Cloning Composition and Logical Inferences in Neural Networks Using Variable-Free Logic

Helmar Gust and **Kai–Uwe Kühnberger**
Institute of Cognitive Science
Katharinenstr. 24
University of Osnabrück
{hgust, kkuehnbe}@uos.de

Abstract

In this paper, we will exemplify compositionality issues of neural networks using logical theories. The idea is to implement first-order logic on the neural level by using category theoretic methods in order to get a variable-free representation of logic with only one operation (composition). More precisely, logic as well as neural networks are represented as algebraic systems. On the underlying algebraic level it is possible to consider compositionality aspects of first-order logical formulas and their realization by a neural network. We will demonstrate the approach with some well-known logical inferences using a straightforward implementation of a simple backpropagation network.

Introduction

The syntactic structure of formulas of classical first-order logic (FOL) is recursively defined. Therefore it is possible to construct new formulas using old formulas by applying a recursion principle. The same is true for computing the semantic value of such formulas, because the meaning of a complex expression is determined by the meaning of the parts constituting the complex expression (Hodges 1997). This compositionality principle of first-order logic is intuitively plausible, because it makes a recursive model-theoretic definition of the meaning of logical expressions possible and it is in accordance with the usual style mathematicians develop approaches based on algebraic theories and, more generally, inductive structures.

On the other side, compositionality principles in artificial neural networks are usually considered as problematic. Even worse generally it is assumed that neural networks are non-compositional on a principal basis: Complex data structures like lists, trees, tables etc. are – if at all – only implicitly used and the representation of structured and complex objects is a non-trivial challenge. Furthermore logical inferences are rarely considered as lying at the heart of the theory of neural networks (Glöckner 1995).

It is well-known that classical logical connectives like conjunction, disjunction, or negation can be represented by neural networks. Furthermore it is known that every Boolean function can be learned by a neural network (Steinbach & Kohut 2002). Although it is therefore straightforward to represent propositional logic with neural networks (Rojas 1996), this is not true for first-order predicate logic and theories involving first-order expressions. The problems are caused by the usage of quantifiers $\forall$ and $\exists$ and the corresponding variables as well as the compositionality of logical theories. Taking into account that many higher cognitive capacities of humans like problem-solving, reasoning, or planning are often based on forms of first-order logical deductions, it seems to be odd that such capacities are problematic for neural networks to be learned. It is therefore no surprise that there was a certain endeavor to solve the representation problem of neural networks during the 90ies in a variety of different accounts ranging from sign propagation (Lange & Dyer 1989), dynamic localist representations (Barnden 1989), to tensor product representations (Smolensky 1990) and holographic reduced representations (Plate 1994). Although these accounts can be seen as major developments in the problem space of representing logical reasoning with neural networks these accounts have certain non-trivial side-effects. Whereas sign propagation as well as dynamic localist representations lack the ability of learning, the tensor product representation results in an exponentially increasing number of elements to represent variable bindings, only to mention some of the problems.

We will present a different approach of modeling (and learning) first-order inferences using neural networks. Because of the fact that the usage of variables and the properties of quantification are at the heart of the problem, a (partial) solution can be achieved by using a variable-free logic induced by a category called a topos (Goldblatt 1984). In such a category theoretic representation only objects and arrows exist and the task is to construct new arrows given certain other arrows. The evaluation of terms and formulas correspond to the construction of certain arrows between objects of the category. For example, the semantic value of a formula with one variable is an arrow from the universe into the set of truth values $\{true, false\}$. The computation of a semantic value, i.e. the reference of a term or the assignment of a formula with one of the truth values *true* or *false* corresponds to a *construction* of an arrow using given arrows guided by principles of topos theory. Using this representation of logic we want to bridge partially the obvious

differences between symbolic and subsymbolic approaches by representing logical inferences with artificial neural networks. If this can be achieved then it is possible to draw conclusions with respect to the compositionality principle that is usually assumed to hold when we talk about logical meaning.

