Online Learning of Logic Based Neural Network Structures

Victor Guimarães*[0000−0002−2851−5618] and
Vítor Santos Costa[0000−0002−3344−8237]

CRACS and DCC/FCUP
Universidade do Porto
Porto, Portugal
`victorguimaraes13@gmail.com, vsc@dcc.fc.up.pt`

Abstract. In this paper, we present two online structure learning algorithms for NeuralLog, NeuralLog+OSLR and NeuralLog+OMIL. NeuralLog is a system that compiles first-order logic programs into neural networks. Both learning algorithms are based on Online Structure Learner by Revision (OSLR). NeuralLog+OSLR is a port of OSLR to use NeuralLog as inference engine; while NeuralLog+OMIL uses the underlying mechanism from OSLR, but with a revision operator based on Meta-Interpretive Learning. We compared both systems with OSLR and RDN-Boost on link prediction in three different datasets: Cora, UMLS and UWCSE. Our experiments showed that NeuralLog+OMIL outperforms both the compared systems on three of the four target relations from the Cora dataset and in the UMLS dataset, while both NeuralLog+OSLR and NeuralLog+OMIL outperform OSLR and RDN-Boost on the UWCSE, assuming a good initial theory is provided.

Keywords: Online Learning · Inductive Logic Programming · Meta-Interpretive Learning · Neural Network · Neural-Symbolic Learning and Reasoning · Theory Revision from Examples.

1 Introduction

Neural networks have achieved a great success on a wide range of tasks [14]. However, traditional neural network models cannot take advantage of background knowledge, which may contain additional information about the examples, as well as expert knowledge. On the other hand, Inductive Logic Programming (ILP) [17] is a field of study that tries to learn first-order logic theories in order to describe a set of examples given background knowledge [17], but ILP struggles to deal with numeric features and uncertainty and noise; which are inherent characteristics of real world applications.

The field of Neural-Symbolic Learning and Reasoning tries to combine the strengths of both neural networks and logic systems, in order to obtain models

* Financed by the Portuguese funding agency, FCT — Fundação para a Ciência e a Tecnologia through Ph.D. scholarships 2020.05718.BD.

that are both capable of dealing with numeric features, uncertainty and noise and can also take advantage of existing background knowledge [6].

NeuralLog is a system developed to transform a first-order logic program into a neural network. It receives as input a set of first-order clauses that are used to define the neural network model, and a set of facts that becomes weights in the neural network. Then, those weights are fine-tuned, given a set of examples [9].

Online Structure Learner by Revision (OSLR) is a theory revision algorithm that revises the logic theory to cope with the arrival of new examples [8, 7]. OSLR uses a tree structure representation of the logic theory, and applies revision operators to this structure in order to improve the theory to the new examples.

In this paper, we propose two structure learning algorithms for NeuralLog: NeuralLog+OSLR, which is a ported version of the OSLR algorithm to use NeuralLog as inference engine; and NeuralLog+OMIL, which uses the same underlying theory revision mechanism used by NeuralLog+OSLR (and OSLR), but applies a new revision operator, based on the Meta-Interpretive Learning (MIL) system Metagol [18].

Metagol is a MIL system that have recently been shown to achieve good performance when learning logic theories from examples [18]. MIL systems use a higher-order logic theory in order to define the hypotheses space and the searching mechanism of first-order theories from examples. To the best of our knowledge, it is the first time MIL is applied to learn first-order logic theories online, where new examples arrive over time.

We compared our approach with the original OSLR [7] and RDN-Boost [12] for link prediction tasks in three different datasets: the Cora [21] and the UWCSE [24] datasets, which were also used on [7]; and the UMLS [13] dataset. Our experiments show that NeuralLog+OMIL outperforms OSLR and RDN-Boost on three of the four target relations from the Cora dataset and in the UMLS dataset. While NeuralLog+OSLR and NeuralLog+OMIL outperform OSLR and RDN-Boost on the UWCSE, assuming a good initial theory is provided.

The remainder of the paper is as follows: in Section 2, we briefly give the background knowledge in order to understand this work; in Section 3, we present our two structure learning algorithms for online theory revision for NeuralLog; in Section 4, we present the performed experiments and obtained results; in Section 5, we present the works related to ours; and we conclude and propose directions for future work in Section 6.

2 Background Knowledge

In this work, a first-order logic program is a set of Horn clauses [11]. A Horn clause has the form of $b(.) \leftarrow p_1(.) \wedge \cdots \wedge p_n(.)$. where $b(.)$ is called the head of the clause and the set of p_i represents a conjunction and is called the body of the clause. The terms between parentheses can be either constants, represented by a string starting with a lowercase letter; or variables, represented by a string starting with an upper case letter. b and p_i are predicate names, and the predicate name followed by its terms is called an atom. A literal is either an atom or the negation of an atom. First-order logic functions and negation are not considered in this work, although we will refer to the atoms in the body of a clause as literals, to distinguish them from the atom of the head. A fact is a Horn clause with empty body, where all terms are constant.

