \ldots \wedge L_n \in Z\} \\ Z_A & = & \{a \leftarrow \neg a^* \in Z\} \cup \{a^* \leftarrow \neg a \in Z\} \\ Z_T & = & Z \setminus (Z_{IC} \cup Z_G \cup Z_A) \end{array}$$

Then $W = \mu(Z)$ is the propositional abductive context $(T_1, G_1, IC_1, A_1, D_1)$ such that

$$\begin{array}{rcl} T_1 & = & Z_T \cup Z_G \cup \{a \leftarrow a^\dagger \mid \{a \leftarrow \neg a^*\} \in Z_A\} \\ G_1 & = & \{goal\} \\ IC_1 & = & Z_{IC} \\ A_1 & = & \{a^\dagger \mid \{a \leftarrow \neg a^*\} \in Z_A\} \\ D_1 & = & \emptyset \end{array}$$

$$
T_1 = \left\{ \begin{array}{l}
car(c1) \\
car(c2) \\
car(c3) \\
wont\_start(c1) \leftarrow flat\_battery(c1) \\
wont\_start(c2) \leftarrow flat\_battery(c2) \\
wont\_start(c3) \leftarrow flat\_battery(c3) \\
wont\_start(c1) \leftarrow no\_fuel(c1) \\
wont\_start(c2) \leftarrow no\_fuel(c2) \\
wont\_start(c3) \leftarrow no\_fuel(c3) \\
\boxed{wont\_start(c1) \leftarrow spark\_plugs\_dirty(c1)} \\
\boxed{wont\_start(c2) \leftarrow spark\_plugs\_dirty(c2)} \\
wont\_start(c3) \boxed{\leftarrow spark\_plugs\_dirty(c3)} \\
spark\_plugs\_dirty(c3) \\
fuel\_gauge\_empty(c2) \\
head\_lights\_work(c1)
\end{array} \right\} \cup \left\{ \begin{array}{l}
broken\_gauge(c1) \leftarrow broken\_gauge^{\dagger}(c1) \\
broken\_gauge(c2) \leftarrow broken\_gauge^{\dagger}(c2) \\
broken\_gauge(c3) \leftarrow broken\_gauge^{\dagger}(c3) \\
no\_fuel(c1) \leftarrow no\_fuel^{\dagger}(c1) \\
no\_fuel(c2) \leftarrow no\_fuel^{\dagger}(c2) \\
no\_fuel(c3) \leftarrow no\_fuel^{\dagger}(c3) \\
flat\_battery(c1) \leftarrow flat\_battery^{\dagger}(c1) \\
flat\_battery(c2) \leftarrow flat\_battery^{\dagger}(c2) \\
flat\_battery(c3) \leftarrow flat\_battery^{\dagger}(c3)
\end{array} \right\}
$$

$$
\cup \left\{ \begin{array}{l}
goal \leftarrow wont\_start(c1) \boxed{\wedge \neg spark\_plugs\_dirty(c1)} \\
goal \leftarrow wont\_start(c2) \boxed{\wedge \neg spark\_plugs\_dirty(c2)} \\
\boxed{goal \leftarrow wont\_start(c3) \wedge \neg spark\_plugs\_dirty(c3)}
\end{array} \right\}
$$

$$
IC_1 = \left\{ \begin{array}{l}
\leftarrow flat\_battery(c1) \boxed{\wedge head\_lights\_work(c1)} \\
\boxed{\leftarrow flat\_battery(c2) \wedge head\_lights\_work(c2)} \\
\boxed{\leftarrow flat\_battery(c3) \wedge head\_lights\_work(c3)} \\
\leftarrow no\_fuel(c1) \wedge broken\_gauge^{*}(c1) \boxed{\wedge \neg fuel\_gauge\_empty(c1)} \\
\boxed{\leftarrow no\_fuel(c2) \wedge \neg broken\_gauge(c2) \wedge \neg fuel\_gauge\_empty(c2)} \\
\leftarrow no\_fuel(c3) \wedge broken\_gauge^{*}(c3) \boxed{\wedge \neg fuel\_gauge\_empty(c3)} \\
\leftarrow broken\_gauge^{*}(c1) \wedge broken\_gauge^{*}(c1) \\
\leftarrow \neg broken\_gauge^{*}(c1) \wedge \neg broken\_gauge^{*}(c1) \\
\leftarrow broken\_gauge^{*}(c3) \wedge broken\_gauge^{*}(c3) \\
\leftarrow \neg broken\_gauge^{*}(c3) \wedge \neg broken\_gauge^{*}(c3)
\end{array} \right\}
$$

$$
G_1 = \{\ goal\ \}
$$

$$
\begin{array}{rcl}
A_1 & = & \{\ no\_fuel^{\dagger}(c1), flat\_battery^{\dagger}(c1), broken\_gauge^{\dagger}(c1), broken\_gauge^{*}(c1)\ \} \\
& \cup & \{\ no\_fuel^{\dagger}(c2), flat\_battery^{\dagger}(c2), broken\_gauge^{\dagger}(c2)\ \} \\
& \cup & \{\ no\_fuel^{\dagger}(c3), flat\_battery^{\dagger}(c3), broken\_gauge^{\dagger}(c3), broken\_gauge^{*}(c3)\ \}
\end{array}
$$

Figure 2: Propositional abductive context for the *classic cars* example (where boxes identify clauses and literals removed by the grounding process)

Example 3.1. *The result of applying the above transformation to the abductive context of Example 2.1 is shown in Figure 2 — which, for convenience, also shows, within boxes, the clauses and literals removed by* LPARSE. *Note that the omitted rules can never be true and the omitted constraints can never be violated.*

STEP 2: Obtaining a Definite Theory

To avoid potential problems resulting from the unrestricted use of negation in the theory (which could result in networks where some states have no fixpoints) we use a well-known equivalence between negation and abduction which allows us to treat each negative literal $\neg p(t_1, \ldots, t_n)$ as an abducible $p^*(t_1, \ldots, t_n)$ subject to integrity constraints stating that an atom and its negation cannot both be true or both be false in the same model. Intuitively, we are free to assume the falsity of any atom if it is consistent to do so. As formalised by the function η, below, we add such constraints for all ground atoms appearing negatively in the context. For convenience, this function uses some notation for representing the positive form of an expression obtained by replacing all negative literals by their abducible proxies.

Definition 3.4 (*)**.** *Let $C = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m$ be a clause. Then C^* is the (definite) clause $C^* = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge C_1^* \wedge \ldots \wedge C_m^*$.*

Definition 3.5 (η)**.** *Let $W = (T_1, G_1, IC_1, A_1, D_1)$ be a propositional abductive context. Then $\eta(W)$ is the abductive context, $(T_2, G_2, IC_2, A_2, D_2)$ such that*

$$
\begin{array}{lcl}
T_2 & = & \{C^* \,|\, C \in T_1\} \\
G_2 & = & \{C^* \,|\, C \in G_1\} \\
IC_2 & = & \{C^* \,|\, C \in IC_1\} \\
 & \cup & \left\{ \begin{array}{l} \leftarrow a \wedge a^* \\ \leftarrow \neg a \wedge \neg a^* \end{array} \;\middle|\; a \in \mathcal{A}^-_{T_1 \cup IC_1 \cup G_1} \right\} \\
A_2 & = & A \cup \{a^* \,|\, a \in \mathcal{A}^-_{T_1 \cup IC_1 \cup G_1}\} \\
D_2 & = & D_1
\end{array}
$$

STEP 3: Adding Clauses for Abduction

We now add some extra clauses that, when translated into a neural network, will perform a systematic search through the space of abductive solutions.

As in the method of Ray & Garcez [9], we use a binary counter whose output determines which abducibles are assumed and which are not. In earlier work, the output of this counter followed a simple binary sequence: 000, 001, 010, 011, 100, etc. But, in this paper, we wish to keep the number of abducibles as small as possible for as long as possible. Thus we use a modified search sequence (called a Banker's sequence) where the number of true bits increases monotonically: 000, 100, 010, 001, 110, etc.

In fact, we use two cascaded Banker's sequences, one for pure abducibles $a^\dagger \in A_2$ and one for (the negations of) the negative literals $a^* \in A_2$. The clauses which define these counters are formalised in the theory C below. Firstly, C^{1,n_1} denotes a counter with n_1 bits $a^1_{n_1}, \ldots, a^1_1$ bits corresponding to the abducibles $a^\dagger_{n_1}, \ldots, a^\dagger_1$. Secondly, C^{2,n_2} denotes a counter with n_2 bits $a^2_{n_2}, \ldots, a^2_1$ bits corresponding to the abducibles $a^*_{n_2}, \ldots, a^*_1$. Each counter i has $HOLD^i$ and $MOVE^i$ conditions that determine its behaviour when a global $next$ signal is applied to the network. If $HOLD^i$ is true, it will keep its current value. If $MOVE^i$ is true, it will advance to the next value in the Banker's sequence. If neither is true, the counter will reset to zero.

The definition of each counter $C^{i,n}$ is split into five macros $C^{i,n}_1$ to $C^{i,n}_5$ that together provide a logical specification of the counter state transition table. In brief, b^i_j is true iff the j^{th} bit of counter i is true; c^i_j is true iff b^i_j is true and b^i_{j-1} is false; d^i_j is true iff c^i_j is true and c^i_k is false for all $k < j$; e^i_j is true iff b^i_j is false and b^i_{j-1} is true; and all^i is true iff all the bits of counter i are true.

$$
\begin{array}{lcl}
C & = & C^{1,n_1} \cup C^{2,n_2} \quad \text{where} \\
C^{i,n} & = & \bigcup_{k=1}^{5} C^{i,n}_k \quad \text{such that} \\
C^{i,n}_1 & = & \bigcup_{k=0}^{n} \left\{ \begin{array}{l} b^i_k \leftarrow \bigwedge_{j=1}^{n} \neg d^i_j \wedge b^i_{n-k} \wedge MOVE^i \\ b^i_k \leftarrow \bigwedge_{j=1}^{n} \neg d^i_j \wedge e^i_{n-k} \wedge MOVE^i \\ b^i_k \leftarrow b^i_k \wedge \neg next \\ b^i_k \leftarrow b^i_k \wedge HOLD^i \end{array} \right\} \\
C^{i,n}_2 & = & \left\{ \begin{array}{l} b^i_n \leftarrow \bigwedge_{j=0}^{n} \neg b^i_j \wedge MOVE^i \\ all^i \leftarrow \bigwedge_{j=0}^{n} b^i_j \end{array} \right\} \\
C^{i,n}_3 & = & \bigcup_{k=2}^{n} \bigcup_{j=0}^{k-1} \left\{ \; b^i_{k-2-j} \leftarrow b^i_j \wedge d^i_k \wedge MOVE^i \; \right\} \\
C^{i,n}_4 & = & \bigcup_{k=1}^{n} \left\{ \begin{array}{l} e^i_k \leftarrow \neg b^i_k \wedge b^i_{k-1} \\ c^i_k \leftarrow b^i_k \wedge \neg b^i_{k-1} \\ d^i_k \leftarrow \bigwedge_{j=1}^{k-1} \neg c^i_j \wedge c^i_k \\ b^i_{k-1} \leftarrow d^i_k \wedge MOVE^i \\ a^i_k \leftarrow ABD^i_k \end{array} \right\} \\
C^{i,n}_5 & = & \bigcup_{k=1}^{n} \bigcup_{j=k+1}^{n} \left\{ \; b^i_j \leftarrow b^i_j \wedge d^i_k \wedge MOVE^i \; \right\} \\
MOVE^1 & = & next \wedge all^2 \\
MOVE^2 & = & next \wedge \neg all^2 \\
HOLD^1 & = & \neg all^2 \\
HOLD^2 & = & false \\
ABD^1_k & = & b^1_k \text{ for all } k \\
ABD^2_k & = & \neg b^2_k \text{ for all } k
\end{array}
$$

The 2^{nd} counter increments until its maximum is reached, whence it resets. The 1^{st} counter holds until the 2^{nd} resets, whence it increments. To preferentially *minimise* the number of assumed abducibles the outputs $a^1_{n_1} \ldots a^1_1$ of the 1^{st} counter are obtained by copying the corresponding bits $b^1_{n_1} \ldots b^1_1$. To preferentially *maximise* the number of atoms assumed false, the outputs $a^2_{n_2} \ldots a^2_1$ of the 2^{nd} counter are obtained by negating the corresponding bits $b^2_{n_2} \ldots b^2_1$.

As in [9], we advance the counter if a fixpoint fp is reached that violates the goal $goal$ or integrity constraints ic. Previously, this was done by a clock with a constant period determined by worst case propagation delay through the network generated by the program. For efficiency, we now add clauses to detect as soon as a fixpoint is reached. These are given by the theory S below, which uses four new propositions $z', z^+, z^-, z^\$$ to compare the current state of each atom z (in a set Z of atoms) with their previous states.

In brief, z' denotes the previous state of z; z^+ is true iff the current and previous states are both true; z^- is true iff they are both false; and $z^\$$ is true iff they are the same. The $w1$-$w4$ are wait signals that just give the counters enough time to compute their next value before the network decides whether the current abducibles are a solution (in which case the signal $soln$ is activated) or whether another set of abducibles are needed (in which case the signal $next$ is activated). fp indicates when a fixpoint has been reached. The signal $done$ indicates when all solutions are exhausted.

$$
\begin{array}{rcl}
S & = & S_1^Z \cup S_2^Z \cup S_3^Z \quad \text{where} \\
S_1^Z & = & \bigcup_{z \in Z} \left\{ \begin{array}{l} z' \leftarrow z \\ z^+ \leftarrow z \wedge z' \\ z^- \leftarrow \neg z \wedge \neg z' \\ z^\$ \leftarrow z^+ \\ z^\$ \leftarrow z^- \end{array} \right\} \\
S_2^Z & = & \left\{ \begin{array}{l} w_1 \leftarrow next \\ w_2 \leftarrow w_1 \\ w_3 \leftarrow w_2 \\ w_4 \leftarrow w_3 \\ \\ fp \leftarrow \bigwedge_{z \in Z} z^\$ \wedge \bigwedge_{i=1}^{4} \neg w_i \wedge \neg next \end{array} \right\} \\
S_3^Z & = & \left\{ \begin{array}{l} next \leftarrow \neg next \wedge fp \wedge ic \\ next \leftarrow \neg next \wedge \neg goal \wedge fp \\ soln \leftarrow \neg next \wedge goal \wedge fp \wedge \neg ic \\ done \leftarrow all^1 \wedge all^2 \wedge fp \end{array} \right\}
\end{array}
$$

To properly specify these additional clauses, it is necessary to state the parameters n_1, n_2 and Z in C and S. This is done by the function δ below, which defines n_1 as the number of abducibles $a^\dagger$, n_2 as the number of negations a^*, and Z as the set of all atoms appearing (positively or negatively) in the context. In addition, this function adds a special atom ic into the head of every integrity constraint, and adds the literal $\neg next$ into the body of every rule to ensure the network is fully relaxed whenever a new solution is attempted. The resulting network architecture is summarised in Figure 3.

Definition 3.6 (δ). *Let $(T_2, G_2, IC_2, A_2, D_2)$ be a propositional abductive context with $G_2 = \{goal\}$ and $D_2 = \emptyset$. Now let $n_1 = |\{a^\dagger \in A_2\}|$, let $n_2 = |\{a^* \in A_2\}|$, let $Z = A^+_{T_2 \cup G_2 \cup IC_2} \cup A^-_{T_2 \cup G_2 \cup IC_2}$, and let R be the program*

$$
\begin{array}{rcl}
R & = & \left\{ \begin{array}{l} H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg next \\ \mid H \leftarrow B_1 \wedge \ldots \wedge B_n \in T_2 \end{array} \right\} \\
& \cup & \left\{ \begin{array}{l} ic \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg next \\ \mid \leftarrow B_1 \wedge \ldots \wedge B_n \in IC_2 \end{array} \right\}
\end{array}
$$

Then, $P = \delta(T_2, G_2, IC_2, A_2)$ is the program $R \cup C \cup S$.

STEP 4: Obtaining a neural network

Now we translate the clauses obtained so far into a network using the method of Ray & Garcez [9], recalled below, which is based on well-known neurosymbolic techniques.

Definition 3.7 (θ). *Let P be a logic program, then $\theta(P)$ is the network (N, E) such that*

$$
N = \bigcup_{r \in \mathcal{G}_P} \left\{ \begin{array}{l} r, H, B_1, \ldots, B_n, C_1, \ldots, C_m \\ \mid r = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m \end{array} \right\}
$$

$$
E = \bigcup_{r \in \mathcal{G}_P} \left\{ \begin{array}{l} (r, H), (B_1, r), \ldots, (B_n, r), (C_1, r), \ldots, (C_m, r) \\ \mid r = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m \end{array} \right\}
$$

and for all $r = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m \in \mathcal{G}_P$

$$
t(r) = n - 1/2 \qquad \begin{array}{ll} t(H) = 1/2 & w(r, H) = 1 \\ t(B_i) = 1/2 & w(B_i, r) = 1 \\ t(C_j) = 1/2 & w(C_j, r) = -1 \end{array}
$$

STEP 5: Computing Abductive Solutions

In the last step, we compute the answers with the neural network. Starting from a relaxed network, we activate the node $next$. Then, on every other subsequent state, we check whether the network has reached a fixpoint where $soln$ activated. If it has, then the current abducibles are recorded and we see which of the $goal$ rules is activated in order to extract the successful ground instances of the query. If more solutions are required then we activate $next$ in order to seek the next answer. We stop this process when a suitable answer is computed or $done$ is true, meaning the entire search space has been explored.

4 Conclusion and future work

We believe that the integration of abductive and inductive inference is necessary to develop improved learning and reasoning approaches. In this paper we presented a neurosymbolic approach for first-order abductive inference. Unlike most other such approaches, we do not impose any restrictions on the use of negation or recursion. However, the ASP grounding procedure assumes that all variables have finite domains, which limits the use of function symbols. We have implemented our method using the C++ programming language and applied it to the abductive context shown in Figure 1. In this way, we correctly obtained the minimal solutions followed by all of the other non-minimal solutions. Even in this simple example we note that the search time increases significantly if a naive grounding is used in place of our more efficient ASP approach. We also note that the time needed to compute all minimal solutions is significantly less than the time needed to compute all (non-minimal) solutions. In this sense we have improved existing methods of neurosymbolic abduction. However, the complexity of the approach is still exponential in the number of abducibles, which limits its practical use. In future work, we will compare our neural approach with related symbolic approaches and investigate ways of parallelising our method.

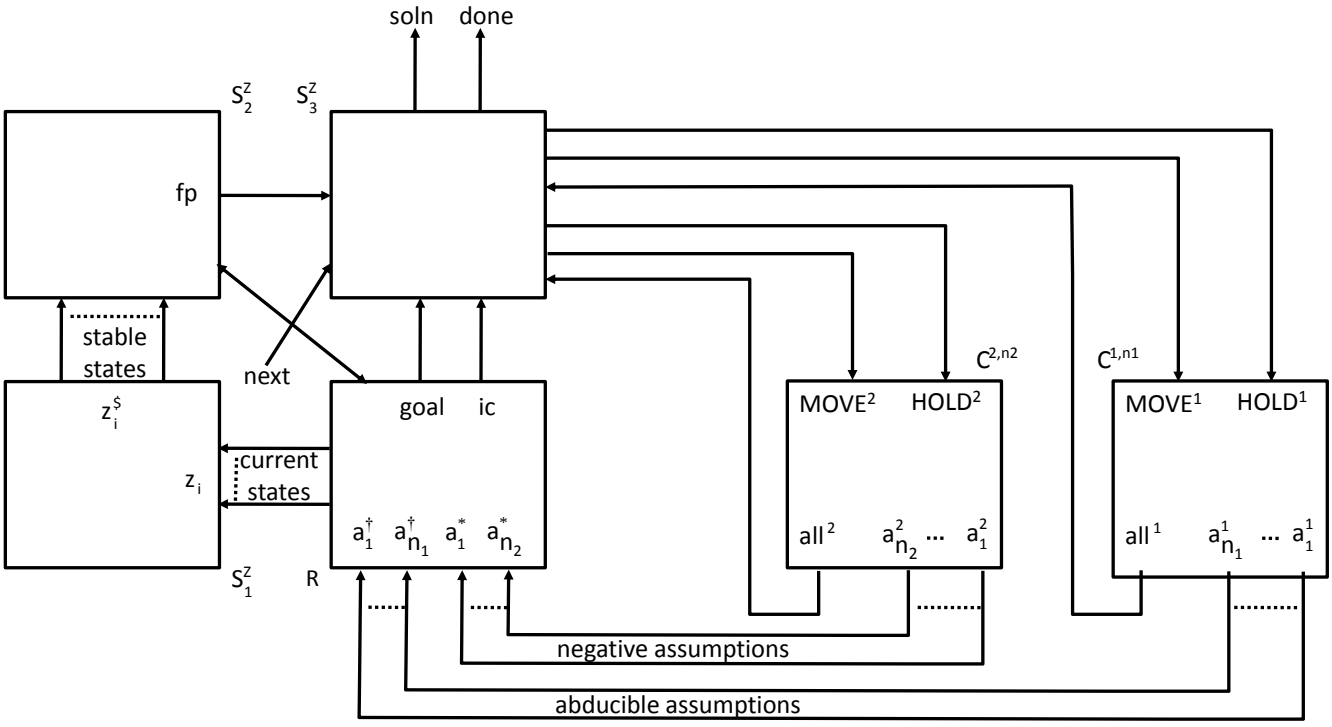


Figure 3: Network architecture showing the logical interactions between the counters C^{1,n_1}, C^{2,n_2} control blocks S_1^Z, S_2^Z, S_3^Z, and the ground program R

References

[1] A. Abdelbar, M. El-Hemaly, E. Andrews, and D. Wunsch II. Recurrent neural networks with backtrack-points and negative reinforcement applied to cost-based abduction. *Neural Networks*, 18(5-6):755–764, 2005.