The paper will have the following structure: First, we will present the basic ideas of variable-free first-order logic using a representation of FOL induced by category-theoretic means in a topos. Second, we will discuss roughly the problems neural networks have in representing structured objects and compositionality. Third, we will examine the overall architecture of the account, the structure of neural networks to code variable-free logic, and the construction of inferences in deduction processes. Fourth, we will discuss a simple example of how predicate logic inferences can be learned by a neural network with a relatively small error. Last but not least, we will add some concluding remarks.

Variable-Free Logic

Classical first-order logic uses variables in order to express general laws. A general law can either express that if for an arbitrary element of the universe a certain property $P_1(x)$ holds, then the property $P_2(x)$ holds as well or that there is a certain element of the universe for which two predicates P_3 and P_4 apply. Hence, those rules have the following form.

$$\forall x : P_1(x) \rightarrow P_2(x)$$
$$\exists x : P_3(x) \wedge P_4(x)$$

In classical logic, variables are not only used to express quantification but also to syntactically mark multiple occurrences of terms. Variable management is usually a problematic issue in logic. In particular the problem arises that algorithms have certain difficulties with quantified variables: The non-decidability of FOL is a direct consequence of this fact.

In order to circumvent this problem a certain endeavor was invested to develop a theory of variable-free logic. A prominent approach is the category theoretic approach using the concept of a topos (Goldblatt 1984). Intuitively a topos is a category that has all limits (products, coproducts, pullbakcs, pushout etc.), that allows exponents, and that has a subobject classifier. The intrinsic properties of a topos induce a semantics on the logical constructions.

Formally, a category $\mathbf{C}$ consists of a set of objects $|\mathbf{C}|$, a set of arrows (or morphisms) $Ar(\mathbf{C})$, and two functions $dom : Ar(\mathbf{C}) \rightarrow |\mathbf{C}|$ (called domain) and $cod : Ar(\mathbf{C}) \rightarrow |\mathbf{C}|$ (called codomain) relating the arrows to the objects. Furthermore a partial composition operation on $Ar(\mathbf{C})$ is defined with the following properties: $g \circ f$ if $cod(f) = dom(g)$, $dom(g \circ f) = dom(f)$, and $cod(g \circ f) = cod(g)$. Finally, the composition of arrows is associative, i.e. $(h \circ g) \circ f = h \circ (g \circ f)$, and for every object c in $|\mathbf{C}|$ there is an identity arrow $id_c \in Ar(\mathbf{C})$ such that $f \circ id_{dom(f)} = f$ and $id_{cod(f)} \circ f = f$

Table 1: The translation of first-order formulas into arrows in a topos

LOGIC	TOPOS
Constant c	$c : ! \rightarrow U$
Function f	$f : U^n \rightarrow U$ or $f : ! \rightarrow U^{U^n}$
Term t	$t : ! \rightarrow U$
Predicate p	$p : U^n \rightarrow \Omega$ or $p : ! \rightarrow \Omega^{U^n}$
1-ary connective	$\Omega \rightarrow \Omega$
2-ary connectives	$\Omega \times \Omega \rightarrow \Omega$
Quantifiers	$\Omega^{X \times Y} \rightarrow \Omega^X$
Closed formula A	$A : ! \rightarrow \Omega$

The type of categories we are interested in is a so-called topos. The properties of a topos $\mathbf{C}$ that are important for our account can be summarized as follows:

- $\mathbf{C}$ has products $a \times b$ and exponents a^b
- $\mathbf{C}$ has pullbacks
- $\mathbf{C}$ contains a terminal object !
- $\mathbf{C}$ has a truth value object Ω together with a subobject classifier $true : ! \rightarrow \Omega$
- For every morphism f there exists an epi-mono factorization

A prototypical category that satisfies the properties of a topos is the category **SET**. The objects of **SET** are sets, the morphisms set theoretic functions. Products $a \times b$ can simply be identified with Cartesian products of sets a and b, exponents a^b with the set of functions $f : b \rightarrow a$. Given two functions $f : a \rightarrow c$ and $g : b \rightarrow c$ an example of a pullback is a triple $\langle D, f', g' \rangle$ where

$$D = \{\langle x, y \rangle \mid x \in a, y \in b, f(x) = g(y)\}$$
$$f'(\langle x, y \rangle) = y$$
$$g'(\langle x, y \rangle) = x$$