NeuralLog treats facts of a logic program as numeric matrices, and the rule inference is performed as algebraic operations in those matrices [9]. In addition, each NeuralLog fact has an associated weight that influences the final result of the rule inference. Those weights can be learned by the neural network, in order to best fit a set of examples.

Inductive Logic Programming (ILP) is a subfield of machine learning which is concerned with finding logic theories to describe a set of examples, given background knowledge [25]. Meta-Interpretive Learning (MIL) uses a higher-order logic theory, in order to define the hypotheses space of possible first-order logic theories and to learn a first-order theory from the examples [18]. In a higher-order logic, the predicate names in the rules can be variables, and the logic inference system should find the substitution of the name in order to prove the rule. Metagol [18] is a MIL system which uses a modified Prolog meta-interpreter that resolves a higher-order theory similarly to a first-order SLD-Algorithm [29].

Another approach to learn first-order logic theories is by revising an initial (partially correct) logic theory, in order to adapt it to new examples. This approach is called *theory revision from examples* [3, 26]. This characteristic of starting from a (possibly empty) initial theory in order to adapt it to new examples makes theory revision a suitable candidate to be applied to online environment, where new examples are arriving over time.

3 Online Theory Revision with NeuralLog

In this section, we present the two structure learning algorithms proposed in this work. We start by presenting NeuralLog+OSLR, our implementation of OSLR. Then, we present NeuralLog+OMIL, an online implementation of MIL.

3.1 Online Structure Learner by Revision

Online Structure Learner by Revision (OSLR) [7] is a system developed to learn logic theories in an online fashion, originally based on ProPPR [28]. It relies on theory revision techniques in order to adapt an existing theory to cope with new arriving examples. In this subsection, we present NeuralLog+OSLR, our implementation of OSLR that learns theories in the NeuralLog language.

The top-level revision algorithm implemented by OSLR is as follows: when new examples arrive, they are placed into a tree structure that represents the logic theory; then, a revision is proposed to the point of the theory that has the biggest potential to bring a gain for the theory; after, the revision is evaluated against an accepting criterion; finally, the revision is either accepted or rejected and the algorithm evaluates the next revision, until no more revision points are changed from the current examples, when the algorithm waits for new examples.

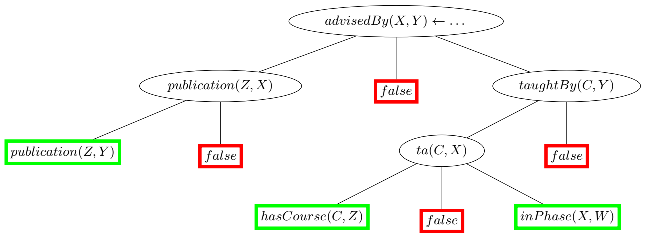


Fig. 1: Tree Structure Representation of the UWCSE Theory Example

Table 1: Theory Example for the UWCSE Dataset

$advisedBy(X,Y) \leftarrow taughtBy(C,Y) \wedge ta(C,X) \wedge hasCourse(C,Z).$
$advisedBy(X,Y) \leftarrow taughtBy(C,Y) \wedge ta(C,X) \wedge inPhase(X,W).$
$advisedBy(X,Y) \leftarrow publication(Z,X) \wedge publication(Z,Y).$

NeuralLog+OSLR follows the OSLR top-level algorithm with minimal changes, in order to make it better suited for neural networks. In the remainder of this subsection, we summarize the OSLR algorithm in order to keep this work self-contained. In addition, we point out the differences between NeuralLog+OSLR and the original OSLR. However, we refer the reader to [8, 7], for a more detailed explanation of OSLR.

Data Representation Online Structure Learner by Revision (OSLR) starts by constructing a tree representation of a, possible empty, theory for each target predicate. This tree structure represents all the rules in the theory, whose head predicate is equal to the target predicate. The root of the tree represents the head of the rules for the target predicate, while the other nodes represent the literals in the body of the rules. The level immediately after the head represents the literals in the first position in the body of the rules, and there will be a node for each different literal in the first position. The next level represents the literals in the second position and so on. For each internal (non-leaf) node in the tree, a default *false* node is appended as child. The *false* nodes are always leaves.

This tree structure plays two roles: identifying the revision points on the theory; and storing the examples that shall be used to revise these points. Figure 1 shows an example of the tree representation of the theory shown in Table 1.

Each path from the root of the tree to a non-false leaf represents a rule in the theory. Rules that are a subset of another rule are not considered. The leaves represent the possible revision points and are shown as squares in Image 1.