[2] B. Ayeb, S. Wang, and J. Ge. A Unified Model For Neural Based Abduction. *IEEE Transactions on Systems, Man and Cybernetics*, 28(4):408–425, 1998.

[3] A. Goel and J. Ramanujam. A Neural Architecture for a Class of Abduction Problems. *IEEE Transactions on Systems, Man and Cybernetics*, 26(6):854–860, 1996.

[4] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[5] A.C. Kakas and P. Mancarella. Generalized Stable Models: a Semantics for Abduction. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 385–391. Pitman, 1990.

[6] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[7] P. Lima. Logical Abduction and Prediction of Unit Clauses in Symmetric Hopfield Networks. In *Artificial Neural Networks, 2*, volume 1, pages 721–725. Elsevier, 1992.

[8] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to abductive multiadjoint reasoning. In *Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, volume 2443 of *Lecture Notes in Computer Science*, pages 213–222, 2002.

[9] O. Ray and A. d'Avila Garcez. Towards the integration of abduction and induction in artificial neural networks. In *Proceedings of the ECAI'06 Workshop on Neural-Symbolic Learning and Reasoning*, pages 41–46, 2006.

[10] J. Reggia, Y. Peng, and S. Tuhrim. A Connectionist Approach to Diagnostic Problem-Solving Using Causal Networks. *Information Sciences*, 70:27–48, 1993.

[11] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligece*, 138(1-2):181–234, 2002.

[12] R. Vingrálek. A connectionist approach to finding stable models and other structures in nonmonotonic reasoning. In *Proceedings of the Second International Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 60–81. MIT Press, 1993.

[13] C. Zhang and Y. Xu. A Neural Network Model for Diagnostic Problem Solving with Causal Chaining. *Neural Networks and Advanced Control Strategies*, 54:87–92, 1999.

Using attentive focus to discover action ontologies from perception

Amitabha Mukerjee
Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
amit@cse.iitk.ac.in

Abstract

The word "symbol", as it is used in logic and computational theory, is considerably different from its usage in cognitive linguistics and in everyday life. Formal approaches that define symbols in terms of other symbols ultimately need to be grounded in perceptual-motor terms. Based on cognitive evidence that the earliest action structures may be learned from perception alone, we propose to use attentive focus to identify the agents participating in an action, map the characteristics of their interaction, and ultimately discover actions as clusters in perceptuo-temporal space. We demonstrate its applicability by learning actions from simple 2D image sequences, and then demonstrate the learned predicate by recognizing 3D actions. This mapping, which also identifies the objects involved in the interaction, informs us on the argument structure of the verb, and may help guide syntax. Ontologies in such systems are learned as different granularities in the clustering space; action hierarchies emerge as membership relations between actions.

1 Introduction

Learning the concepts for concrete objects require the perceptual system to abstract across visual presentations of these objects. In contrast, modeling actions present a more complex challenge [Fleischman and Roy, 2005], [Sugiura and Iwahashi, 2007]. Yet actions are the central structure for organizing concepts; the corresponding language units (verbs) also acts as "heads" (predicates) in sentences, controlling how an utterance is to be interpreted. Typically the structure for an action/verb includes a set of possible constituents that participate in the action, and also some constraints on the type of action (e.g. the type of motion that may constitute "A chases B").

In this work, we consider the learning of the structure of actions, based on image sequences. Cognitively, there is evidence that some action schemas are acquired through perception in a pre-linguistic stage [Mandler, 2004]; later these are reinforced via participation, and may eventually seed linguistic aspects such as argument structure.

We postulate that a key aspect of this process is the role of perceptual attention [Regier, 2003],[Ballard and Yu, 2003]. Thus, an action involving two agents may involve attention shifts between them, which helps limit the set of agents participating in the action. The set of agents participating in an action eventually generalizes to the argument structure. In [Ballard and Yu, 2003], human gaze was directly tracked and matched with language fragments, and verbs such as "picking up" and "stapling" were associated with certain actions. However, the verbal concepts learned were specific to the context, and no attempt was made to generalize these into action schemas, applicable to new scenes or situations. Top down attention guided by linguistic inputs is used to identify objects in [Roy and Mukherjee, 2005]. More recently, in [Guha and Mukerjee, 2007] attentive focus is used to learn labels for simple motion trajectories, but this is also restricted to a particular visual domain.

1.1 From Percept to Concept to Symbol

The word "symbol", as it is used in logic and computational theory is considerably different from its usage in cognitive linguistics and in everyday life. The OED defines it as "Something that stands for, represents, or denotes something else". This meaning carries over to the cognitive usage, where it is viewed as a tight coupling of a set of mental associations (the *semantic pole*) with the psychological impression of the sound (the *phonological pole*) [**?**]. Formally, however, a symbol is detached from any meaning, it is just a token constructed from some finite alphabet, and is related only to other such tokens. A computer system dealing with such symbols can define many relations with other symbols, but finds it difficult to relate it to the world, and this makes it difficult also to keep the relations between symbols up to date. The objective of this work is to try to align a symbol to a perceptual stimulus, so as to provide grounding for the symbols used in language or in reasoning.

In other work, we have addressed the question of

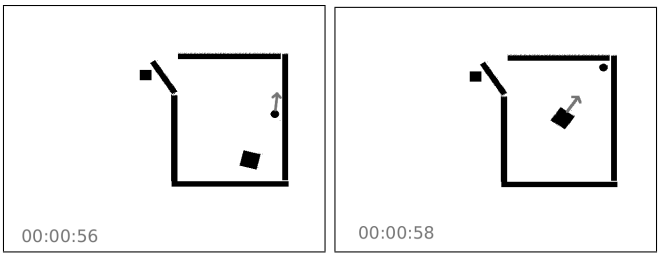


Figure 1: *Scenes from 2D video: "Chase"*: Three agents, "big square", 'small square" and "circle" play and chase each other. Velocities are shown as gray arrows.

learning the language label (or the phonological pole) of a symbol [Satish and Mukerjee, 2008]. Here we focus on modeling the semantic pole, especially with respect to action ontologies. Such models, called *Image Schema* in Cognitive Linguistics [Langacker, 1999] or *Perceptual Schema* in Experimental Psychology [Mandler, 2004], involve abstractions on low-level features extracted from sensorimotor modalities (positions and velocities), as well as the argument structure.

We ask here if, given a system that is observing a simple 2D scene (see fig. 1) with shapes like squares and circles chasing each other, is it possible for it to cluster all 2-agent interactions in some meaningful way into a set of action schemas? If so, do these action schemas relate reliably to any useful conceptual structures? Further, is there any possibility of learning any relationships between these action schemata, thus constructing a primitive ontology? Note that all this has to take place without any language, witout any human inputs in any form.

Constructing such action templates has a long history in Computer vision, but most gather statistics in view-specific ways with an emphasis on recognition [Xiang and Gong, 2006; **?**]. We restrict ourselves to two-object interactions, using no priors, and our feature vectors are combinations of relative position and velocity vectors of the objects (we use a simple inner product). We perform unsupervised clustering on the spatio-temporal feature space using the Merge Neural Gas algorithm [Strickert and Hammer, 2005]; the resulting clusters constitute our action schemas. By considering different levels of cluster granularity in the unsupervised learning process, we also learn subsets of coarse concepts as finer action concepts, resulting in an action hierarchy which may be thought of as a rudimentary ontology.

Having learned the action schema based on a given input, we apply it to recognize novel 2-body interactions in a 3D fixed camera video, in which the depth of a foreground object is indicated by it's image y-coordinate. We show that the motion features of humans can be labelled using the action schemas learned.

2 Analysis: Role of Attentive Focus

One of the key issues we explore in this work is the relevance of perceptual attention. It turns out that restricting computation to attended events somehow results in a better correlation with motions that are named in language. This may reflect a bias in conceptualization towards actions that attract attention. Like other models that use attention to associate agents or actions to language [Ballard and Yu, 2003; Guha and Mukerjee, 2007], we use attentive focus to constrain the region of visual salience, and thereby the constituents participating in an action. We use a computational model of dynamic visual attention [Singh *et al.*, 2006] to identify agents possibly in focus.

In order to analyze the different types of motion possible in the scene, we first perform a qualitative analysis of the motions. We assume that all objects have an intrinsic frame with a privileged "front" direction defined either by its present direction of motion, or by the last such observed direction. Let the reference object be A, then the pose of located object B w.r.t. the frame of A can be described as a 2-dimensional qualitative vector [Forbus *et al.*, 1987], where each axis is represented as $\{-, 0, +\}$ instead of quantitative values. This results in eight possible non-colliding states for the pose of B. In each pose, the velocity of B is similarly encoded, resulting in 9 possible velocities (including non-moving).

This results in 72 possible relations, and distinguishing the situation when the reference object A is moving, from that when it is stationary, results in a total of 144 possible states. Linguistic labels(Come-Close(CC), Move-Away(MA), Chase(CH), Go-Around(GoA), Move-Together(MT), Move-Opposite(MO)) are manually assigned to each of these qualitative relative motion states. The motion in nearly half the states do not appear to have clear linguistic terms associated with them, and these undenominated interactions are left empty. The remaining classes assigned are shown in Figure 2. Qualitative classification for the frames in Fig.1 is shown in Fig. 3.

Next, we analyze the frequency of these cases observed on the Chase video. Fig. 2 compares the frequency of the qualitative states with non-stationary first object, in the situation where all possible object pairs are considered (no attentive focus), versus that where using attentive cues pairs of agents attended to within a temporal window of 20 frames become candidates for mutual interaction; all other agent pairings are ignored. The frequency of indeterminate qualitative cases are 58% in the first situation and 24% in the second. Thus, attentive focus biases the learning towards relations that we have names for in language.

3 Visual Attention

We consider a bottom-up model of visual attention (not dependent on task at hand) [Itti, 2000]. Here we consider a model designed to capture bottom-up attention in dynamic scenes based on motion saliency [Singh *et al.*,

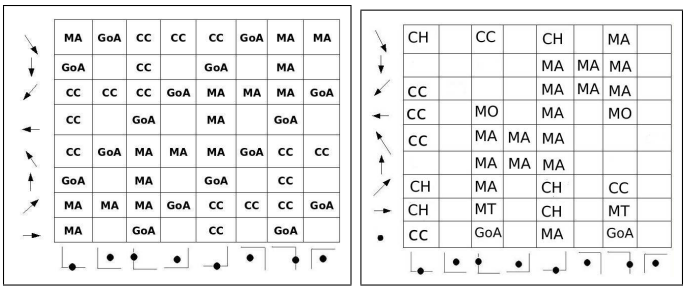


Figure 2: Qualitative analysis of two object interaction: Single frame qualitative classification for *(a)* stationary first object and *(b)* when the first object is moving horizontally to the right. X-axis gives the different positions of the second object and Y-axis gives the different velocity directions(including zero velocity) of second object w.r.t. the first object at origin. Cases when motion does not have a simple English label are blank. Others labels are: Come Closer (CC), Move {Away,Opposite, Together} (MA,MO,MT), Chase (CH) and Go Around (GoA)

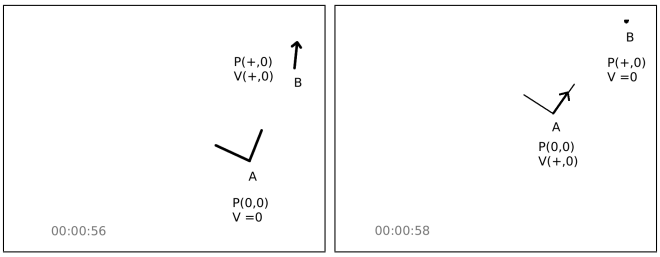


Figure 3: Single frame qualitative classification for the frames in Fig.1. The big square is taken as the first object. The labels assigned are MA(*left*) and CC(*right*). P and V refer to the position and velocity in the reference frame with origin at the first object and x-axis along its velocity.

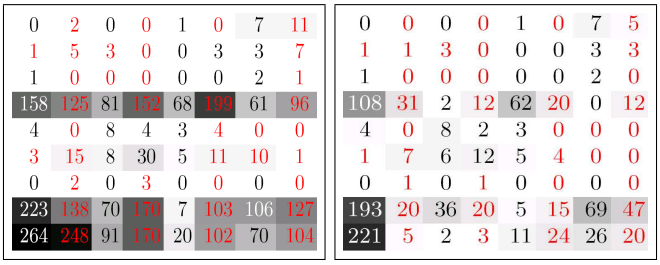


Figure 4: Without and with Attention: Frequency map for Single frame qualitative classification for the case of non-stationary first object. % of the feature vectors that can not be labelled (given in Red) is 58% without attention, and 24% with attentive focus.

2006]. Objects are taken as the attentive foci instead of pixels. Motion saliency map is computed from optical flow, a confidence map is introduced to assign higher salience to objects not visited for a long time. A small foveal bias is introduced to mediate in favour of proximal fixations against large saccadic motions. Winner-Take-All network on the combined saliency map gives the most salient object for fixation.

Name	Formula
pos·velDiff	$(\vec{x}_B - \vec{x}_A) \cdot (\vec{v}_B - \vec{v}_A)$
pos·velSum	$(\vec{x}_B - \vec{x}_A) \cdot (\vec{v}_B + \vec{v}_A)$

Table 1: Dyadic Features Formulae. A and B refer to the two objects; A is said to be the Reference Object (The more salient, usually the larger of the two objects, is taken as Reference Object) and B the Located Object in the feature computation; $\vec{v_A}$ refers to velocity vector of A; $\vec{x_A}$ refers to position vector of A; and '$\cdot$' refers to the inner product of the vectors.

4 Unsupervised Perceptual Clustering

Perceptual systems return certain abstractions of the raw sensory data - "features" - which are used for recognition, motor control, categorization, etc. In this work we use two features that capture the interaction of two agents. All learning takes place in the space of these two features, (Table 1); the first feature captures the combination of relative position and velocity, the second the relative position and magnitude.

These feature vectors are then clustered into categories in an unsupervised manner based on a notion of distance between individuals. We use the Merge Neural Gas(MNG) algorithm[Strickert and Hammer, 2005] for unsupervised learning which has been shown to be well-suited for processing complex dynamic sequences as compared to the other existing models for temporal data processing like Temporal Kohonen map, Recursive SOM etc. This class of temporal learning algorithms are more flexible with respect to the state specifications and time history compared to HMMs or VLMMs. MNG algorithm performs better than other unsupervised clustering algorithms like K-Windows [Vrahatis *et al.*, 2002], DBSCAN [Ester *et al.*, 1996] because of the utilization of the temporal information present in the frame sequences unlike the other algorithms.

4.1 Merge Neural Gas algorithm

The Neural Gas algorithm [Martinetz and Schulten, 1994] learns important topological relations in a given set of input vectors (signals) in an unsupervised manner by means of a simple Hebb-like learning rule. It takes a distribution of high-dimensional data, P(ξ) and returns a densely connected network resembling the topology of the input.

For input feature vectors arriving from temporally connected data, the basic neural gas algorithm can be generalized by including explicit context representation which utilizes the temporal ordering present in the feature vectors of the frames, resulting in the Merge Neural Gas algorithm [Strickert and Hammer, 2005]. Here, a *Context* vector is adjusted based on the present winning

Table 2: *Clustering Accuracy*: The i^{th} row, j^{th} column gives the number of i^{th} action labels in the j^{th} NG Cluster. % is the fraction of vectors of an action correctly classified to the total vectors of that type. Total Classification Accuracy(TCA) is the % of total vectors correctly classified .

Action	C1	C2	C3	C4	Tot	%	TCA
CC	399	6	10	29	444	90	
MA	16	311	5	48	380	82	84
Chase	21	59	149	154	383	79	

neuron data. Cluster labels for the frames are obtained in the final iteration of the algorithm based on the winner neuron.

5 Concept Acquisition: Chase video

Unsupervised clustering using the Merge Neural Gas algorithm is used on the feature vectors from the video, corresponding to object pairs that were in attentive focus around the same time. Salient objects in a scene are ordered by a computational model of bottom-up dynamic attention[Singh *et al.*, 2006]. The most salient object is determined for each frame, and other objects that were salient within k frames before and after (we use $k = 10$) are considered as attended simultaneously. Dyadic feature vectors are computed for all object pairs in these $2k$ frames.

Owing to the randomized nature of the algorithm, the number of clusters varies from run to run. Clusters with less than ten frames are dropped. With the *aging* parameter set to 30, the number of clusters came out to be four in 90% of the runs; the set of four clusters with highest total classification accuracy (refer Table 2) are considered below.

In order to validate these clusters with human concepts, we asked three subjects (Male, Hindi-English/ Telugu-English bilinguals, Age-22, 20 and 30) to label the scenes in the video. They were shown the video twice and in the third viewing they were asked to speak out one of three action labels (CC, MA, Chase) which was recorded. Given the label and the frame when this was uttered, the actual event boundaries and participating objects for the groundtruth data were assigned by inspection. In case of disagreement, we took the majority view.

The percentage accuracies shown in table 2 do not reflect the degree of match, since although an event may last over 15 frames, even if 10 frames have been detected, it is usually quite helpful. This can be seen in 6 which present results along a time line for *Chase*; each row reflects a different combination of agents (small square, big square, circle). At first glance, figures like 6 would seem to reflect a higher accuracy than 84% in table 2.

A surprising result was found when by experimenting with the *edge aging* parameter in the Merge Neural Gas

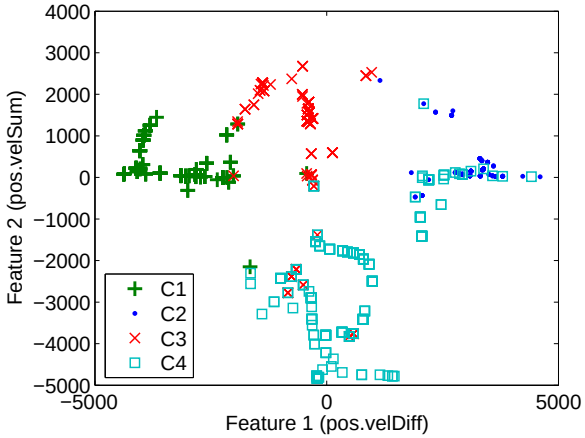


Figure 5: *Feature Vectors of the Four Clusters from the MNG Algorithm*: CC - C_1, MA - C_2, Chase(Reference Object is the Chaser) - C_3, Chase(Reference Object is the Leader) - C_4; The clusters reflect the spatio-temporal proximity of the vectors.

Figure 6: Comparison of Human and Algorithm Labelling of "chase" over first 1500 frames. Because of our choice of reference object, frames in first row are in C_4 and second row are in C_3.