In category theory, products and pullbacks are so-called limit constructions. They allow the construction of new arrows. For example, in the case of a Cartesian product $a \times b$ the following condition holds: if arrows $f : c \rightarrow a$ and $g : c \rightarrow b$ are given, then there exists a unique arrow $h : c \rightarrow a \times b$ such that the corresponding diagram commutes. We will crucially use the possibility of constructing new arrows – provided some other arrows are given – in the account presented in this paper. The terminal object ! in **SET** is the one-element set $\{0\}$, because for every arbitrary set a there is exactly one function from a into $\{0\}$. The subobject classifier can be constructed by taking $! = \{0\}$ and $\Omega = \{0, 1\}$ and *true* mapping 0 to 1. Factorization is clear in **SET**.

There is a straightforward way of interpreting first-order logical formulas in a topos (Goldblatt 1984). Table 1 summarizes the crucial connections. First of all, notice that variables are not explicitly represented in a topos. Constants and terms are represented as arrows from the final object ! to the universe U. In other words, one could say that this arrow

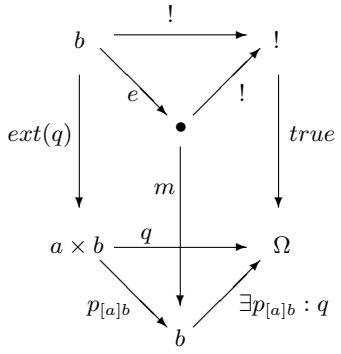

Figure 1: The construction of the formula $\lambda y.(\exists x : q(y, x))$ as an arrow $\exists p_{[a]b} : q$. The expression $p_{[a]b}$ is the projection from $a \times b$ into b, Ω denotes the truth value object.

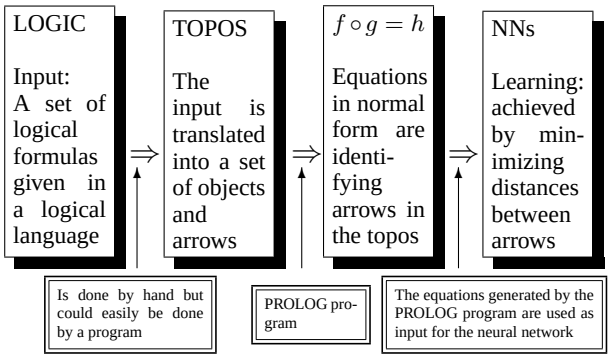

Figure 2: The general architecture of the account.

determines the reference of the term. Similarly functions f of arity n are arrows from the n-ary product of the universe to the universe. Predicates of arity n are represented by arrows from the n-ary product of the universe U into the truth value object. As usual, logical connectives are arrows from the truth value object Ω to the truth value object Ω in the case of unary connectives (e.g. negation) and arrows from the product of the truth value object with itself into the truth value object (e.g. conjunction). Interesting cases concern free and quantified variables: Formulas containing free variables are considered as complex predicates (i.e. elements of Ω^X where X is the range of the sequence of free variables), whereas quantified formulas correspond to an operation mapping predicates to predicates. Quantification over a sequence of variables with a corresponding range Y reduces the complex predicate $\Omega^{X \times Y}$ (where X is the range of the remaining free variables) to the complex predicate Ω^X represented in the topos by an arrow $\Omega^{X \times Y} \rightarrow \Omega^X$. Closed formulas are formulas without any free variables (hence X is the empty set $\emptyset$).

The idea how to interpret truth values of formulas is to construct new arrows between objects provided some arrows are given as input. Clearly this is done in a compositional way by using existing arrows: the unique construction principle in a topos is the composition of arrows. We sketch the idea how an existential quantifier can be constructed in **SET** (Figure 1). The construction of the arrow $\exists p_{[a]b} : q$ corresponding to the logic expression $\lambda y.(\exists x : q(y, x))$ works as follows. First, we need to justify the construction of the arrow $ext(q)$. This can be achieved by the fact that **SET** has pullbacks. To justify the arrows e and m we notice that $exp(q) \circ p_{[a]b}$ allows an epi-mono factorization represented by the arrows e and m. Finally the arrow $\exists p_{[a]b} : q$ can be constructed because a subobject classifier exists.