When a new example arrives, it is passed through the tree to decide where it will be placed. The example starts in the root, then it is recursively passed through the nodes in the tree as described: for each node u in the children nodes of the current node v, if the (partial) rule from the root to u proves the example, we pass the example down to u. If the example is not proved by any of the children nodes of v, it is placed in the false node connected to v. This process repeats until the example reaches the leaves of the tree. If the example is proved by more than one child node, it goes to all the nodes that prove it.

Theory Revision After placing the arrived examples in the correspondent leaves, all the leaves that received examples are candidates to be revised. OSLR uses a heuristic function in order to sort the revision points to prioritize the ones that may have the bigger impact in the evaluation of the theory. This heuristic is simply the number of misclassified examples in the leaf. The number of misclassified examples is the number of positive examples, in the case of a false leaf; or the number of negative examples, in the case of a non-false leaf.

In order to propose the revision of the theory, it uses revision operators that are applied to the revision points. There are two possible operations to revise the tree: adding new nodes to the tree; or removing nodes from the tree. OSLR applies all the possible operators to each revision point and uses the examples contained in the point in order to evaluate the revision on the theory.

Adding Node. It can be applied to both false and non-false leaves. When applied to a false leaf, the new nodes are used in order to generate a new rule that starts from the root until the parent of the false leaf. This new rule will be added to the theory in order to make the theory more generic and is an attempt to prove positive examples that fell in the false leaf. On the other hand, adding node to a non-false leaf extends the path from the root to the new leaf, thus, extending the rule and making the theory more specific, which is an attempt to avoid proving false examples in the non-false leaf. There are two algorithms to select the nodes to be added, both relying on the concept of the bottom clause [15]: the hill-climbing algorithm, which tries to add a candidate literal at a time, until certain stop criteria is met; and the relational path-finding algorithm [22], which tries to find a path between the variables of the example.

Deleting Node. It can be applied to non-false leaves or to false leaves whose parent has a single non-false child and this non-false child is also a leaf; this approach is described as *Alternative 1* in [8]. When applied to a false leaf, it deletes the literal represented by its sibling node, in an attempt to make the theory more generic, in order to prove the positive examples in the false leaf. When applied to a non-false leaf, it deletes the rule represented by the leaf, in an attempt to make the theory more specific, avoiding proving negative examples.

After applying all the possible revision operators to a given revision point, the operator that better improves the performance of the theory, given the examples in the revision point, is selected to be evaluated against the acceptance criteria, which will be described below.

The adding node operator is actually two distinct revision operators, one for the hill-climbing and another for the relational path-finding. Thus, alongside the deletion operator, they all compete among each other and the one that achieves the best result for the revision point is selected.

Accepting the Revision Once the operator that has the biggest metric in a given revision point is selected, OSLR uses a threshold to decide if the improvement of the revised theory over the current theory is significant.

This threshold is based on the Hoeffding's bound [10] and is given by $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$ where R is the size of the range of the given metric, n is the number of examples used in the evaluation and δ is a parameter defined by the user. An improvement larger than ϵ means that the probability of the revised theory to be actually better than the current theory is $1 - \delta$.

The examples used to evaluate the theories are the ones in the revision points. If the improvement of the revised theory over the current one is greater than ϵ, the revision is accepted and the examples used to evaluate it are discarded. Otherwise, the revision is discarded and the examples are unchanged. After either case, the algorithm continues to try to revise the remaining revision points, if there are any revision points left to be revised.

Clause Modifiers In OSLR, after a revision is accepted, a feature, in the ProPPR language [28], is generated for the rule that was modified by the revision. This is the point where our implementation differs the most from the original OSLR. NeuralLog language does not support the ProPPR features, although, in some cases, they might be similar to the addition of a literal to the rule whose weight should be learned by the neural network. As such, instead of computing the features in the same way OSLR does, which would select a subset of terms in the rule to have associated weights to be learned, we create a unique weight for each rule, which is learned by the neural network and is independent of the instantiation of the terms in the rule. Our experiments using OSLR showed that the difference between this approach and the original one is minimal.

The addition of the weight is done by a clause modifier, which appends the weight to the body of the revised rules. These weights are represented in the form of a literal, with a unique constant for each rule; and this literal is marked as learnable in the NeuralLog language.

In addition to appending a literal to the rule with a unique constant, we have two more clause modifiers that are useful for the neural network construction. The first one is a clause modifier that appends a literal to the rule with a term from the head of the rule. This modifier is used to append an activation function for the rule, whose term must be the last variable in the head of the rule.

The other modifier changes the predicate name of the head of the rule to another name, by appending a suffix at the end of the name, this modifier is useful to learn a set of rules that indirectly proves the examples. For instance, suppose one wants to learn examples from the predicate $p/2$ without changing rules that have the $p/2$ predicate in the head. One could add the rule $p(X, Y) \leftarrow p_1(X, Y)$. to the background knowledge and use a clause modifier to change the head of the rules, learned from the examples, from $p/2$ to $p_1/2$. This is specially useful in the definition of neural networks, because it allows us to isolate the learned part of the theory and to add neural network components around it. For instance, the addition of biases and output functions for the target predicates.