Table 3: *Hierarchical clustering*: Using a larger number of clusters reveals a sub-classification; e.g. frames classified as CC in Table 2, are now in $C_1, C_5, or C_6$, reflecting two cases of $CC_{one-object-static}$, or one case of $CC_{both-moving}$.

	C1	C2	C3	C4	C5	C6	C7	C8
CC	201	3	9	20	189	21	1	0
MA	8	126	4	45	9	1	181	6
Chase	1	9	142	151	13	9	32	26

Table 4: *Relevance of Argument Order*: Value at i^{th} row, j^{th} column gives number of vectors that were originally in Cluster i and now assigned to Cluster j when object order was switched in dyadic feature vectors. Note that C3 and C4, the clusters corresponding to *Chase*, are flipped.

	C1	C2	C4	C3
Cluster 1	390	20	11	15
Cluster 2	9	323	15	29
Cluster 3	6	12	1	145
Cluster 4	22	48	152	9

algorithm. The number of clusters increase as aging parameter is decreased, and at one stage eight clusters were formed (edge aging parameter=16). The Total Classification Accuracy (TCA) was about 51 and we would have discarded the result, but inspecting the frames revealed that the clusters may be reflecting what appeared to be hierarchy of action types. Thus cluster C_1 from the earlier classification (majority correlation=CC) was broken up into C_1, C_5, C_6. C_1 was found to contain frames where both objects are moving towards each other whereas C_5 contains frames where the smaller object is stationary and the other moves closer. Thus *Come-Closer* and *Move-Away* appear to be sub-classified into 3 classes (two *one object static* cases, and one *both moving* case). This 'finer' classification is given in Table 3.

5.1 Argument order in Action Schemas

In another experiment, we investigated the importance of argument ordering by re-classifying the same frames, but reversing the order of the objects used in the dyadic vector computation. Earlier, if the larger object was $arg1$ or reference object, now it became $arg2$ or non-reference object. If the corresponding concept changed, especially if it flipped, this would reflect a semantic necessity to preserve the argument order; otherwise the arguments were commutative. Using the coarser clusters, we observe that the argument order is immaterial since the majority relation is unchanged (black) for C1 and C2 (CC,MA respectively). On the other hand, both C3 and C4 (correlations with Chase) are flipped (Table 4). Thus, the fact that argument order is important for *Chase* is learned implicitly within the action schema itself. The non-commutativity of $CC_{one-object-static}$ and $MA_{one-object-static}$ could not be established because of the skewed distribution of frames in the input video amongst the two sub-classes for the action verbs.

5.2 Comparison with K-Windows Clustering

We compare the clustering accuracy obtained by the unsupervised Merge Neural Gas algorithm with K-windows algorithm [Vrahatis *et al.*, 2002]. K-Windows is an improvement of K-Means clustering algorithm with a better time complexity and clustering accuracy. We set the value of k in this algorithm to 4 and run it on the input feature vectors obtained after attentive pruning. The initial cluster points for the algorithm are set randomly. Table 5 gives the clustering results obtained.

Table 5: *Clustering Accuracy by K-Windows*: The value of *'k'* is set to 4. The i^{th} row, j^{th} column gives the number of i^{th} action labels in j^{th} Cluster. % is the fraction of vectors of an action correctly classified to the total vectors of that type. Total Classification Accuracy(TCA) is the % of total vectors correctly classified .

Action	C1	C2	C3	C4	Total	%	TCA
CC	277	19	51	97	444	62	
MA	29	234	61	56	380	61	59
Chase	95	83	91	114	383	54	

The lower accuracy (as compared to results in Table 2) is expected because K-windows treats each feature vector as a separate entity without utilizing the information present in the temporal ordering of the frames.

6 Recognizing actions in 3D

In order to test the effectiveness of the clusters learned, we test the recognition of motions from a 3D video of three persons running around in a field (Fig.7). In human classification of the action categories (into one of *CC, MA, Chase*), the dominant predicate in the video, (777 out of 991 frames), is Chase.

In the image processing stage, the system learns the background over the initial frames based on which it segments out the foreground blobs. It is then able to track all the three agents using the Meanshift algorithm. Assuming camera height near eye level, the bottom-most point in each blob corresponds to that agent's contact with the ground, from which its depth can be determined within some scaling error (157 frames with extensive occlusion between agents were omitted). Given this depth, one can solve for the lateral position - thus, we are able to obtain, from a single view video, the (x, y) coordinates for each agent in each frame, within a constant scale. Based on this, the relative pose and motion parameters are computed for each agent pair, and therefrom the features as outlined earlier. Now these feature vectors are classified using the action schemas (coarse clusters) al-

Figure 7: *Test Video :* Scenes from the 3D video

Table 6: Distribution of Chase frames(ground truth) from the 3D video across the Neural gas clusters

	C1	C2	C3	C4	Chase total	%
Chase	13	15	496	253	777	96

ready obtained from the Chase video (2D) (Table 6).

7 Discussion and Conclusion

We have outlined how our unsupervised approach learns action schemas of two-agent interactions resulting in an action ontology. The image schematic nature of the clusters are validated by producing a description for a 3D video. The approach provided here underlines the role of concept argument structures in aligning with linguistic expressions, and that of bottom-up dynamic attention in pruning the visual input and in aligning linguistic focus.

Once a few *basic* concepts are learned, other concepts can be learned without direct grounding, by using conceptual blending mechanisms on the concept itself. These operations are often triggered by linguistic cues, resulting in new concepts, as well as their labels being learned together, in a later stage. Indeed, the vast majority of our vocabularies are learned later purely from the linguistic input [Bloom, 2000]. But this is only possible because of the grounded nature of the first few concepts, without which these later concepts cannot be grounded. Thus the perceptually grounded nature of the very first concepts are crucial to subsequent compositions.

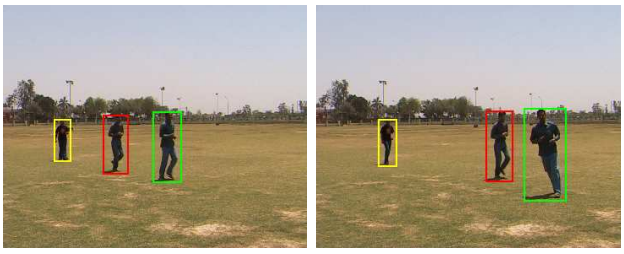


Figure 8: *Image Schemas identified for actions*: "Red *Chase* Green", "*Move Away*(Red,Yellow)", "*Move Away*(Green,Yellow)"

References

[Ballard and Yu, 2003] Dana H. Ballard and Chen Yu. A multimodal learning interface for word acquisition. In *International Conference on Acoustics,Speech and Signal Processing(ICASSP03)*, volume 5, pages 784–7, April 2003.

[Bloom, 2000] Paul Bloom. *How Children Learn the Meanings of Words*. MIT Press, Cambridge, MA, 2000.

[Ester *et al.*, 1996] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xug. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[Fleischman and Roy, 2005] Michael Fleischman and Deb Roy. Why verbs are harder to learn than nouns: Initial insights from a computational model of intention recognition in situated word learning. In *Proceedings of the 27th Annual Meeting of the Cognitive Science Society*, 2005.

[Forbus *et al.*, 1987] Kenneth D Forbus, Paul Nielsen, and Boi Faltings. Qualitative kinematics: A framework. In *IJCAI*, pages 430–436, 1987.

[Guha and Mukerjee, 2007] Prithwijit Guha and Amitabha Mukerjee. Language label learning for visual concepts discovered from video sequences. In Lucas Paletta, editor, *Attention in Cognitive Systems. Theories and Systems from an Interdisciplinary Viewpoint*, volume 4840, pages 91–105. Springer LNCS, Berlin / Heidelberg, 2007.

[Itti, 2000] L. Itti. *Models of Bottom-Up and Top-Down Visual Attention*. PhD thesis, Pasadena, California, Jan 2000.

[Langacker, 1999] Ronald Wayne Langacker. *Grammar and Conceptualization*. Berlin/New York: Mouton de Gruyer, 1999.

[Mandler, 2004] J M Mandler. *Foundations of Mind*. Oxford University Press, 2004.

[Martinetz and Schulten, 1994] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[Regier, 2003] Terry Regier. Emergent constraints on word-learning: A computational review. *Trends in Cognitive Sciences*, 7:263–268, 2003.

[Roy and Mukherjee, 2005] Deb Roy and Niloy Mukherjee. Towards situated speech understanding: visual context priming of language models. *Computer Speech and Language*, 19(2):227–248, 2005.

[Satish and Mukerjee, 2008] G. Satish and A. Mukerjee. Acquiring linguistic argument structure from multimodal input using attentive focus. In *7th IEEE International Conference on Development and Learning, 2008. ICDL 2008*, pages 43–48, 2008.

[Singh *et al.*, 2006] Vivek Kumar Singh, Subranshu Maji, and Amitabha Mukerjee. Confidence based updation of motion conspicuity in dynamic scenes. In *Third Canadian Conference on Computer and Robot Vision*, 2006.

[Strickert and Hammer, 2005] Marc Strickert and Barbara Hammer. Merge som for temporal data. *Neurocomputing*, 64:39–71, 2005.

[Sugiura and Iwahashi, 2007] Komei Sugiura and Naoto Iwahashi. Learning object-manipulation verbs for human-robot communication. In *WMISI '07: Proceedings of the*

2007 workshop on Multimodal interfaces in semantic interaction, pages 32–38, New York, NY, USA, 2007. ACM.

[Vrahatis *et al.*, 2002] Michael N Vrahatis, Basilis Boutsinas, Panagiotis Alevizos, and Georgios Pavlides. The new k-windows algorithm for improving thek -means clustering algorithm. *Journal of Complexity*, 18(1):375–391, March 2002.

[Xiang and Gong, 2006] T. Xiang and S. Gong. Beyond tracking: Modelling activity and understanding behaviour. *International Journal of Computer Vision*, 67(1):21–51, 2006.

Text-based Reasoning with Symbolic Memory Model

Kun Tu[1,2] **and Hava Siegelmann**[2]
[1] College of Computer Science and Engineering, South China University of Technology
[2] Department of Computer Science, University of Massachusetts Amherst
hava@cs.umass.edu

Abstract

In this paper, a sentence-based reasoning model is introduced for the prediction of new individual activities by means of memory reconsolidation that enables the integration of incoming evidence with related past experience. Both the evidence and previous experience are stored in extended semantic networks (ESN) as memory. They are then processed in Bayesian networks for inferring new and unified memory. Symbolic approaches, which focus on the structural aspect of language, ensure the correct extraction of the key information of words according to the context. Effective mechanisms for information propagation, Bayesian networks (BN) construction and combination are adopted to enable inference reasonable and adaptive to different scenarios based on the topic domain. Our model is compared to other reasoning systems through experiments. The results show that our model can both deduce more implicit information from texts, and avoid some incorrect reasoning caused by confusing data in the knowledgebase.

1 Introduction

Successful reasoning on activities from text should be more adaptive to different scenarios than other data-driven inference models that focus on structure and parameter leaning. For that, a preprocessing must include both the integration with commonsense knowledge and the representation of events after parsing the sentences [Klein and Manning, 2003].

Previous approaches have applied some language processing operations and built effective reasoning models. One method establishes "coreference mappings" of data in a memory system to reduce the number of ambiguous sentence interpretations [Livingston and Riesbeck, 2009]; another builds a large commonsense knowledgebase to make inference from key words [Liu and Singh, 2004]. These methods, although efficient in many cases, lack a systematic approach for disambiguation of word senses and scenario topics [Dahlgren, 1988], and hence may lead to partially incorrect inference.

Additionally, some data from commonsense knowledge are not always true given a specific scenario. For example, the assertion "*if someone is a lawyer, he practices law*" could sound correct, however, there are non-practice lawyers and lawyers are not likely to practice law during "their holidays" [Kathleen and Joyce, 1989].

To improve on the inference of individual activities from simple sentences, we propose a reasoning model with the characteristic of memory reconsolidation. Memory reconsolidation can be thought as an information-processing procedure where a recall of memory can be updated or strengthened as a result of integration of incoming information into the pre-existing "memory network"[Tulving and Thomson, 1973]. During this procedure, information similar in meanings, topics or scenarios within a domain is activated and selectively moved to working memory for cognitive process. Inspired by this phenomenon, we build a robust and scenario-adaptive memory system with extended semantic networks (ESN) as the symbolic representation of the language. Stanford Parser [Klein and Manning, 2003] is embedded to parse sentences. Key information such as subjects, verb and objects is extracted into working memory and referred to WordNet [Fellbaum, 1998] and VerbNet [Kipper *et al.*, 2006] for disambiguation of word senses and topics. Ontology categories of noun phrases, built from WordNet and stored in directed acyclic graphs, describe the features of entities of individual subjects. The relations of entities(representing activities, events or features of entities), are stored as sub-symbolic memory in multiple Bayesian networks (BN). The BNs are built from a)statistical analysis of information in ESN, b)sentences containing causal relation and c)other knowledge bases. Related BNs are adaptively selected by matching the keywords of topics in a scenario, and are combined together if they have nodes in common.

The selection and combination of BNs enable inference processes to avoid inaccurate conclusion and obtain more implicit information than previous models.

2 Related Work

We use the following programs to enable our memory model:

1) Stanford parser, a Java program, is used to analyze the grammatical structure of sentences and obtain subjects, objects and predicates [Klein and Manning, 2003]. 2) WordNet is an English lexical database that provides multiple meanings and topic domains of a word. It also groups words into sets of cognitive synonyms and indicate the relations of the words. (e.g. "dog" belongs to "canine", "canine" belongs to

"carnivore", "placental" and "mammal".) We use it to construct an acyclic graph for word disambiguation.

3) VerbNet classifies verbs into classes. Each class is assigned with thematic roles, selectional restrictions on the arguments and syntactic frames. For example, the Class "Hit-18.1" has a constraint on the syntax of the class members, showing the subject of "hit" should be a human or animal. Such constraints are used to disambiguate the meanings from WordNet and ConceptNet.

4) ConceptNet provides a commonsense knowledgebase that can describe concepts of nouns and provides causal relations between predicates. The knowledge in ConceptNet is expressed as five-tuple assertions ("relation type", "A", "B", "f", "I"), where "relation type" indicates the relation of "A" and "B", f is the number of times a fact is uttered in their training corpus, i counts how many times an assertion was inferred during the 'relaxation' phase. For example, (CapableOf "animal" "grow" "f=2; i=2;") has a relation type "CapabaleOf", which indicates the category of "animal" is capable to perform the activity "grow". Since some relation types provide causal relations of the verb phrases (VP), we build two-node Bayesian networks from each of the assertions and use them for inference on focused topics.

3 Information Representation

Predicates in sentences describe the features or activities of the subjects. Some predicates can break into a form of "joiner + object", where the joiner contains a verb, indicating the relations between the subjects and the objects (see Figure 1(a)) . From a graphical point of view, a vertex can symbolically represent a subject or an object while an edge can represent their relation. For this reason, our extended semantic networks(ESNs) are used as a memory that represents the information derived from sentences .

The ESNs focus on the relations between the subject and the object. Since the meaning of a word can be captured by the distribution of commonly co-occurring words or phrases [Landauer and Dumais, 1997]. The semantic roles of verbs have been characterized with nouns, and were shown to predict the brain activity associated with the meanings of nouns [Mitchell *et al.*, 2008]. The relation in an ESN contains a verb that helps to understand the subjects in a certain scenario.

3.1 Definition of Extend Semantic Network

Suppose $S = (s_1, s_2, \ldots, s_n)$ is a set of sentences, an extended semantic network to store S is a graph $G' = (V', E')$, where $V' = \{v_k | k = 1, 2, \ldots, m\}$ is a set of vertices representing the concepts of subjects or objects in the sentences in S; $E' = \{e_d(V_i, V_j) | d = 1, 2, \ldots;\ i, j = 1, 2, \ldots, m\}$ is a set of edges representing the relations between vertex V_i and V_j, which are indicated in the sentences in S.

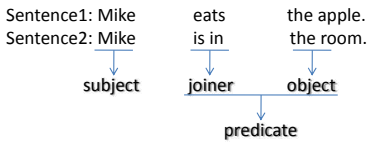
Notice that $e_d(V_i, V_j)$ is the d-th edge from vertex V_i to V_j, which means that there can be more than one relations between two concepts. Vertex V_i can be the same as V_j (that is, $i = j$), in which case it is a unary relation of V_i (e.g. edge $e_1(V_3, V_3)$ in Figure 1(b)) representing a intransitive verb or a property of a vertex.

In our model, the memory system has two kinds of memory: a long-term memory and a working memory. The long-term memory is an ESN that stores all the refined information from working memory. The working memory is a temporarily created ESN and receives information from the Stanford Parser, long-term memory and inferences from BNs. The working memory provides a disambiguation mechanism to ensure the accuracy of information by topic matching. Then, the disambiguated information would be either stored into the long-term memory as an update or propagated to BNs as evidence for inferences.

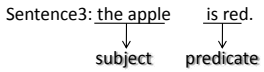
3.2 Parsing a Sentence

The subject and predicate of an input sentence are first extracted with the Stanford Parser.

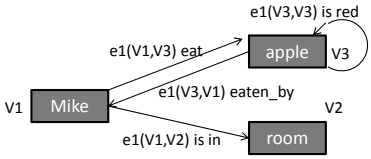
Figure 1 shows the storing of sentences in an ESN. In Figure 1(a), the subject "*Mike*" and the objects "*apple*" and "*room*" are stored as vertices. The joiners "*eat*" and "*is in*" in the Verb Phases (VP) are stored as directed edges between vertices (Figure 1(c)).



(a) predicates containing an object



(b) a predicate with no objects



(c) symbolic representation in ESN

Figure 1: Parsing sentences into an ESN.

3.3 The Vertices

Vertices in an ESN denote noun phrases (NP) from subjects and objects in sentences. They represent the entities of people or things in the world. Features and constraints are added to these vertices to ensure the robustness of the memory system.

Ontology Categories

Ontology categories classify the vertices according to the word senses. WordNet provides such hierarchical category data with regard to the word meanings.

Our model uses WordNet to build directed acyclic graphs to represent these categories. Each category has descriptions of its features and activities. Figure 2 shows a sub-graph of the category graph generated from data in WordNet. Suppose "research professor" belongs to the categories of "professor" and "researcher". In WordNet, "professor" is a kind

of "academician" who "works at a college or university". After extracting the subject and predicate in the annotation of "professor" with the Stanford Parser, "professor" can be assigned to the category of "academician" and the feature activities are "work at a college" or "work at a university". Similarly, "research professor" is also in the descendant categories of "educator" ("someone who educates young people") and "researcher" ("scientist who devotes himself to research").

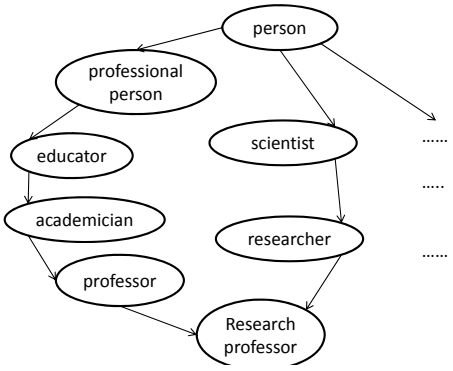


Figure 2: part of a directed acyclic graph for categories

In our category graph, a category can obtain the features or activities either from its parent categories or by importing data from the knowledgebase in ConceptNet.

The ontology categories are important to the entities in working memory because they can help to disambiguate the word sense and provide predicates to connect BNS(see next section).

Possible States of a ESN Vertex

During information processing in the human brain, some neurons are highly active, whereas others remain silent. This leads to the formation of neural circuits for specific memory. In analogy with biological neural memory, a vertex in an ESN has different states: 1)active, 2)semi-active and 3)inactive .

Table 1 shows the differences among the three states of a vertex. An active vertex represents an entity ready for inference. It comes from a subject or an object in a sentence or from the long-term memory. Its meaning is disambiguated and its renewed relations in the working memory with other entities will be directly updated to the long term memory. A semi-active vertex will become an active vertex if it is linked to another active vertex once there exists a new relation between them. An inactivated vertex is not for inference and remains in the long term memory. This can reduce the size of ESNs, which makes computation less complex.

state	Long-term memory	Working Memory	For inference	information update	assigned a category
active	Yes	Yes	Yes	Yes	Yes
semi-active	No	Yes	Yes	No	No
inactive	Yes	No	No	No	No

Table 1: Three states of a vertex in an ESN.

