Using FCA for Encoding Closure Operators into Neural Networks*

Sebastian Rudolph

University of Karlsruhe
Institute AIFB
rudolph@aifb.uni-karlsruhe.de

Abstract. After decades of concurrent development of symbolic and connectionist methods, recent years have shown intensifying efforts of integrating those two paradigms. This paper contributes to the development of methods for transferring present symbolic knowledge into connectionist representations. Motivated by basic ideas from formal concept analysis, we propose two ways of directly encoding closure operators on finite sets in a 3-layered feed forward neural network.

1 Introduction

On the scientific quest for enabling machines to fulfill more and more sophisticated cognitive tasks, two basic antagonistic paradigms evolved.

On one side, one tries to capture (in a top-down manner, essentially by introspection and psychological experiments) the basic entities of human thought and their interplay in terms of symbols[1], symbol manipulation systems and formal logic. Also approaches involving conceptual graphs mainly fall into that class.

On the other side, advances in biology and medicine have provided bottom-up insights into the human way of information processing via networks of neurons. So, a contrary approach – started by [2] – tries to employ these findings by simulating neural structures (although this is mostly done in an extremely simplified way).

The interest in the integration of symbolic methods based on computational logic with artificial neural networks (also known as connectionist systems) has grown significantly in the last years. As a motivating goal of those efforts appears to combine the advantages of both approaches: While symbolic systems are superior in dealing (i.e., representing and reasoning) with structured data, connectionist systems show impressive capabilities when it comes to learning on larger datasets and generalizing the results to new input. See [3] for an overview of this prospering research area.

* This is an extended version of a paper presented at NeSy07 – the Third Workshop on Neural-Symbolic-Integration at IJCAI 2007. Sebastian Rudolph is supported by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project and by the European Union under the NeOn project (IST-2005-027595).

[1] As opposed to the rich semiotic meaning of the term *symbol*, we use this word in a more syntactic sense following [1].

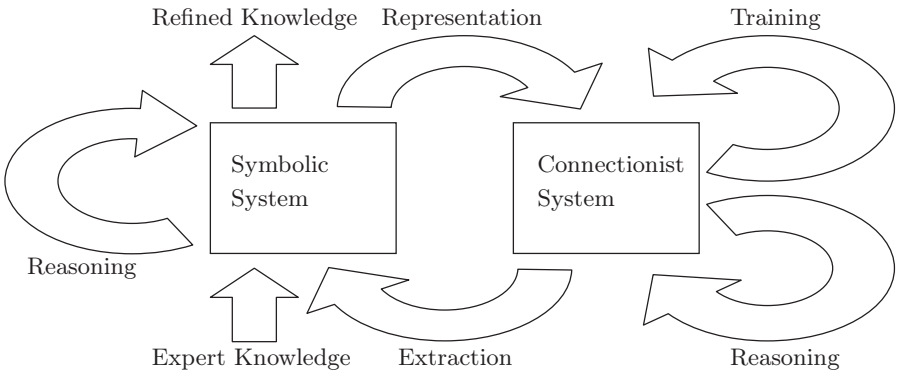

Fig. 1. The neural-symbolic learning cycle.

The neural-symbolic learning cycle depicted in Fig. 1 (see also [1]) proposes a general framework for organizing a neural-symbolic integration. In our paper, we focus on the representation subtask, i.e. encoding explicitly prespecified background knowledge. In particular, we investigate ways of canonically encoding *closure operators* into neural networks. Closure operators on attribute or feature sets arise naturally in various domains; whenever the validity of some features enforces the validity of others (as in human associative thinking and logic entailment to name just two extremes of a wide spectrum), this can be described by closure operators. So assume a neural network for some purpose has to be designed, where some rule-like partial information about the network's desired behavior is already known and can be stated in form of implications on the feature set. We now look for a neural network obeying those prescribed rules (which can then be trained on an example set to acquire further behavior). Previous approaches [4, 5] tackle this problem by assigning a node of the network to each implication. We propose a contrary approach, where – roughly speaking – network nodes don't take the role of enforcing wanted features but preventing unwanted ones. This approach is motivated by the mathematical area of formal concept analysis.