3.2 Online Meta-Interpretive Learning

In this work, we propose a novel approach by combining the MIL hypotheses search strategy with the online learning mechanism from OSLR. We call this **O**nline **M**eta-**I**nterpretive **L**earning approach as NeuralLog+OMIL.

To achieve such a goal, we created a new revision operator based on the MIL search strategy used by Metagol, which proposes revisions to nodes in the OSLR tree representation. Then, we use the same NeuralLog+OSLR machinery, replacing the three original revision operators by our MIL revision operator.

In order to reuse NeuralLog+OSLR machinery we have to adapt the MIL algorithm to work with the OSLR tree structure. In Metagol, a higher-order clause, such as $P(X, Y) \leftarrow Q(X, Z) \wedge R(Z, Y)$, is directly applied to the target relation and the resolution system searches for substitutions to the higher-order variables (representing predicate names) which proves the positive examples.

Instead of applying the higher-order clauses to the input examples, NeuralLog+OMIL creates a target atom to be proved by the higher-order program, based on the revision point to which the MIL operator is applied.

In order to propose the revision of the theory, we apply each higher-order clause to solve the target atom generated by the revision point. Then, each higher-order variable of this clause is replaced by a valid predicate from the background knowledge. For each possible first-order rule, the revision point is replaced by the body of the rule and the theory is evaluated against the examples of the revision point. If the operator is applied to the false leaf, it uses the new atoms to create a new rule whose body starts with the path from the root to the parent of the false leaf node in the tree. Finally, if the first-order theory evaluation is greater than the Hoeffding's bound threshold, the algorithm returns it, otherwise, it continues to the next first-order theory, if any.

In order to give the OMIL operator more information to propose the revision, we create a different target atom, depending on the revision point.

Root node. When the operator is applied to the false leaf of the root node, we use the predicate of the example as target atom; in this case, the operator tries to learn a rule to directly predict the examples.

Propositional literal. When the literal has no variables, the target atom would have a special predicate, only used inside this operator, with the same variables as the head of the rule. This will inform the operator about the input and output variables of the rule.

Literal connected to the output. When the target literal is connected to the output, it will be the target atom. In this way, the same terms of the target literal will be used by the operator.

Literal disconnected from the output. When the literal is not connected to the output, the target atom will have the special predicate name, and the variables will be the input variable of the disconnected literal and the output variable of the rule. As such, the operator will be able to find a body which connects the input of the literal to the output of the rule, closing this path.

These modifications to the target atom gives the OMIL operator enough information to find meaningful rules: (1) having the information about the input and output variables of the rule, in the first two cases; (2) the information about the final part of a path in the rule, in the third case; and (3) the information about an open path and the output variable of the rule, in the last case. The use of this information will depend on the higher-order theory. However, it allows the user to define meta-rules that might: (1) create new paths, (2) replace the final part of an existing path, or (3) close an existing open path; respectively.

In order to restore the deletion behaviour from OSLR, we added two special rules to the higher-order theory: $P(X, Y) \leftarrow true.$ and $P(X, Y) \leftarrow false.$

The *true* literal is a special literal that is always true. Since the body of a rule is a conjunction (logic **AND**) of literals, the addition of a literal that is always true does not change the result of the conjunction, as such it can be removed from the rule. When the true rule is applied to a literal, the literal is replaced by the true literal, that is not added to the rule, since it will have no effect on it, thus, the application of this rule represents a literal removal. On the other hand, the $false$ literal is a special literal that is always false. As such, a rule whose body includes a $false$ literal will always fail to be proved, and can be removed from the theory. Thus, the application of the false rule to append a literal in the body of a rule will result in the deletion of the rule from the theory.

Applying any of these rules to a false leaf would produce no effect. For the true rule, it would generate a rule whose body is a subset of another rule, which is not allowed by the OSLR algorithm. For the false rule, it would result in an attempt of creating a new rule with the $false$ literal in its body, which would be excluded and the theory would remain unchanged.

After the proposal of the modification of the theory by the operator, the clause modifiers are applied to the modified rule as usual, which will be the rule formed by the existing tree and the literal in the body of the clause generated by the higher-order theory.

4 Experiments

In order to show the capabilities of NeuralLog+OSLR and NeuralLog+OMIL, we compared them with OSLR in online learning of logic theories, for link prediction, in three distinct datasets: the Cora dataset, which is a citation matching dataset [21]; the Unified Medical Language System (UMLS), which is a medical dataset [13]; and the UWCSE dataset, which describes relations between professors and students in the University of Washington [24]. In addition, we also include the comparison with RDN-Boost [12], a system that learn Relational Dependency Networks (RDNs) [19], which was also presented in [8, 7].

