

SHERLOCK - An Interface for Neuro-Symbolic Networks*

Ekaterina Komendantskaya and Qiming Zhang

School of Computing, University of Dundee, Dundee, Scotland

Abstract

We propose SHERLOCK - a novel problem-solving application based on neuro-symbolic networks. The application takes a knowledge base and rules in the form of a logic program, and compiles it into a connectionist neural network that performs computations. The network's output signal is then translated back into logical form. SHERLOCK allows to compile logic programs either to classical neuro-symbolic networks (the "core method"), or to inductive neural networks (CILP) — the latter can be trained using back-propagation methods.

1 Introduction

We take the ideas of neuro-symbolic integration to the level of software engineering and design. That is, we do not consider theoretical aspects of neuro-symbolic integration here, but take its synthetic principle to be our main software engineering principle. So, which methods could software engineering borrow from the area of neuro-symbolic integration? Here, we offer one possible answer, but see also [Cloete and Zurada, 2000].

Declarative programming languages, and especially logic programming, have one important underlying idea — they are designed to be syntactically similar to the way people reason. Logic programming, for example, is one of the easiest languages to teach students with non-technical background or general public alike. Also, it is feasible to parse natural language into logic programming syntax. Therefore, the strength of logic programming from the software engineering point of view is that it makes for a general and easily accessible interface for users with diverse backgrounds.

Neural networks, on the other hand, offer both massive parallelism and ability to adapt. However, it would seem almost impossible to imagine that a person with non-technical background easily masters neural networks as part of his working routine, alongside with a web-browser or a text editor. It is common that industrial applications of neural networks are designed and maintained by specialists, while non-specialist users do not have ways to edit the applications. This is why

*The work was supported by EPSRC, UK; Postdoctoral Fellow research grant EP/F044046/2.

neural network applications are often problem-specific. Such applications could be made more general and user-friendly if the users were given a nice easy interface to manipulate neural networks at a level of natural language.

For example, consider a police officer who has just come to a crime scene and wishes to record all evidence available. To be efficient, the police officer uses a small portable computer that has a problem-solving assistant. What should this assistant be like? Neural network software would come in handy, because it can be trained as new evidence is obtained; also — it can be fast due to parallelism. On top of this neural software, though, it is best to have an easy interface allowing the officer to enter data in the form of a natural language.

We propose SHERLOCK — an application that allows the user to type in the knowledge base in the language close to the natural language, and then rely on the compiler that transforms the problem into a suitable neural network. The network will attempt to solve the problem; and once the solution is found — it outputs the answer in a logical form. Thus, SHERLOCK successfully implements the full *neuro-symbolic cycle*, [Hammer and Hitzler, 2007; d'Avila Garcez *et al.*, 2008].

Additionally, as we show in the poster and Section 3, SHERLOCK can be embedded into a bigger knowledge-refining cycle. In this case, we rely upon the backpropagation learning that CILP (cf. [d'Avila Garcez *et al.*, 2002]) offers.

SHERLOCK software relates to the work of [Gruau *et al.*, 1995] proposing a neural compiler for PASCAL; and the programming languages AEL, NETDEF [Siegelmann, 1994] designed to be compiled by neural networks. SHERLOCK differs from the previous similar work in two respects. It is the first fully automated neural compiler for *declarative* languages we know of. Also, in the cited works the main emphasis was on building a fully functional compiler for a programming language; here our emphasis is not on creating a neural compiler for PROLOG *per se*; but building a compiler sufficient to handle knowledge bases and reason over them.

2 Design of SHERLOCK

SHERLOCK provides an editor which allows to write and edit information in logical form; it then transforms the information into connectionist neural network; finally, it translates the outcome of the neural-symbolic system back to the logic programming syntax.

```

C:\Users\Katya\Desktop\Visual Logic\Samples\Robbery.txt
File Edit Format Deduction Help
[Icons]
Domains
Suspect = {Harry,Jane,Stephen}.
Predicates
HaveKeys=( Suspect ).
HaveSmallFeet=( Suspect ).
Smoke=( Suspect ).
Criminal=( Suspect ).
Goals
?Criminal(X).
Clauses
HaveKeys(Harry).
HaveKeys(Jane).
HaveKeys(Stephen).
HaveSmallFeet(Stephen).
HaveSmallFeet(Jane).
Smoke(Stephen).
Criminal(X) :- HaveKeys(X);HaveSmallFeet(X);Smoke(X).

