A Best-first Backward-chaining Search Strategy based on Learned Predicate Representations

Alexander Sakharov
Synstretch, Framingham, MA, U.S.A.

Keywords: Knowledge Base, First-order Logic, Resolution, Backward Chaining, Neural-symbolic Computing, Tensorization.

Abstract: Inference methods for first-order logic are widely used in knowledge base engines. These methods are powerful but slow in general. Neural networks make it possible to rapidly approximate the truth values of ground atoms. A hybrid neural-symbolic inference method is proposed in this paper. It is a best-first search strategy for backward chaining. The strategy is based on neural approximations of the truth values of literals. This method is precise and the results are explainable. It speeds up inference by reducing backtracking.

1 INTRODUCTION

The facts and rules of knowledge bases (KB) are usually expressible in first-order logic (FOL) (Russell and Norvig, 2009). Typically, KB facts are literals. Quantifier-free implications $A \Leftarrow A_1 \wedge ... \wedge A_k$, where $A, A_1, ..., A_k$ are literals, are arguably the most common form of rules in KBs. All these literals are positive in Prolog rules. In general logic programs, rule heads are positive. These rules are equivalent to disjunctions of literals in classical FOL. These disjunctions are known as non-Horn clauses, and the disjunctions corresponding to Prolog rules are called Horn clauses. Any FOL formula can be expressed by such set of non-Horn clauses that their conjunction is equisatisfiable with this formula (Nie, 1997).

Resolution methods work on sets of non-Horn clauses. These methods have become a de facto standard for inference in KBs and logic programming (Russell and Norvig, 2009). Their success is a major reason for the popularity of non-Horn clauses as a knowledge representation format. Complete inference methods for FOL including resolution are inherently slow. Multiple strategies and heuristics speeding up resolution procedures have been developed. SLD resolution, which is also known as backward chaining, is a complete strategy for Horn clauses. Faster but incomplete inference procedures are also acceptable for KBs. Prolog also utilizes an incomplete inference procedure (Stickel, 1992).

The inefficiency of inference in FOL and even in its Horn fragment prompted attempts to replace inference with neural networks (NN) (Rocktäschel, 2017; Serafini and d'Avila Garcez, 2016; Dong et al., 2019; Marra et al., 2019; Van Krieken et al., 2019; Sakharov, 2019). Most commonly, it is done via predicate tensorization. Objects are embedded as real-valued vectors of a fixed length for the use in NNs. Predicates are represented by one or more tensors of various ranks which are learned. The truth values of ground atoms of any predicate P are approximated by applying a symbolically differentiable function σ to an algebraic expression. The range of σ is the interval $[0,1]$. One corresponds to *true*, and zero corresponds to *false*. The expression is composed of the tensors representing P, embeddings of the constants that are P arguments, tensor contraction operations, and symbolically differentiable functions.

One key advantage of this machine learning approach over inference is that approximation of truth values of ground atoms is fast. Assuming that σ and other functions from the aforementioned expression are efficiently implemented, the approximation takes a linear time over the size of the tensors representing a predicate. Unfortunately, there are serious cons to this approach.

This approach is limited to ground atoms. If the result of an approximation is around 0.5, it is not possible to draw any conclusion about the truth value of an atom. Approximation results may not be reliable. Their accuracy is not known in advance and previous results do not provide any assurance for the same for future results. The truth values yielded by NNs are not explainable because machine learning does not pro-

Sakharov, A.
A Best-first Backward-chaining Search Strategy based on Learned Predicate Representations.
DOI: 10.5220/0010299209820989
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 982-989
ISBN: 978-989-758-484-8

vide any justification for the results. In many AI tasks such as automatic code generation, robotic planning, etc., the aim is actually a derivation itself not the mere knowledge of the truth value. The approximation of truth values based on NNs does not contribute to these tasks.

Hybrid approaches that combine symbolic reasoning and machine learning models are considered the most promising (Marcus, 2020). To the best of author's knowledge, there are no known hybrid methods that retain the accuracy of inference, produce derivations, and take advantage of learned predicate representations. This work introduces such method. It is a search strategy for backward chaining. This search strategy utilizes learned predicate representations in order to make better choices at every inference step, and thus, to reduce backtracking which usually consumes the vast portion of time during inference.