Link prediction is the task to find the entities related to another entity, given a relation. In this case, given a query $?-p(a, X)$, we would like to find the substitutions of X that match the positive examples, without matching the negative ones. We call p the target relation, a the input entity and the substitution of X are the output entities.

4.1 Datasets

Cora. The Cora dataset contains four target relations, *Same Author*, *Same Bib*, *Same Title* and *Same Venue*. We ran each of these relations separately.

UMLS. It is a dataset between biomedical entities. We selected the *Affects* as target relation, since it is the most frequent relation in this dataset, as it is done in [27]. Since it has no negative examples, we generated approximately 2 negative examples for each positive example, by selecting a random output entity that appears in the target relation, but is not related to the input entity, following the Local Closed World Assumption (LCWA) [5].

UWCSE. This dataset has one target relation *Advised by*, that relates the students with supervisors. The UWCSE dataset also contains ternary facts, which are not supported by NeuralLog. Thus, we converted them to binary, by concatenating two terms that always appear together in the theory. We also added two additional predicates to extract either term, given the concatenated form.

Table 2 shows the statistics of the datasets, for a total of 6 target relations. Since the number of negative examples in the UWCSE is much bigger than the positive ones, we downsample the set of negative examples to be twice as much as the number of positives ones, as it is done in [12].

Table 2: Size of the Datasets

Relation	# Positives	# Negatives
Same Author	488	66
Same Bib	30,971	21,952
Same Title	661	714
Same Venue	2,887	4,976
Affects	1,022	-
Advised by	113	16,601

4.2 Simulating the Online Environment

In order to properly evaluate the online systems, we use these datasets to simulate an online environment by reproducing the procedure used in [8, 7]. We split each target relation into $N + 1$ iterations, where iteration 0 has only the background knowledge and each of the following iterations has approximately $|E|/N$ examples, where $|E|$ is the total number of examples of each relation.

We pass each iteration, in order, to the system. When an iteration arrives, the current model is tested with it, before training. Then, the system trains on this iteration and the evaluation of each iteration is recorded. This evaluation procedure is known as *Prequential* [2]. Since RDN-Boost is designed for batch learning, we transformed each iteration of the online learning environment into a batch learning task by appending all the examples from the previous iteration to it. Then, we applied RDN-Boost to each of those tasks.

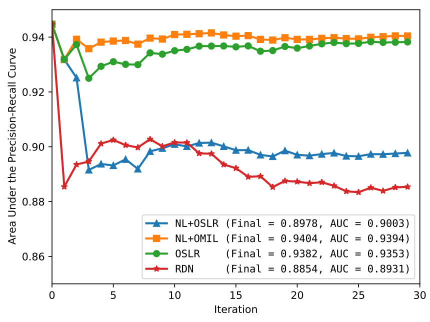
Following the procedure from [8, 7], we set the number N of iterations to approximately 30 for all the target relations except the *Advised by* which was set to 91. We ran each experiment 30 times and reported the average of the area under the Precision-Recall curve as the evaluation metric. The Hoeffding's bound δ parameter was set to 10^{-3} and updated to half its value, each time a revision was accepted. OSLR only considers the number of unrelated examples to compute the Hoeffding's bound, however, we had to relax this restriction for the UMLS dataset, given the reduced number of unrelated examples.

NeuralLog systems add a weight for each rule, as well as an activation function. Also, a bias is added to the output of the target relation, which then passes through an output function. The weights and biases are parameters to be learned by the neural network. After each accepted revision, the neural network adjusts its parameters on the same set of examples used by the revision. It uses the adagrad optimization algorithm to reduce the mean squared error with L2-regularisation for 10 epochs, with a learning rate of 0.01 and the l2 $\lambda = 0.01$. For OSLR, we replaced its feature generation by another one that creates a single weight for each rule, in order to be closer related to NeuralLog, although this change did not have a big impact in the final result.

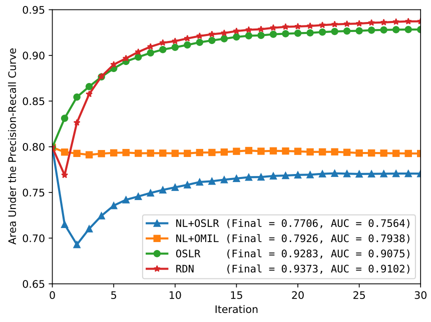
4.3 Results

We now present the results of the experiments. All pairs of systems were compared for statistical significance using two-tailored paired t-test with $p < 0.05$. There is a statistical significance between the pairs, unless stated otherwise.