The Problem of Composing Formulas in Neural Networks

Rule-based inferences like in logical calculi are difficult to realize in artificial neural networks (Shastri & Ajjanagadde 1990). The first problem arises because of the fact that logical inferences are performed on structured objects, namely on variables, terms, atomic formulas, and complex formulas. All these different entities can only be represented in a neural network by a vector $x \in \mathbb{R}^n$ where $n \in \mathbb{N}$. Therefore a structural distinction between these entities cannot be represented explicitly. The same problem arises with respect to the semantic value of a certain logical expression. Whereas the symbolic computation of a semantic value of a formula is a highly structured process, no such an analogy exists on the side of neural networks.

A possible way out of this gap is to use algebraic characterizations of both, logic and neural networks. This account provides a possibility to find a common language for both accounts. An important approach was proposed in (Pollack 1990) later further generalized in (Sperduti 1993). Pollack uses implicitly algebraic means to represent structured objects like trees and stacks over a finite universe in neural networks. Another more explicit direction was taken by (Glöckner 1995) where the author shows which classes of algebras can be represented by multilayer perceptrons. This approach reconstructs the compositional structure of logic by the composition of networks.

In contrast to this idea, we will not use multilayered perceptrons to approximate functions, rather we will use a neural network to represent the behavior of functions and predicates by approximating the composition process of these entities. More precisely, not the structural properties of these entities will be represented (simply because all these entities are viewed as atomic in our account), rather the behavior in the composition process is modeled. In order to achieve this, we want to use the topos representation of logical terms and formulas in order to implement a backpropagation algorithm on a neural network that is able to perform (and learn) elementary first-order inferences.

Learning Evaluations of Formulas

The idea of modeling logical evaluations using neural networks is based on the idea that compositionality in logic can be reduced to compositionality principles in a topos, i.e. the construction of arrows resulting in a commuting diagram. The existence of such commuting diagrams can be interpreted as equations between two terms on the term level and equivalences of complex expressions on the formula level.

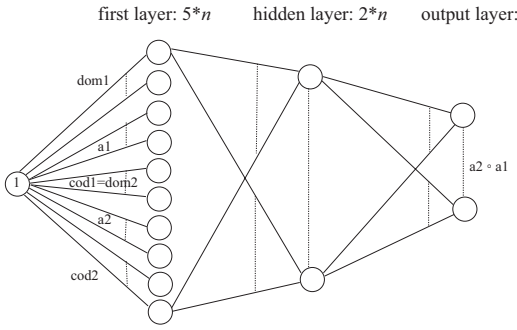

Figure 3: The structure of the neural network that learns composition of first-order formulas.

The Overall Architecture

Figure 2 depicts the general architecture of the presented account in four steps: First, input data is given by a set of logical formulas (determining a partial theory) relative to a given logical language $\mathcal{L}$. Second, this set of formulas is translated into objects and arrows of a topos based on the fundamental theorem that first-order predicate logic can be represented by a topos.[1] Third, a PROLOG program is generating equations in normal form $f \circ g = h$ identifying new arrows in the topos. It order to make this work, we developed a simple topos language $\mathcal{L}_T$ to code the definitions of objects and arrows in a way such that they can be processed by the program components. The idea is that a given arrow f can be used to generate new equations like $id \circ f = f$, $f \circ id = f$ and so on. Last but not least, these equations are used to train a neural network. The structure of the used neural network will be described below.

The motivation of the proposed solution is based on the idea that we need to transform an interpretation function I of classical logic into a function $I' : \mathbb{R}^n \to \mathbb{R}^m$ in order to make it appropriate as input for a neural network. The first step to achieve this can loosely be associated with reification and elimination of variables, both standard techniques commonly used in AI: Formulas of first-order predicate logic are interpreted as objects and arrows in a topos. The second step is motivated by the challenge to represent logical formulas as equations and finally to represent formulas as equations in a real-valued vector space. In the last step, a necessary issue is to hard-wire certain principles like the one that *true* and *false* is maximally distinct.