2 RESOLUTION

Resolution is perhaps the most practical inference method. The resolution calculus works on Skolemized FOL formulas in the conjunctive normal form. The conjuctions are viewed as sets of disjunctions of literals, i.e. non-Horn clauses. The resolution calculus has two rules: resolution and factoring. The resolution rule produces disjunction

$$A_1\theta \vee ... \vee A_{i-1}\theta \vee A_{i+1}\theta \vee ... \vee A_k\theta \vee B_1\theta \vee ... \vee B_{j-1}\theta \vee B_{j+1}\theta \vee ... \vee B_m\theta$$

from two disjunctions $A_1 \vee ... \vee A_k$ and $B_1 \vee ... \vee B_m$ where substitution θ is the most general unifier of A_i and $\neg B_j$. The factoring rule produces disjunction

$$A_1\theta \vee ... \vee A_{i-1}\theta \vee A_{i+1}\theta \vee ... \vee A_k\theta$$

from disjuction $A_1 \vee ... \vee A_k$ where substitution θ is the most general unifier of A_i and A_j. Factoring can be combined with the resolution rule. We assume reader's familiarity with resolution. Please refer to (Chang and Lee, 1973) for details. In this paper, we consider KB rules that are equivalent non-Horn clauses, and KB facts that are literals.

Unconstrained resolution may be very inefficient. What makes resolution practical is the availability of strategies that constrain branching at every resolution step by prohibiting certain applications of the resolution rule. These strategies include set of support resolution, unit resolution, linear resolution, etc. Some of them are complete for FOL, some are not.

Backward chaining can be applied to non-Horn clauses as well (Sakharov, 2020). In this strategy, one of any two resolved literals is a rule head or a fact. It is an incomplete strategy for non-Horn clauses but it is relatively efficient. Complete inference procedures for FOL and its extensions are more suitable for theorem provers. Inference in KBs is supposed to be faster, even at the expense of completeness. Unlike theorem provers, the number of facts and rules involved in KB inference may be huge, which slows down the inference. The use of incomplete strategies is also justified by the fact that KBs are almost always incomplete.

Backward chaining is often explained in terms of goal lists (sets). The set of negations of literals of one disjunction is considered as an original list of goals. Every resolvent is also viewed as a goal list that is comprised of negations of its literals. We follow this tradition. Due to this explanation, backward chaining is interpreted as inference based on generalized Modus Ponens (Russell and Norvig, 2009). Goals have to be derived, not refuted. Not only this interpretation makes backward chaining more explainable, it is also more pertinent to our best-first strategy. This strategy aims to pick the facts or rules that lead to goal lists whose elements are more likely derivable.

Various search strategies can be used in implementations of resolution. Search strategies determine the order in which literals or disjunctions are resolved. These strategies include depth-first, breadth-first, iterative deepening, and others. Prolog relies on the depth-first strategy. It is incomplete but efficient. Unit preference (Russell and Norvig, 2009) is one well-known best-first search strategy for resolution. Facts are resolved before rules under unit preference. OTTER (McCune, 2003) conducts best-first search on the basis of rule weight. Lighter rules are preferred. Longer rules tend to have a higher weight.

The OTTER's search strategy is perhaps the closest to the strategy presented in this paper. In the examples given later, we compare the two. For certainty, we assume that the rule weight equals the number of symbols in the rule including variables, constants, functions, predicates, and negations. Resolution strategies should include some form of loop detection (Shen et al., 2001). There also exist optimization techniques that make resolution implementations more efficient. One notable example of these techniques is tabling (Swift, 2009).

Non-Horn clauses may contain Skolem functions or constants. We refer to both of them as Skolem functions for short. They are introduced in the process of eliminating existential quantifiers from FOL formulas (Chang and Lee, 1973). Skolem functions are not evaluable because they are unknown. It is fair to assume that all other functions in KB rules are evaluable. Any term with a Skolem function at the top can be unified with a variable only. Usually, the majority of rules do not contain Skolem functions.

When predicates are approximated by NNs and rules contain Skolem functions, subsymbolic representations of Skolem functions, or other functions for that matter, can be learned like subsymbolic representations of predicates (Sakharov, 2019). Like predicates, Skolem functions are represented by one or more tensors of various ranks. These representations of Skolem functions map embeddings of function arguments into embeddings of function results. If learned subsymbolic representations of predicates, Skolem functions, and the functions occurring above Skolem functions in terms are available, then it is possible to approximate the truth value of any ground atom regardless of whether it contains Skolem functions or not.