```

Figure 1: SHERLOCK's interface.

SHERLOCK consist of the following components:

1. A code editor, in which the users can write a general logic program in a prolog-like declarative language;
2. A translator, which can analyse syntax and semantics of the logic program to set up neural-symbolic systems according to the logic program;
3. A model of the “core method” neural networks [Hammer and Hitzler, 2007], and a model of CILP-neural networks [d’Avila Garcez *et al.*, 2002];
4. An interpreter;
5. An output reader.

The Figure 1 shows SHERLOCK's interface together with a data base written in syntax similar to logic programming. The answer would be all the names that satisfy the rule for “Criminal”.

3 Knowledge Refining using SHERLOCK

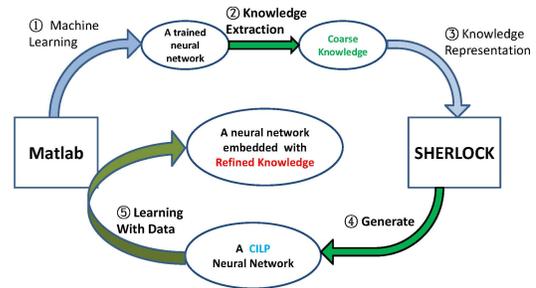
Knowledge refining is one of the important features in human reasoning. We wish to insert background (or “coarse”) knowledge into a neural network and obtain refined knowledge by learning with example data. CILP is suitable to do knowledge refining: it has the capability to present background knowledge into neural networks, and it can use back-propagation to get networks trained with examples.

We propose a novel approach to build knowledge refining systems based on SHERLOCK:

1. Coarse knowledge is obtained from the trained neural network using one of the standard extraction techniques.
2. Then it is expressed in the first order language in SHERLOCK.
3. A CILP neural network is obtained.
4. CILP is trained with the data, and the embedded knowledge is refined.

We test this model on the famous cancer data set from the UCI Machine Learning Repository. The final neural network

A Knowledge Refining System



has a performance of 96.7%. The performance of the final neural network cannot be improved by setting a better training goal while a general neural network can. This implies the knowledge embedded in the CILP neural network is sensitive to certain kinds of data.

We summarise the properties of this model as follows:

1. It provides a methodology to obtain knowledge in any domain by using both induction and deduction.
2. If the knowledge obtained in Step 1 is reasonable, the final neural network will remain a clear structure, which could be interpreted to symbolic knowledge. Otherwise, the neural network is just an ordinary supervised trained neural network.
3. The final neural network has a very good performance in terms of learning. Besides, it seems that the neural network owns an ability to detect some faulty data due to the knowledge embedded in it.

Sherlock software and its user manual can be downloaded from <http://www.computing.dundee.ac.uk/staff/katya/sherlock/>

References

[Cloete and Zurada, 2000] I. Cloete and J. M. Zurada. *Knowledge-Based Neurocomputing*. MIT Press, 2000.

[d’Avila Garcez *et al.*, 2002] Arthur d’Avila Garcez, K. B. Broda, and D. M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer-Verlag, 2002.

[d’Avila Garcez *et al.*, 2008] Arthur d’Avila Garcez, L. C. Lamb, and D. M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Cognitive Technologies. Springer-Verlag, 2008.

[Gruau *et al.*, 1995] Frédéric Gruau, Jean-Yves Ratajszczak, and Gilles Wiber. A neural compiler. *Theor. Comput. Sci.*, 141(1&2):1–52, 1995.

[Hammer and Hitzler, 2007] B. Hammer and P. Hitzler. *Perspectives of Neural-Symbolic Integration*. Studies in Computational Intelligence. Springer Verlag, 2007.

[Siegelmann, 1994] H. Siegelmann. Neural programming language. *Conf. of AAAI*, 1994.