3.4 The Directed Edges

In a sentence, a predicate with an object is regarded as the form of "joiner + object". Edges in the ESNs represent the joiners. They denote the relations between vertices in ESNs and have features that are important to the inference in BNs:

1) **The indicator of the evidence**. Edges are indicated as evidence if they are generated from input sentences. These edges will be directly stored in both the long-term and working memory regardless of their probabilities.

2) **The probability of an edge**. The non-evidence edges are generated from inferences by BNs. Their probabilities can change after each inference. In a working memory, a non-evidence edge is added or removed depending on whether its probability is beyond or below a given threshold. After inferences in the working memory are finished, the non-evidence edges are transported to the long-term memory.

3) **Edges representing a passive voice**. As each BN in our model predicts the activities of only one subject, the interaction of two entities cannot be inferred in one BN. A reversed edge representing the passive voice of a verb is used in the ESN to ensure information can be propagated among the BNs for different entities. For example (Figure 1), "Mike eats the apple". An edge $e_3(V_3, V_1)$ representing "be eaten by" will be created, and a node representing "be eaten by Mike" in a BN if the activities of the apple need to be inferred.

4) **Unary and binary relation**. Edges connecting two different vertices indicate binary relations. Unlike standard SNs, loop edges are allowed as unary relations in our ESN when the predicate has no object(e.g. "stops"), or when the predicate only describes a property of its subject (e.g. "is red" $e_1(V_3, V_3)$ in Figure 1(b)).

5) **Multiple edges between two vertices**. Multiple relations between two vertices are represented as multiple edges in the ESN. In referring to Figure 1(c), suppose also "Mike likes the room", then another edge "like", denoted as $e_2(V_1, V_2)$, would connect "Mike" to the "room".

3.5 Disambiguation for Meanings

Words with multiple senses are classified into multiple categories in WordNet and cause ambiguities. We suggest operations that can disambiguate the word senses:

1) choosing categories with constraints in VerbNet :

For example, "a bat catches insets". The "bat" can be a kind of "mammal" or a kind of "club" for ball game. VerbNet constrain that "Catch" should have a human or animal as agent, so the meaning of "bat" should be a "mammal"(belonging to the "animal" category) not a "club" for sport.

2)selecting the word sense by matching the topics in the working memory and those in BNs:

E.g. "bank" has more than four meanings in its noun form in WordNet and thus can be related to topics such as "deposit", "depository financial institution", "flight maneuver", "slope" and so on. The sentence "he goes to the bank" can be confusing because the "bank" may relate to any of the topics. Two BNs containing conditional probability $P(depositmoney|gotobank)$ and $P(jumpintothewater|gotothebank)$ can limit the meaning to "bank building" and "sloping land beside water" within the topic domains "deposit" or "slope, water". Other information

in the working memory (e.g. a vertex representing "river", has the same topic "water" as "sloping land beside water") can then decide that the "bank" should be related to a "slope".

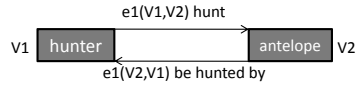
4 An Adaptive BN Mechanism for Reasoning

Our model focuses on update new activities of entities based on new information in memory. Each BN at a time infers activities of only one subject and takes in predicates from working memory to infer new ones.

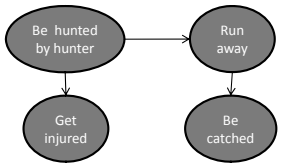
During inference, activities in the working memory are regarded as in the same scenario, and BNs are selected to the topic domain only.

4.1 The Bayesian Network Structure

BNs are used in our model to update information in ESNs. Each BN is constructed with the causal relations between predicates. Thus, the agent of the predicate should be the same as the inferred ESN. Each BN has a vector containing words and representing a topic domain. Multiple BNs will be used for different entities in the working memory. Figure 3 shows two BNs for the hunter and antelope.



(a) hunting scenario in an ESN



(b) a BN for the antelope

(c) a BN for the hunter

Figure 3: Propagating information from an ESN to BNs.

The Nodes

We choose the predicates as the nodes of BNs. Figure 3 illustrates how the predicate "be hunted by hunter" is passed from the working memory to a BN as a node .

Evidence in BNs

In BNs, each node has a property called "evidence". At the beginning of inference, nodes are set to "evidence" if they match the edges of evidence in the working memory. "be hunted by hunter"(Figure 3(b)) and "hunt antelope" (Figure 3(c)) are viewed as evidence.

4.2 BN Construction

We propose a mechanism for automaitc self-generation of BN nodes. There are two ways to construct a BN in our model.

The first way is to learn from experience. After an update in the long-term memory, if two predicates of a subject co-appear for a large number of times, they are considered to have causal relation and are extracted as a predicate pattern pair. The predicate pattern here consists of a verb phrase and the specific object category.

For example, an "alarm sounds" and 8 out of 12 "person"s hear it. The vertex "alarm" and the 12 vertices of different "person" are activated and moved to the working memory. The 12 "person"s are in the same category of human according to WordNet. So the predicate pattern is "be heard by + human". "Sound" and "heard by human" are converted to BN nodes. Both predicates share "alarm" as their subject.

VerbNet has the syntax restriction to ensure the verb phrase "be heard by" to have a correct "agent" alarm and object "human". The edge "be heard by human" only goes from the vertex "alarm" to the other 12 "human" vertices. When counting the frequency of "be heard by", the maximum number should be 12. Thus $p(heardby = true|sound = true)$ = 8/12 =2/3. Another conditional distribution we can get from the data is $p(sound|heardby)$. The conditional distribution is initialized according to the frequency.

The second method is to extract information from other knowledge bases. For example, BN can be constructed from data in the eight of the twenty semantic relations in ConceptNet as described in Table 2.

relation type	probability function	probability value
(PrerequisiteEventOf "A","B")	$P(A = true\|B = true)$	0.9
	$P(A = true\|B = false)$	0.2
(FirstSubeventOf "A","B")	$P(A = true\|B = true)$	0.7
	$P(A = true\|B = false)$	0.3
(EffectOf "A","B")	$P(A = true\|B = true)$	0.7
	$P(A = true\|B = false)$	0.2
(CapableOf "A","B")	$P(A = true\|B = true)$	0.7
	$P(A = true\|B = false)$	0.1
(SubeventOf "A","B")	$P(A = true\|B = true)$	0.9
	$P(A = true\|B = false)$	0.1
(MotivationOf "A","B")	$P(A = true\|B = true)$	0.6
	$P(A = true\|B = false)$	0.1
(DesirousEffectOf "A","B")	$P(A = true\|B = true)$	0.6
	$P(A = true\|B = false)$	0.4
(IsA "A","B")	$P(B = true\|A = true)$	1
	$P(B = true\|A = false)$	0.6

Table 2: Integrate information to a BN from ConceptNet.

Since some commonsense knowledge bases do not provide the probabilities of causal relations, we initialize the probability distributions to default values(Table 2). The probabilities of nodes without a parent are set to 0.5.

BN Selection and Connection

BNs need to be selected and joined together as a new large Bayesian net for the activity prediction.

The BN selection is based on the predicates in the sentences and the topics in the working memory. The first step is to search BNs whose nodes represent the same predicate as

in the sentences. E.g. for the sentence "He goes to the bank", BNs containing probability $P(depositmoney|gotobank)$ or $P(jumpintothewater|gotothebank)$ are all selected.

The second step is to remove BNs with a different topic from the working memory. E.g. if the working memory has entities in the categories related to "money" or edges related to financial operation, BNs without these topics are removed.

An adjacency matrix M is built for combining BNs. If BN i and BN j share a common predicate, then $M_{ij} = M_{ji} = 1$, otherwise, the element in the matrix is 0. By multiply the adjacency matrix, a path connecting the BNs can be found. This approach saves us from learning and inferring about unrelated data, and hence reduces much complexity.

The Joint Distribution Function

Suppose the shared node s has parents $s_k \in pa_i(s)$ in BN i and parents $y_l \in pa_j(s)$ in BN j, where $y_l \notin pa_i(s)$. the conditional distribution $p(s|pa_i(s))$ in BN i and $p(s|pa_j(s))$ in BN j are known and the new joint conditional distribution $p(s|pa_i(s), pa_j(s))$ should be initialized for the new network.

5 Making Inference on Evidence

There are several main steps to infer new information from new sentences:

1)A new entity from arriving sentences are automatically created as an active vertex in the working memory. If there are other entities have relation to the new entity, they are brought to the working memory as well.

2)Classify the entities into categories.

3)Predicates in the coming sentences are evidences. New predicates can be inferred from the evidence via BNs. BNs containing the to-be-inferred predicates are target BNs. They are selected for the next step.

4)Find a link path from the target BN to the BN containing evidence with the adjacency matrix. (e.g. fig.5) (If the link does not exist, then the evidence does not affect the activities in that target BN.)

5)Combine the BNs in the path to a large BN. Set the joint conditional distribution values of the share nodes.

6)Search a d-connecting path from the evidence node to the target node.

7)Calculate the probability of the variables in the BN

8)Update the working memory and the long-term memory

6 Case Study

In the following, we will compare our model and Direct Memory Access(DMAP) on the data provided by [Livingston and Riesbeck, 2009]. A scenario is built to compare the inference result of our system and ConceptNet.

6.1 Comparison with DMAP

DMAP uses a story of bombing attack at U.S. soldiers to test how well the model can understand the text. One of the stories is as follow: "An attack occurred in Afghanistan. The bombing was performed by Al Qaeda. The attack occurred on July 18, 2008. The attack targeted United States soldiers."

The aim of DMAP is to integrate the information of "bombing" and "attack" by using a language pattern to map the two words to the same reference in its memory system.E.g, the sentence "the bombing was performed by Al Qaeda" is represented as (performedBy, Bombing-54 Al-Qaeda). DMAP then searches if there are assertions such as (performedBy ?attack Al-Qaeda) to map the two words. DMAP successed in integrating the information only if "bombing" and "attack" share the same assertion pattern (performedBy xxx Al-Qaeda). However, if there is another keyword that share the assertion pattern as "bombing", DMAP would be confused. For example, if the "Al-Qaeda performed a celebration" after the attack, the three word "bombing", "attack" and "celebration" would share the same pattern "performed by Al-Qaeda". Thus, DMAP could falsely reason that "bombing" is equal to "celebration".

In our model, a disambiguation for the word sense is performed in the working memory by using WordNet. The meaning of "bombing" can be referred to "bomb", which has a meaning "throw bombs or attack with bombs". The keyword "attack" in the annotation is extracted by the Stanford Parser and matches the "attack" in the sentences. In this way, the word "celebration" can be omitted as an unrelated word.

6.2 Comparison with ConceptNet

Suppose there are three sentences in a scenario: "Mike swings a bat", "John throws baseball" and "Jones catches the ball". ConceptNet and our model will use the same knowledgebase to reason on the sentences. The following assertions can be found in the database in ConceptNet(explanation of the assertion can be seen in section 2):

1. *(CapableOf "batter" "hit ball" "f=2;i=1;")*
2. *(SubeventOf "play tennis" "hit ball" "f=7;i=1;")*
3. *(CapableOf "baseball player" "hit ball" "f=7;i=1;")*
4. *(CapableOf "bat" "hit ball" "f=2;i=0;")*
5. *(MotivationOf "play tennis" "hit ball" "f=3;i=0;")*
6. *(Isa "batter" "baseball player" "f=2;i=0")*
7. *(CapableOfReceivingAction "baseball pitcher" "throw baseball" "f=2;i=0;")*
8. *(CapableOf "baseball player" "throw ball" "f=3;i=1;")*
9. *(SubeventOf "play baseball" "throw ball" "f=2;i=1;")*
10. *(CapableOf " baseball pitcher" "throw ball" "f=4;i=1;")*
11. *(SubeventOf "play football" "throw ball" "f=2;i=0;")*
12. *(Capableof "cathcer" "catch ball")*
13. *(CapableOf "batter" "swing bat" "f=2;i=0;")*
14. *(Isa "catcher" "baseball player" "f=2; i=0")*

By matching the keywords, ConceptNet can get the following results about "Mike": "Mike might be a batter.(from assertion 13)" "Mike can hit ball.(from assertion 1)", "Mike might be a baseball player.(from assertion 3)" "Mike might play tennis(from assertion 2)" "Mike might be a bat.(from assertion 4)" Notice that "tennis" and "baseball" are different sports and "Mike" cannot play them at the same time. Additionally, ConceptNet cannot infer the exact activities of "John" because there are four different assertions regarding "throw ball"(from assertion 8 to 11).

In our model, the probability generated from data in ConceptNet is initialized as in Table 2. Joint conditional

distribution is initialized as follow:$P(S|x_k, y_l) = 0.8$, $P(S|\bar{x_k}, y_l) = P(S|x_k, \bar{y_l}) = 0.55$,$P(S|\bar{x_k}, \bar{y_l}) = 0.1$.

BNs were constructed and combined according to the knowledge base(Figure 4).

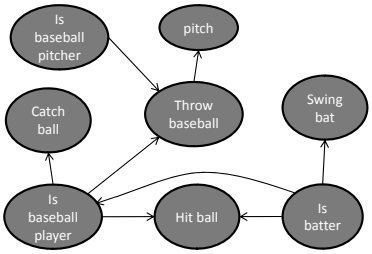


Figure 4: Constructed BN

Table 3 summaries the inferred result. We choose the threshold of probability for adding an edge in memory as 0.65, hence the first three row in table 3 are selected ass new inferred information and added to long-term memory. Notice that "is a baseball pitcher" is not in the D-connecting path given the evidence "swing bat". This means it is independent of "swing bat". Table 4 shows the inference about John's activities. Our concluded inferences, as shown in Table 3 and Table 4, are more reasonable than that in the ConceptNet.

Predicate for "Mike"	probability	value
Is batter	$P(Isbatter)$	0.77
Hit ball	$P(Hitball)$	0.67
Is baseball player	$P(Isbaseballplayer)$	0.86
Throw ball	$P(Throwball)$	0.62
Is baseball pitcher	$P(Isbaseballpitcher$	N/A
Catch ball	$P(Catchball)$	0.62

Table 3: probability of Mike's activities.

Predicate for "Mike"	probability	value
Is baseball player	$P(Isbaseballplayer)$	0.84
Pitch	$P(Pitch)$	0.70
Is baseball pitcher	$P(Isbaseballpitcher$	0.65
Hit ball	$P(Hitball)$	0.61
Catch ball	$P(Catchball)$	0.61
Is batter	$P(Isbatter)$	0.60
Swing bat	$P(Swingbat)$	0.50

Table 4: probability of John's activities.

7 Discussion and Future Work

In this paper, we built a reasoning model to represent and infer new information from texts. Experiments compare DMAP, ConceptNet and our model. Results show that our model is both robust and scalable because the disambiguation mechanism enables it to avoid inaccurate reasoning caused by confusing data. Through calculation, our model can obtain reasonable probabilities of the activities of entities.

The parameters of BNs are set to default values when BNs are first constructed. In the future, we will add a training mechanism of BNs to adjust the parameters after the BN combination. The threshold for adding edges should also be adjusted according to a specific scenario.

8 Acknowledgement

We thank Chinese Scholarship Council and the ONR grant "Symbolic and Continuous Representation in Adaptive Memory with Efficient Inference" number N00014-09-1-0069.

References

[Dahlgren, 1988] K. Dahlgren. *Using Commonsense Knowledge to Disambiguate Word Sense*. Natural Language Understanding and Logic Programming. Amsterdam, North Holland, 2 edition, 1988.

[Fellbaum, 1998] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[Kathleen and Joyce, 1989] Dahlgren Kathleen and McDowell Joyce. Knowledge representation for commonsense reasoning with text. *Computational Linguistics*, 5(3):149–170, 1989.

[Kipper *et al.*, 2006] Karin Kipper, Anna Korhonen, Neville Ryant, and Martha Palmer. Extending verbnet with novel verb classes. In *Fifth International Conference on Language Resources and Evaluation (LREC 2006)*, Genoa, Italy, 2006.

[Klein and Manning, 2003] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. pages 3–10, 2003.

[Landauer and Dumais, 1997] T. K. Landauer and S. T. Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychol. Rev*, 104:211–240, 1997.

[Liu and Singh, 2004] H. Liu and P. Singh. Conceptnet: A practical commonsense reasoning toolkit. *BT Technology Journal*, Volume 22, 2004.

[Livingston and Riesbeck, 2009] K. Livingston and C. K. Riesbeck. Resolving references and identifying existing knowledge in a memory based parser. In *the AAAI 2009 Spring Symposium*, Learning by Reading and and Learning to Read, California, USA, 2009.

[Mitchell *et al.*, 2008] Tom M. Mitchell, Svetlana V. Shinkareva, Andrew Carlson, Kai-Min Chang, Vicente L. Malave, Robert A. Mason, and Marcel Adam Just. Predicting human brain activity associated with the meanings of nouns. *Science*, 320(5880):1191–1195, 2008. 10.1126/science.1152876.

[Tulving and Thomson, 1973] E. Tulving and D. M. Thomson. Encoding specificity and retrieval processes in episodic memory. *Psychological Review*, 80(5):352–373, 1973. ISI Document Delivery No.: Q9393 Times Cited: 1541 Cited Reference Count: 91 AMER PSYCHOLOGICAL ASSOC.

Extracting Propositional Rules from Feed-forward Neural Networks by Means of Binary Decision Diagrams

Sebastian Bader
Department of Computer Science, University of Rostock, Germany
sebastian.bader@uni-rostock.de

Abstract

We discuss how to extract symbolic rules from a given binary threshold feed-forward network. The proposed decompositional approach is based on an internal representation using binary decision diagrams. They allow for an efficient composition of the intermediate results as well as for an easy integration of integrity constraints into the extraction. We also discuss some experimental results indicating a good performance of the approach.

1 Introduction

During the training process, neural networks acquire knowledge by generalising from raw data. Unfortunately, this learnt knowledge is hidden in the weights associated to the connections and humans have no direct access to it. One goal of rule extraction is the generation of a human-readable description of the output units behaviour with respect to the input units. Usually, the result is described in form of if-then rules, giving conditions that activate (or inactivate) a given output unit. Rule extraction from connectionist systems is still an open research problem, even though a number of algorithms exists. For an overview of different approaches we refer to [Andrews *et al.*, 1995] and [Jacobsson, 2005]. Extraction techniques can be divided into *pedagogical* and *decompositional* approaches. While the first conceives the network as a black box, the latter decomposes the network, constructs rules describing the behaviour of the simpler parts, and then re-composes those results.

In [Bader *et al.*, 2007], we proposed the CoOp-algorithm, a decompositional approach for the extraction of propositional rules from feed-forward neural networks. Here, we discuss an extension of this approach. In this new extension, binary decision diagrams (BDD) are used to store intermediate results, i.e., rules extracted from single units (perceptrons). This representation has three advantages:

1. results are stored in a very compact form,
2. intermediate results can easily be combined, and
3. integrity constraints can easily be incorporated.

After a presentation of all necessary concepts in Section 2, we discuss the proposed extension of the CoOp-approach, i.e., the extraction of BDDs from simple perceptrons. Afterwards, we discuss how to compose the intermediate results and how to incorporate integrity constraints. In Section 6 first experimental results are presented and further work is discussed in Section 7.

2 Preliminaries

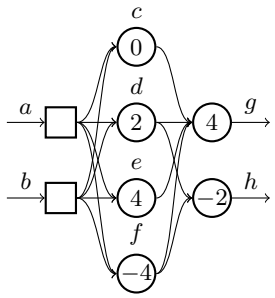
In this section, we introduce some necessary concepts. After defining feed-forward neural network and the rule-extraction problem, we discuss binary decision diagrams.

