Using neural networks for relational learning

Hendrik Blockeel HENDRIK.BLOCKEEL@CS.KULEUVEN.AC.BE
Werner Uwents
Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium

Abstract

Relational learners need to be able to handle the information contained in a set of related tuples. Most current relational learners are biased either towards the use of aggregate functions that summarize that set, or towards checking the existence of specific kinds of elements in that set. Learning patterns that contain a combination of both is a challenging task. In this paper we introduce a neural networks based approach to relational learning, where the neural net that is learned can actually represent such a combination. This capacity is illustrated on toy problems, but several questions are open with respect to learnability of more complicated concepts.

1. Introduction

Non-relational ("propositional") learners can be said to learn a function $f(x)$ where the domain of x is a cartesian product of different domains. That is, x is described by a fixed number of attributes, for each of which a single value is given; or, in yet other words, x is a single tuple.

In relational learning, a single instance is described by a tuple together with a set of other tuples related (linked) to it. We will adopt the terminology from relational databases here. A database contains a set of relations, each of which is a set of tuples. Tuples may be linked to each other through foreign key relationships. Thus, if we have relations R and S in a relational database, a tuple r of R forms a propositional description of some entity, whereas r together the set of all $s \in S$ that are linked to r, forms a relational description of the same entity.

Assuming we disallow set-valued attributes, the information contained in a set of related tuples can in practice not be represented accurately as a single tuple (for instance, because its size is unbounded), it can only be approximated. We can then distinguish two approaches: those that describe the set with a number of features that is chosen in advance (e.g., its cardinality) and then learn from a propositional representation (these are also called "propositionalization approaches"), and those that search a potentially infinite space of features for the most relevant ones; this feature construction is then part of the learning process.

In relational algebra terminology, the features that are used to describe sets are usually of the general form $\mathcal{F}(\sigma_C(S))$ where $\mathcal{F}$ is some kind of aggregation function, C is a condition that elements of S may or may not fulfill, and $\sigma_C(S)$ is the set of all elements of S that fulfill C. The set S, when classifying a tuple t, is obtained by joining t with any other tuples to which it is linked, joining those tuples again with linked tuples, and so on, up to a certain maximum number of consecutive joins.

Now it is instructive to look at how relational learners fill in $\mathcal{F}$ and C. In propositionalization approaches, the feature itself, i.e., the combination of $\mathcal{F}$ and C, is defined in advance. Any combination can be identified by the user as potentially relevant. There is no automatic feature construction.

Other approaches construct features using some structured search. These typically focus their search on either $\sigma_C(S)$ or $\mathcal{F}$. Inductive logic programming (ILP) (Muggleton & De Raedt, 1994) algorithms typically construct a complex $\sigma_C(S)$, where S is several levels deep and C may be a complex conjunction of conditions; but $\mathcal{F}$ reduces to testing the emptiness of $\sigma_C(S)$ (logical clauses, as learned in ILP, are existentially quantified), in other words, $\mathcal{F}(X) = [count(X) > 0]$.

Yet other approaches, typically described in a relational databases framework, allow several predefined aggregation functions $\mathcal{F}$, but apply them only to S itself, not to a selection of S. Knobbe et al. (2002) are, to our knowledge, the first to present a method that performs a systematic search in a hypothesis space (in this case, that of "selection graphs") where hypotheses combine aggregation and selection. Their approach is however limited to monotone aggregation functions ($\mathcal{F}$ is monotone if $S \subseteq S' \Rightarrow \mathcal{F}(S) < \mathcal{F}(S')$), which limits its applicability somewhat (for instance, Sum and Average are not monotone), and to selecting aggregate functions from a limited set given by the user.

Finally, some approaches may learn any aggregation function, not only predefined ones; an example of these is presented by Perlich and Provost (2003), who classify tuples based on the distribution of linked tuples. These authors also present a classification of relational data mining tasks, stating that most current relational systems handle tasks in categories 1 or 2 (involving no aggregation or uni-dimensional aggregation). Categories 3 and 4 involve multi-dimensional aggregation (aggregating over several variables, of the same relation for category 3, of different relations for category 4). (They also have a category 5, not relevant for this discussion). ILP-like methods are the only ones that consider tasks up to level 4, albeit with strong limitations with respect to the aggregate functions that they allow. Knobbe et al. (2002) partially lift this limitation, by allowing any aggregate function that is a member of a user-defined set of monotone functions. To our knowledge, no approaches currently combine a complicated selection with less limited kinds of aggregation.