3 BEST-FIRST SEARCH STRATEGY

Our strategy is designed for backward chaining. This strategy selects the goal list that is either the best candidate or is comparable with the the best one. Given that, it is fair to call it best-first. This strategy may work as a supplement to other search strategies which will be called base strategies in this case. If a base strategy leaves multiple choices, then our strategy is invoked. For example, our best-first strategy can be used to select individual facts in addition to employing unit preference.

A KB may have evaluable predicates which are implemented as boolean recursive functions outside of the KB rules. Literals of evaluable predicates are not resolved. Instead, their truth values are calculated as soon as all arguments are bound to constants. For any goal list, this evaluation is done before all other goals are resolved. Equality is an evaluable predicate.

An extensional predicate is a predicate that is wholly defined by a set of ground literals. Usually, facts of extensional predicates are ground literals without functions. They constitute the bulk of a KB, and the number of rules is much smaller. Extensional predicates do not need subsymbolic representations either. The truth values of their literals present in the KB are known. The truth value of any absent literal with constant arguments is approximated as zero.

Our strategy relies on the following property of backward-chaining derivations:

Proposition 1. *Every goal in any backward-chaining derivation is derivable or is a fact.*

Since every step of backward chaining is interpreted as an application of generalized Modus Ponens, the proof of Proposition 1 is a straightforward induction on the depth of derivations. In general, resolution derivations may contain literals whose negations are not derivable. Due to this, the interpretation of resolvents as goal lists is limited to backward chaining.

Suppose $\mathcal{D}(v)$ denotes the domain of variable v and $\mathcal{D}(P_n)$ denotes the domain of the n-th argument of predicate P. For any ground literal B, let $[B]$ denote its truth value, i.e. 0 (*false*) or 1 (*true*). For any set of goals $\{B_1,...,B_b\}$ with free variables $u_1...u_w$, its truth value $[B_1,...,B_b]$ can be expressed as:

$$[B_1,...,B_b] = \max_{u_1 \in \mathcal{D}(u_1)...u_w \in \mathcal{D}(u_w)} \min_{i=1...b} [B_i] \quad (1)$$

We assume that the arguments of any ground literal without Skolem functions are evaluated. If the truth value of a ground atom is unknown, then a subsymbolic representation of the predicate of this literal (and subsymbolic representations of its Skolem functions if any) can be used to approximate the truth value. If r is the approximation of the truth value of ground atom A, then the truth value of $\neg A$ is approximated as $1-r$.

We use the same notation $[B_i]$ for the approximation of the truth value of ground literal B_i. It is given by a NN, or NNs in the presence of Skolem functions. The same formula can be used for approximating the truth values of arbitrary sets of goals. Based on these observations, Formula 1 will be used as the objective function for our best-first search strategy.

Heuristics can be incorporated into this objective function. If it is known that certain literals are less or more likely derivable, then their truth value approximations $[B_i]$ are adjusted as $[B_i]^p$. The value of p is close to one. It is more than one if literal B_i is less likely derivable, and p is less than one if B_i is more likely derivable. For instance, literals with Skolem functions are expected to be less likely derivable due to limited unification opportunities for them, and their truth value approximations can be adjusted. Finding suitable values of p requires experimentation.

The factoring rule has a higher priority than the resolution rule. It is always applied as soon as possible because this rule reduces the list of goals. Our best-first strategy is limited to resolution steps. A KB may contain a gigantic number of facts for some predicates, and this leads to huge branching factors in relevant resolution steps. Therefore, branching reduction for resolution steps involving facts is crucial.

The evaluation of the objective function for any resolvent R is possible only if the domains of all variables from R are finite. Even in this case, the evaluation may be too costly. The objective function is approximated by iterating over finite subsets of domains $\mathcal{D}(u_1),...,\mathcal{D}(u_w)$. If the evaluation or approximation

of the objective function does not yield a definite result, then an arbitrary resolvent is picked.