Figure 3 depicts the structure of a neural network that is used in order to model the composition process of evaluating terms and formulas. Each object of the topos is represented as a point in an n-dimensional real-valued unit cube. In the example used in this paper, $n = 5$. Each arrow in the topos is again represented as a point in the n-dimensional real-valued unit cube together with pointers to the respective domain and codomain. The input of the network is represented by weights from the initial node with activation 1. This allows the backpropagation of the errors into the representation of the inputs of the network. The input of the network represents the two arrows to be composed by the following parts:

- The domain of the first arrow
- The representation of the first arrow
- The codomain of the first arrow which must be equal to the domain of the second arrow
- The representation of the second arrow
- The codomain of the second arrow

These requirements lead to a net with $5 \cdot n$ many input values (first layer in Figure 3). The output of the neural net is the representation of the composed arrow. Up to now we have no good experience how many hidden nodes we need. In the example, we use $2 \cdot n$ many nodes.

In order to enable the system to learn logical inferences, some basic arrows have static (fixed) representations. These representations correspond directly to the role of these objects in the underlying topos. Here is a list of these objects:

- The truth value *true* : $(1.0, 1.0, 1.0, 1.0, 1.0)$
- The truth value *false* : $(0.0, 0.0, 0.0, 0.0, 0.0)$

Notice that the truth value *true* and the truth value *false* are considered to be maximally distinct. All other objects and arrows are initialized with random values. The defining equations of the theory and the equations corresponding to the used categorical constructions (like products) are used to train the neural network.

Inference Example

We want to give a simple example of how a neural network can learn logical inferences on the basis of equations and static objects that are listed above.

Table 2: Example code of the objects and arrows

```
!.                      # the terminal object
@.                      # the truthvalue object
! x ! = !.
u.                      # the universe
static  t: ! --> @,     # true
static  f: ! --> @.     # false
not: @     --> @,       # negation
->:  @ x @ --> @.       # implication
not o t = f,
not o f = t,
-> o t x t = t,
-> o t x f = f,
-> o f x t = t,
-> o f x f = t.
```

In Table 2, elementary logical operators and equations are defined specifying the behavior of classical connectives like the composition of the implication or the product of *true* $\times$ *true* results in *true*. Furthermore static arrows are used to define *true* and *false* and static objects are used to define the terminal object ! and the truth value object Ω. The coding of topos entities in $\mathcal{L}_T$ is straightforward: in the first part we define objects and arrows and in the second part we provide the defining equations. Table 3 summarizes the important constructs. Furthermore $\mathcal{L}_T$ provides a simple macro

[1] Compare (Goldblatt 1984). Although this could quite easily be done automatically by a program, we still do the translation by hand.

Table 3: The specification of $\mathcal{L}_T$

$\mathcal{L}_T$	Intended Interpretation
`!`	Terminal object !
`@`	Truth value objects Ω
`t`	Truth value $true$
`f`	Truth value $false$
`Y x Z`	Product object of Y and Z
`y x z`	Product arrow of y and z
`! X`	Terminal arrow of X
`x: Y --> Z`	Definition of an arrow
`y o z`	Composition of arrows x and y

mechanism to allow a compact coding for complex equations (compare the definition of `==>` in Table 4).

In Table 4, an extension of the premises of the classical Socrates syllogism is modeled. The information coded is not only that all humans are mortal (and Socrates is human) but also that all mortals ascend to heaven and everyone in heaven is an angle. As constants not only *Socrates* but also *robot* and *something* are introduced with the additional information that *robot* is not human. There is no knowledge about *something*. These types of information are represented by equations. For example, the composition of the arrow *human* and *socrates* is resulting in *true* representing that "*Socrates* is human". Slightly more difficult is the representation of universally quantified expressions like the following predicate logic formula:

$$\forall x : human(x) \rightarrow mortal(x)$$

The equation is constructed as follows:[2]

$$\forall(\Rightarrow \circ human \times mortal \circ d) = true$$

This is equivalent with

$$\Rightarrow \circ human \times mortal \circ d = true \circ !$$

The diagonal arrow $d : U \rightarrow U \times U$ is composed with the arrows for predicates $human : \Omega \rightarrow \Omega$ and $mortal : \Omega \rightarrow \Omega$ and the arrow for the implication $\Rightarrow: \Omega \times \Omega \rightarrow \Omega$. Notice further that the construction is only possible because a topos guarantees the existence of the relevant arrows.

