Propositional Rule Extraction from Neural Networks under Background Knowledge

Maryam Labaf[1,2], Pascal Hitzler[1], and Anthony B. Evans[2]

[1] Data Semantics (DaSe) Laboratory, Wright State University, Dayton, OH, USA
[2] Dept. of Math. and Stat., Wright State University, Dayton, OH, USA

Abstract. It is well-known that the input-output behaviour of a neural network can be recast in terms of a set of propositional rules, and under certain weak preconditions this is also always possible with positive (or definite) rules. Furthermore, in this case there is in fact a unique minimal (technically, *reduced*) set of such rules which perfectly captures the input-output mapping.

In this paper, we investigate to what extent these results and corresponding rule extraction algorithms can be lifted to take additional background knowledge into account. It turns out that uniqueness of the solution can then no longer be guaranteed. However, the background knowledge often makes it possible to extract simpler, and thus more easily understandable, rulesets which still perfectly capture the input-output mapping.

1 Introduction

The study of rule extraction from trained artificial neural networks [2,8,15] addresses the desire to make the learned knowledge accessible to human interpretation and formal assessment. Essentially, in the propositional case, activations of input and output nodes are discretized by introducing an arbitrary threshold. Each node is interpreted as a propositional variable, and activations above the threshold are interpreted as this variable being "true", while activations below the threshold are interpreted as this variable being "false". If $\mathcal{I}$ denotes the power set (i.e., set of all subsets) of the (finite) set $\mathcal{B}$ of all propositional variables corresponding to the nodes, then the input-output function of the network can be understood as a function $f : \mathcal{I} \to \mathcal{I}$: For $I \in \mathcal{I}$, we interpret each $p \in I$ as being "true" and all $p \notin I$ as being "false". The set $f(I)$ then contains exactly those propositional variables which are "true" (or activated) in the output layer.

In propositional rule extraction, one now seeks sets P_f of propositional rules (i.e., propositional Horn clauses) which capture or approximate the input-output mapping f. In order to obtain such sets, there exist two main lines of approaches. The first is introspective and seeks to construct rules out of the weights associated with the connections between nodes in the network, usually proceeding in a layer-by-layer fashion [8]. The second is to regard the network as a black box and to consider only the input-output function f. This was, e.g., done in [15] where it was shown, amongst other things, that a positive (or definite) ruleset can always be extracted if the mapping f is monotonic, and that there is indeed a unique *reduced* such ruleset; we will provide sufficient technical details about these preliminary results in the next section.

However, rulesets extracted with either method are prone to be large and complex, i.e., from inspection of these rulesets it is often difficult to obtain real insights into what the network has learned. In this paper, we thus investigate rule extraction under the assumption that there is additional background knowledge which can be connected to network node activations, with the expectation that such background knowledge will make it possible to formulate simpler rulesets which still explain the input-output functions of the networks, if the background knowledge is also taken into account.

The motivation for this line of work is the fact that in recent years there has been a very significant increase in the availability of structured data on the World Wide Web, i.e., it becomes easier and easier to actually find such structured knowledge for all different kinds of application domains. That this is the case is, among other things, a result of recent developments in the field of *Semantic Web* [4,12], which is concerned with data sharing, discovery, integration and reuse, and where corresponding standards, methods and tools are being developed. E.g., structured data in the form of *knowledge graphs*, usually encoded using the W3C standards RDF [3] and OWL [11], has been made available in ever increasing quantities for over 10 years [5,17]. Other large-scale datasets include Wikidata [20] and data coming from the schema.org [9] effort which is driven by major Web search engine providers.