In this paper, we explore the use of neural networks for relational learning. The approach we propose, does combine complicated, non-predefined, selections and aggregations, and hence can be positioned in category 4. It can be considered a supervised learning approach, where aggregations and selections are learned in parallel, tuned to each other in order to provide maximal information with respect to the tuple to be classified. This paper presents preliminary work, showing the promise of the approach, but leaving many questions open. We define relational neural networks in Section 2, present some experiments in Section 3, and conclude in Section 4.

2. Relational Neural Nets

A standard feedforward neural network has a fixed number of inputs, all with a well-distinguished effect on the output; in other words, it represents a function $f(x)$ where x is a single tuple. In order to use a neural network for relational learning, we need to extend it with a capacity for handling sets. More specifically, for a given instance, we need to be able to feed a set of tuples into the network for any one-to-many relationship that the instance participates in.

We first discuss how we can handle a set of inputs using a recurrent neural network; then we show how a relational neural network structure can be derived from a relational database schema. Finally we compare this approach to some other approaches that use neural networks for relational learning.

2.1. Handling set inputs using recurrent neural networks

A recurrent neural network is a standard feedforward neural network in which feedback loops are introduced; that is, the outputs of certain neurons are fed back into their inputs (either directly or indirectly).

For 2-layer recurrent neural networks, three architectures are typically distinguished:

- Elman: the output of each layer 1 (hidden layer) node is fed back into each layer 1 node
- Jordan: the output of each layer 2 node is fed back into each layer 1 node
- Willams-Zipfer: all layer 1 and layer 2 outputs are fed back into all layer 1 nodes (thus combining the Elman and Jordan architectures).

The feedback loops give the network a kind of memory, which allows one to feed multiple input vectors into the network for a given training example. Recurrent networks are typically used for time series, where the prediction at a given time point may depend not only on inputs for that time point but also on previous inputs. The input vectors at times $t-k$, $t-k+1$, $\dots$, $t-1$, t are then typically fed into the network one after another and the output produced by the network after seeing these inputs is compared with the desired output y_t at time t. Reduction of the difference between y_t and the network output can be done using a gradient descent procedure that can be implemented using a variant of the standard backpropagation algorithm. A typical method used is backpropagation through time, where the recurrent network is unfolded into a multilayer feedforward neural network to which standard backpropagation is applied.

While recurrent neural networks are often used for time series prediction, they appear to be less commonly used for handling sets of input vectors. The main difference between these two is that in time series prediction, the input for a given instance is an ordered set of vectors, rather than an unordered set (which is what we need here). Obviously, any function of an unordered set can be represented as a function of an ordered set, but the converse does not hold. The fact that we want to learn an order-invariant function could in principle be used to constrain the neural net's parameters so that only order-invariant functions can be learnt. At this stage we have not investigated exactly how this could be done.

2.2. Relational neural networks

Assume we have a relational database schema where R_T is the target relation, and $R_1, \dots, R_n$ are other relations in the database. We denote the attribute sets of R_i by U_i. In the following we will assume a classification setting for ease of discussion, but everything applies equally well to any predictive setting.

Given any relation R, we define

- $S_1(R)$: $R_i \in S_1(R)$ iff each tuple $t \in R$ is connected to exactly one tuple in R_i (i.e., there is a one-to-one or many-to-one relationship between R and R_i, in which R participates completely)

- $S_{01}(R)$: $R_i \in S_{01}(R)$ iff each tuple $t \in R$ is connected to at most one tuple in R_i (again one-to-one or many-to-one, but partial participation)
- $S_n(R)$: $R_i \in S_n(R)$ iff each tuple $t \in R$ is connected to zero, one, or more tuples in R_i (partial or complete participation of R in a one-to-many or many-to-many relationship with R_i)
- $S_u(R)$: $R_i \in S_u$ iff R_i is a relation of the relational database not in $S_1(R)$, $S_{01}(R)$ or $S_n(R)$ (i.e., it is not directly connected to R)

Given a tuple t in the target relation R_T, we want to classify it based on the information contained in the tuple and in any tuples linked to this tuple. For a relation R_i we use U_i to denote the original attribute set of that relation and I_i to denote the attribute set actually used as input to our classifier; one might expect $I_i = U_i$ but there will be some small differences. For the target table, $I_T = U_T - \{C\}$ where C is the class attribute.