Finally, test equations are represented in Table 4. They correspond to the logically possible combinations of $mortal$, $human$, and $angle$ on the one hand and $Socrates$, $robot$, and $something$ on the other. All combinations can be either *true* or *false*.

Results of the Inference Example

The input generated by the Prolog program is feeded into the neural network. The result of an example run is then given by the errors of the test equations. These test equations query whether the composition of *angle* and *robot* is *false*, whether the composition of *angle* and *robot* is *true*,

[2]Notice that the following are expressions in the topos not logical expressions.

Table 4: Example code of an extension of the famous "Socrates inference"

```
# predicates of the theory
  human, mortal, heaven, angel:  u ---> @ .
  X ==> Y: -> o X x Y o d u = t o ! u .
  human  ==> mortal.
  mortal ==> heaven.
  heaven ==> angel.
#individuals
distinctive
  socrates, robot, something: ! ---> u.
  human o socrates = t.
  human o robot    = f.
# test the learned inferences
tests
  mortal o something = t,
  mortal o something = f,
  mortal o robot     = t,
  mortal o robot     = f,
  mortal o socrates  = t,
  mortal o socrates  = f,
  heaven o something = t,
  heaven o something = f,
  heaven o socrates  = t,
  heaven o socrates  = f,
  heaven o robot     = t,
  heaven o robot     = f,
  angel  o something = t,
  angel  o something = f,
  angel  o socrates  = t,
  angel  o socrates  = f,
  angel  o robot     = t,
  angel  o robot     = f.
```

whether the composition of *angle* and *socrates* is false etc. The results of test run of our example is depicted below:

```
Tests:
angel o robot       = f          0.636045
angel o robot       = t          0.886353
angel o socrates    = f          2.197811
angel o socrates    = t          0.011343
angel o something   = f          0.053576
angel o something   = t          2.017303
heaven o robot      = f          0.454501
heaven o robot      = t          1.080864
heaven o socrates   = f          2.153396
heaven o socrates   = t          0.013502
heaven o something  = f          0.034890
heaven o something  = t          2.111497
mortal o socrates   = f          1.985289
mortal o socrates   = t          0.030935
mortal o robot      = f          0.195687
mortal o robot      = t          1.488895
mortal o something  = f          0.025812
mortal o something  = t          2.159551
```

The system convincingly learned that *Socrates* is mortal, ascended to *heaven*, and is an *angle*. Furthermore it learned that the negations of these consequences are false. In other words, the system learned the transitivity of the implication in universally quantified formulas. With respect to *robot* the system evaluates with a relatively high certainty that *robot* is not mortal. The other two properties are undecided. In the case of *something* relatively certain knowledge for the system is that *something* is neither in heaven nor mortal nor an angle.

Error Rate of the Inference Example

The maximal error of the neural network of 10 runs with $1.6 \cdot 10^6$ iterations is depicted in Figure 4. The curves show

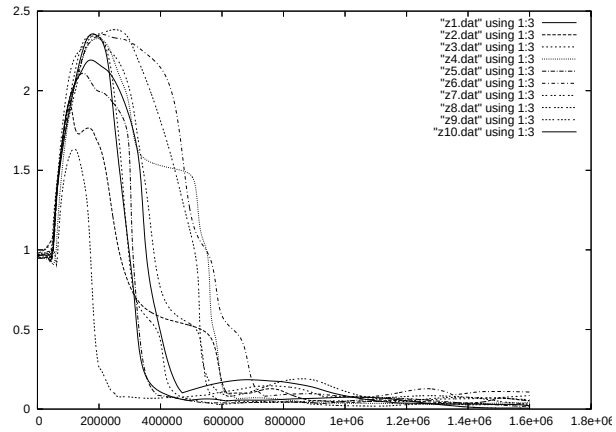

Figure 4: The maximal error of the neural network with respect to the extended Socrates example (10 runs with $1.6 \cdot 10^6$ iterations).