In Section 2.1 we will introduce the basic notions *closure operator* and *implication* and show their correspondence. Section 2.2 will sketch the elementary ideas of *formal concept analysis*, based on which we will unfold our representation approach. Very briefly, Section 2.3 will recall the notion of a *3-layered feedforward network*. Section 3 then combines the approaches and provides two

ways of encoding a formal context's closure operator into a neural network of the specified kind. In Section 4, we show how the approach can be applied to propositional logic programs, where it can be used to compute models. Finally, in Section 5, we conclude and give topics for ongoing research.

2 Preliminaries

2.1 Closure Operators and Implications

In this section, we will introduce two notions – closure operator and implications – and show their tight correspondence.

The following considerations are based on an arbitrary set M. Intuitively, it may be conceived as a set of *features* or *attributes* or *atomic propositions*, depending on the modelled problem. Many of the definitions and theoretical results presented in this and the next section apply to arbitrary sets, however, when it comes to questions of practical realization and computability, finiteness of M has to be presumed.

We will first define the fundamental notion of a closure operator. Roughly speaking, applying such an operator to a set can be understood as a minimal extension of that set in order to fulfill certain properties.

Definition 1. *Let M be an arbitrary set. A function $\varphi : \mathcal{P}(M) \to \mathcal{P}(M)$ (where $\mathcal{P}(M)$ denotes the powerset of M) will be called*

- EXTENSIVE, *if $A \subseteq \varphi(A)$ for all $A \subseteq M$,*
- MONOTONE, *if $A \subseteq B$ implies $\varphi(A) \subseteq \varphi(B)$ for all $A, B \subseteq M$, and*
- IDEMPOTENT, *if $\varphi(\varphi(A)) = \varphi(A)$ for all $A \subseteq M$.*

If φ is extensive, monotone, and idempotent, we will call it a CLOSURE OPERATOR. *In this case, we will additionally call*

- *$\varphi(A)$ the* CLOSURE *of A,*
- *A* CLOSED, *if $A = \varphi(A)$.*

The family of all closed sets is also called CLOSURE SYSTEM. Furthermore, any closure system constitutes a *lattice* with set inclusion as the respective order relation.

Mark that the notion of a closure operator is ubiquitous in both human associative thinking and classical (at least monotonic) logics.[2]

In the sequel, we show in which way closure operators are closely related to implications.

Definition 2. *Let M be an arbitrary set. An* IMPLICATION *on M is a pair (A, B) with $A, B \subseteq M$. To support intuition, we write $A \twoheadrightarrow B$ instead of (A, B).*[3]

[2] For example, in classical first order logic, taking the set of all consequences $cons(\Phi) := \{\varphi \mid \Phi \models \varphi\}$ of a formula set Φ is a closure operator.

[3] To facilitate reading we will occasionally omit the parentheses, i.e., we will write $a, b \twoheadrightarrow c$ instead of $\{a, b\} \twoheadrightarrow \{c\}$.

For $C \subseteq M$ and a set $\mathfrak{I}$ of implications on M, let $C^{\mathfrak{I}}$ denote the smallest set with $C \subseteq C^{\mathfrak{I}}$ that additionally fulfills

$$A \subseteq C^{\mathfrak{I}} \quad \textit{implies} \quad B \subseteq C^{\mathfrak{I}}$$

for every implication $A \twoheadrightarrow B$ in $\mathfrak{I}$.[4] *If $C = C^{\mathfrak{I}}$, we call C* $\mathfrak{I}$-CLOSED.