In order to motivate the rest of the paper, consider the following very simple example. Assume that the input-output mapping P of the neural network without background knowledge is

$$p_1 \wedge q \to r \qquad\qquad p_2 \wedge q \to r$$

and that we also have background knowledge K in form of the rules

$$p_1 \to p \qquad\qquad p_2 \to p.$$

We then obtain the simplified input-output mapping P_K, taking background knowledge into account, as

$$p \wedge q \to r.$$

The example already displays a key insight why background knowledge can lead to simpler extracted rulesets: In the example just given, p serves as a “more general” proposition, e.g., p_1 could stand for “is an apple” while p_2 could stand for “is a banana”, while p could stand for “is a fruit”. If we now also take, e.g., q to stand for “is ripe” and r to stand for “can be harvested”, then we obtain a not-so-abstract toy example, where the background knowledge facilitates a simplification because it captures both apples and bananas using the more general concept “fruit”.

In this paper, we will formally define the setting for which we just gave an initial example. We will furthermore investigate to what extent we can carry over results regarding positive rulesets from [15] to this new scenario with background knowledge. We will see that the pleasing theoretical results such as uniqueness of a solution no longer hold. However, existence of solutions can still be guaranteed under the same mild conditions as in [15], and we will still be able to obtain algorithms for extracting corresponding rulesets.

The rest of the paper will be structured as follows. In Section 2 we will introduce notation as needed and recall preliminary results from [15]. In Section 3 we present the results of our investigation into adding background knowledge. In Section 4, we briefly discuss related work, and in Section 5 we conclude and discuss avenues for furture work.

2 Preliminaries

We recall notation and some results from [15] which will be central for the rest of the paper. For further background on notions concerning logic programs, cf. [13].

As laid out in the introduction, let $\mathcal{B}$ be a finite set of propositional variables, let $\mathcal{I}$ be the power set of $\mathcal{B}$, and we consider functions $f : \mathcal{I} \to \mathcal{I}$ as discretizations of input-output functions of trained neural networks. In this paper, we consider only positive (or definite) propositional rules, which are of the form $p_1 \wedge \cdots \wedge p_n \to q$, where q and all p_i are propositional variables. A set P of such rules is called a (propositional) logic program. For such a rule, we call q the *head* of the rule, and $p_1 \wedge \cdots \wedge p_n$ the *body* of the rule.

A logic program P is called *reduced* if all of the following hold.

1. For every rule $p_1 \wedge \cdots \wedge p_n \to q$ in P we have that all p_i are mutually distinct.
2. There are no two rules $p_1 \wedge \cdots \wedge p_n \to q$ and $r_1 \wedge \cdots \wedge r_m \to q$ in P with $\{p_1, \ldots, p_n\} \subseteq \{r_1, \ldots, r_m\}$.

To every propositional logic program P over $\mathcal{B}$ we can associate a semantic operator T_P, called the *immediate consequence operator*, which is the function

$$T_P : \mathcal{I} \to \mathcal{I} :$$
$$T_P(I) = \{q \mid \text{there exists } p_1 \wedge \cdots \wedge p_n \to q \text{ in } P \text{ with } \{p_1, \ldots, p_n\} \subseteq I\}.$$

This operator is well-known to be monotonic in the sense that whenever $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.

We make some additional mild assumptions: We assume that the propositional variables used to represent input and output nodes are distinct, i.e., each propositional variable gets used either to represent an input node, or an output node, but not both. Technically, this means that $\mathcal{B}$ can be partitioned into two sets $\mathcal{B}_1$ and $\mathcal{B}_2$, i.e., $\mathcal{B} = \mathcal{B}_1 \dot\cup \mathcal{B}_2$, and we obtain the corresponding power sets $\mathcal{I}_1$ and $\mathcal{I}_2$ such that $T_P : \mathcal{I}_1 \to \mathcal{I}_2$.

While the definition of the immediate consequence operator just presented is very common in the literature, we will now give a different but equivalent formalization, which will help us in this paper. For any $I = \{p_1, \ldots, p_n\} \subseteq \mathcal{B}$, let $c(I) = p_1 \wedge \cdots \wedge p_n$. In fact, whenever $I \subseteq \mathcal{B}$, in the following we will often simply write I although we may mean $c(I)$, and the context will make it clear

Algorithm 1: Reduced Definite Program Extraction

Input: A monotone mapping $f : \mathcal{I}_1 \to \mathcal{I}_2$.
Output: P, a definite logic program with $T_P(I) = f(I)$ for all $I \in \mathcal{I}_1$.