3.1 Tabling

Tabling enables an efficient approximate evaluation of the objective function. In contrast to tabling for resolution (Swift, 2009), values $[B_i]$ for ground literals B_i are tabled. These values are calculated in advance and persisted for use in all derivations. Since the aim of our search strategy is to find goal lists with the highest values of $[B_i]$, only these values are tabled. For atom $P(c_1,...,c_p)$ where $c_1,...,c_p$ are constants, the values $[P(c_1,...,c_p)] > t$ and the values $[\neg P(c_1,...,c_p)] > t$ are tabled. Here, t is a constant that is slightly smaller than 1.

To guarantee a fast access to tabled values, they are stored in hash tables. Every entry in such hash table $\mathcal{H}_P$ for predicate P is a key-value pair of one of the two following forms: $P(c_1,...,c_p) \rightarrow [P(c_1,...,c_p)]$, $\neg P(c_1,...,c_p) \rightarrow [\neg P(c_1,...,c_p)]$. Algorithm 1 populates $\mathcal{H}_P$. The total number of iterations in this algorithm is limited by constant $\overline{n}$.

It is fair to say that all computer data types are countable (enumerable). This applies even to the types that represent continuous sets such as real numbers because the number of bits representing these values on a computer is limited by a constant. For any computer data type, there exists a recursive function that is a bijection of natural numbers into the set of values of this type. We reuse the notation $\mathcal{D}(...)$ for denoting this bijection for the respective domain. Let p be the number of arguments of predicate P. As usual, $|A|$ denotes the cardinality of set A.

Algorithm 1.

$\mathcal{H}_P := \emptyset$; $n_1 := min\{\overline{n}, |\mathcal{D}(P_1)|\}$; ...
$n_p := min\{\overline{n}, |\mathcal{D}(P_p)|\}$;
for $i_1 := 1...n_1$;...;$i_p := 1...n_p$ **do**
  if $[P(\mathcal{D}(P_1,i_1),...,\mathcal{D}(P_p,i_p))] > t$ **then**
    $\mathcal{H}_P := \mathcal{H}_P \cup P(\mathcal{D}(P_1,i_1),...,\mathcal{D}(P_p,i_p)) \rightarrow$
      $[P(\mathcal{D}(P_1,i_1),...,\mathcal{D}(P_p,i_p))]$
  end if
  if $[P(\mathcal{D}(P_1,i_1),...,\mathcal{D}(P_p,i_p))] < 1-t$ **then**
    $\mathcal{H}_P := \mathcal{H}_P \cup \neg P(\mathcal{D}(P_1,i_1),...,\mathcal{D}(P_p,i_p)) \rightarrow$
      $1-[P(\mathcal{D}(P_1,i_1),...,\mathcal{D}(P_p,i_p))]$
  end if
end for

Functions $\mathcal{D}(...)$ can be viewed as sampling functions. It is beneficial if they are consistent with the probability distributions. It is assumed that Algorithm 1 iterates over entire finite domains. It can be achieved by adjusting the value of t for larger domains. Countable domains are recursively defined. Functions $\mathcal{D}(...)$ are expected to count the values constructed from other values after the values from which they are built, and the former are expected to have lower probabilities than the latter. Normally, real numbers are expected to have a Gaussian distribution centered at zero. And functions $\mathcal{D}(...)$ are supposed to comply with that (Knuth, 1998, chapter 3.4.1).

3.2 Objective-function Approximation

For any resolution step, the set $\mathcal{S}$ of possible resolvents is generated incrementally, one resolvent at a time, and no more than $\hat{n}$ resolvents are generated. Subsymbolic representations may not be available for some predicates or Skolem functions. The resolvents containing literals with such predicates or Skolem functions are not included in $\mathcal{S}$. We assume that the implementation of a KB guarantees fast access to the facts that may unify with a given literal. For instance, a hash table could be created for storing facts of every predicate with a large number of facts.

Let $u_1,...,u_w$ be the variables from resolvent $\{R_1,...,R_r\} \in \mathcal{S}$. Note that w is usually small, it is zero or one for many resolvents in derivations of ground literals. Let $\theta(i_1,...,i_w)$ denote substitution $\{u_1/\mathcal{D}(u_1,i_1),...,u_w/\mathcal{D}(u_w,i_w)\}$, where $i_1,...,i_w$ are natural numbers. Every $[R_i\theta(i_1,...,i_w)]$ can be evaluated or approximated.