It is well known that for a given set $A \subseteq M$ and implication set $\mathfrak{I}$, $A^{\mathfrak{I}}$ can be computed in linear time with respect to $|\mathfrak{I}|$ (see [6] or [7]). As can be easily seen, for any set $\mathfrak{I}$ of implications on any set M, $(.)^{\mathfrak{I}}$ is a closure operator. Moreover, for any closure operator $\varphi : \mathcal{P}(M) \to \mathcal{P}(M)$, there exists (at least) a set $\mathfrak{I}$ of implications on M such that $(.)^{\mathfrak{I}} = \varphi$.[5]

An elementary observation from logic becomes particularly obvious in this setting: a contradiction implies everything. Thus, if, say, two elements $a, b \in M$ are contradictory, this can be expressed by the implication $\{a, b\} \twoheadrightarrow M$. In the sequel, we will use the shorthand $a, b \twoheadrightarrow \bot$ for these special cases.

2.2 Formal Concept Analysis

The mathematical theory of formal concept analysis mainly deals with conceptual hierarchies which are generated from basic data structures encoding object-attribute relationships. Thereby, it provides a rather applied access to lattice theory For a comprehensive introduction into formal concept analysis, see [8].

In this section, we sketch the basic definitions and some results from formal concept analysis, as far as they are needed for this work. We start by defining the central underlying data structure.

Definition 3. *A* (FORMAL) CONTEXT $\mathbb{K}$ *is a triple (G, M, I) with*

- *an arbitrary set G called* OBJECTS,
- *an arbitrary set M called* ATTRIBUTES,
- *a relation $I \subseteq G \times M$ called* INCIDENCE RELATION

We read gIm as: "object g has attribute m".

Intuitively, a formal context is represented by a so-called cross table, where each row is associated to an object, each column to an attribute, and crosses indicate which object has which attributes.

Definition 4. *Let $\mathbb{K} = (G, M, I)$ be a formal context. We define a function $(.)^I : \mathcal{P}(G) \to \mathcal{P}(M)$ with*

$$A^I := \{m \mid gIm \ \textit{ for all } g \in A\}$$

[4] Note, that this is well-defined, since the mentioned properties are closed wrt. intersection.

[5] A naïve way to achieve this: given φ, let $\mathfrak{I} = \{A \twoheadrightarrow \varphi(A) \mid A \subseteq M\}$.

for $A \subseteq G$. Furthermore, we use the same notation to define the function $(.)^I : \mathcal{P}(M) \to \mathcal{P}(G)$ where

$$B^I := \{g \mid gIm \quad \text{for all } m \in B\}$$

for $B \subseteq M$.

For convenience, we sometimes write g^I instead of $\{g\}^I$ and m^I instead of $\{m\}^I$.

Applied to an object set, this function yields all attributes common to these objects; by applying it to an attribute set we get the set of all objects having those attributes. The following facts are consequences of the above definitions:

Proposition 1.

- *$(.)^{II}$ is a closure operator on G as well as on M.*
- *For $A \subseteq G$, A^I is a $(.)^{II}$-closed set and dually*
- *for $B \subseteq M$, B^I is a $(.)^{II}$-closed set.*

The next definition shows how a conceptual hierarchy can be built from a formal context.

Definition 5. *Given a formal context $\mathbb{K} = (G, M, I)$, a* FORMAL CONCEPT *is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A = B^I$, and $B = A^I$.*

We call the set A EXTENT *and the set B* INTENT *of the concept (A, B). Let (A_1, B_1) and (A_2, B_2) be formal concepts of a formal context. We call (A_1, B_1) a* SUBCONCEPT *of (A_2, B_2) (written: $(A_1, B_1) \leq (A_2, B_2)$) if $A_1 \subseteq A_2$. Then, (A_2, B_2) will be called* SUPERCONCEPT *of (A_1, B_1).*

Proposition 2. *The concept intents of a formal concept are exactly those attribute sets closed with respect to $(.)^{II}$.*

It is well known from FCA that the set of all formal concepts of a formal context together with the subconcept-superconcept-order form a complete lattice, the so called *concept lattice*.

2.3 On Neural Networks

In this section, we recall the notion of a particular neural network giving a formal definition that we will build upon in the subsequent sections.