Initialization: $P = \emptyset$.
Choose a total linear order $\prec$ on $\mathcal{I}_1$, such that for any $I_i, I_j \in \mathcal{I}_1$ with $i < j$ we have $|I_i| < |I_j|$.
for all $I = \{p_1, \dots, p_n\} \in \mathcal{I}_1$, chosen in ascending order according to $\prec$ **do**
  for all $q \in f(I)$ **do**
    if there is no $q_1 \wedge \dots \wedge q_n \to q$ in P with $\{q_1, \dots, q_n\} \subseteq I$ **then**
      add the rule $p_1 \wedge \dots \wedge p_n \to q$ to P.
    end if
  end for
end for
Return P as result.

which notation is meant; e.g., if I appears as part of a logical formula, then we actually mean $c(I)$.

Now, given a logic program P and $I \in \mathcal{I}_1$, we obtain

$$T_P(I) = \{q \in \mathcal{B}_2 \mid I \wedge P \models q\},$$

where $\models$ denotes entailment in propositional logic. Please note that we use another common notational simplification, as $I \wedge P$ is used to denote $I \wedge \bigwedge_{R \in P} R$.

In [15], the following was shown.

Theorem 1. *Let $f : \mathcal{I}_1 \to \mathcal{I}_2$ be monotonic. Then there exists a unique reduced logic program P with $T_P = f$. Furthermore, this logic program can be obtained using Algorithm 1.*

If we drop the precondition on f to be monotonic, then Theorem 1 no longer holds, because of the fact mentioned above that immediate consequence operators are always monotonic.

We will now investigate Theorem 1 when considering additional background knowledge. It will be helpful to have the following corollary from Theorem 1 at hand.

Theorem 2. *Given a logic program P, there is always a unique reduced logic program Q with $T_P = T_Q$.*

Proof. Given P, we know that T_P is monotonic. Now apply Theorem 1.

Let us give an example for reducing a given program. Let $\mathcal{B}_1 = \{p_1, p_2, p_3\}$ and $\mathcal{B}_2 = \{q_1, q_2\}$ be input and output sets, respectively, and consider the logic program P given as

$$\begin{aligned} &p_1 \wedge p_2 \to q_1 && p_1 \wedge p_2 \wedge p_3 \to q_1 \\ &p_1 \wedge p_3 \to q_1 && p_1 \to q_2 \\ &p_1 \wedge p_2 \to q_2. \end{aligned}$$

Applying Algorithm 1 then yields the reduced program

$$p_1 \wedge p_2 \to q_1 \qquad\qquad p_1 \wedge p_3 \to q_1$$
$$p_1 \to q_2.$$

3 Rule extraction with Background Knowledge

We consider the following setting. Assume P is a logic program which captures the input-output function of a trained neural network according to Theorem 1. Let furthermore K be a logic program which constitutes our background knowledge, and which may use additional propositional variables, i.e., propositional variables not occurring in P. We then seek a logic program P_K such that, for all $I \in \mathcal{I}_1$, we have

$$\{q \in \mathcal{B}_2 \mid I \wedge P \models q\} = \{q \in \mathcal{B}_2 \mid I \wedge K \wedge P_K \models q\}. \tag{1}$$

In this case, we call P_K a *solution* for (P, K).

3.1 Existence of Solutions

We next make two more mild assumptions, namely (1) that no propositional variable from $\mathcal{B}_2$ appears in K, and that (2) propositional variables from $\mathcal{B}_1$ appear only in bodies of rules in K. The first is easily justified by the use case, since we want to explain the network behaviour, and the occurrence of variables from $\mathcal{B}_2$ in K would bypass the network. The second is also easily justified by the use case, which indicates that network input activations should be our starting point, i.e. the activations should not be altered by the background knowledge.