Algorithm 2 implements resolvent selection from $\mathcal{S}$. This algorithm uses the hash tables $\mathcal{H}$ created by Algorithm 1. The values of $[R_1\theta(i_1,...,i_w)]$,...,$[R_r\theta(i_1,...,i_w)]$ are evaluated or approximated for given values of $i_1,...,i_w$ in every iteration. If all of them are greater than t, then this resolvent is selected and the algorithm terminates. Otherwise, the next iteration for other values of $i_1,...,i_w$ starts. If Algorithm 2 yields *nil*, then an arbitrary resolvent is picked. The same constant $\overline{n}$ limits the values of $i_1,...,i_w$. We say that ground literal $P(c_1,...,c_n)$ is outer if it contains such argument c_i that $c_i = \mathcal{D}(P_i,k)$ and $k > \overline{n}$. Constant $\underline{n}$ limits the number of iterations over arguments of outer literals, $\underline{n} < \overline{n}$.

This resolvent selection procedure avoids NN calculations when possible. These are relatively fast compared to derivations but picking up values from a hash table is much faster than NN calculations. Initially, outer literals do not occur as keys in $\mathcal{H}$ tables. The approximated truth value of any ground literal is calculated once during a derivation, and then it is added to the respective hash table. These new hash table entities are not reused in other derivations.

Since all modern programming environments

support multi-threading, the approximation of $[R_1\theta(i_1,...,i_w),...,R_r\theta(i_1,...,i_w)]$ can be done in parallel for all resolvents from $\mathcal{S}$ provided that operations on hash tables are synchronized. For simplicity, a serial version of Algorithm 2 is presented here. A parallel version could work significantly faster.

Algorithm 2.

for $\{R_1,...,R_r\} \in \mathcal{S}$ **do**
  $n_1 := min\{\overline{n}, |\mathcal{D}(u_1)|\}$; ...
  $n_w := min\{\overline{n}, |\mathcal{D}(u_w)|\}$;
  for $i_1 := 1...n_1;...;i_w := 1...n_w$ **do**
    $h := -1$;
    for $j = 1...r$ **do**
      $h := \mathcal{H}_{R_j}.get(R_j\theta(i_1,...,i_w))$;
      if $h = -1 \wedge outer(R_j\theta(i_1,...,i_w)) \wedge$
          $i_1 \leq \underline{n} \wedge ... \wedge i_w \leq \underline{n}$ **then**
        $h := [R_j\theta(i_1,...,i_w)]$;
        $\mathcal{H}_{R_j} := \mathcal{H}_{R_j} \cup R_j\theta(i_1,...,i_w) \rightarrow h$;
      end if
      if $h < t$ **then break**; **end if**
    end for
    if $h \geq t$ **then return** $\{R_1,...,R_r\}$; **end if**
  end for
end for
return *nil*;

In the worst case, the time of retrieving elements of $\mathcal{S}$ is proportional to KB size $\overline{b}$. Suppose $\overline{a}$ is the maximum time of the neural approximation of the truth value of a ground atom, and $\overline{r}$ is the maximum length of a resolvent. If $\overline{w}$ is the maximum number of variables in a resolvent from $\mathcal{S}$, then the total number of executions of the innermost cycle of Algorithm 2 is no more than $\hat{n}\overline{r}\overline{n}^{\overline{w}}$, and at most $\hat{n}\overline{r}\underline{n}^{\overline{w}}$ of them involve NN approximations. Hence, the time complexity of Algorithm 2 is $O(\max\{\hat{n}\overline{r}\overline{n}^{\overline{w}}, \overline{a}\hat{n}\overline{r}\underline{n}^{\overline{w}}, \overline{b}\})$. This estimate is based on the assumption that a single operation on a hash table takes a constant time.

Objective-function approximation is less expensive for such resolvents that one of the two resolved literals is a fact because these resolvents usually have fewer variables than other possible resolvents for the same resolution step. Given that, it may be worth limiting $\mathcal{S}$ to resolvents derived from facts for some KBs.

3.3 Examples

The base strategy in these examples is such depth-first search that literals are resolved according to their order in rule bodies, and preference is given to facts and then rules with a shorter length.