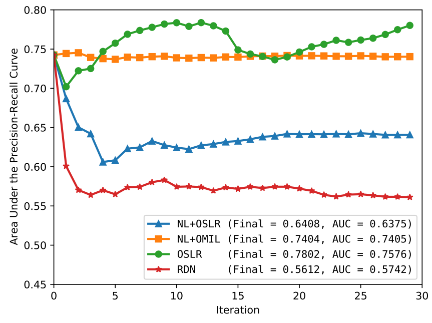
Figure 2 shows the evaluation of the systems in the Cora dataset over the epochs. As can be seen, NeuralLog+OMIL outperforms both OSLR and RDN-Boost over all iterations for the *Same Author* and the *Same Venue* relations, while it underperforms OSLR and RDN-Boost on all iterations on the *Same Bib* relation. For the *Same Title* relation, NeuralLog+OMIL has a stable behaviour over the iterations, while OSLR have some ups and downs. However, OSLR is able to achieve a better result in the final iteration, where all the examples are used, and also has a better overall result, measured by the area under the curve of iterations. NeuralLog+OSLR performed worse than OSLR in all relations of the Cora dataset, but it is able to outperform RDN-Boost in all relation, except for the *Same Bib*. There was no statistical difference between NeuralLog+OMIL and OSLR, and for NeuralLog+OSLR and RDN-Boost for the area under the curve and the final result, in the *Same Author* relation. There were no statistical difference between OSLR and RDN-Boost for the area under the curve and the final result, in the *Same Bib* relation. Finally, there were no statistical difference between NeuralLog+OMIL and OSLR for the area under the curve and the final result, in the *Same Title* relation.



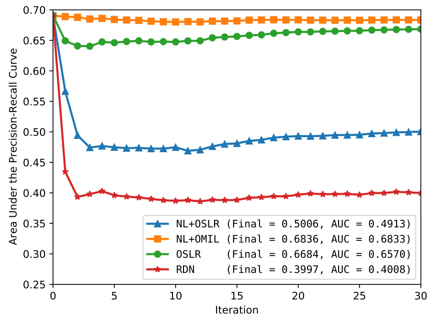
(a) Same Author



(b) Same Bib



(c) Same Title



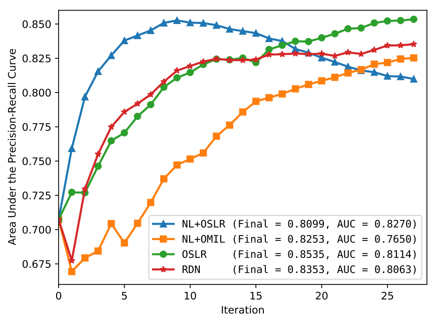
(d) Same Venue

Fig. 2: The Evaluation of the Cora Dataset

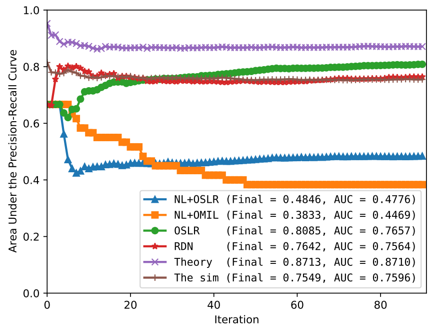
Figure 3 shows the results of the experiments for the UMLS and UWCSE datasets. On Figure 3a, we can see that NeuralLog+OMIL, OSLR and RDN-Boost get better as new examples arrive, ending with an evaluation greater than NeuralLog+OSLR. However, NeuralLog+OSLR performs much better than the other systems on the initial iterations, achieving a better overall evaluation, given the area under the curve. There was no statistical difference between NeuralLog+OSLR and NeuralLog+OMIL nor between NeuralLog+OMIL and RDN-Boost, for the result of the final iteration. Also, there was no statistical difference between the area under the curve between OSLR and RDN-Boost.

In order to evaluate the impact of an initial theory, we performed three experiments with the UWCSE dataset, using a hand-crafted theory provided by Alchemy[1]. Since Alchemy supports a more complex logic language, we removed the rules whose logic feature were not supported by both OSLR and NeuralLog. Then, we used two sets of theory: a complete set, containing more rules, which are more specific; and a simplified version of this theory, containing only some generic rules from the complete set. The *Theory* lines in the figures show the

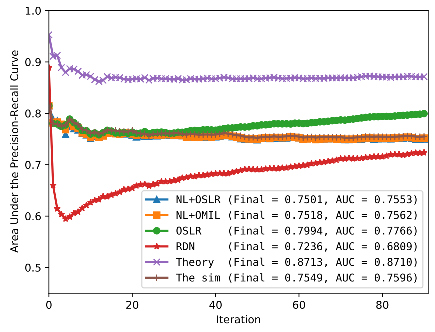
[1] http://alchemy.cs.washington.edu/



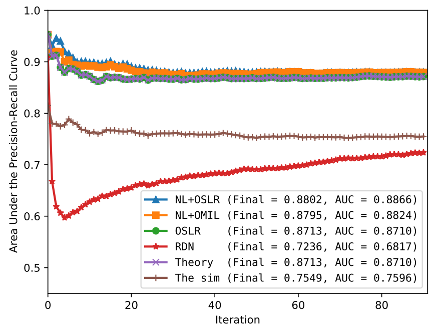
(a) UMLS



(b) UWCSE



(c) UWCSE
Simplified Initial Theory



(d) UWCSE
Complete Theory

Fig. 3: The Evaluation of the UMLS and UWCSE Datasets

result of the initial theory, while the *The sim* shows the results of the simplified theory; both theories were inferred by OSLR, without any training.