Definition 6. *A* 3-LAYERED FEEDFORWARD NETWORK *is defined as a tuple $N = (\mathcal{I}, \mathcal{H}, \mathcal{O}, t, w)$ where*

- *$\mathcal{I}, \mathcal{H}, \mathcal{O}$ are finite disjoint sets called* INPUT NODES, HIDDEN NODES, *and* OUTPUT NODES,
- *$t : (\mathcal{I} \cup \mathcal{H} \cup \mathcal{O}) \to \mathbb{R}$ is the* THRESHOLD FUNCTION, *and*
- *$w : (\mathcal{I} \times \mathcal{H}) \cup (\mathcal{H} \times \mathcal{O}) \to \mathbb{R}$ is the* WEIGHT FUNCTION.

Clearly, neural networks are intended as computational models, i.e. they are designed to calculate something. Hence given a neural network we can define a function capturing its computational behaviour.

Definition 7. *Given a 3-layered feedforward network N as specified in Definition 6, the* ASSOCIATED NETWORK FUNCTION $f_N : \mathcal{P}(\mathcal{I}) \to \mathcal{P}(\mathcal{O})$ *is defined in the following way: For a given argument set S, we define the set $\mathcal{A}_S \subseteq \mathcal{I} \cup \mathcal{H} \cup \mathcal{O}$ of* ACTIVATED NEURONS *as follows (using the shortcut $\chi_A(a) = |\{a\} \cap A|$):*

- *for every $i \in \mathcal{I}$, we set $i \in \mathcal{A}_S$ exactly if $\chi_S(i) - t(i) > 0$,*
- *for every $h \in \mathcal{H}$, we set $h \in \mathcal{A}_S$ exactly if $\sum_{i \in \mathcal{I}} \chi_{\mathcal{A}}(i) w_{ih} - t(h) > 0$, and*
- *for every $o \in \mathcal{O}$, we set $h \in \mathcal{A}_S$ exactly if $\sum_{h \in \mathcal{H}} \chi_{\mathcal{A}}(i) w_{ho} - t(o) > 0$.*

Finally, we set $f_N(S) = \mathcal{A}_S \cap \mathcal{O}$.

This definition exactly mirrors the usual way of calculating with neural networks, presuming the Heaviside step function as activation function.

In the sequel, we aim at the special case of simulating a closure operator $\varphi : \mathcal{P}(M) \to \mathcal{P}(M)$ with this kind of neural network, i.e., input and output layer correspond to the same set (namely M).

3 Encoding of Closure Operators inspired by FCA

The basic idea for this paper is to use formal contexts to represent closure operators. In particular (as we have seen in Section 2.2), for a formal context $\mathbb{K} = (G, M, I)$, the function $(.)^{II} : \mathcal{P}(M) \to \mathcal{P}(M)$ is a closure operator on M. Moreover, *any* closure operator on a finite set M can be represented by an appropriate formal context.[6]

So, in this section, we propose two canonical ways to translate a formal context into a 3-layered feedforward network, which – given a set $A \subseteq M$ – computes its closure A^{II}.

The intuition hereby is to identify the hidden layer neurons with the object set of the formal context. We realize the $(.)^{II}$-operator by first applying $(.)^I$ to A (which by definition yields an object set represented by the activated neurons in the hidden layer) and, afterwards, applying $(.)^I$ to A^I thus obtaining the closure of the attribute set at the output layer.

Definition 8. *For a given formal context $\mathbb{K} = (G, M, I)$, we define a corresponding 3-layered feedforward network $N_{\mathbb{K}}$ in the following way:*

- $\mathcal{I} = \{i_m \mid m \in M\}$
- $\mathcal{O} = \{o_m \mid m \in M\}$
- $\mathcal{H} = \{h_g \mid g \in G\}$
- $t(i) := 0.5$ *for all* $i \in \mathcal{I}$

[6] One method, how to construct a formal context with this property will be explicated in Section 4.