Transitive Closure. Suppose binary extensional predicate E specifies edges of a directed graph. Our task is to find such paths connecting pairs of nodes in the graph that all interim nodes on these paths have property P. Graph path search algorithms can be adopted to tackle this task, but the ultimate goal of automatic programming is to have a declarative specification of this problem and to expect that solutions are derived without coding an algorithm. The following two rules define predicate T as the transitive closure of E with an additional constraint given by P. Derivations of goals $T(a,b)$, where a and b are nodes, give desired paths. Arguably, it is the simplest possible specification of the task. This specification is applicable to infinite graphs too.

$T(x,y) \Leftarrow E(x,y)$
$T(x,y) \Leftarrow E(x,z) \wedge P(z) \wedge T(z,y)$

Initially, the first rule is selected for goal $T(a,b)$ by the base strategy. If $E(a,b)$ is not a fact, then the search backtracks and the second rule is applied to this goal. For goal list $(E(a,z),P(c),T(z,b))$, the base strategy picks up the first literal. Algorithm 2 should select such fact $E(a,c)$ that c is on a desired path because both $[P(c)]$ and $[T(c,b)]$ are expected to be close to one if the neural approximations of the truth values of P and T are accurate. If $P(c)$ derivation does not involve backtracking, then all backtracking relates to the first rule, and it is induced by the base strategy. In comparison to our best-first strategy, the search strategy based on rule weight would not help select better facts and thus would not reduce backtracking.

Minimization Operator. The following rules specify predicates M, N, and P. Q, R, and T are extensional predicates satisfying closed world assumption, i.e. if atom A is not present in a KB, then $\neg A$ holds.

$M(x,1) \Leftarrow P(x,1) \wedge Q(1)$
$M(x,y) \Leftarrow \neg(y=1) \wedge P(x,y) \wedge Q(y) \wedge N(x,y-1)$
$N(x,1) \Leftarrow \neg P(x,1)$
$N(x,1) \Leftarrow \neg Q(1)$
$N(x,y) \Leftarrow \neg(y=1) \wedge \neg P(x,y) \wedge N(x,y-1)$
$N(x,y) \Leftarrow \neg(y=1) \wedge \neg Q(y) \wedge N(x,y-1)$
$P(x,y) \Leftarrow T(x) \wedge R(f(x),y)$
$\neg P(x,y) \Leftarrow S(x,y)$
$\neg P(x,y) \Leftarrow \neg R(f(x),y)$

Derivation of $M(a,z)$, where a is a constant, binds variable z with the minimum natural number c satisfying both $P(a,c)$ and $Q(c)$. Suppose $T(a)$, $R(f(a),100)$, and $Q(100)$ hold whereas $S(a,i)$ is not derivable and at least one of $R(f(a),i)$, $Q(i)$ does not hold for every $i = 1...99$. If the NN representation of N is accurate, then $[N(a,n)]$ is close to zero for $n \geq 100$ whereas $[N(a,1)],...,[N(a,99)]$ are close to one. After unsuccessfully exercising the first M rule, the second M rule is applied, and then the first P rule is applied, and $T(a)$ is resolved.

$R(f(a),z)$ is resolved with fact $R(f(a),100)$ because 100 is the only value of z for which $[R(f(a),z)] = 1$, $[Q(z)] = 1$, $[\neg(z = 1)] = 1$, and $[N(a,z-1)]$ is close to one. After that, $Q(100)$ is resolved with a fact, and $\neg(100 = 1)$ is evaluated. Derivation of $N(a,99)$ is not expected to involve backtracking. The third N rule is selected by Algorithm 2 for such $i \in \{1,...,99\}$ that $\neg P(a,i)$ is derivable. The fourth N rule is selected for the rest. Due to the best-first strategy, the second P rule is avoided if the NN representation of S is accurate. Depending on the rules related to S, the second P rule could be a source of extensive backtracking while attempting to derive $\neg P(a,i)$ for multiple values of i. The search strategy based on rule weight would be counterproductive in this example. It would always select the second P rule.

4 RELATED WORK

Paper (d'Avila Garcez et al., 2019) is a survey of research in the area of neural-symbolic computing. The aforementioned works (Rocktäschel, 2017; Serafini and d'Avila Garcez, 2016; Dong et al., 2019; Van Krieken et al., 2019) use rules or FOL formulas to train NNs based on subsymbolic predicate representations. These NNs are used to approximate the truth values of ground formulas. The methods from these papers do not incorporate inference. Function-free rules with unary and binary predicates are used for training NNs in (Rocktäschel, 2017). Neural Logic Machines (Dong et al., 2019) are capable of representing non-cyclic Horn clauses over finite object domains. Differentiable Reasoning (Van Krieken et al., 2019) employs function-free FOL formulas. Logic Tensor Networks (Serafini and d'Avila Garcez, 2016) accept all FOL formulas.