For any tuples $t_i \in R_i$ where $R_i \in S_1(R_T)$, the information in t_i can be added to t by joining R_i with $\{t\}$: a single tuple is thus obtained. $I_i = U_i$ for these tuples.

For any $R_i \in S_{01}(R_T)$, an outer join with $\{t\}$ also yields exactly one tuple, which may however contain null values for the attributes of R_i. As neural networks do not have a distinguished encoding for null values, we will use an extra attribute E_i that indicates whether the link to R_i yielded a joining tuple or not. $I_i = U_i \cup \{E_i\}$.

For $R_i \in S_n(R_T)$, an indefinite number of tuples t_{i1}, …, t_{in} may be linked to a single $t \in R_T$. In order to allow this set of tuples to influence the classification of t, a recurrent neural network will be constructed. The tuples t_{ij} will be represented using their original attribute values plus an extra attribute E_i which is always true for these tuples. The fact that no tuples t_{ij} exist will be indicated using a single input tuple where $E_i = false$.

Tuples in $R_i \in S_u(R_T)$ are not directly linked to tuples in R_T, but may be linked indirectly. For now we ignore these.

Based on the above, we can propose several options for constructing a relational neural network that classifies $t \in R_T$ based on its own attribute values as well as those of (directly) related tuples. Two options that we will explore here, are:

Option 1: For each relation involved, including the target relation, a different neural network (called a "component") is constructed. The outputs of these networks are combined by a single perceptron. The components are standard feedforward neural networks for the target relation R_T and for any relations in $S_1(R_T)$ and $S_{01}(R_T)$, with as inputs the I_i as defined above. The components for $S_n(R_T)$ are recurrent neural networks.

Option 2: For each t, we can construct a tuple t' with attributes

$$I_T \cup \Big(\bigcup_{i:R_i \in S_1(R_T)} I_i\Big) \cup \Big(\bigcup_{i:R_i \in S_{01}(R_T)} I_i\Big) \cup \Big(\bigcup_{i:R_i \in S_n(R_T)} O_i\Big)$$

with I_i as defined above, and O_i a set of attributes the values of which are the outputs of a 2-layer recurrent neural network with as inputs U_i.

Of course, other options are possible as well, but we restrict ourselves in this paper to these two. Of these two, Option 1 is the more restricted one: a simpler network is constructed in which the results of different components are combined by one perceptron. Option 2 creates a larger and more expressive network; it also has an extra structural parameter, namely the number of outputs $|O_i|$ of the recurrent networks (which we assume here to be the same for all recurrent networks).

Note that the O_i attributes in Option 2 can be seen as aggregate functions that summarize the set of R_i-tuples related to t. Thus, this approach is closer to the propositionalization approach, where a propositional learner learns from a fixed set of attributes, some of which summarize set information. It differs from classical propositionalization approaches in that these aggregates are now learned, instead of predefined.

Note that with Option 2, the technique of adding to t the O_i attributes that summarize related tuples, can be repeated for those related tuples, thus also incorporating information in indirectly linked tuples (from relations in $S_u(R_T)$).

2.3. Related work

The existing work that is probably closest to our approach, is a line of work in the neural networks community on learning from structured data using recursive neural networks or folding architecture networks (Goller & Küchler, 1996; Sperduti & Starita, 1997; Frasconi et al., 1998). These authors describe how to learn from structured data (e.g., logical terms, trees, graphs), and discuss tasks like the identification of substructures, but they do not aim at learning aggregation functions. Those tasks relate to the tasks we consider, more or less in the same way as ILP relates to our approach. Our approach however is not essentially different, and many existing results on learnability of recursive neural networks may carry over to our setting.

A number of neural network based approaches have been defined in the ILP setting (Botta et al., 1997; Basilio et al., 2001); the neural networks in these ap-

ID	Gender	Father	Mother
albert	M	bob	alice
peter	M	bob	alice
alice	F	jack	mary
bob	M	james	anne
...	...	...	...