If we drop assumption (2) just stated, then existence of a solution cannot be guaranteed: Let $\mathcal{B}_1 = \{p_1, p_2\}$, let $\mathcal{B}_2 = \{q_1, q_2\}$. Then, for the given programs $P = \{p_1 \to q_1, p_2 \to q_2\}$ and $K = \{p_1 \to p_2\}$ there is no solution for (P, K). To see this, assume that P_K be a solution for (P, K). Then because $p_2 \wedge P \models q_2$ we obtain that $p_2 \wedge K \wedge P_K \models q_2$. But then $p_1 \wedge K \wedge P_K \models q_2$ although $p_1 \wedge P \not\models q_2$, i.e., P_K cannot be a solution for (P, K).

If condition (2) from above is assumed, though, a solution always exists.

Proposition 1. *Under our standing assumptions on given logic programs P and K, there always exists a solution for (P, K) which is reduced.*

Proof. Because rule heads from K never appear in P, we obtain

$$\{q \in \mathcal{B}_2 \mid I \wedge P \models q\} = \{q \in \mathcal{B}_2 \mid I \wedge K \wedge P \models q\}$$

for all $I \in \mathcal{I}_1$, i.e., P is always a solution for (P, K). Existence of a reduced solution then follows from Theorem 2.

Our interest of course lies in determining *other* solutions which are simpler than P.

Algorithm 2: Construct all reduced solutions for (P, K)

Input: Logic programs P and K with $T_P : \mathcal{I}_1 \to \mathcal{I}_2$ and $T_K : \mathcal{I}_1 \to \mathcal{I}_3$ which satisfy our standing assumptions, where $\mathcal{B}_2 = \{q_1, \ldots, q_n\}$.
Output: All the reduced solutions for (P, K).

Set $\mathcal{S} = \emptyset$ and $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_3$.
Set $\mathcal{I}$ to be the power set of $\mathcal{B}$.
Set $\mathcal{R}$ to be the power set of $\mathcal{I}$.
for all $(R_1, \ldots, R_n) \in \mathcal{R}^n$ **do**
  for all $i \in \{1, \ldots, n\}$ **do**
    $Q_i = \{c(B) \to q_i \mid B \in R_i\}$
  end for
  Set $Q = \bigcup_{i \in \{1,\ldots,n\}} Q_i$.
  if T_Q is a solution for (P, K) **then**
    Apply Algorithm 1 to T_Q to obtain a reduced program S with $T_S = T_Q$.
    if $S \notin \mathcal{S}$ **then**
      Add S to $\mathcal{S}$.
    end if
  end if
end for
Return $\mathcal{S}$ as result.

Proposition 2. *There exist logic programs P and K which satisfy our standing assumptions, such that there are two distinct reduced solutions for (P, K).*

Proof. Let $\mathcal{B}_1 = \{p_1, p_2, p_3\}$ and $\mathcal{B}_2 = \{q\}$. Then consider the programs P as

$$p_2 \wedge p_3 \to q \qquad\qquad p_1 \wedge p_3 \to q$$

and K as

$$\begin{aligned} p_2 \wedge p_3 &\to r_1 & \qquad p_1 \wedge p_3 &\to r_1 \\ p_1 &\to r_2 & p_2 &\to r_2. \end{aligned}$$

The two logic programs

$$\begin{aligned} P_{K_1} &= \{r1 \to q\} \qquad \text{and} \\ P_{K_2} &= \{p_3 \wedge r_2 \to q\} \end{aligned}$$

are then both reduced solutions for (P, K).

We will see later in the proof of Theorem 3, that the number of reduced solutions is actually worst-case exponential in the combined size of P and K.

3.2 Algorithms

We first present a naive algorithm for computing all reduced solutions for given (P, K). It is given as Algorithm 2 and it uses a brute-force approach to check all possible logic programs which can be constructed over the given propositional variables, whether they constitute a solution for (P, K). For each such solution, it then invokes Algorithm 1 to obtain a corresponding reduced program, which is then added to the solution set. The algorithm is quite obviously correct and always terminating, and we skip a formal proof of this.