A simple hybrid algorithm combining logical reasoning with the approximation of truth values is suggested in (Sakharov, 2019). This algorithm searches for partial derivations. Once such ground literal occurs in a derivation that its approximated truth value is close to 1, this literal is considered a fact not requiring further derivation. This hybrid algorithm may yield no derivation at all, its results may not be explainable. The reliability of the results is dependent of the accuracy of the approximation of the truth values of ground atoms.

Differentiable reasoning for the function-free fragment of FOL is suggested in (Minervini et al., 2020). Unification is done by finding the nearest neighbors in the Euclidean space of predicate embeddings. An approximate proof is constructed by solving an optimization problem. In contrast to symbolic reasoning, differentiable reasoning may not be reliable but it is appropriate for areas with imprecise knowledge such as natural language. Extending the method of (Minervini et al., 2020) onto rules containing atoms with functions is problematic due to unification.

In other research on the neural representation of knowledge, individual predicates are not directly represented via NNs but are rather blended into a graph NN that represents multiple predicates along with rules (Zhang et al., 2020). Nonetheless, this graph NN can be used to approximate the truth values of ground literals. Potential advantages of graph NNs for the neural representation of knowledge are advocated in (Lamb et al., 2020). In DeepLogic (Cingillioglu and Russo, 2019), tensor NNs represent function-free normal logic programs. Combined with embeddings of ground atoms, these NNs are used to approximate the truth values of ground atoms.

Deep logic models proposed in (Marra et al., 2019) are two-layer NNs. The first layer represents predicates, and its output is approximated truth values of atoms. The second layer represents constraints on the outputs of the first layer. These constraints are expressed by FOL formulas over predicates of the first layer.

DeepProbLog (Manhaeve et al., 2018) is a hybrid system that combines probabilistic logic programming with NNs. The latter are used to calculate probabilities associated with ground atoms in disjunctions annotated with probabilities. These annotated disjunctions are transformed into probabilistic logic programs. Exact inference for these programs is intractable, and approximate inference is more complicated than FOL inference (Kimmig et al., 2010). TensorLog (Cohen et al., 2020) is another system integrating probabilistic logical reasoning with NNs. TensorLog programs are function-free, they contain unary or binary predicates, and probabilities are associated with facts only.

The resolution calculus can be reformulated for non-Skolemized FOL formulas. For this purpose, formulas $\exists y(A_1 \vee ... \vee A_k)$ are considered in addition to disjunctions of literals and a new inference rule is introduced (Mints, 1993). Resolution with the three rules is complete for FOL. Resolvents do not contain existential quantifiers, they are non-Horn clauses without Skolem functions. In relation to our best-first search strategy, this approach eliminates inaccuracy introduced by the subsymbolic approximation of Skolem functions. However, the applicability of backward chaining and search strategies to this variant of resolution has not been investigated, and thus,

this variant is not ready for practical use yet.

5 CONCLUSION

Our method is neural-symbolic but its results are derivations, and they are precise and explainable. If some neural predicate representations are not reliable, this has effect on search speed only. Our best-first strategy is expected to work best for derivations of goals that are sets of ground literals. Fortunately, the primary purpose of KBs and logic programs unlike theorem provers is to derive facts that are expressible by ground literals. As the examples have shown, our best-first strategy may dramatically reduce backtracking for typical derivation tasks. Nonetheless, further experiments are necessary in order to get quantitative results for benchmark KBs.

The effect of our strategy on the speed of inference depends on the accuracy of neural predicate representations. As of the time of writing this paper, evaluations of the accuracy of NN representations of predicates are sparse. Algorithm 2 is presented in a general form. It can be easily adjusted to various knowledge representation formats including those deviating from FOL.

Our best-first search strategy can be applied to any resolution method, not just to backward chaining, but Algorithm 2 will not give positive results for resolvents containing literals whose negations are not derivable. This strategy could be extended onto FOL with equality. In this case, the selection of resolvents should be additionally done for paramodulation or other other rules enabling derivations in FOL with equality (Chang and Lee, 1973). Our objective function can be directly applied to the paramodulation rule.