As can be seen in Figures 3b, OSLR outperforms both NeuralLog systems and RDN-Boost, when they all start from an empty theory, however, it cannot outperform the complete theory. On the other hand, when the systems start from the simplified theory (Figure 3c), OSLR can improve over the simplified theory, however, not yet outperforming the complete theory; while NeuralLog systems stay close to the performance of the simplified theory and RDN-Boost performed worse than the theory itself. Finally, when starting from the complete theory (Figure 3d), both NeuralLog+OSLR and NeuralLog+OMIL are able to improve over the complete theory, with a slight advantage for NeuralLog+OSLR, while OSLR is only capable of achieving the same performance as the complete theory and RDN-Boost, again, performed worse than the theory. However, neither NeuralLog+OSLR nor NeuralLog+OMIL were able to change the complete theory, showing that the improvement in this dataset was due to the NeuralLog inference mechanism itself.

This demonstrates the strength of revision theory methods, that are capable of improving the results of initial existing theories, even when the theories are only partially correct, corroborating the results already found in other works such as [23, 4, 20]. On the other hand, RDN-Boost is not able to change the initial theory and cannot fix possible mistakes of the theory.

For the empty initial theory, there is no statistical difference between: NeuralLog+OSLR and NeuralLog+OMIL, in either metrics; OSLR and the simplified theory nor OSLR and RDN-Boost, for the area under the curve; RDN-Boost and the simplified theory, in either metrics. For the initial complete theory, there is no statistical difference between: NeuralLog+OSLR and NeuralLog+OMIL, in either metrics; NeuralLog+OSLR and OSLR, for the final iteration; NeuralLog+OSLR and the complete theory, for the final iteration; NeuralLog+OMIL and OSLR for the final iteration; NeuralLog+OMIL and the complete theory, for the final iteration; OSLR and the complete theory, in either metrics. For the initial simplified theory, there is no statistical difference between: NeuralLog+OSLR and NeuralLog+OMIL, in either metrics; and NeuralLog+OSLR/NeuralLog+OMIL and the simplified theory, in either metrics.

5 Related Work

Neural-Symbolic Learning and Reasoning studies the combination of ILP with deep learning [6], which can leverage deep learning the ability of handling relational data while addressing the problem of uncertainty and noise from ILP.

TensorLog [1] is a system closely related to NeuralLog. Both TensorLog and NeuralLog store logic facts in matrix form and perform logic inference through numeric operations on those matrices. However, NeuralLog differs from TensorLog in the way the neural network is built from the logic theory. Furthermore, NeuralLog is more flexible than TensorLog, mainly because it treats numeric attributes as logic terms that can be easily manipulated through logic rules. Differently from TensorLog, NeuralLog also supports rules containing free variables, which might be important for some tasks.

MIL have already been applied in Iterated Structural Gradient [27], to learn theories for ProPPR, a Stochastic Logic Programming system [16]. However, ProPPR uses an inference mechanism that cannot be easily integrated with neural networks. MIL is well suited to integrate with NeuralLog, since the higher-order theory allows the user to define a template in order to create the relational part of the network. This template can be used to append this relational part to an existing neural network.

We implemented the OSLR theory revision algorithm as our online learning mechanism [8, 7]. We opted for this algorithm because it has a clear separation between the structure learning algorithm and the underneath inference mechanism, which allowed us to easily port it to work with NeuralLog. Finally, the flexibility of OSLR allowed us to implement the new MIL revision operator to apply MIL online. To the best of our knowledge, it is the first time that MIL is applied to online learning tasks.

6 Conclusion

In this paper we presented two online structure learning algorithms based on NeuralLog [9], NeuralLog+OSLR and NeuralLog+OMIL. Both learning algorithms are based on Online Structure Learner by Revision (OSLR) [8, 7]. NeuralLog+OSLR is a port of OSLR, using NeuralLog as inference engine; while NeuralLog+OMIL uses the underlying mechanism from OSLR, but with a revision operator based on Meta-Interpretive Learning (MIL) [18]. We compared our proposal with OSLR [8, 7] and RDN-Boost [12] on link prediction task in three different datasets: Cora [21], UMLS [13] and UWCSE [24]. Our experiments showed that NeuralLog+OMIL outperforms OSLR and RDN-Boost on three of the four target relations from the Cora dataset and in the UMLS dataset. While NeuralLog+OSLR and NeuralLog+OMIL outperformed OSLR and RDN-Boost on the UWCSE, whenever a good initial theory is provided.