four characteristic phases: in the first phase (up to 50,000 iterations), the randomly chosen representations for the input arrows and objects remains relatively stable. During the second phase (between 50,000 and approx. 200,000 iterations) the maximal error dramatically increases due to the rearrangement of the input representations. In the third phase (between 200,000 and 600,000 iterations) the maximal error rapidly decreases which is again connected with the reorganization of the input representation. In most cases, the maximal error decreases in the fourth phase (between 600,000 and above iterations), but the input representations stay relatively stable.

The models approximated by the network behave as expected: Test equations which are logically derivable *true* or *false* will be mapped in all models to *t* or *f* respectively. Those equations for which no logical deduction of the truth value is possible, are more or less arbitrarily distributed between f and t in the set of models. Nevertheless, the models seem to tend to realize a closed world interpretation, i.e. the truth value of $mortal(robot)$, $heaven(robot)$, and $angle(robot)$ tend to be *false*.

Conclusions

In this paper, we presented first results of modeling predicate logical inferences by neural networks. The idea was to to use a variable-free representation of first-order logic in a topos and to reduce logical inferences to constructions of arrows via composition in a topos. This can be implemented in a neural network and the example shows that quantified inferences can be learned by the system with a surprisingly small error. The complexity of determining the truth value of a query depends only on the complexity of the query and not on the number of inference steps needed for a logical deduction. The features of the system can be summarizes as follows:

- The network is able to learn
- The static pre-established connections of the network are minimal
- Variables of logical formulas are not (explicitly) represented, hence there is no binding problem

All features above give a solution to the problem of learning first-order inferences by neural networks without the trouble generated by accounts mentioned in the introduction.

Future work concerns, first, a careful evaluation of the resulting models. Until now, we do not precisely know which types of models are generated by the system. Second, a natural idea would be to use the system for non-monotonic reasoning. Third, the topology of the representation space needs to be examined. Connected with this issue is the question whether concepts can be extracted from configurations of the network.

References

Barnden, J.A. 1989. Neural net implementation of complex symbol processing in a mental model approach to syllogistic reasoning. In Proceedings of the International Joint Conference on Artificial Intelligence, 568-573.

Fodor, J. & Pylyshyn, Z. 1988. Connectionism and cognitive architecture: A critical analysis, *Cognition* 28: 3-71.

Goldblatt, R. 1984. *Topoi, the categorial analysis of logic. Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam.

Glöckner, I. 1995. Verwendung neuronaler Netze zur Emulation logischen Schließens. Master thesis, Institute of Cognitive Science, University of Osnabrück.

Gust, H. 2000. Quantificational Operators and their Interpretation as Higher Order Operators. M. Böttner & W: Thümmel, *Variable-free Semantics*, 132-161, Osnabrück.

Hodges, W. 1997. *A Shorter Model Theory*. Cambridge University Press.

Lange, T. & Dyer, M.G. 1989. High-level inferencing in a connectionist network. Technical report UCLA-AI-89-12.

Plate, T. 1994. Distributed Representations and Nested Compositional Structure. PhD thesis, University of Toronto.

Pollack, J. 1990. Recursive distributed representation. *Artificial Intelligence* 46:77-105.

Rojas, R. 1996. *Neural Networks – A Systematic Introduction*. Springer, Berlin, New York.

Shastri, L. & Ajjanagadde, V. 1990. From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences* 16: 417-494.

Smolenski, P. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence* 46(1–2): 159–216.

Sperduti, A. 1993. Labelling RAAM. Technical Report TR 93–029, International Computer Science Institute (ICSI), University of Berkeley, Berkeley, CA.

Steinbach, B. & Kohut, R. 2002. Neural Networks – A Model of Boolean Functions. In Steinbach, B. (ed.): *Boolean Problems, Proceedings of the 5th International Workshop on Boolean Problems*, 223-240, Freiberg.