Extracting Reduced Logic Programs from Artificial Neural Networks

Jens Lehmann, Sebastian Bader

INTERNATIONAL CENTER FOR COMPUTATIONAL LOGIC

TECHNISCHE UNIVERSITÄT DRESDEN

Pascal Hitzler

Universität Karlsruhe (TH)

AIFB

Motivation

- Trained networks *solve* problems, but we cannot access their domain knowledge.
- Analysis and verification of trained neural networks impossible.

Example: TD-Gammon (Tesauro 1995)
Backgammon program based on standard neural network architecture.
Learns by playing against itself.
Reaches almost professional player's strength by itself.

We play Backgammon *symbolically*. The learning of some known strategic insights could be observed when the program was trained.

Can we learn from the network how to play better?

Propositional knowledge extraction

Basic idea: Extract propositional logic program following [Hölldobler & Kalinke 1994].

Two main approaches:

1. Structural: Analysis of the network weights in order to find pre-defined *patterns* in substructures.
   Combination of the findings.
   E.g. [Garcez, Broda & Gabbay, AI 2001]
2. Global: Identify truth values with numbers.
   Identify input+output nodes with propositional atoms.
   Interpret input-output behaviour as logical inference between conjunctions.

We obtaind some foundational results on the global approach.

Propositional knowledge extraction

Setting:

- No restriction on network architecture.
- Networks map input activation to output activations.
- We assign truth values to activations, e.g. 0=false, 1=true, or to activation ranges, e.g. [0,.1]=false.
- We assign propositional atoms to all input and output nodes. Particular cases (e.g. recurrent architectures) may force us to identify some input and output nodes by assigning the same atom.

Each input resp. output activation is a propositional *interpretation*.
Network is mapping from input to output, i.e. on interpretations.
$\rightsquigarrow$ Encoding of network I/O as T_P-operator

(P ... propositional logic program).

Extraction by full exploration

For definite programs, i.e. no negation as failure

1. Let f be a mapping from I_P to I_P.
2. Initialize $P = \emptyset$.
3. For every interpretation $I = \{r_1, \ldots, r_a\} \in I_P$ and each element $h \in f(I)$ add a clause $h \leftarrow r_1, \ldots, r_a$ to P.
4. Return P as result.

Extraction by full exploration

Example

Let $I_P = \{p, q\}$ and f be obtained by full exploration of some network.

We use normal full exploration and obtain P. $T_P = f$ holds.

$$
\begin{aligned}
f = \{ \quad \emptyset &\mapsto \{p\} \\
\{p\} &\mapsto \{q\} \\
\{q\} &\mapsto \emptyset \\
\{p, q\} &\mapsto \{p, q\}\}
\end{aligned}
\qquad\qquad
\begin{aligned}
P = \{p &\leftarrow \neg p, \neg q; \\
p &\leftarrow p, q; \\
q &\leftarrow p, \neg q; \\
q &\leftarrow p, q \quad \}
\end{aligned}
$$

- In general, algorithm produces highly redundant programs.

Propositional knowledge extraction: improved

A clause $C_1 : h \leftarrow p_1, \ldots, p_a, \neg q_1, \ldots, \neg q_b$ is said to *subsume* $C_2 : h \leftarrow r_1, \ldots, r_c, \neg s_1, \ldots, \neg s_d$, iff we have $\{p_1, \ldots, p_a\} \subseteq \{r_1, \ldots, r_c\}$ and $\{q_1, \ldots, q_b\} \subseteq \{s_1, \ldots, s_d\}$.

A program P is called *reduced* if the following properties hold:

1. There are no clauses C_1 and C_2 with $C_1 \neq C_2$ in P, such that C_1 subsumes C_2.
2. A predicate symbol does not appear more than once in any clause body.
3. No clause body contains a predicate and its negation.

Theorem:

If P is a definite reduced program,
then P is the smallest program among all definite Q with $T_Q = T_P$.

Propositional knowledge extraction: improved

Let $f : I_P \to I_P$ be a given mapping, as obtained e.g. from a neural network, and consider I_P to be totally ordered in some way such that I is before K in the ordering if $|I| < |K|$. Let Q be an initially empty program.

For all interpretations $I \in I_P$, in sequence of the assumed ordering, do the following:

- Let $I = \{p_1, \ldots, p_n\}$. For every $q \in f(I)$, check whether a clause $q \leftarrow q_1, \ldots, q_m$ with $\{q_1, \ldots, q_m\} \subseteq I$ is already in Q. If not, then add the clause $q \leftarrow p_1, \ldots, p_n$ to Q.

Return Q as the result.

Result: The unique reduced program Q with $T_Q \equiv f$.

Extraction by full exploration

For normal programs (including negation as failure))

1. Let f be a mapping from I_P to I_P.
2. Initialize $P = \emptyset$.
3. For every interpretation $I = \{r_1, \dots, r_a\} \in I_P$, we have $B_P \setminus I = \{s_1, \dots, s_b\}$.
4. For each element $h \in f(I)$ add a clause $h \leftarrow r_1, \dots, r_a, \neg s_1, \dots, \neg s_b$ to P.
5. Return P as result.

Propositional knowledge extraction: normal programs

- Definite case: There is always a least program to be extracted.
- Normal case: In general, no least such program exists.

The following programs are equivalent:

$$P_1 = \{p \leftarrow \neg p, \neg r; \quad p \leftarrow p, r; \quad p \leftarrow \neg p, q \quad \}$$

$$P_2 = \{p \leftarrow \neg p, \neg r; \quad p \leftarrow p, r; \quad p \leftarrow q, r \quad \}$$

A full search easily reveals that the given two programs are minimal.