Feed-forward artificial neural networks (ANNs), also called *connectionist systems*, consist of simple computational units (neurons) which are connected. The set of units $\mathcal{U}$ together with the connections $\mathcal{C} \subseteq \mathcal{U} \times \mathcal{U}$ form an acyclic directed graph. In this paper we concentrate on networks with units applying the ± 1-threshold function. Such a neural network can be represented as a 6-tuple $\langle \mathcal{U}, \mathcal{U}_{\mathsf{inp}}, \mathcal{U}_{\mathsf{out}}, \mathcal{C}, \omega, \theta \rangle$. $\mathcal{U}_{\mathsf{inp}}, \mathcal{U}_{\mathsf{out}} \subseteq \mathcal{U}$ denote input and output units of the network, i.e., are sources and sinks of the underlying graph. The functions $\omega : \mathcal{C} \to \mathbb{R}$ assign a weight to every connection and $\theta : \mathcal{U} \to \mathbb{R}$ a threshold to every unit. Every unit u has an activation value $\mathsf{act}_u \in \{-1, +1\}$ which is set from outside for input units, or computed based on the activation value of its predecessor units and the threshold $\theta(u)$ as follows:

$$\mathsf{act}_u = \begin{cases} +1 & \text{if } \sum_{c=(v,u)\in\mathcal{C}} \mathsf{act}_v \cdot \omega(c) \geq \theta(u) \\ -1 & \text{otherwise} \end{cases}$$

Figure 1 shows a simple network serving as running example throughout the paper.

Because every unit can be active ($\mathsf{act}_u = +1$) or inactive ($\mathsf{act}_u = -1$) only, we can associate a propositional variable u to it, which is assumed to be true if and only if the unit u is active, and we use $\bar{u}$ to denote the negation of u. Furthermore, we can characterise network inputs as interpretations $I \subseteq \mathcal{U}_{\mathsf{inp}}$ of the propositional variables $\mathcal{U}_{\mathsf{inp}}$. We use $\mathsf{act}_u(I)$ to denote the state of unit u if all input units contained in I are active and all other input units are inactive. Using this notation, we can define the rule extraction problem as follows: The *rule extraction problem* for a given node u of a feed-forward



ω	c	d	e	f	g	h
a	1	-2	5	2		
b	1	1	-3	1		
c					1	0
d					2	-3
e					3	0
f					5	-2

Figure 1: A simple network serving as running example. The threshold are shown within the nodes and the connection weights are shown on the right.

threshold network $\langle \mathcal{U}, \mathcal{U}_{\mathsf{inp}}, \mathcal{U}_{\mathsf{out}}, \mathcal{C}, \omega, \theta \rangle$ is the construction of a propositional formulae F over $\mathcal{U}_{\mathsf{inp}}$ such that for all interpretations I we find

$$\mathsf{act}_u(I) = \begin{cases} +1 & \text{if } I \models F \\ -1 & \text{otherwise} \end{cases}$$

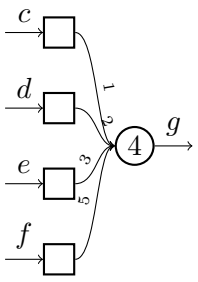
I.e., we are looking for propositional formula stating necessary and sufficient conditions (in terms of the activation of input units) such that the unit u is active.

Usually not all input combinations make sense in a given application domain, because some of them would correspond to invalid states of the world. We use the term *valid inputs* to denote the set of allowed input combinations. Even more important for the extraction is the fact that all training samples are taken from this subset. Therefore, the network learns to solve a task under the implicit conditions hidden in the selection of inputs. *Integrity constraints* are a way to make those conditions explicit during the extraction. An integrity constraint is a formula IC over $\mathcal{U}_{\mathsf{inp}}$ describing the set of valid inputs $V \subseteq \mathcal{P}(\mathcal{U}_{\mathsf{inp}})$ as follows: For all $I \subseteq \mathcal{U}_{\mathsf{inp}}$ we find $I \models \text{IC}$ if and only if $I \in V$. Using integrity constraints, we can reformulate the extraction problem as follows: The rule extraction problem for a given network N and a given integrity constraint IC is the construction of a propositional formulae F over $\mathcal{U}_{\mathsf{inp}}$ such that for all interpretations I with $I \models \text{IC}$ we find

$$\mathsf{act}_u = \begin{cases} +1 & \text{if } I \models F \\ -1 & \text{otherwise} \end{cases}$$

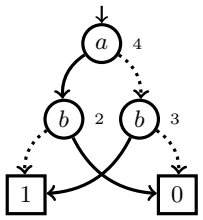
Networks can be decomposed into their basic building blocks, namely single units together with their incoming connections. Those single units can be seen as simple sub-networks (perceptrons) consisting of a number of inputs and a single output unit, together with the corresponding weighted connections. To simplify the notations we use $\mathcal{P}_p = \langle \theta, \mathcal{I}, \omega \rangle$ to denote the perceptron corresponding to the unit p together with its threshold θ, the set of predecessor units $\mathcal{I}$ and the weight function ω. Figure 2 shows the perceptron for the output unit g.

Binary decision diagrams (BDD) are a data structure to represent propositional formulae in a very compact way and to manipulate them easily. A nice introduction



$$\mathcal{P}_g = \langle \theta, \mathcal{I}, \omega \rangle \text{ with}$$
$$\theta = 4$$
$$\mathcal{I} = \{c, d, e, f\}$$
$$\omega(x) = \begin{cases} 1 & \text{if } x = c \\ 2 & \text{if } x = d \\ 3 & \text{if } x = e \\ 5 & \text{if } x = f \end{cases}$$

Figure 2: The perceptron corresponding to unit g.



$$BDD = \langle \prec, 0, 1, R, N \rangle \text{ with}$$
$$R = 4$$
$$N = \{\langle 2, b, 0, 1 \rangle, \langle 3, b, 1, 0 \rangle, \langle 4, a, 2, 3 \rangle\}$$
$$\text{pf}(2) = (b \wedge \bot) \vee (\neg b \wedge \top) = \neg b$$
$$\text{pf}(3) = (b \wedge \top) \vee (\neg b \wedge \bot) = b$$
$$\text{pf}(4) = (a \wedge \neg b) \vee (\neg a \wedge b)$$

Figure 3: A simple BDD, nodes are annotated with their variables and their ID on the right. High branches are depicted as solid and low branches as dashed lines. On the right you find the underlying data structure and the logic formulae corresponding to the internal nodes.

can be found for example in [Andersen, 1999]. Intuitively, a BDD is a directed acyclic graph with a variable associated to every node and such that all nodes $n \neq 0, 1$ have exactly two successors, called high and low branch of n. The nodes 0 and 1 are the sinks of the graph. We use $\langle \prec, 0, 1, R, N \rangle$ to refer to a BDD with sinks 0 and 1, a set of nodes N and a root-node with identifier R. And we use $\langle i, v, h, l \rangle$ to denote the node with identifier i, with variable $v = \text{var}(i)$, and with high and low branch pointing to the nodes with identifiers h and l, respectively.

Usually a BDD is assumed to be ordered and reduced. It is called ordered iff there exists a linear order $\prec$ on the variables and the successors of a node are marked with variables that are bigger with respect to $\prec$. It is called reduced if no two nodes for the same variable have identical high and low branch, and for no node high and low-branch coincide.

BDDs represent propositional formulae in if-then-else normal form. The corresponding formula for a given node is defined recursively as follows:

$$\text{pf}(0) := \bot \qquad \text{pf}(1) := \top$$
$$\text{pf}(i) := (\text{var}(i) \wedge \text{pf}(h)) \vee (\neg\, \text{var}(i) \wedge \text{pf}(l))$$

Figure 3 shows a simple BDD using a graphical representation as well as the underlying data structure and the corresponding logic formulae for every node.

3 From Perceptrons to Search Trees

In this section, we discuss an algorithm to extract a BDD from a single unit such that the BDD represents necessary and sufficient condition on the inputs to turn the unit active. Following [Bader *et al.*, 2007], we define input patterns I as subsets of the inputs $\mathcal{I}$ of a given perceptron $\mathcal{P}_p = \langle \theta, \mathcal{I}, \omega \rangle$ which are assumed to be active. The inputs not contained in I can be either active or inactive. And we define the corresponding minimal input $i_{\min}(I)$ as follows:

$$i_{\min}(I) = \sum_{a \in I} \omega(a) - \sum_{a \in \mathcal{I} \setminus I} |\omega(a)|$$

The minimal input is computed by adding the contribution of the fixed inputs $\sum_{a \in I} \omega(a)$ and the minimal input caused by all other inputs. A perceptron is called *positive*, if all weights are positive. For the following constructions, we assume the perceptrons to be positive. In Section 5, we discuss how to apply the extraction to arbitrary perceptrons.

The construction of BDDs below is based on the search trees described in [Bader *et al.*, 2007]. These search trees contain a node for every possible input pattern. Children of a given node correspond to input patterns which contain exactly one symbol more and all nodes are sorted with respect to their minimal inputs. If the minimal input of some node exceeds the threshold, that node is marked (I.e., the corresponding input pattern represents a sufficient condition to turn the perceptron active). The complete tree is pruned by removing all those nodes for which no descendant is marked and all those nodes which are descendants of marked nodes. The construction of a pruned tree is shown in Algorithm 1. Figure 4 shows the full and the resulting pruned search tree on top of it.

Input: A positive perceptron $\mathcal{P}_p^+$.
Output: A pruned search tree.

1 Fix an order $\prec$ such that $b \prec c$ if $\omega(b) \geq \omega(c)$.
2 Create a root node for the empty input pattern.
3 Add a child labelled x for each input symbol x (sorted wrt. $\prec$).
4 **foreach** *newly added node labelled* y **do**
5 Add a new child c for every symbol z with $y \prec z$ (sorted wrt. $\prec$).
6 Label c with the corresponding pattern I.
7 Mark c if $i_{\min}(I) > \theta(p)$.
8 Remove all descendants of marked nodes.
9 Remove all nodes for which no descendant is marked.

Algorithm 1: Constructing a pruned search tree.

Exploiting the structure of these search trees, we can easily construct BDDs representing conditions to turn the perceptron active. I.e., we find the perceptron to be active for all those input patterns which, understood as interpretation, turn the logic formula corresponding to the BDD true.

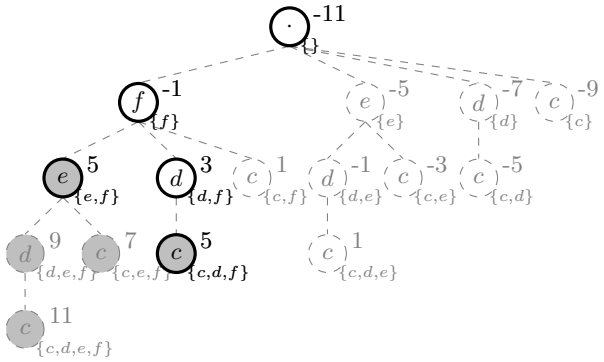


Figure 4: The pruned search tree for the perceptron $\mathcal{P}_g$ from above. The underlying full tree is depicted in grey using dashed lines. The nodes contain the newly added symbol and are annotated with the corresponding input pattern and the resulting minimal input. All nodes for which the minimal input exceed the threshold of $\theta(g) = 4$ are shown with grey background.

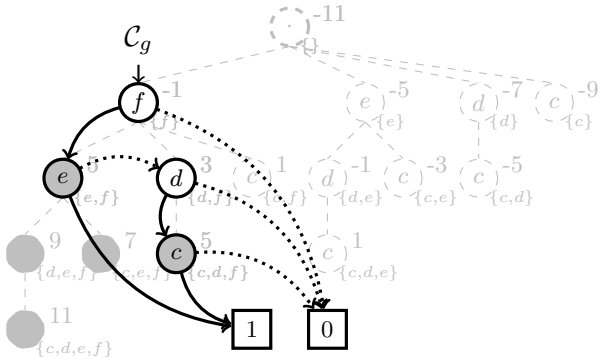


Figure 5: The BDD corresponding to the pruned search tree from Figure 4.

4 From Search Trees to BDDs

Before presenting an algorithm to construct BDDs from a given pruned search tree, we introduce some further notations. Every node in the search tree is represented as a pair $\langle I, C \rangle$ with I being the corresponding input pattern and C being the set of children. $\mathrm{id}(n)$ denotes a unique identifier for the node n (e.g., the corresponding input pattern, or some index), this identifier is also used as internal index for the BDD nodes. We assume $\mathrm{id}(n) := 0$ if there is no node n. $\mathrm{var}(n)$ denotes the symbol which is added to the input pattern at node n.

The construction of a BDD for a given search tree is shown as Algorithm 2. This algorithm transforms a search tree into a BDD, by traversing the tree in a left-depth-first manner. A node's high branch points to 1, if its minimal input exceeds the threshold. Otherwise, it points to its left-most child, or to 0 if there is no child. The low-branch points to the right sibling, or to 0 if there is none. The result for the perceptron $\mathcal{P}_g$ is shown in Figure 5.

Input: A search tree T for $\mathcal{P}_p = \langle \theta, \mathcal{I}, \omega \rangle$ wrt. $\prec$.
Output: A corresponding OBDD $\langle \prec, 0, 1, R, N \rangle$.

1 **if** T *is empty* **then**
2 | $R = 0$ and $N = \{\}$
3 **else if** T *contains only the root node* **then**
4 | $R = 1$ and $N = \{\}$
5 **else**
6 | $R = \mathrm{id}(r_l)$ for the leftmost child of the root.
7 | $N = \{\}$.
8 | **foreach** *leaf node* n *in* T **do**
9 | | Add $\langle \mathrm{id}(n), \mathrm{var}(n), 1, l \rangle$ to N with $l = \mathrm{id}(r)$ for the right sibling r.
10 | **foreach** *node* $\langle I, C \rangle$ *in* T *with left sibling* l **do**
11 | | Let $\langle \mathrm{id}(l), \mathrm{var}(l), h_l, l_l \rangle$ be the node for l
12 | | Let $\langle \mathrm{id}(l_l), \mathrm{var}(n), h_{l_1}, l_{l_1} \rangle$ be the node for the leftmost child l_l of l
13 | | **if** $\mathrm{mci}(l) - 2\omega(l) + 2\omega(n) > \theta$ **then**
14 | | | Add $\langle \mathrm{id}(n), \mathrm{var}(n), l_{l_1}, l_n \rangle$ to N with $l_n = \mathrm{id}(r_n)$ for the right sibling r_n of n
15 | **foreach** *other non-root node* $\langle I, C \rangle$ **do**
16 | | Add a node $\langle \mathrm{id}(n), \mathrm{var}(n), \mathrm{id}(c), l \rangle$ to N for the leftmost child c and $l = \mathrm{id}(r_n)$ for the right sibling r_n of n and $l = 0$ if there is none

Algorithm 2: Constructing a BDD.

Please note, that the BDD can be constructed without constructing the search tree first. The tree is used only to describe the underlying ideas. All conditions tested in Algorithm 2 can be tested by expanding the tree step-by-step. Looking a little closer at the constructed search tree we find that some sub-trees have an identical internal structure, which is exemplified in Figure 6. If the condition tested in Line 13 of Algorithm 2 is fulfilled, two neighbouring sub-trees are structured identically. $\mathrm{mci}(n)$ denotes the minimum of all minimal inputs associated to nodes below n. Please note, that $\mathrm{mci}(n)$ can be computed without expanding the sub-tree by looking at the associated input pattern.

The mentioned structural equivalence can be exploited by using a shortcut into the already constructed BDD and thus preventing the expansion of an identical sub-tree. Figure 6 contains a number of those shortcuts, e.g., one from node $\{b\}$ to the node $\{c, a\}$, because the children of $\{b\}$ are annotated the same way as the node below and right of $\{c, a\}$. Please note that this identity can be recognised without expanding the second sub-tree, i.e., the construction of whole tree below $\{b\}$ can be avoided.

The condition on Line 13 is fulfilled whenever the perceptron shows a so called n-of-m behaviour, i.e., if there are m inputs from which n suffice to turn the perceptron active. In this case, there will be n equivalent sub-trees, which can be shortcut. As discussed in [Towell and Shavlik, 1993], this occurs quite frequently while training neural networks.

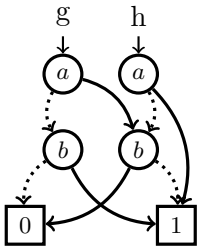


Figure 7: The global BDD for the network from Figure 1.

5 Composition of Intermediate Results

In the previous section, we have been concerned with positive perceptrons only. But we can easily turn every perceptron into a positive one, by multiplying negative weights by -1 and inverting the corresponding input symbols. By doing so, we can apply the algorithm to all output units of a given network, and obtain a BDD describing necessary conditions with respect to the predecessor units that turn the output unit active. But some of the input symbols may have been inverted. I.e., we need another algorithm to construct BDDs stating conditions which turn a perceptron inactive. Due to the symmetry of the threshold function, we find this algorithm to be dual to Algorithm 1 and 2. I.e., by inverting the order and the inequalities we obtain an algorithm that constructs such a BDD.

Once we have extracted the BDD for a given output unit, we can continue by substituting the nodes testing non-input nodes (i.e., nodes not corresponding to input units of the network) by their corresponding BDDs. A non-negated node is replaced by the BDD as constructed above, and negated nodes are replaced by the dual BDDs. As mentioned above, BDDs have been designed to allow for an efficient manipulation of logic formulae. And in fact it is straightforward to compose the intermediate results into an overall diagram by simply replacing the nodes. But this is not the best approach, because the resulting 'global' BDD would not be ordered any more. But while expanding the BDD, we can keep it ordered (and reduced) as described in [Andersen, 1999]. After expanding all non-input nodes, we obtain a final BDD representing necessary and sufficient conditions on the network's input to turn a given output unit active.

Using BDDs as internal data structure has some further advantages. We can actually extract all output units into the same global BDD. Doing so leads automatically to a sharing of intermediate results, because common substructures are contained only once within this BDD. Figure 7 shows the final 'global' BDD for the network from Figure 1. Please note that the right node labelled b is used for both output units g and h.

Furthermore, we can integrate integrity constraints in a straightforward fashion. Instead of starting with an empty BDD, we extract the output nodes into a BDD representing the integrity constraints. To exemplify this, a network has been trained to the Encode-Decoder task. It contains 8 input, 8 output units and 3 hidden units

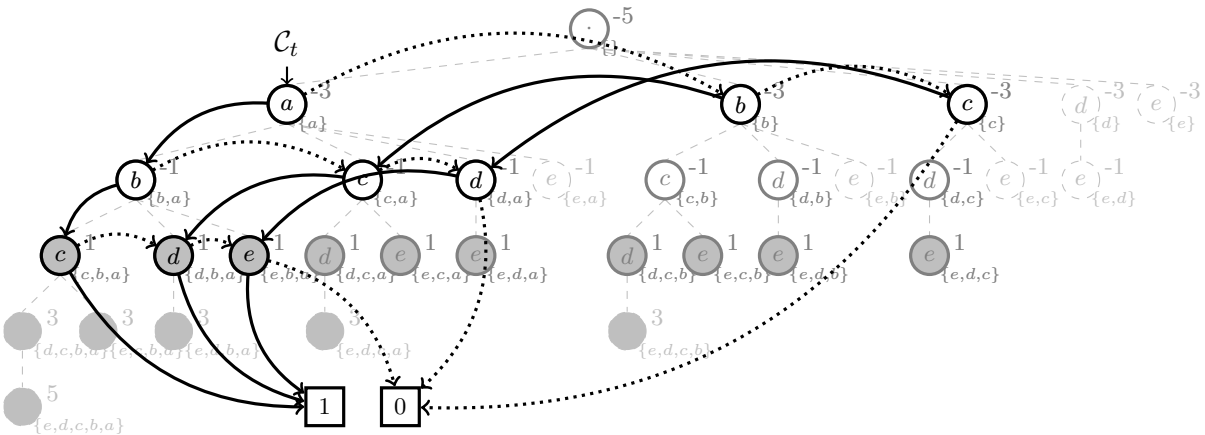


Figure 6: A larger BDD and its underlying search tree. The tree has been constructed for a perceptron with 5 inputs whose weights are all 1 and with threshold 0. I.e., this perceptron is active if 3 of the five inputs are +1. Please note that there are certain symmetries in the tree: All label and minimal inputs of the sub-trees of node $\{b\}$ coincide with those of the right sub-trees of $\{a\}$ if a is substituted by b. This has been exploited by linking from $\{b\}$ to $\{c, a\}$.

and is trained to learn the identity mapping for all inputs in which exactly one unit is active. I.e., the network has to learn a compressed representation within the hidden layer. But applying the algorithm presented above yields an unwanted result shown in Figure 8 on the left. Using the integrity constraint that at most one input is active at a time yields the BDD shown on the right. 900 nodes have to be constructed (including all intermediate results while constructing the BDD) for the 'normal' BDD, but only 124 while using the integrity constraint. This shows the advantage of using integrity constraints right from the beginning of the extraction process. Usually they are used to refine the extraction result afterwards. This would be possible here as well by simple computing the conjunction of the 'normal' BDD with one representing the integrity constraint. But starting with the constraint avoids the construction of many intermediate nodes which would be removed afterwards.