As future work, we would like to experiment with NeuralLog structure learning algorithms on more datasets, including mixing the relation part of the neural network with propositional neural network models. For instance, the combination of models for natural language processing with relational tasks.

References

1. Cohen, W.W., Yang, F., Mazaitis, K.: Tensorlog: A probabilistic database implemented using deep-learning infrastructure. J. Artif. Intell. Res. **67**, 285–325 (2020)
2. Dawid, A.P.: Present position and potential developments: Some personal views: Statistical theory: The prequential approach. Journal of the Royal Statistical Society. Series A (General) **147**(2), 278–292 (1984)
3. De Raedt, L.: Logical and Relational Learning. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 1 edn. (2008)
4. Duboc, A.L., Paes, A., Zaverucha, G.: Using the bottom clause and modes declarations on FOL theory revision from examples. Machine Learning **76**(1), 73–107 (2009)
5. Galárraga, L.A., Teflioudi, C., Hose, K., Suchanek, F.: Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In: Proceedings of the 22Nd International Conference on World Wide Web. pp. 413–422. WWW '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2488388.2488425
6. Garcez, A.d., 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: AAAI Spring Symposium Series (2015)
7. Guimarães, V., Paes, A., Zaverucha, G.: Online probabilistic theory revision from examples with ProPPR. Machine Learning **108**(7), 1165–1189 (Jul 2019)
8. Guimarães, V.: Online Probabilistic Theory Revision from Examples: A ProPPR Approach. Master's thesis, PESC, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil (2018)

9. Guimarães, V., Costa, V.S.: Neurallog: A neural logic language (2021)
10. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**(301), 13–30 (1963)
11. Horn, A.: On sentences which are true of direct unions of algebras. The Journal of Symbolic Logic **16**(1), 14–21 (1951)
12. Khot, T., Natarajan, S., Kersting, K., Shavlik, J.: Gradient-based boosting for statistical relational learning: the markov logic network and missing data cases. Machine Learning **100**(1), 75–100 (2015)
13. Kok, S., Domingos, P.: Statistical predicate invention. In: Proceedings of the 24th International Conference on Machine Learning. p. 433–440. ICML '07, Association for Computing Machinery, New York, NY, USA (2007)
14. LeCun, Y., Bengio, Y.: The handbook of brain theory and neural networks. chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. MIT Press, Cambridge, MA, USA (1998)
15. Muggleton, S.: Inverse entailment and progol. New Generation Computing **13**(3), 245–286 (1995)
16. Muggleton, S.: Stochastic logic programs. In: New Generation Computing. vol. 32, pp. 254–264. Academic Press, Cambridge, Massachusetts, EUA (January 1996)
17. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. J. Log. Program. **19/20**, 629–679 (1994)
18. Muggleton, S.H., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. Mach. Learn. **100**(1), 49–73 (2015)
19. Neville, J., Jensen, D.: Relational dependency networks. Machine Learning **8**(Mar), 653–692 (2007)
20. Paes, A., Zaverucha, G., Costa, V.S.: On the use of stochastic local search techniques to revise first-order logic theories from examples. Machine Learning **106**(2), 197–241 (2017)
21. Poon, H., Domingos, P.: Joint inference in information extraction. In: Proceedings of the 22Nd National Conference on Artificial Intelligence. AAAI'07, vol. 1, pp. 913–918. AAAI Press, Vancouver, British Columbia, Canada (July 2007)
22. Richards, B.L., Mooney, R.J.: Learning relations by pathfinding. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92). pp. 50–55. San Jose, CA (July 1992)
23. Richards, B.L., Mooney, R.J.: Automated refinement of first-order horn-clause domain theories. Machine Learning **19**(2), 95–131 (1995)
24. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1), 107–136 (2006)
25. Shan-Hwei Nienhuys-Cheng, R.d.W.a.: Foundations of Inductive Logic Programming. Lecture Notes in Computer Science 1228 : Lecture Notes in Artificial Intelligence, Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 1 edn. (1997)
26. Shapiro, E.Y.: Algorithmic program debugging. The MIT Press, Cambridge, Massachusetts, EUA, 1 edn. (1983)
27. Wang, W.Y., Mazaitis, K., Cohen, W.W.: Structure learning via parameter learning. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. p. 1199–1208. CIKM '14, Association for Computing Machinery, New York, NY, USA (2014)
28. Wang, W.Y., Mazaitis, K., Lao, N., Cohen, W.W.: Efficient inference and learning in a large knowledge base. Machine Learning **100**(1), 1–26 (2015)
29. Warren, D.H.D., Pereira, L.M., Pereira, F.: Prolog - the language and its implementation compared with lisp. SIGPLAN Not. **12**(8), 109–115 (1977)