– $w_{i_m h_g} = w_{h_g o_m} = \begin{cases} 0 & \text{if } gIm \\ -1 & \text{otherwise.} \end{cases}$
– $t(n) := -0.5$ *for all* $n \in \mathcal{H} \cup \mathcal{O}$

Next, we will prove that indeed the associated network function $f_{N_\mathbb{K}}$ corresponds to the closure operator $(.)^{II}$, i.e., for all $A \subseteq M$, we have that $A^{II} = \{m \mid o_m \in f_{N_\mathbb{K}}(\{i_{\tilde{m}} \mid \tilde{m} \in A\})\}$

Proposition 3. *Let $\mathbb{K} = (G, M, I)$ be a formal context and $N_\mathbb{K}$ the corresponding neural network. Then*

1. *for every $A \subseteq M$, activating (exactly) the set $\{i_m \mid m \in A\}$ of input neurons leads to an activation of (exactly) the set $\{h_g \mid g \in A^I\}$ of hidden neurons and*
2. *for every $B \subseteq G$, activating (exactly) the set $\{h_g \mid g \in B\}$ of hidden neurons leads to an activation of (exactly) the set $\{o_m \mid m \in B^I\}$ of output neurons.*

Proof. Consider the hidden layer neuron h_g representing the object $g \in G$. Now, since $A^I = \{g \mid gIm \text{ for all } m \in A\}$ we have that $g \in A^I$ exactly if g has all attributes from A. Obviously, this is the case if and only if

$$\sum_{m \in M} \chi_{\mathcal{A}_A}(i_m) w_{i_m h_g} = \sum_{m \in A} w_{i_m h_g} = 0 > -0.5.$$

The second claim is proved in exactly the same manner.

The next corollary then follows immediately be the definition of $(.)^{II}$ as twofold application of $(.)^I$.

Corollary 1. *$N_\mathbb{K}$ computes $(.)^{II}$.*

This approach is quite close to formal concept analysis since the neurons of the hidden layer directly correspond to the object set of the represented formal context. The negative weights are necessary due to the fact that $(.)^I$ is (in both variants) an *antitone* function (i.e. $A \subseteq B$ implies $B^I \subseteq A^I$).

However, this can be overcome by a simple "work around": instead of mirroring the functions $A \mapsto A^I$ and $B \mapsto B^I$ (for $A \subseteq M$ and $B \subset G$), one could use the functions $A \mapsto M \setminus A^I$ and $B \mapsto (M \setminus B)^I$ instead. Both of them are monotone and can hence be modelled with only positive weights, and still their composition yields the wanted operator $(.)^{II}$. In the sequel, we will elaborate this idea.

Definition 9. *For a given formal context $\mathbb{K} = (G, M, I)$, we define a corresponding 3-layered feedforward network $\tilde{N}_\mathbb{K}$ in the following way:*

– $\mathcal{I} = \{i_m \mid m \in M\}$
– $\mathcal{O} = \{o_m \mid m \in M\}$
– $\mathcal{H} = \{h_g \mid g \in G\}$
– $t(i) := 0.5$ *for all* $i \in \mathcal{I}$
– $t(o_m) := -0.5 + |\{g \in G \mid \neg gIm\}|$ *for all* $o_m \in \mathcal{O}$

– $w_{i_m h_g} = w_{h_g o_m} = \begin{cases} 0 \text{ if } gIm \\ 1 \text{ otherwise.} \end{cases}$
– $t(h) := 0.5$ *for all* $h \in \mathcal{H}$

Proposition 4. *Let* $\mathbb{K} = (G, M, I)$ *be a formal context and* $\tilde{N}_{\mathbb{K}}$ *the corresponding neural network. Then*

1. *for every* $A \subseteq M$*, activating (exactly) the set* $\{i_m \mid m \in A\}$ *of input neurons leads to an activation of (exactly) the set* $\{h_g \mid g \in G \setminus A^I\}$ *of hidden neurons and*
2. *for every* $B \subseteq G$*, activating (exactly) the set* $\{h_g \mid g \in B\}$ *of hidden neurons leads to an activation of (exactly) the set* $\{o_m \mid m \in (G \setminus B)^I\}$ *of output neurons.*

Proof. 1. Consider the hidden layer neuron h_g representing the object $g \in G$. Now, since $A^I = \{g \mid gIm \text{ forall } m \in A\}$, we have that $g \in A^I$ exactly if g has all attributes from A. Obviously, this is the case if and only if

$$\sum_{m \in A} w_{mg} = 0 < 0.5.$$

Therefore, any h_g being activated must be in $G \setminus A^I$.