6 Experimental Evaluation

To evaluate the approach a Prolog implementation has been used to gather some statistics. The results are shown in Table 1. The table shows average numbers for different numbers of inputs, the size of the full search tree, the number of minimal input patterns, the size of the corresponding BDD and the number of BDD nodes per input pattern. All numbers have been collected from 100 random perceptrons per size. The extraction using the full search tree is not feasible due to the exponential growth. The number of input patterns is a conservative lower bound for the size of the pruned search tree, because those trees have at least one node per minimal input pattern. The result shows that the use of BDD proposed here yields a very compact representation. Even though the number of nodes in the BDD grows, the ratio (node/IP) of size of the BDD and the number of minimal coalitions decreases.

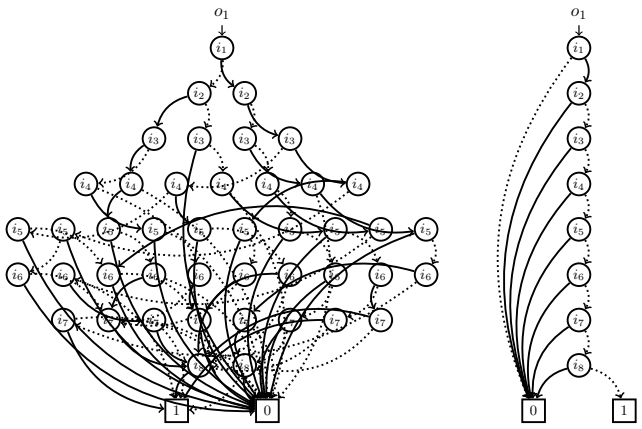


Figure 8: The result of extracting output unit 1 from an 8-3-8 encoder-decoder network. The BDD on the left is the result of the 'normal' extraction. On the right the constraint that at most one unit is active has been incorporated.

$\lvert\mathcal{I}\rvert$	$\lvert$T$\rvert$	#IPs	$\lvert$BDD$\rvert$	node/IP
1	2	0.58	2.58	4.448
5	32	4.31	8.49	1.969
10	1024	63.51	49.97	0.786
15	32768	1270.45	313.25	0.246
20	1048576	25681.70	1863.90	0.072

Table 1: The size of the full search tree ($|$T$|$), the number of minimal input patterns as a lower bound for the size of the pruned search tree (#IPs) and the corresponding BDDs ($|$BDD$|$) for different number of inputs ($|\mathcal{I}|$).

A second experiment has been performed to show the effect of the usage of integrity constraints while extracting the BDDs. A network with 6 inputs, 4 hidden and 2

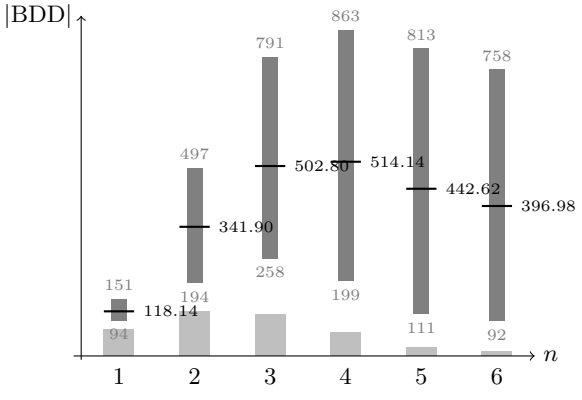


Figure 9: Resulting BDD sizes of the extraction for different $\max_n$-integrity constraints. The bars indicate minimal, maximal and average sizes of the BDDs. The size of the sub-BDD for the constraint is shown in grey.

output units has been used for the experiment. The possible inputs have been constrained by a $\max_n$ integrity constraint for $0 \leq n \leq 6$. The results are presented in Figure 9. For every n the experiment has been conducted for the same 100 randomised networks and the following numbers have been collected: the size of the sub-BDD encoding the constraint, the minimal, maximal and average size of the final BDD. Please note that the numbers show the total number of internal nodes constructed for the BDD, i.e., including all necessary intermediate nodes. For $n = 1$, i.e., the biggest restriction, we obtain very small BDDs. The size of the BDD grows up to $n = 4$ and decreases again for $n > 4$. From those observations we can conclude that the incorporation of integrity constraints into the extraction process can lead to big savings in terms of nodes constructed for the final BDD. Without their use during the extraction, we would have to construct the BDD corresponding to $n = 6$. This big BDD of ≈ 400 nodes would have to be refined with respect to the constraints afterwards. There seem to be cases (e.g., for $n = 4$) where the use of integrity constrain yields larger BDDs, but nonetheless, the final BDD does not have to be revised afterwards, and the difference is not too big.

7 Conclusions and Future Work

A novel approach for the extraction of propositional rules from feed-forward networks of threshold units has been presented. After decomposing the network into perceptrons, binary decision diagrams representing preconditions that activate or inactivate the perceptron have been extracted. Those intermediate representations can be composed using the usual algorithms for BDDs, or they can be combined during their construction by extracting one into the other. The latter approach does also allow for an incorporation of integrity constraints – already during the extraction of the intermediate results. As already mentioned in [Bader *et al.*, 2007], the pruned search trees constructed above are related to the approach presented in [Krishnan *et al.*, 1999]. But due to a different order, we do not need to expand them completely, which would otherwise be necessary.

The extraction as presented here is applicable to all feed-forward networks composed of binary threshold units computing ± 1-threshold function. This limitation can be softened by allowing arbitrary symmetric threshold functions. The symmetry is necessary to construct negative and positive forms of the perceptron without changing the global network function.

Finally, we discussed first experimental results indicating a good performance of the approach. On the one hand, we obtain a very compact representation and on the other hand, we circumvent the construction of non-necessary intermediate results while incorporating integrity constraints right from the start.

Nonetheless, much remains to be done. In particular, the extraction for non-threshold units has to be studied. For the encoder-decoder experiments mentioned above the algorithm has simply been applied to networks computing the symmetric hyperbolic tangent as activation function. Interestingly, the result coincide with our expectations. This is due to the fact, that networks when trained to compute crisp decisions tend to behave like threshold networks. But the details of this need to be investigated in the future. Furthermore, a detailed analysis of the performance is necessary, in particular using networks trained for real-world problems. The approach as presented here detects equivalent sub-BDDs for n-of-m patterns. But there are more cases for equivalent sub-BDDs [Mayer-Eichberger, 2008]. Those have to be integrated into the extraction procedure. It would also be interesting to study the evolution of a network during the training process by repeatedly applying the extraction method and compare the results.

Acknowledgements The author is thankful for the comments of Valentin Mayer-Eichberger and two anonymous reviewers.

References

[Andersen, 1999] H. R. Andersen. An introduction to binary decision diagrams. Lecture Notes, 1999.

[Andrews *et al.*, 1995] R. Andrews, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge–Based Systems*, 8(6), 1995.

[Bader *et al.*, 2007] S. Bader, S. Hölldobler, and V. Mayer-Eichberger. Extracting propositional rules from feed-forward neural networks — a new decompositional approach. In *Proceedings of the 3rd International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'07*, January 2007.

[Jacobsson, 2005] H. Jacobsson. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263, 2005.

[Krishnan *et al.*, 1999] R. Krishnan, G. Sivakumar, and P. Bhattacharya. A search technique for rule extraction from trained neural networks. *Non-Linear Anal.*, 20(3):273–280, 1999.

[Mayer-Eichberger, 2008] V. Mayer-Eichberger. Towards solving a system of pseudo boolean constraints with binary decision diagrams. Master's thesis, Universidade Nova de Lisboa, SEP 2008.

[Towell and Shavlik, 1993] G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

Symbol emergence in design

Amitabha Mukerjee, Madan M Dabbeeru
Indian Institute of Technology Kanpur
amit@cse.iitk.ac.in, mmadan@iitk.ac.in

Abstract

A key step in mapping the more conceptual stages of design onto computational systems involves identifying a vocabulary and ontology. While a number of high-level ontologies have been proposed, these are difficult to ground in terms of actual design instances, and manual definitions of the symbols are often incomplete and difficult to maintain. As an alternative, we propose an "infant designer" paradigm which abstracts patterns for the "functionally feasible regions" (FFR) while evaluating many individual configurations in the design space. These learned FFR patterns (which may arise due to minimal levels of functional acceptability, or from optimization) often embody dependency relationships among the design parameters, i.e. the good designs lie along lower-dimensional manifolds in the design parameter space. We show how such manifolds exist in several design situations; each combination of the original design parameters may be thought of as a "chunk"; the space of these chunks models only the "good designs". Next, we show how the patterns defined based on these chunks constitute image schemas, which may be implicit (e.g. the pattern for an FFR), or explicit (where the relationship is observable). These patterns or image schemas are incipient semantic model leading to symbols. We present examples of how such image schemas are arrived at with the help of universal motor design.

1 Efforts towards standardizing the design vocabulary

Evolving a standardized vocabulary for design has emerged as an important focus in engineering design. Possible applications include developing design repositories [Bohm *et al.*, 2005], computer assisted conceptual design [Gero and Fujii, 2000], etc. It is clear that vocabularies are structured, that is there are considerable relations between terms. Often, this is viewed as an ontology or as a structured relationship that captures a part of the semantics of these terms. One popular view of the engineering system considers the flow of energy, information, etc, and proceeds downward into detailed design. With its roots in value engineering ideas from the 1940s, these notions were seeded by the analysis in Pahl and Beitz [Pahl and Beitz, 19881996] and a particularly influential study by Welch and Dixon [Richard and Dixon, 1994], leading to modern ontological models like the widely used *functional basis* model [Hirtz *et al.*, 2002] or implementations on ontology tools [Nanda *et al.*, 2007; Szykman *et al.*, 2001].

The above represents the human-engineered approach to defining symbols. This type of approach is initially tempting because it tends to meet immediate applications, but a long history in knowledge-based systems has shown it to be brittle, i.e. subject to failure under even minor deviations in the domain. In general, it may be that symbols are more meaningfully developed by abstracting from existing data. The novel contribution of this paper is to show that at least in certain types of design tasks, lower-dimensional surfaces are revealed by multi-objective optimization. The intrinsic dimensions in these pareto-surfaces might constitute one approach to obtaining "symbols" directly from experiential data as opposed to engineering them by programming definitions / rules. These approaches are detailed further in section 1.2 and section 3, but first we look more closely at the term "symbol", and what is understood by its semantics.

1.1 The semantics of design symbols

Unfortunately the term "symbol", as it is used in the logic and computational theory is considerably different from its usage in cognitive linguistics and in everyday life. In the latter usage, symbols are imbued with meaning grounded on experience, whereas in the formal usage, it is merely a token constructed from some finite alphabet, and is related only to other such tokens. If we present an analogy, a blind man knows "red" is a different color from "blue" and "green" but his understanding of red is dramatically different from that of a sighted person, because the semantic pole is not connected to direct experience.On the other hand, "symbol" has come to be understood in cognitive science (and also traditionally in linguistics, e.g. de Saussure ([De Saussure, 19161986]), as the tight binding of the of the psychological impression of the sound (the "phonological pole") with the mental image of the meaning (the *semantic pole*) [Langacker, 1986]. The mental image or image schema includes all sorts of associations and

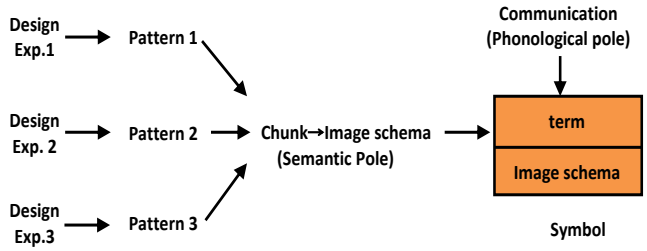


Figure 1: *Emergence of symbols based on experience*: Often the same abstract pattern (or *chunk*) appears in many experiences (e.g. the notion of "containment" for peg in hole, bolt in latch, plug in sink, etc.). If a chunk is valuable in compactly representing many situations, it has a higher likelihood of being communicated, thus acquiring a phonological pole and becoming a symbol. A symbol can then form other associations besides the initial chunk, all of which together constitute its semantic pole or *image schema*.

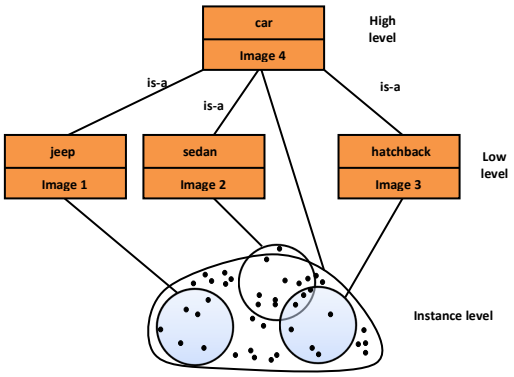


Figure 2: *Abstraction starts with ground instances*: Symbols like "hatchback", "sedan", or "jeep" may correspond an abstract pattern or "image schema", which is used to identify instances as belonging to a symbol category, but also in composing symbols, and in interpreting higher abstractions. Primitive design ontologies like *is-a* arise when instances already known as sedans or hatchbacks are also labelled as "car" by a trusted user. Similarly, other relations e.g. "jeeps can drive over rough terrain" would also be learned through usage and become part of the image schema. The number of such associations for each symbol is often very large, and limiting these to a few user-determined definitions is a major contributor to brittleness in knowledge systems.

is somewhat different for each user, though social convention ensures a degree of overlap between mental images within the language community.

However, the notion of symbol is more far-reaching than communication. It turns out that to some extent, the symbols help divide up the world into classes, and eventually, it may reflect changes in how we think. For instance, Korean language makes a distinction between spatial tight-fit situations, *kkita*, (as in "put the cap on the pen", "hand in glove") from other usages of "in" or "on". Infants growing up in English and Korean linguistic environments were sensitive to both contrasts, but English children appear to lose this sensitivity around the time they start acquiring language, suggesting that the language construct may have weakened their sensitivity to these changes [McDonough *et al.*, 2003].

On the other hand, incompatibility of design vocabulary is rarely a problem between humans (that's why exceptions often become memorable). If designers A and B are talking, and A does not have a particular symbol λ, its image-schema may emerge through a small amount of discussion; in many cases, just a single example may be enough to stretch an existing concept $\lambda\prime$ in A to the current one. Of course, the new symbol $\lambda\prime$ remains imprecise, and designer A is aware of it, and subsequent uses of $\lambda\prime$ will serve to ground it. All this is possible because the semantic pole for the human is a complex, elastic set of associations that cannot be defined in terms of a single predicate or even a range, it is the set of all situations where the symbol may be encountered (figure 2). All these associations need to be learned, and cannot be inferred based on a single definition (not to mention issues such as nonmonotonicity); hence the programmer-given single definition, usually created to demonstrate the example at hand, is a hopelessly inadequate semantics for a design symbol; and that is why we need bottom-up symbol discovery in order to ground a design vocabulary.

1.2 Bottom-Up Semantics in design

An alternative that has been proposed for modeling design concepts is to attempt to move more towards the human process, to learn symbols based on design experience[Gero and Fujii, 2000]. The human design process is a constant, motivated exploration of the design space, e.g. through sketching. All the while, the designer is focusing on the designs that are "good" in some functional sense, and eventually, some kinds of patterns emerge as the common characteristics of these designs. This is one sense in which sketches "talk back" to the designer [Goldschmidt, 2003]. These patterns result in constraints whereby many of the initial design variables can be combined, a process cognitively known as *chunking* [Gobet *et al.*, 2001].

For example, in designing a padlock, we may learn that the shackle diameter increases roughly in proportion with body size. Thus these two parameters can then be brought down to a single chunk. These chunks, which limit the choices used in "good designs", may be what are used by expert designers [Gross, 1986].

An early attempt at discovering patterns in the design space of shapes may be seen in relation to 2D shapes in the work of [Park and Gero, 1999]. [Moss *et al.*, 2004] have developed a system in which a design observer agent considers trends among good designs and try to extracts chunks. Similarly a recent approach by [Sarkar *et al.*, 2008], considers Singular Value Decomposition (SVD) on a co-occurance matrix of matrix of variables and constraints to identify the relations between different variable groups.

However, none of these proposals attempt to learn their symbols in a grounded manner, and therefore lack the flexibility of the human designer. By *grounded*, we refer to the progressive manner in which a human designer learns her concepts - the more abstract ones are based on earlier, concrete concepts, but are still presented through instances. In the end, many concepts are grounded in terms of a number of experiential instances. For a human designer, this learning cannot be limited to the years of training as a designer, but must include all of her knowledge about the world, the so called commonsense knowledge. Thus, the fact that a fat peg will not go into a thin hole is part of her prior knowledge. Indeed, it is likely that the process by which she acquires these patterns, built upon many layers of pre-existing knowledge, may be similar in some salient ways with her earliest learning.

In this work, we propose to take the first step towards building such a grounded semantics, which we call the birth of symbols. In a human design scenario, say while "talking" to a sketch, a designer may get a conscious awareness of a constraint without verbalizing it - this is referred to as *reification*, becoming real - and is a key step in forming new symbols. Sometimes, amorphous implicit schemas, which are formed well before we are aware of them [Gladwell *et al.*, 2005] are incipient symbols, but they need to prove their mettle before they become true symbols. This interpretation is in line with a long tradition in psychology and linguistics, that symbols are "aware" or conscious [Mandler, 2004].

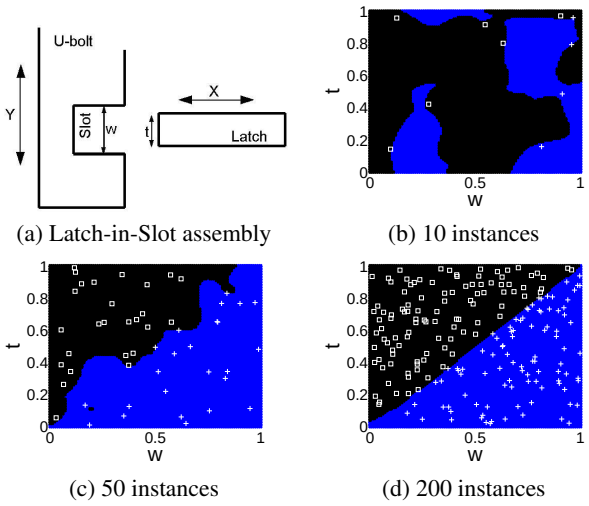
2 Infant designer

A system learning symbols is like a baby who is first discovering regularity of object behaviour in the world. She can make various choices, and evaluate them based on some notion of function. Considering the peg-in-hole task just alluded to, we see how she might learn the concept that a peg must be smaller than a hole.

The functional model considered is simple - the design is functionally feasible if the peg can go in (actually our system computes the configuration space - the penetration region disappears when $w > t$). We consider a horizontal version of the peg-in-hole - a latch is entering a slot on a bolt, say. Figure 3 shows how after evaluating a number of instances in the design space of latch-widths w and slot-widths t; in (w, t) space, a clear 45 degree line emerges, separating the "good designs" from the bad.

Does this constitute symbolic knowledge for the infant designer? Most likely not. However, it is something that might become a symbol as she acquires other concepts that she can refer to. What is interesting in the results of figure 3 is how, after experiencing just a few instances, the pattern is inchoate, so the baby keeps trying to insert the fat square into the smaller circle, filling up the negative (black) area of the figure. Eventually the defining boundary becomes sharper, and at some point it can be said to knows the principle, at least implicitly.

At the next step for our infant designer, we consider the concept that a designer knows as "fit". By now our infant learner will attempt to insert pegs only if they are smaller than



(a) Latch-in-Slot assembly (b) 10 instances

(c) 50 instances (d) 200 instances

Figure 3: *Learning through experience that latch-must-be-smaller-than-slot ($w > t$).* (a) A latch of thickness t is fitted to a slot of width w. The learned patterns are shown in (w, t)-space in (b)-(d). The quality of the learned pattern varies greatly with degree of experience: results shown for a multi-layer perceptron after experiencing 10,50, and 200 design instances.

the slot. The function is defined in terms of the degree of *fit* - how much does it wiggle? Defining the wiggle in terms of the area of the free-space in the configuration space, we see that if the wiggle desired is very small, we get the situation on the left, and if it is very large, we get the situation on the right. Eventually, the learner learns the concept of "fit" as a chunk (composed as $w - t$) - thus, given a level of fit, it imposes a constraint where w and t are related in a manner where they constitute a one-dimensional chunk instead of two independent variable.