The given algorithm is of course too naive to be practically useful for anything other than toy examples. Still, it is worst-case optimal, as the following theorem shows – note that Algorithm 2 has exponential runtime because of line 4.

Theorem 3. *The problem of finding all solutions to (P, K) is worst-case exponential in the combined size of P and K.*

Proof. Let n be any positive integer. Define the logic program P_n to consist of the single rule $p_1 \wedge \cdots \wedge p_n \to q$ and let

$$K_n = \{p_i \to r_{i,1}, p_i \to r_{i,2} \mid i = 1, \ldots, n\}.$$

Then, for any function $f : \{1, \ldots, n\} \to \{1, 2\}$, the logic program

$$P_f = \{r_{1,f(1)} \wedge \cdots \wedge r_{n,f(n)} \to q\}$$

is a reduced solution for (P_n, K_n). Since there exist 2^n distinct such functions f, the number of reduced solutions in this case is 2^n, so their production is exponential in n, while the combined size of P_n and K_n grows only linearly in n.

A more efficient algorithm for obtaining only *one* reduced solution is given as Algorithm 3. It is essentially a combination of Algorithms 1 and 2.

Proposition 3. *Algorithm 3 is correct and always terminating.*

Proof. Like Algorithm 1, Algorithm 3 checks all combinations of $I \in \mathcal{I}_1$ and $q \in T_P(I)$ and makes sure that there are rules in the output program such that $I \wedge K \wedge S \models q$. The rules for the output program are checked one by one in increasing length until a suitable one is found. Note that the rule $I \to q$ is going to be checked at some stage, i.e. the algorithm will either choose this rule, or a shorter one, but in any case we will eventually have $I \wedge K \wedge S \models q$. This shows that the algorithm always terminates and that we obtain $I \wedge K \wedge S \models q$ for all $q \in T_P(I)$.

In order to demonstrate that the algorithm output S is indeed a solution for (P, K), we also need to show that for all $q \in \mathcal{B}_2$ and $H \in \mathcal{I}_1$ we have that $H \wedge K \wedge S \models q$ implies $q \in T_P(H)$. This is in fact guaranteed by line 11 of Algorithm 3, i.e. the algorithm output S is indeed a solution for (P, K).

We finally show that the output of the algorithm is reduced. Assume otherwise. Then there are $I_1 \to q$ and $J \to q$ in S with $I_1 \subsetneq J$. By our condition on the order we thus have $I_1 \prec J$ and so we know that $I_1 \to q$ was added to S earlier in the algorithm than $J \to q$. now let us look at the instance of line 12 in Algorithm 3 when the rule $J \to q$ was added to S. In this case (using notation from the algorithm description, and S denoting the current S at that

Algorithm 3: Reduced solution for (P, K)

Input: Logic programs P and K with $T_P : \mathcal{I}_1 \to \mathcal{I}_2$ and $T_K : \mathcal{I}_1 \to \mathcal{I}_3$ which satisfy our standing assumptions.
Output: A reduced solution for (P, K).

Set $S = \emptyset$ and $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_3$.
Set $\mathcal{I}$ to be the power set of $\mathcal{B}$.
Choose a total linear order $\prec$ on $\mathcal{I}$, such that for any $I_i, I_j \in \mathcal{I}$ with $i < j$ we have $|I_i| < |I_j|$.
for all $I = \{p_1, \ldots, p_m\} \in \mathcal{I}_1$, chosen in ascending order according to $\prec$ **do**
  for all $q \in T_P(I)$ **do**
    if $I \wedge K \wedge S \not\models q$ **then**
      Set endloop = false.
      Choose first $J = \{b_1, \ldots, b_n\} \in \mathcal{I}$ according to $\prec$.
      while endloop = false **do**
        if $I \wedge K \wedge S \wedge (J \to q) \models q$ **then**
          if $\{H \in \mathcal{I}_1 \mid H \wedge K \wedge S \wedge (J \to q) \models q\} \subseteq \{H \in \mathcal{I}_1 \mid q \in T_P(H)\}$ **then**
            Add the rule $J \to q$ to S and set endloop = true.
          end if
        else
          Choose next $J = \{b_1, \ldots, b_n\} \in \mathcal{I}$ according to $\prec$.
        end if
      end while
    end if
  end for
end for
Return S as a result.