2. Now, consider the output layer neuron o_m representing the attribute $m \in M$. If B is activated in the hidden layer, o_m will be activated exactly if

$$\sum_{g \in B} w_{gm} = |\{g \in B \mid \neg gIm\}| > -0.5 + |\{g \in G \mid \neg gIm\}|$$

Yet, due to $B \subseteq G$, this can only be the case iff $\{g \in B \mid \neg gIm\}| = |\{g \in G \mid \neg gIm\}|$ which is equivalent to the statement that gIm for all $g \in G \setminus B$. Hence, o_m is activated exactly if $m \in (G \setminus B)^I$.

Corollary 2. $\tilde{N}_{\mathbb{K}}$ *computes* $(.)^{II}$.

Proof. Due to the preceding proposition, applying $\tilde{N}_{\mathbb{K}}$ to an attribute set A will first activate the hidden neurons representing $G \setminus A^I$ and then the output neurons representing $(G \setminus (G \setminus A^I))^I = (A^I)^I = A^{II}$

As already mentioned, using this type of network will activate exactly those hidden layer neurons *not* contained in A^I, if A is entered.

An interesting feature of both presented networks is their symmetry: for all $m \in M$ and $g \in G$, $w_{i_m h_g} = w_{h_g o_m}$. Although this puts structural constraints on the neural network and might therefore hamper the application of learning strategies, it might be useful from a quite different point of view: in cases, where the neural network will be hardwired, input and output layer could be identified and calculation be done in a "back-and-forth manner" using the links twice for every calculation.

4 Application to propositional logic programs

In this section, we will show, how the presented strategy can be applied in a propositional logic programming scenario.

Logic programming is especially suited for this approach, since

- any logic program essentially consists of a set of implications and hence
- entailment can (at least in the negation-free case) therefore be described by a closure operator on the ground facts.

Consequently, one can assign to every logic program an operator T_P which applied to a set of ground facts intuitively calculates the immediate consequences by "applying" each implication once. The entailment closure operator can then be simulated by iteratively applying T_P until a fixed point is reached. [5] presents an approach to encode T_P into a recurrent 3-layered neural network, by assigning every implication to a node of the middle layer. To make this clear, consider the following example.

Imagine, some kind of animal has to be determined via some tests. Let furthermore the only available tests be to indicate whether the animal is a mammal, a bird, a monkey, a donkey, an owl, a fowl or a frog. Hence $M := \{\text{donkey}, \text{monkey}, \text{mammal}, \text{frog}, \text{bird}, \text{owl}, \text{fowl}\}$. Then the implications presented in Fig. 2 characterize the setting:

$$
\begin{aligned}
\text{monkey} &\multimap \text{mammal}\\
\text{donkey} &\multimap \text{mammal}\\
\text{owl} &\multimap \text{bird}\\
\text{fowl} &\multimap \text{bird}\\
\text{monkey, donkey} &\multimap \bot\\
\text{owl, fowl} &\multimap \bot\\
\text{mammal, bird} &\multimap \bot\\
\text{mammal, frog} &\multimap \bot\\
\text{bird, frog} &\multimap \bot
\end{aligned}
$$

Fig. 2. Implication representing the knowledge in our example.

Following [4], the neural network corresponding to the T_P-operator representing those implications interpreted as a logic program would look like the one represented in Fig, 3.