Of course, from a machine learning perspective, both these examples are rather elementary. Our objective in presenting it is merely to emphasize the role of even the earliest knowledge in many advanced design situations. These two concepts are also among our earliest knowledge achievements; typically, infants learn containment (peg in hole) by about 3 months, and tight vs loose by 5 months [Casasola *et al.*, 2003]. Many cognitive scientists believe that our concepts of abstraction, including the *is-a* crucial to constructing hierarchies, is a metaphorical extension of containment [Lakoff and Johnson, 1999].

3 Symbol emergence

As the designer matures from infancy, we can consider the more general process by which symbols form. These may correspond to the stages shown in figure 5. At first, the designer explores with instances in the design space, distinguishing the good designs from the bad. Eventually a subset of the design space emerges as the Functionally Feasible region (FFR), or the space of "good designs". Often, FFRs correspond to narrow bands of functional feasibility. This

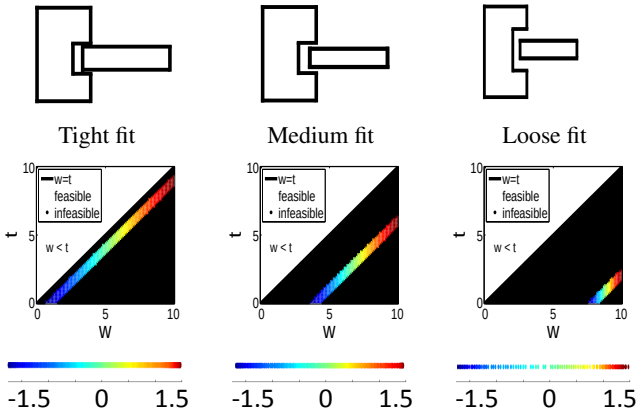


Figure 4: *Birth of the image-schema for "fit":* An insertion task with different kinds of fit are shown in the top row and the corresponding design spaces (w, t) with feasible and infeasible regions are shown below. The function is given as the amount of play available (amount of free-motion or wiggle). If the desirable wiggle is specified, the two-dimensional design space is effectively reduced to one since a relation emerges between the feasible w and t. This mapping or image schema is a early prototype of the concept of "fit".

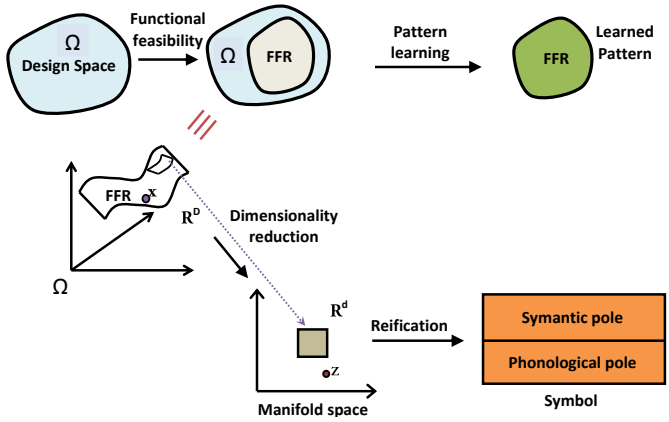


Figure 5: *The symbol emergence process*: our main interest is to discover and learn structural or behavioral chunks that result in good designs, corresponding to functionally feasible regions (FFRs) in the designs space. FFRs typically reflect multiple functional criteria, and may be obtained from some approximate optimization, or from user specified minimal functional criteria. A set of FFR instances can be used to learn a pattern of functional feasibility, the quality of this pattern improves with experience as earlier. Once the FFR is sufficiently rich, one may also discover that they lie along some low-dimensional manifold (R^d) embedded in the high-dimensional design space R^D ($d \ll D$). The lower dimensional space is then a chunked representation for the initial design space. If this relation becomes conscious, it may then become a design symbol.

may be because they are the result of (possibly unconscious) multi-objective optimization - thus, if there are k design objectives, then they constitute a $k-1$ surface in the objective space. In continuum design situations (i.e. the search space is continuous and not combinatorial), if the function measures that map from the design variable space to the objective space are continuous, their Jacobians would be well-posed, and the near neighbours in the objective space may correspond to near neighbours in the design space. While this assumption is flawed for a large class of difficult optimization problems (e.g. Quadratic assignment), it often holds for a large if not preponderant fraction of real tasks. Thus, in such situations, we may designs that lie along a $k-1$ pareto-surface (or "manifold") in the objective space (shown as a folded patch in the figure), and a similar lower-dimensional manifold in the design space as well. Each dimension of this lower dimensional space reflects an inter-relation between independent design parameters (e.g. the shackle diameter and the lock size). Sometimes, some of these dimensional mappings or chunks may recur in many design situations - this makes the chunk useful, which is an important criteria for becoming a symbol. In the interim, the designer may use these chunks with a dim awareness of it for a long period, even several years. Later, a label may get attached to it, and many other associations would eventually accrue to this term / image-schema pair; it would then constitute a truly reified symbol.

Thus a key aspect of design symbol formation is *dimensionality reduction*, - i.e. finding low-dimensional patterns in high-dimensional space. There are two classes of dimensionality reduction algorithms - linear methods like PCA or ICA [Bishop, 2006], or nonlinear approaches, which may be global (Isomaps [Tenenbaum *et al.*, 2000]) or local (Locally Linear Embedding or LLE [Saul and Roweis, 2003] and Laplacian Eigenmaps [Belkin and Niyogi, 2002]). Here we present some results based on the LLE algorithm, which is an eigenvector method that works based on the assumption that the same weighted sum between neighbours would hold both in the high and the low dimensional spaces (algorithm 1).

3.1 Universal Motor example

We illustrate the working of the process based on the Universal Motor,which has been well studied in the product family design literature [Simpson, 1998]. The design space consists of eight design variables: N_c (number of wire turns on armature) N_s (number of turns on each field pole), A_{wa} (cross-section area of armature wire), A_{wf} (cross-section area of the field wire), r_o (radius of motor), t (thickness of stator) , I: (current drawn by motor), L (stack length). Function is measured through a set of performance behaviors: strength, mass, energy and efficiency. The corresponding performance metrics in terms of these design variables can be $\pi_{torque}(\vec{v}) = \frac{N_c \phi I}{\Pi}$, $\pi_{mass}(\vec{v}) = mass_{windings} + mass_{armature} + mass_{windings}$, $\pi_{power}(\vec{v}) = V_t I - I^2(R_a + R_s) - 2I$, and $\pi_{efficiency}(\vec{v}) = \frac{\pi_{power}}{V_t I}$. (following [Simpson, 1998]). We may now consider that the feasible designs have (i) the magnetizing intensity $H < 5000$ and (ii) the outer

Algorithm 1 Local Linear Embedding

1. Compute the neighbors X_j of each data point, X_i.
2. Compute the weights W_{ij} that best reconstruct each data point X_i from its neighbors, minimizing the reconstruction error ($\epsilon(W) = \sum_i |X_i - \sum_j W_{ij} X_j|^2$) by constrained linear fits.
3. Compute the vectors Γ_i best reconstructed by the weights W_{ij}, minimizing the quadratic form ($\Phi(\Gamma) = \sum_i |\Gamma_i - \sum_j W_{ij} \Gamma_j|^2$) by its bottom nonzero eigenvectors.

radius of the stator r_o greater than the thickness of the stator t.

We next outline two experiments designed to reveal the inter-relationships in the parameter space when it comes to the optimized designs. The results suggest that the optimized designs are not scattered uniformly across the design space, but reveal certain inter-relations between the design parameters. Thus, the initial parameter space of 8 parameters may actually constitute only two independent parameters when it comes to the optimized designs. While these results hold only for these design classes, the implications might be more general, and imply far-reaching consequences in obtaining symbols as dimension-reducing patterns in continuous parameter space of a wide ranging set of problems. However, whether these results will scale up to other remains a subject of considerably more research; the results below only indicate that this may be so.

3.2 Two-dimensional design space

In an initial experiment, we consider a minimal parameter set for the universal motor - modeling the design variability in terms of only two design parameters L and I, while keeping other parameters constant [Simpson, 1998]. For a desired functional range of power 280 W$< \pi_{power} <$ 295, the FFR (the valid designs resulting from this constraint) is shown in Figure 6(a). These lie along a small band, which can be thought of as a curved 1-D manifold (with a slight thickness). 6(b).

The mapping between the nonlinear feasible region (Fig. 6 (b)) and the one-dimensional chunk for it below (Fig. 6 (c)) shows the continuity of mapping between these. If we take three data points A,B, and C in L, I space. Let us say X = [A B C], each data point is a real-valued vector, with of dimensionality 2. With the help of Local Linear Embedding (LLE) algorithm [Roweis and Saul, 2000], we construct a neighborhood preserving mapping from L, I space to Γ. The three points A= $(32.0, 4.09)$, B= $(22.5, 3.5455)$ and C= $(10.5, 12.000)$ and their corresponding mappings in the lower-dimensional manifold are $\gamma_A = -0.2102$, $\gamma_B = -0.1430$ and $\gamma_C = 0.0007$.

This reduction of the two design parameters to a single γ represents the first stage of symbol formation. If, later, this γ chunk is discovered in other situations, then a label, say "gavagai", may attach to it. Then as the term "gavagai" may spread in the design community, and might occur in many

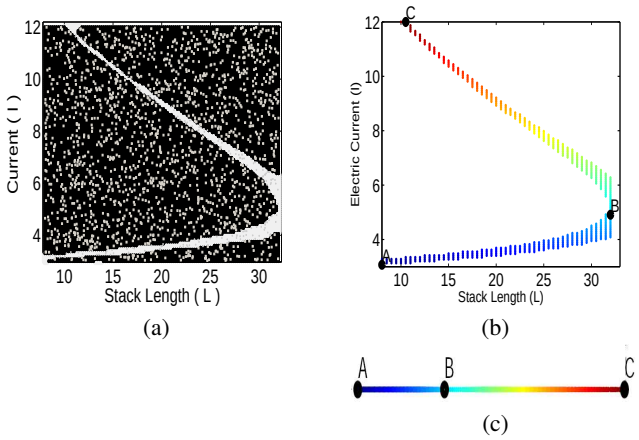


Figure 6: *Chunking on the L, I subspace for Universal Motors*: (a) The implicit constraint on the L, I subspace of the Design Space is learned for 2000 design instances under the functional specification 280 W$< \pi_{power} <$ 295 W. (b) The feasible designs in the L, I subspace. (c) The mapping onto a low-dimensional (1-D) space; this reveals that for good designs, the stack-length L and the motor current I are related. A,B, C: individual design instances in (b) and (c).

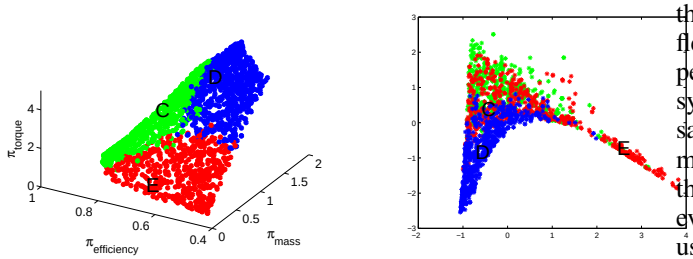
other situations, and each such association would form part of the semantics of the term *gavagai*. A computational system that learns this term in this way would need to participate in such discussions in the design community to keep its semantics current. This is another reason why static programmed machine semantics, even if they can capture all the usages at a given point of time, fail in the long run as human usage changes.

3.3 High-dimensional spaces: Multi-Objective Optimization

If we are to consider the eight-dimensional design space for the Universal motor, a more useful approach towards finding FFRs may be to consider a multi-objective optimization problem based on a set of performance metrics. If design solution A is better than solution B in all the functional criteria, we say that A *dominates*— B. The set of all non-dominated solutions is the non-dominated front or *pareto-front*, and usually lies along a surface in the space of objective functions. For the Universal motors example, the multi-objective optimization problem may be formulated as follows:

Multi-Objective Optimization

$$\begin{array}{ll} \text{Minimize} & \pi_{mass}(\underline{v}) \\ \text{Maximize} & \pi_{efficiency}(\underline{v}) \\ \text{Maximize} & \pi_{torque}(\underline{v}) \\ \text{Subject to} & g_1(\underline{v}) \equiv r - t > 0 \\ & g_2(\underline{v}) \equiv 5000 - H > 0, \\ & g_3(\underline{v}) \equiv 2.0 - \pi_{Mass} \geq 0, \\ & g_4(\underline{v}) \equiv 0.5 \leq \pi_{torque} \leq 5.0, \\ & g_5(\underline{v}) \equiv 300 \leq \pi_{Power} \leq 600 \\ & g_6(\underline{v}) \equiv \pi_{efficiency} - 0.15 \geq 0 \end{array} \quad (1)$$



(a) (b)

Figure 7: *The non-dominated front for the Universal motor.* (a) The non-dominated solutions (pareto-front) in the 3-objective space of mass, efficiency and torque. (b) The manifold space corresponding to the map from the high-dimensional design space $D = 8$ to low-dimensional design space $d = 2$ obtained by LLE. Note that the distribution of colours are non-uniform in the two maps, but they remain segregated (with some noise).

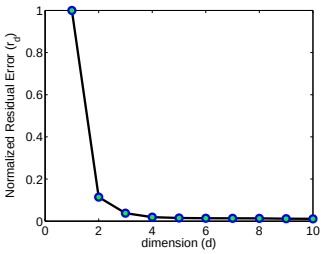


Figure 8: *Dimensionality of manifold for Universal Motors based on* . The FFR data is mapped onto manifolds of different dimensions, and then mapped back to the original design space and the error is estimated. The error drops sharply from 1-D to 2-D manifold, and then less sharply. The knee of the curve at "2" is indicative of the intrinsic dimensionality of the space.

We use the well known NSGA-II [Deb, 2001] evolutionary algorithm, with population size 2000, and probability of crossover 0.8, mutation probability 0.33 and 0.1 (for real/ binary). The estimated pareto front for maximizing both the torque (π_{torque}) and efficiency ($\pi_{efficiency}$) while minimizing the mass (π_{mass}) is shown in Fig. 7(a). The designs in this non-dominated front in objective space are identified in the original 8-parameter design space. We now attempt to see if these 8-D points actually constitute a lower dimensionality manifold, by considering the reconstruction error when mapped to differing dimensionalities from 2 to 8 (figure Fig. 8; the sharp knee at 2 indicates considerable information abstraction, and Fig. 7(b) shows the mapping to a 2-dimensional space obtained by LLE. This mapping reveals that neighbours in the high dimensional space remain nearby in the lower-dimensional space at least for this universal motor problem.

The results here signify that for the universal motor, obtaining the FFR as a 2-dimensional non-dominated surface in objective space can lead to a dimensionality reduction to 2 in the design space as well. These two dimensions possibly reflect inter-relations between the original eight parameters that pertain to the better designs in the design space. In terms of symbol formation, these two dimensions ("XX" and "YY", say), if they are found repeatedly in other domains as well, may eventually become symbols. With sufficient experience, the relation between these two parameters and the design may eventually be encoded into design rules: e.g. "higher YY is usually associated with the more efficient designs". Subsequent experience may also alter the way we understand these chunks, and therefore rules like the above that are built on it; through this demonstration we are primarily arguing that by keeping these symbols grounded, it would be possible to keep updating their semantics *and* their inter-relations (the rules), thus providing a truly flexible symbol system, in contrast to static symbol systems.

We must be careful to point however, that in general a $k-1$-dimensional pareto-surface in objective space may not map to an equivalent manifold in design space - there are a large number of situations where the performance metrics mapping from design space to objective space are not so well-behaved, and such results may not hold. Nonetheless, even if a subset of design parameters are well-behaved, at least some dimensionality reduction may occur in these spaces. To obtain an estimate of the dimension of the manifold for our data set, we use the technique based on the idea that a dimensionality reduction algorithm should preserve information on a global scale, so that the inverse mapping error should be minimal. For a given input dataset $X = \{X_1, \ldots X_N\} \subset R^D$, the dimensional reduction algorithm such as LLE provide a reduced dimensional representation $Y = \{Y_1, \ldots Y_N\} \subset R^d$ of the original data set X. How to determine the reduced-dimensionality d is not clear; one approach may be to consider several d's and select that which minimizes the residual bijection error $(r_d) = \sum_i ||f_d^{-1}(f_d(X_i - X_i)||$,[Martin and Backer, 2005] where$f_d : X \rightarrow Y$ is the map produced by LLE. By observing the behavior of r_d for different values of d shown in Fig. 8 we can suggest the intrinsic dimension for the universal motor is most likely 2; i.e. the initial space of 8 parameters can, given these optimization conditions, be reduced to two incipient "symbols".

4 Conclusion

The main contributions of this work is the proposal that non-linear manifold learning may constitute an important step in discovering latent relationships among the many parameters that define how the world works. A key constraint is our incomplete characterization of the situations in which such a lower-dimensional characterization would exist.

Among the work that would need to be done next is to the conjoints of more than one symbol; i.e. given the design elements each as an individual symbol, we need to be able to say what the conjunction of these elements (the syntax) will do, and whether the resulting object - a design instance - will be adequate to meet the design task or not. Again, depending on the "good designs" that emerge in the process, a combination of symbols may come to be designated as a symbol on its own right, leading to the birth of abstract symbols.

The argument presented here implies that in the long run, to create viable computer vocabularies for design or AI, we must train the systems to learn these relationships, by experiencing many design and real world situations. This may be done in an accelerated manner, but the system must be exposed to something like the vast array of experiences of a human - or possibly many more, since the abstraction processes as computationally available today may not be as efficient. As different systems are deployed in solving different problems, their somewhat differing input sets would result in somewhat different abstractions for the same symbols. These resulting design agents may therefore be somewhat less predictable than current computers, but such is the price of flexibility.

References

[Belkin and Niyogi, 2002] M. Belkin and P. Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. *Advances in Neural Information Processing Systems*, 1:585–592, 2002.

[Bishop, 2006] C.M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[Bohm *et al.*, 2005] M.R. Bohm, R.B. Stone, and S. Szykman. Enhancing Virtual Product Representations for Advanced Design Repository Systems. *Journal of Computing and Information Science in Engineering*, 5:360, 2005.

[Casasola *et al.*, 2003] Marianella Casasola, Leslie B. Cohen, and Elizabeth Chiarello. Six-month-old infants' categorization of containment spatial relations. *Child Development*, 74:679–693, 2003.

[De Saussure, 19161986] F. De Saussure. *Course in general linguistics*. Open Court, 1916/1986.

[Deb, 2001] Kalyanmoy Deb. *Multi-Objective imization using Evolutionary Algorithms*. Chichester, John Wiley and Sons, Ltd., 1 edition, 2001.

[Gero and Fujii, 2000] JS Gero and H. Fujii. A computational framework for concept formation for a situated design agent. *Knowledge-Based Systems*, 13(6):361–368, 2000.

[Gladwell *et al.*, 2005] M. Gladwell, H. Finder, and C-SPAN (Television network. *Blink: The power of thinking without thinking*. Penguin Books, 2005.

[Gobet *et al.*, 2001] F. Gobet, P.C.R. Lane, S. Croker, P.C.H. Cheng, G. Jones, I. Oliver, and J.M. Pine. Chunking mechanisms in human learning. *Trends in Cognitive Sciences*, 5(6):236–243, 2001.

[Goldschmidt, 2003] G. Goldschmidt. The backtalk of self-generated sketches. *Design Issues*, 19(1):72–88, 2003.

[Gross, 1986] Mark Donald Gross. *Design as Exploring Constraints*. PhD thesis, Department of Architecture ,Massachusetts Institute of Technology, February 1986.

[Hirtz *et al.*, 2002] J. Hirtz, R.B. Stone, D.A. McAdams, S. Szykman, and K.L. Wood. A functional basis for engineering design: Reconciling and evolving previous efforts. *Research in Engineering Design*, 13(2):65–82, 2002.

[Lakoff and Johnson, 1999] George Lakoff and Mark Johnson. *Philosophy in the Flesh: The embodied mind and its challenge to Western thought*. Basic Books, New York, 1999.