moment) we know that $I \wedge K \wedge S \wedge (J \to q) \models q$ and $I \wedge K \wedge S \not\models q$. This implies $I \wedge K \wedge S \models J$, and because $I_1 \subseteq J$ we obtain $I \wedge K \wedge S \models I_1$. But we also have already observed that $I_1 \to q$ is already contained in S at this stage, and thus we obtain $I \wedge K \wedge S \models q$, which contradicts the earlier statement that $I \wedge K \wedge S \not\models q$. We thus have to reject the assumption that S is not reduced; hence S is indeed reduced. This completes the proof.

To close, we give a somewhat more complex example. Let $\mathcal{B}_1 = \{p_1, p_2, p_3\}$ and $\mathcal{B} = \{q_1, q_2, q_3, q_4\}$. Consider the program P as

$$
\begin{array}{ll}
p_1 \to q_1 & p_2 \to q_1 \\
p_1 \to q_2 & p_2 \to q_2 \\
p_1 \to q_3 & p_2 \to q_3 \\
p_1 \to q_4 & p_2 \wedge p_3 \to q_4
\end{array}
$$

and K as

$$p_1 \to r_1 \qquad p_2 \wedge p_3 \to r_1$$
$$p_1 \to r_2 \qquad p_2 \to r_2$$
$$p_2 \to r_3.$$

Then there is only one reduced solution P_K for (P, K), which is

$$r_2 \to q_1 \qquad r_2 \to q_2$$
$$r_2 \to q_3 \qquad r_1 \to q_4.$$

Note, that P_K is simpler and shorter than P.

4 Related Work

It would be out of place to have a lengthy discussion of related work in neural-symbolic integration, or even just on the topic of rule extraction, in this brief paper. We hence limit ourselves to some key pointers including overview texts. We already discussed the rule-extraction work [15] on which our work is based, and [8] which pursues a different approach based on inspecting weights. For more extensive entry points to literature on neural-symbolic integration we refer to [2,6,7,10] and to the proceedings of the workshop series on Neural-Symbolic Learning and Reasoning.[3]

Regarding the novel aspect of this work, namely the utilization of background knowledge for rule extraction, we are not aware of any prior work which pursues this. However, concurrently the second author has worked on lifting the idea to the application level in [19], by utilizing description logics and Semantic Web background knowledge in the form of ontologies and knowledge graphs [12] together with the DL-Learner system [16] for rule extraction. The results herein, which are constrained to the propositional case, can be considered foundational for the more application-oriented work currently pursued along the lines of [19].

We are also greatful that a reviewer pointed out a possible relationship of our work with work laid out in [18] in the context of abduction in logic programming. Looked at on a very generic level, the general abduction task is very similar to our formulation in equation (1), which means that the field of abduction may indeed provide additional insights or even algorithms for our setting. On the detail level, however, [18] differs significantly. Most importantly, [18] consideres literals or atoms as abducibles, i.e., an explanation consists of a set of literals, while in our setting explanations are actually rule sets. Another difference is that [18] considers logic programs under the non-monotonic answer set semantics, i.e., logic programs with default negations, while we consider only logic programs without negation in our work – it was laid out in much detail in [15] that propositional rule extraction under negation has a significantly different dynamics. Nevertheless, the general field of abduction in propositional logic programming may provide inspiration for further developing our approach, but working out the exact relationships appears to be a more substantial investigation.

[3] `http://neural-symbolic.org/`

5 Conclusions and Further Work

We have investigated the issue of propositional rule extraction from trained neural networks under background knowledge, for the case of definite rules. We have shown that a mild assumption on the background knowledge and monotonicity of the input-output function of the network suffices to guarantee that a reduced logic program can be extracted such that the input-output function is exactly reproduced. We have also shown that the solution is not unique. Furthermore, we have provided algorithms for obtaining corresponding reduced programs.