The set $\{donkey, fowl\}$ demonstrates that, in general, T_P may have to be applied several times to calculate the closure, since

$$T_P(\{\text{donkey}, \text{fowl}\}) = \{\text{donkey}, \text{mammal}, \text{fowl}, \text{bird}\}$$

and

$$T_P(\{\text{donkey}, \text{mammal}, \text{fowl}, \text{bird}\}) = M.$$

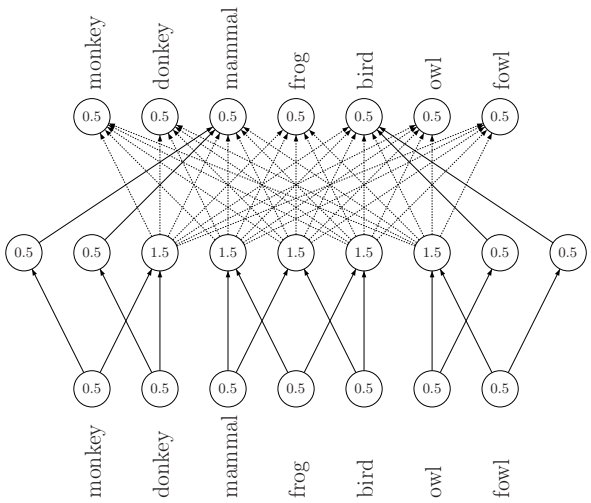

Fig. 3. Neural network corresponding to the T_P-operator of our propositional setting. All weights are set to 1. The dotted lines are those indicating a contradiction.

Now we consider how our method would apply. So, we have to find a formal context $\mathbb{K} = (G, M, I)$, where $A^{II} = A^{\mathfrak{I}}$ for all $A \subseteq M$. One possibility to do so is to consider the lattice of all $\mathfrak{I}$-closed sets. Fig. 4 represents this.

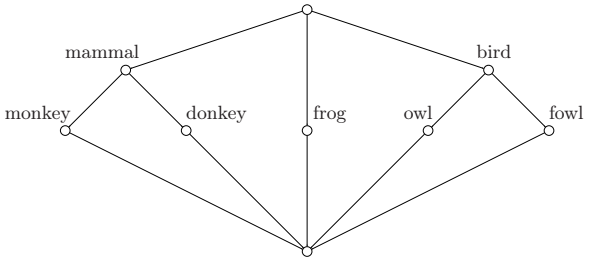

Fig. 4. Lattice of the $\mathfrak{I}$-closed sets.

Yet, a well-known result of FCA provides a direct way to find a minimal set of objects for a formal context that is supposed to generate a given lattice. One has to take all supremum-irreducible elements as objects. Looking at the diagram, the supremum-irreducible elements are exactly those having only one lower neighbour. In our particular case, these are exactly all upper neighbours of the bottom element. Hence, we can derive the formal context depicted in Fig. 5.

	monkey	donkey	mammal	frog	bird	owl	fowl
g_1					×		×
g_2					×	×	
g_3				×			
g_4		×	×				
g_5	×		×				

Fig. 5. The formal context $\mathbb{K}$ corresponding to the closure operator to describe.

According to the preceding section, there are two ways of using this kind of formal context to define a neural network that computes the closure of a given set directly (i.e., no manyfold application – likewise no recurrent organisation – of the net would be necessary).

The first one (corresponding to the definition of $N_{\mathbb{K}}$) is shown in Fig. 6. Note that all drawn edges correspond to weights of -1.

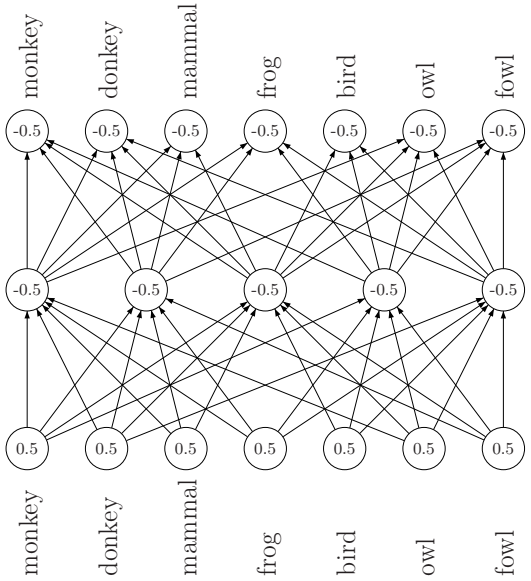

Fig. 6. Neural network corresponding to the consequence operator of our propositional setting. All weights are set to -1.