Any heuristic search strategy would be especially beneficial for inductive theories because the induction rule is the source of infinite branching (Hutter, 1997) but adaptation of the objective function to the induction rule is problematic. Nonetheless, our objective function can used for the induction rule limited to literals (Sakharov, 2020). If the selection does not yield a positive result, then the induction rule should be abandoned for the respective derivation step.

REFERENCES

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic logic and mechanical theorem proving*. Academic press.

Cingillioglu, N. and Russo, A. (2019). DeepLogic: Towards end-to-end differentiable logical reasoning. In *AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering*, volume 2350. CEUR-WS.org.

Cohen, W., Yang, F., and Mazaitis, K. R. (2020). Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *Journal of Artificial Intelligence Research*, 67:285–325.

d'Avila Garcez, A. S., Gori, M., Lamb, L. C., Serafini, L., Spranger, M., and Tran, S. N. (2019). Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4):611–632.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. (2019). Neural logic machines. In *International Conference on Learning Representations*.

Hutter, D. (1997). Coloring terms to control equational reasoning. *Journal of Automated Reasoning*, 18(3):399–442.

Kimmig, A., Demoen, B., De Raedt, L., Costa, V. S., and Rocha, R. (2010). On the implementation of the probabilistic logic programming language ProbLog. *arXiv preprint arXiv:1006.4442*.

Knuth, D. E. (1998). *Art of Programming, Volume 2: Seminumerical Algorithms*, volume 2. Addison-Wesley.

Lamb, L. C., d'Avila Garcez, A. S., Gori, M., Prates, M. O. R., Avelar, P. H. C., and Vardi, M. Y. (2020). Graph neural networks meet neural-symbolic computing: A survey and perspective. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4877–4884.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). DeepProbLog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759.

Marcus, G. (2020). The next decade in AI: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*.

Marra, G., Giannini, F., Diligenti, M., and Gori, M. (2019). Integrating learning and reasoning with deep logic models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 517–532.

McCune, W. (2003). Otter 3.3 reference manual and guide. Technical report, Argonne National Lab.

Minervini, P., Bosnjak, M., Rocktäschel, T., Riedel, S., and Grefenstette, E. (2020). Differentiable reasoning on large knowledge bases and natural language. In *Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges*, pages 125–142. IOS Press.

Mints, G. (1993). Gentzen-type systems and resolution rule. II. Predicate logic. In *Logic Colloquium'90: ASL Summer Meeting in Helsinki*, pages 163–190. Association for Symbolic Logic.

Nie, X. (1997). Non-Horn clause logic programming. *Artificial Intelligence*, 92(1):243 – 258.

Rocktäschel, T. (2017). Combining representation learning with logic for language processing. *CoRR*, abs/1712.09687.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Sakharov, A. (2019). Hierarchical rules for knowledge representation and learning. In *IEEE 2nd International Conference on Artificial Intelligence and Knowledge Engineering*, pages 167–171.

Sakharov, A. (2020). Hierarchical resolution for structured predicate definitions. In *11th Hellenic Conference on Artificial Intelligence*, pages 202–210.

Serafini, L. and d'Avila Garcez, A. S. (2016). Logic tensor networks: Deep learning and logical reasoning from data and knowledge. In *11th International Workshop on Neural-Symbolic Learning and Reasoning*, volume 1768. CEUR-WS.org.

Shen, Y.-D., Yuan, L.-Y., and You, J.-H. (2001). Loop checks for logic programs with functions. *Theoretical Computer Science*, 266(1-2):441–461.

Stickel, M. E. (1992). A Prolog technology theorem prover: a new exposition and implementation in Prolog. *Theoretical Computer Science*, 104(1):109–128.

Swift, T. (2009). Design patterns for tabled logic programming. In *International Conference on Applications of Declarative Programming and Knowledge Management*, pages 1–19. Springer.

Van Krieken, E., Acar, E., and Van Harmelen, F. (2019). Semi-supervised learning using differentiable reasoning. *Journal of Applied Logics*, 6(4):633–651.

Zhang, Y., Chen, X., Yang, Y., Ramamurthy, A., Li, B., Qi, Y., and Song, L. (2020). Efficient probabilistic logic reasoning with graph neural networks. In *International Conference on Learning Representations*.