We consider our results to be foundational for further work, rather than directly applicable in practice. Our observation that background knowledge can yield simpler extracted rulesets of course carries over to more expressive logics which extend propositional logic.

It is such extensions which we intend to pursue, which hold significant promise for practical applicability: structured information on the World Wide Web, as discussed in the Introduction, is provided in logical forms which are usually non-propositional fragments of first-order predicate logic, or closely related formalisms. In particular, description logics [1], i.e. decidable fragments of first-order predicate logic, form the foundation of the Web Ontology Language OWL. First-order rules are also commonly used [14]. This raises the question how to extract meaningful non-propositional rules from trained neural networks while taking (non-propositional) background knowledge, in a form commonly used on the World Wide Web, into account.

Acknowledgements. The first two authors acknowledge support by the Ohio Federal Research Network project *Human-Centered Big Data*.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edn. (2010)
2. Bader, S., Hitzler, P.: Dimensions of neural-symbolic integration – A structured survey. In: Artëmov, S.N., Barringer, H., d'Avila Garcez, A.S., Lamb, L.C., Woods, J. (eds.) We Will Show Them! Essays in Honour of Dov Gabbay, Volume One. pp. 167–194. College Publications (2005)
3. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1. Turtle – Terse RDF Triple Language. W3C Recommendation (25 February 2014), available at http://www.w3.org/TR/turtle/
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (May 2001)
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story So Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
6. d'Avila Garcez, A., Besold, T.R., de Raedt, L., Földiak, P., Hitzler, P., Icard, T., Kühnberger, K.U., Lamb, L.C., Miikkulainen, R., Silver, D.L.: Neural-symbolic learning and reasoning: Contributions and challenges. In: McCallum, A., Gabrilovich, E., Guha, R., Murphy, K. (eds.) Proceedings of the AAAI 2015

Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches. AAAI Press Technical Report, vol. SS-15-03. AAAI Press, Palo Alto (2015)
7. d'Avila Garcez, A.S., Lamb, L.C., Gabbay, D.M.: Neural-Symbolic Cognitive Reasoning. Cognitive Technologies, Springer (2009)
8. d'Avila Garcez, A.S., Zaverucha, G.: The connectionist inductive lerarning and logic programming system. Applied Intelligence 11(1), 59–77 (1999)
9. Guha, R.V., Brickley, D., Macbeth, S.: Schema.org: evolution of structured data on the web. Commun. ACM 59(2), 44–51 (2016)
10. Hammer, B., Hitzler, P. (eds.): Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence, vol. 77. Springer (2007)
11. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S. (eds.): OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation (11 December 2012), http://www.w3.org/TR/owl2-primer/
12. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. CRC Press/Chapman & Hall (2010)
13. Hitzler, P., Seda, A.K.: Mathematical Aspects of Logic Programming Semantics. CRC Press/Chapman and Hall (2010)
14. Krisnadhi, A., Maier, F., Hitzler, P.: OWL and rules. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P.F. (eds.) Reasoning Web. Semantic Technologies for the Web of Data – 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures. Lecture Notes in Computer Science, vol. 6848, pp. 382–415. Springer (2011)
15. Lehmann, J., Bader, S., Hitzler, P.: Extracting reduced logic programs from artificial neural networks. Appl. Intell. 32(3), 249–266 (2010)
16. Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Machine Learning 78(1-2), 203–250 (2010)
17. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web 6(2), 167–195 (2015)
18. Lin, F., You, J.: Abduction in logic programming: A new definition and an abductive procedure based on rewriting. Artif. Intell. 140(1/2), 175–205 (2002)
19. Sarker, M.K., Xie, N., Doran, D., Raymer, M., Hitzler, P.: Explaining trained neural networks with semantic web technologies: First steps. In: Proceedings of the Twelveth International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'17, London, UK, July 2017 (2017), to appear
20. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM 57(10), 78–85 (2014)