Naive approach: Find minimal programs by complete search $\leadsto$ impractical.
Alternative: Use heuristics for minimizing programs.

$\leadsto$ Heuristic algorithm in paper. [details skipped]

First-order knowledge extraction

Very little work has been done on this.

A general idea:

- Use any initialization method as a base.
- Neural network is point in $\mathbb{R}^n$, where n is number of weights.
- Use any representation methods of logic programs as neural networks.
- Span search space, e.g. conditions on programs which may be extracted.
  Examples: maximum number of body atoms or of term nesting depth.
  $\rightsquigarrow$ discrete points in $\mathbb{R}^n$ via initialization method.
- Program which lies closest to network in $\mathbb{R}^n$ is the extracted program.

Issues:

- How to implement this efficiently?
- What are good parameters?
- Which metric(s) should be used on $\mathbb{R}^n$?

Summary

Extraction of logic programs from artificial neural networks.

- Definite propositional: nice but unexplored in practice
- Normal propositional: messy
- First-order: hardly investigated yet

Thank You!

Jens Lehmann, Steffen Hölldobler — INTERNATIONAL CENTER FOR COMPUTATIONAL LOGIC, TECHNISCHE UNIVERSITÄT DRESDEN

Pascal Hitzler — AIFB Universität Karlsruhe (TH)

www.neural-symbolic.org

Normal knowledge extraction: an algorithm

Let T_P be an immediate consequence operator, and h be a predicate. We call $B = p_1, \ldots, p_a, \neg q_1, \ldots, \neg q_b$ *allowed with respect to h and T_P* if the following two properties hold:

- For every interpretation $I \subseteq B_P$ with $I \models B$ we have $h \in T_P(I)$.
- There is no allowed body $B' = r_1, \ldots, r_c, \neg t_1, \ldots, \neg t_d$ for h and T_P with $B' \neq B$ such that $\{r_1, \ldots, r_c\} \subseteq \{p_1, \ldots, p_a\}$ and $\{t_1, \ldots, t_d\} \subseteq \{q_1, \ldots, q_b\}$.

There are two reason for a clause body B not to be allowed:

- The resulting clause could be wrong.
- There could be a smaller allowed body B', s.t. $h \leftarrow B'$ subsumes $h \leftarrow B$.

Normal knowledge extraction: an algorithm

- The algorithm will incrementally add clauses to an initially empty program.
- Clause to add is chosen from allowed clauses w.r.t. some *score*-function, which is a heuristics for the importance of a clause.
- score computes the number of interpretations for which the program does not yet behave correctly, but for which it would after adding the clause.

The score of a clause $C : h \leftarrow \mathtt{body}$ w.r.t. a program P is defined as

$$\text{score}(C, P) := \Big|\{I \mid I \subseteq B_P \text{ and } h \notin T_P(I) \text{ and } I \models \mathtt{body}\}\Big|.$$

Normal knowledge extraction: an algorithm

T_P^q is the *restricted* consequence operator for $q \in B_P$.
Set $T_P^q(I) = \{q\}$ if $q \in T_P(I)$, and $T_P^q(I) = \emptyset$ otherwise.

Let T_P and $B_P = \{q_1, \ldots, q_m\}$ be the input of the algorithm.
Initialize $Q = \emptyset$. For each predicate $q_i \in B_P$:

1. construct the set S_i of allowed clause bodies for q_i
2. initialize: $Q_i = \emptyset$
3. repeat until $T_{Q_i} = T_P^{q_i}$:
   (a) Determine a clause C of the form $h \leftarrow B$ with $B \in S_i$ with the highest score with respect to Q_i.
   (b) If several clauses have the highest score, then choose one with the smallest number of literals.
   (c) $Q_i = Q_i \cup \{C\}$
4. $Q = Q \cup Q_i$

Return Q as the result.

Normal knowledge extraction: algorithm example

$$
\begin{array}{llll}
T_P = \{\emptyset \mapsto \{p\} & \{q,r\} \mapsto \emptyset & \{p\} \mapsto \emptyset & \{q,s\} \mapsto \{p\} \\
\{q\} \mapsto \{p\} & \{r,s\} \mapsto \emptyset & \{r\} \mapsto \emptyset & \{p,q,r\} \mapsto \{p\} \\
\{r\} \mapsto \emptyset & \{p,q,s\} \mapsto \emptyset & \{p,q\} \mapsto \{p\} & \{p,r,s\} \mapsto \{p\} \\
\{p,r\} \mapsto \{p\} & \{q,r,s\} \mapsto \{p\} & \{p,s\} \mapsto \{p\} & \{p,q,r,s\} \mapsto \{p\}\}
\end{array}
$$

Allowed clause bodies (head p):

$S = \{p, r; \neg p, \neg r, \neg s; q, \neg p, \neg r; q, \neg r, \neg s; p, q, \neg s; p, s, \neg q; q, s, \neg p; q, r, s\}$

Two runs of the algorithm:

$$
\begin{array}{lll}
P_1 = \{p \leftarrow p, r; & p \leftarrow q, \neg p, \neg r; & \\
\quad p \leftarrow \neg p, \neg r, \neg s; & p \leftarrow q, \neg r, \neg s; & \\
\quad p \leftarrow p, s, \neg q; & p \leftarrow q, s, \neg p & \}
\end{array}
\qquad
\begin{array}{lll}
P_2 = \{p \leftarrow p, r; & p \leftarrow \neg p, \neg r, \neg s; & \\
\quad p \leftarrow q, s, \neg p; & p \leftarrow q, \neg r, \neg s; & \\
\quad p \leftarrow p, s, \neg q & & \}
\end{array}
$$