[Langacker, 1986] Ronald W. Langacker. An introduction to cognitive grammar. *Cognitive Science*, (10):l–40, 1986.

[Mandler, 2004] Jean Matter Mandler. *Foundations of Mind : Origins of conceptual thought*. Oxford University Press, New York, 2004.

[Martin and Backer, 2005] S Martin and A Backer. Estimating manifold dimension by inversion error. In *ACM Symposium on Applied Computing*, pages 22–26, 2005.

[McDonough *et al.*, 2003] L. McDonough, S. Choi, and J.M. Mandler. Understanding spatial relations: Flexible infants, lexical adults. *Cognitive Psychology*, 46(3):229–259, 2003.

[Moss *et al.*, 2004] J. Moss, J. Cagan, and K. Kotovsky. Learning from design experience in an agent-based design system. *Research in Engineering Design*, 15(2):77–92, 2004.

[Nanda *et al.*, 2007] J Nanda, H.J. Thevenot, T.W. Simpson, R.B. Stone, M. Bohm, and S.B. Shooter. Product family design knowledge representation, aggregation, reuse, and analysis. *AI EDAM*, 21(02):173–192, 2007.

[Pahl and Beitz, 19881996] G Pahl and W Beitz. *Engineering Design:A Systematic Approach*, pages 199–400. The Design Council/Springer-Verlag, London/Berlin, 1988/1996.

[Park and Gero, 1999] SH Park and J.S. Gero. Qualitative representation and reasoning about shapes. In *Visual and Spatial Reasoning in Design*, volume 99, pages 55–68, 1999.

[Richard and Dixon, 1994] Welch V Richard and John R Dixon. Guiding conceptual design through behavioural reasoning. *Research in Engineering*, 6:169–188, 1994.

[Roweis and Saul, 2000] S.T. Roweis and L.K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding, 2000.

[Sarkar *et al.*, 2008] Somwrita Sarkar, Andy Dong, and John S Gero. A learning and inference mechanism for design optimization problem (re)-formulation using singular value decomposition. In *Proceedings of DETC'08, ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, August 2008.

[Saul and Roweis, 2003] L K Saul and S T Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *The Journal of Machine Learning Research*, 4:119–155, 2003.

[Simpson, 1998] Timothy W. Simpson. *A Concept Exploration Method for Product Family Design*. PhD thesis, Georgia Tech University, Dept Mechanical Engineering, 1998.

[Szykman *et al.*, 2001] S. Szykman, R.D. Sriram, and W.C. Regli. The role of knowledge in next-generation product development systems. *Journal of computing and information Science in Engineering*, 1:3, 2001.

[Tenenbaum *et al.*, 2000] J.B. Tenenbaum, V. Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction, 2000.

A Neural-Symbolic System for Automated Assessment in Training Simulators
A Position Paper

Leo de Penning, Bart Kappé, Karel van den Bosch
TNO Defense, Security and Safety
Soesterberg, The Netherlands
leo.depenning@tno.nl, bart.kappe@tno.nl, karel.vandenbosch@tno.nl

Abstract

Performance assessment in training simulators is a complex task. It requires monitoring and interpreting the student's behaviour in the simulator using knowledge of the training task, the environment and a lot of experience. Assessment in simulators is therefore generally done by human observers. To capture this process in an automated system is challenging and requires innovative solutions. This paper proposes a new module for automated assessment in simulators that is based on Neural-Symbolic Learning and Reasoning and the Recurrent Temporal Restricted Boltzmann Machine (RTRBM). The module is capable of using existing and learning new rules for performance assessment, by observing experts and students performing the training tasks. These rules are used to validate and support the assessment process and to automatically assess student performance in a training simulator. The module will be developed in a three year research project on assessment in driving simulators for testing and examination.

1 Introduction

Performance assessment in training simulators has always been a complex task that is generally performed by human observers. Performance assessment by automated systems is often limited to simple training tasks, because assessing complex tasks requires the modelling of all interrelations between the information present in the simulation, the training tasks, and the constructs being assessed (e.g. competences). Also, when it comes to more subjective assessments (e.g., how 'safe' is the student driving), conventional modelling techniques fall short, as the applied assessment rules are often implicit and difficult to elicitate from the simulation or domain experts.

We propose a new module for automated assessment as part of the Virtual Instruction platform SimSCORM [Penning *et al.*, 2008]. This assessment module will be able to learn new rules from the task description, (real-time) simulation data, related assessment data of domain experts or students and already existing rules (also called background knowledge). These rules can be presented in a human-readable ('*symbolic*') form, facilitating the validation of the assessment rules and supporting the assessment process.

2 Global Architecture

The automated assessment module requires real-time interaction with the simulator(s), the student and human assessors, and a description of the training task, a student profile and the simulated environment. SimSCORM provides a generic platform for definition and presentation of simulation based training content and interaction between the content, its users and the simulation based on international standards (e.g. SCORM, HLA, XML, etc.). Via this platform the automated assessment module can easily access the objects and attributes in the simulation and get information on the student profile and progress.

Figure 1 depicts the automated assessment module (named CogAgent) in the SimSCORM context. SimSCORM provides a player that presents a SCORM based training task to the students and possibly one or more assessors (e.g. teachers, examiners or students) via a (web-based) Learning Management System. This player uses SimAgent to interact with the simulator(s) and CogAgent to do automated performance assessment and learn new assessment rules from observation. Therefore, the player configures CogAgent with information on the training task, measured variables, student profile, assessed constructs and existing symbolic rules. During execution of the training task, assessors can provide feedback on the assessed constructs which will be presented to CogAgent as short-term evaluations (depicted as assessment data). SimAgent will act as a generic interface between the simulator(s) and CogAgent, and pre-processes received data from the simulator(s) based on measured variable descriptions. Based on the information from the player and SimAgent, CogAgent determines an overall (or long-term) evaluation for the assessed constructs which will be presented to the students (and assessors) as assessment result. Parallel to this it uses the measured data and assessment data to adapt the internal knowledge on assessment rules, resulting in new rules that can be validated afterwards. All information, including the symbolic rules, will be encoded in XML as part of the working memory of the agents and will be distributed via SOAP (either locally or via a web-service).

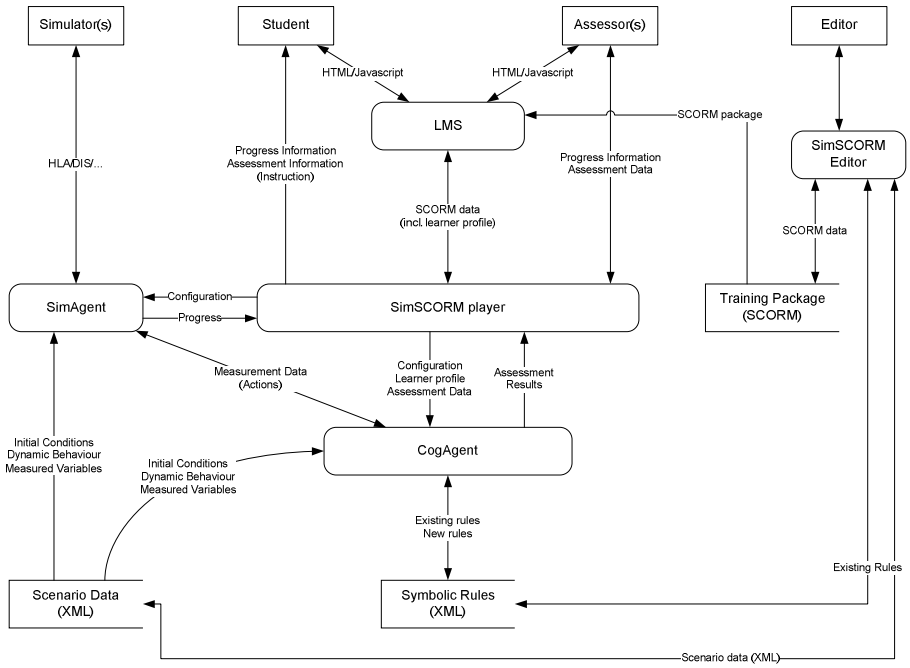


Figure 1. Global architecture of the automated performance assessment module

3 Neural-Symbolic Learning and Reasoning

The CogAgent must be able to learn new rules from observation and existing rules, infer conclusions from these rules and present them in a human readable form. Research on Neural-Symbolic Learning and Reasoning focuses on the integration of learning techniques and architectures from Neural Networks with the symbolic presentation and reasoning techniques in (Fuzzy) Logic Programs (see [Bader and Hitzler, 2005]).

The Neural-Symbolic model proposed for CogAgent is based on the Recurrent Temporal Restricted Boltzmann Machine (RTRBM) [Sutskever *et al.*, 2009] and is depicted in Figure 2. This partially connected symmetric neural network implements an auto-associative memory of its input layers (called visible layers). CogAgent contains three visible layers that represent its beliefs, desires and intentions (introduced by [Bratman, 1999]). Beliefs are variables related to the training task (initial conditions, dynamic behaviour and measured variables) and the student profile. Intentions are variables related to actions or instructions. And desires are variables related to performance assessments (e.g. evaluations or rewards). Beliefs and intentions are directly related to the current state of the context whereas desires will be related to future states as well using Temporal Difference learning [Sutton, 1988]. This technique learns the model to predict a maximum obtainable value for its desires (e.g. overall evaluation scores) based on the current and previous states. Otherwise, the model would only learn to map short-term evaluations, which is not desired in this case.

The hidden layer of the RTRBM is connected to the visible layers with symmetric connections. Each hidden unit represents a rule or relation between one or more visible units. It also contains recurrent hidden-to-hidden connections that enable the RTRBM to learn the temporal dynamics in the visible layers using an algorithm based on contrastive divergence and backpropagation through time. Using this layer we can infer the posterior probability of beliefs, intentions and desires in relation to the state of current and previous beliefs, intentions and desires.

3.1 Symbolic Rules and Fuzzy Atoms

As described in section 2, the rules CogAgent needs to encode, learn and reason about are relations (or causalities) between XML encoded constructs, which will be called atoms hereafter. An XML based atom describes a belief, intention or desire as a function of measured data from the simulator and/or assessment data from the assessors (or students). In case of training simulators this data is often expressed in both continuous and binary values. Therefore we need to use functions in the visible units that can express both. In [Chen and Murray, 2003] sigmoid functions are introduced that contain a 'noise-control' parameter to allow a smooth translation from noise-free deterministic behaviour to binary-stochastic behaviour. These continuous stochastic functions can express both binary and continuous variables. The 'noise-control' parameter controls the steepness of the sigmoid function and can be trained, such that the behaviour of a function dynamically changes according to the distribution of its input values. We will extend our model with such functions to create a Recurrent Temporal Continuous Restricted Boltzmann Machine (RTCRBM).

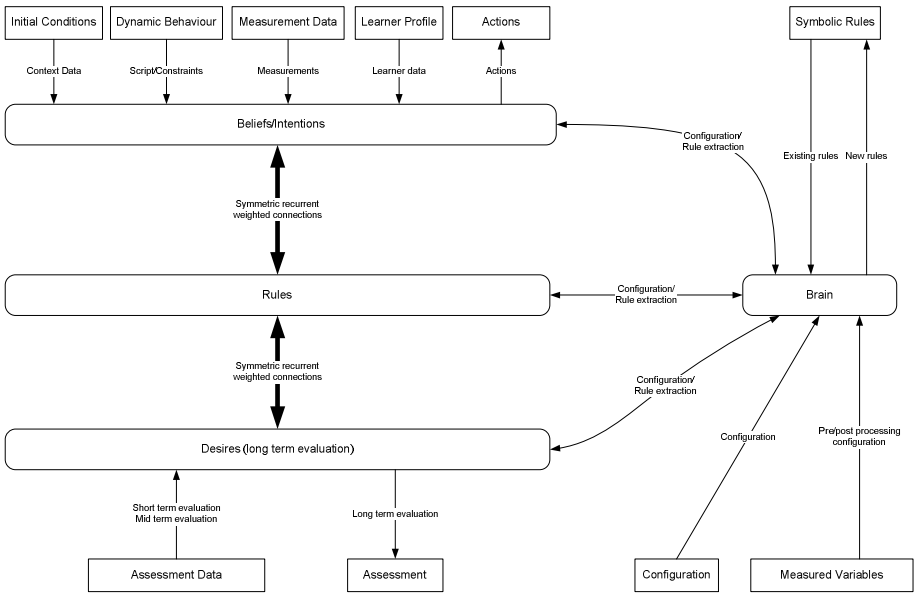


Figure 2. Neural-Symbolic Cognitive Architecture for CogAgent

To express relations between atoms in symbolic rules we propose to use the temporal propositional logic described in Lamb *et al.*, [2007]. This logic contains several modal operators that extend classical modal logic with a notion of past and future. All these operators can be translated to a form that relates only to the immediate previous timestep (denoted by the temporal operator ●). This allows us to encode any rule from this language in the RTCRBM as a combination of visible units (or atoms) and recurrent hidden units that represent applied rules in the previous timestep. For example the proposition $\alpha \mathbb{S} \beta$ denotes that a proposition α has been true *since* the occurrence of proposition β. This can be translated to: $\beta \rightarrow \alpha \mathbb{S} \beta$ and $\alpha \wedge \bullet(\alpha \mathbb{S} \beta) \rightarrow \alpha \mathbb{S} \beta$, where α and β are modelled by visible units and $\bullet(\alpha \mathbb{S} \beta)$ is modelled by a recurrent hidden unit.

We extend this logic with the use of equality and inequality formulas to represent the atoms for continuous variables (e.g. $A{=}x$, $A{<}x$, etc). Note that the atoms for binary variables can also be represented as $A{=}true$ or $A{=}false$, which allows us to handle the outcome of these atoms in the same way as with the continuous atoms. But for readability we will use the classical notion A and $\neg A$.

Due to the stochastic nature of the sigmoid functions used in our model, the atoms can be regarded as fuzzy sets with a Gaussian membership function. This allows as to represent fuzzy concepts, like good and bad or fast and slow or approximations of learned values, which is especially useful when reasoning with implicit and subjective rules. In fact our model can be regarded as a neural-fuzzy system similar to the fuzzy systems described in [Kosko, 1992] and [Sun, 1994].

Now let's take the training task depicted in Figure 3. Using our extended temporal propositional logic, we can describe rules about the conditions, scenario and performance assessment related to this task.

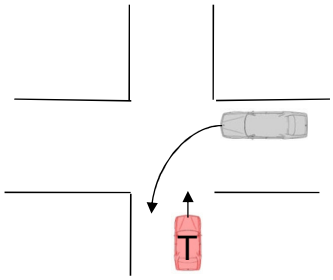


Figure 3. Example training task for driving simulation. The Trainee drives on an urban road, approaching an intersection. The Trainee has to apply the yield-to-the-right-rule.

Example rules for a driver training task:

Conditions:

$(Area = urban)$
$(Weather \geq good)$
$(Time \geq 6) \wedge (Time \leq 18)$

Scenario:

$(Speed > 0) \wedge ApproachingIntersection \rightarrow CrossIntersection$
$ApproachingIntersection \wedge \Diamond(ApproachingTraffic = right)$
$((Speed > 0) \wedge (HeadingIntersection)) \; \mathbb{S} \; (DistanceIntersection < x) \rightarrow ApproachingIntersection$

Assessment:

$ApproachingIntersection \wedge (DistanceIntersection = 0) \wedge (ApproachingTraffic = \text{right}) \wedge \Box(Speed = 0) \rightarrow (Evaluation = good)$
$ApproachingIntersection \wedge (DistanceIntersection = 0) \wedge (ApproachingTraffic = right) \wedge \Diamond(Speed > 0) \rightarrow (Evaluation = bad)$

The rule with temporal operator $\mathbb{S}$, denotes that *ApproachingIntersection* is true when the driver has been driving towards an intersection since a certain distance x to an inter-

section was passed. This rule and the actual value for *x* can be learned from observation by clamping the actual speed, heading and distance to the visible units and the value *true* to the unit for *ApproachingIntersection* when the trainee is approaching the intersection. This can be done by an assessor or the student, but could also be automatically inferred by the model, as explained in the next section.

3.2 Rule encoding and extraction

To encode and extract symbolic rules in symmetric connectionist networks, like the RBM, Pinkas [1995] describes a generic method that directly maps these rules to the energy function of such networks. Therefore he describes an extension to propositional logic, called penalty logic that applies a penalty to each rule. This penalty can be regarded as the "certainty" or "reliability" of a rule and is directly related to the weights of the connections between the units that form this rule. To apply the encoding and extraction algorithms of Pinkas successfully to our model we need extend our temporal propositional logic with the use of penalties. [Sun, 1994] describes a method to map atoms with classical modal operators to real values. We propose to extend this method to create a mapping of atoms and rules with the modal operators used in our model to penalties. Furthermore we need to investigate what changes are required to the algorithms to handle the use of equality formulas and continuous variables. For example, we need to prove that it is possible to infer the correct value for unknown continuous variables in a rule via pattern reconstruction based on known values and (previously) applied rules. And to encode and extract rules with inequality formulas we need to be able to transform these to and from rules that contain only equality formulas.

The penalties that are encoded or learned by our model can be used to rank the rules according to their applicability in a certain context or scenario, giving the students and assessors a nice overview of the applied rules. Also they allow us to solve ambiguities in the application of rules, by using such a ranking to select the most applicable (or reliable) rule in each case.

4 Further Research and Experiments

The model described here is still conceptual and requires further research. To summarize the previous sections, we need to investigate the following topics:

- Is the proposed language for symbolic rules adequate enough to represent the subjective and fuzzy rules applied in performance assessment?
- How to determine the penalties of atoms and rules based on their modalities? And how to map penalties to temporal modalities of rules and atoms?
- How to transform rules with inequality formulas to and from rules with only equality formulas?
- If and how to adapt the rule encoding and extraction methods of Pinkas [1995] to make them applicable to the RTCRBM?
- How to integrate temporal difference learning in the RTCRBM for long term evaluation of desires?

These and many other topics will be investigated in a three year research project on assessment in driving simulators, carried out by TNO in cooperation with the Dutch licensing authority (CBR), Research Center for Examination and Certification (RCEC), Rozendom Technologies and ANWB driving schools. The resulting automated assessment module will be validated in several experiments on a large student population using multiple commercial driving simulators. If successful, the module will be used to support the Dutch driver training and examination program.

References

[Bader and Hitzler, 2005] Sebastian Bader and Pascal Hitzler. Dimensions of neural-symbolic integration - a structured survey. In *We Will Show Them: Essays in Honour of Dov Gabbay, Volume 1. International Federation for Computational Logic*, pages 167-194, College Publications, 2005.

[Bratman, 1999] Michael E. Bratman. Intention, Plans, and Practical Reason. Cambridge University Press, June 1999.

[Chen and Murray, 2003] Hsin Chen and Alan F. Murray. Continuous restricted Boltzmann machine with an implementable training algorithm. In *Vision, Image and Signal Processing, IEE Proceedings*, pages 153-158, 2003.

[Kosko, 1992] Bart Kosko. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall, 1992.

[Lamb *et al.*, 2007] Luís C. Lamb, Rafael V. Borges, Artur S. d'Avila Garcez. A Connectionist Cognitive Model for Temporal Synchronisation and Learning. In *Proceedings of the Conference on Association for the Advancement of Artificial Intelligence (AAAI)*, pages 827-832, 2007.

[Penning *et al.*, 2008] Leo de Penning, Eddy Boot and Bart Kappé. Integrating Training Simulations and e-Learning Systems: The SimSCORM platform. In *Proceedings of the Conference on Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, Orlando, USA, December 2008.

[Pinkas, 1995] Gadi Pinkas. Reasoning, nonmonotonicity and learning in connectionist networks that capture propositional knowledge. In *Artificial Intelligence v.77 n.2*, pages 203-247, September 1995.

[Sutskever *et al.*, 2009] Ilya Sutskever, Geoffrey E. Hinton and Graham W. Taylor. The Recurrent Temporal Restricted Boltzmann Machine. In *Advances in Neural Information Processing Systems 21*, MIT Press, Cambridge, MA, 2009.

[Sun, 1994] Ron Sun. A neural network model of causality. In *IEEE Transactions on Neural Networks, Vol. 5, No. 4*. pages 604-611. July, 1994.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning 3*: pages 9-44, erratum page 377, 1988.