The second network (corresponding to the definition of $\tilde{N}_{\mathbb{K}}$) is shown in Fig. 7. Here all drawn edges carry weight of 1.

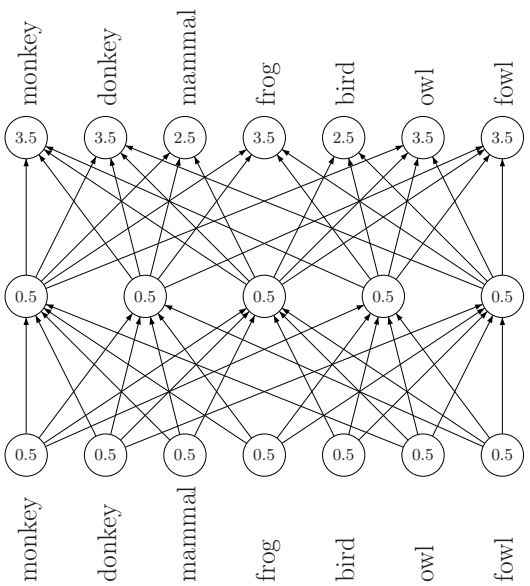

Fig. 7. Neural network $\tilde{N}_{\mathbb{K}}$ corresponding to the consequence operator of our propositional setting. All weights are set to 1.

5 Conclusion and Future Work

In our paper, we presented two new canonical ways for generating neural networks that compute the closure operator of a given finite set. We thereby provide a method to support the representation part of the neural-symbolic learning cycle by presenting an encoding strategy for a kind of background knowledge generically occurring in the area of knowledge processing.

In contrast to other methods, where the closure is approximated iteratively (using a recurrent network), the networks presented in our approach will calculate it directly, i.e., by a single run of the network.

Moreover, as shown by our example, there are cases where this kind of representation is also advantageous in terms of the number of hidden layer neurons needed. In general, this approach seems to be especially beneficial, if the number of implications becomes large.

Naturally, the proposed method requires preprocessing of the implicative information to be encoded. Depending on how this information is given, it has to be transformed into a formal context. The way we presented here – namely generating the whole lattice of the closed sets and identifying the supremum-irreducible elements of it – is certainly not optimal with respect to time costs (in the worst case, the size of the lattice can be $2^{|M|}$). So one important field of future research is to find more efficient methods to convert implicative knowledge into small contexts. On the more theoretical side, also complexity bounds for this kind of task would be of interest.

More generally, we are convinced, that connectionist approaches – if taken into the focus of (up to now more symbolically oriented) fields like conceptual structures – could contribute to a deeper and more comprehensive understanding of either field. This could even pave the way to an integrated neural-symbolic theory of conceptual thinking.

References

1. Bader, S., Hitzler, P.: Dimensions of neural-symbolic integration - a structured survey. In Artemov, S., Barringer, H., d'Avila Garcez, A.S., Lamb, L.C., Woods, J., eds.: We Will Show Them: Essays in Honour of Dov Gabbay. Volume 1. International Federation for Computational Logic, College Publications (2005) 167–194
2. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics **5** (1943) 115–133
3. d'Avila Garcez, A., Broda, K., Gabbay, D.: Neural-Symbolic Learning Systems: Foundations and Applications. Perspectives in Neural Computing. Springer-Verlag (2002)
4. Hölldobler, S., Kalinke, Y.: Towards a massively parallel computational model for logic programming. In: Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing, ECCAI (1994) 68–77
5. Hitzler, P., Hölldobler, S., Seda, A.K.: Logic programs and connectionist networks. J. Applied Logic **2** (2004) 245–272
6. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. J. Log. Program. **1** (1984) 267–284
7. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
8. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997) Translator-C. Franzke.