Table 1. An example instance of the family database.

proaches typically mimic logical inference as it would be made by logic programs or implement numerical computations in them. Again, they do not learn aggregate functions as we do. Ramon and De Raedt (2000) have defined multi-instance neural networks; these are a subset of our relational neural networks designed specifically for the multi-instance case. Ramon et al.'s neural logic programs (Ramon et al., 2002) are somewhat similar to our relational neural networks, with as main differences that they are described in a first order logic framework and that, just like for multi-instance neural networks, specific aggregate functions are encoded in advance by the user, instead of learned (and typically they represent logical conjunctions and disjunctions). We believe that from the point of view of relational learning, the ability to learn aggregate functions is a crucial advantage of our approach.

3. Experiments

To evaluate the potential of this approach, we have performed a number of experiments, varying parameters along a number of dimensions: Option 1 versus Option 2, as discussed above; different architectures for the recurrent components (Elman, Jordan, Williams-Zipfer); different learning approaches (backpropagation through time, evolutionary learning), different parameter settings for these approaches.

3.1. A family database

In this artificially generated toy database, there is a single relation with attributes ID, Gender, Father, Mother; it describes a number of persons and parentship relations between them (Father and Mother are foreign keys to ID). We show an example relation in Table 1. Note that the ID, Father and Mother attributes are present only to identify tuples and to link them to each other; they are not descriptive attributes of which the value will be tested by a hypothesis.

For this toy database, we define a few simple concepts. Given a tuple t, let $S(t)$ be the set of tuples s for which $s.Father = t.ID$ or $s.Mother = t.ID$.

- C1: has at least one son;
  $C1(t) \Leftrightarrow \mathit{count}(\sigma_{Gender=M}(S(t))) > 0$
- C2: has 2 children; $C2(t) \Leftrightarrow \mathit{count}(S(t)) = 2$
- C3: has 2 sons;
  $C3(t) \Leftrightarrow \mathit{count}(\sigma_{Gender=M}(S(t))) = 2$
- C4: has one son and one daughter;
  $C4(t) \Leftrightarrow (\mathit{count}(\sigma_{Gender=M}(S(t))) = 1 \wedge \mathit{count}(\sigma_{Gender=F}(S(t))) = 1)$
- C5: has one son or two daughters;
  $C5(t) \Leftrightarrow \mathit{count}(\sigma_{Gender=F}(S(t))) = 1 \vee \mathit{count}(\sigma_{Gender=M}(S(t))) = 2)$
- C6: has a grandson; $C6(t) \Leftrightarrow \mathit{count}(\{s \in S(t) | \mathit{count}\sigma_{Gender=F}(S(s))) > 0\}) > 0$

Note that C1 is a concept that could be learned by any ILP system: it involves some selection criterion but a trivial aggregation function. C2 is a concept that could easily be learned by any system where aggregates over sets of related tuples are predefined. Concept C3 is a more complicated concept that involves an aggregate over a selection of related tuples. C4 is again an ILP-like concept, but a slightly more complicated one than C1. Similarly, C5 is a slightly more complicated concept than C3, combining two functions that themselves combine aggregation and selection. C6 is an example of a concept that takes into account information "two steps away" from the tuple to be classified.

Most existing systems can learn these only with a relatively strong bias; e.g., in an ILP system typically the allowed aggregate function as well as any conditions that are allowed in the argument of this aggregate function would have to be given in advance.

The experimental setting is as follows: each combination of relational (Option 1, Option 2) and recurrent (Elman, Jordan, Williams-Zipfer) architecture is tried on all concepts C1-C6. For each architecture-concept combination, five runs with different random initializations of the network weights are made. Each run uses the same training set, which consists of 2/3 of the dataset. Of these five randomized runs, the one with highest training accuracy is chosen and evaluated on the remaining, unseen, examples (1/3 of the dataset). (Using a separate validation set would allow to separate the overfitting risk of an architecture from its true generalization power; for these initial exploratory experiments we found it unnecessary to have this separation, but it would be desirable for more sophisticated experiments.)

It turned out that for both relational architectures, using backpropagation through time, the network is able to learn correctly (with 100% test set accuracy) all the concepts. In general, some tuning of the network parameters was necessary for this, which is not unexpected; and usually not all random initializations of the network allowed it to converge to the correct concept. The results for evolutionary learning were similar.

3.2. Other Datasets

We have also attempted to evaluate the proposed approach on two benchmark datasets: the Musk dataset (Dietterich et al., 1997), a multi-instance problem on which several relational learning approaches have been compared, and the Financial dataset (Berka, 2000) for which also Knobbe et al. (2002) have reported results.

Due to inefficiencies in our current implementation, we have been able to evaluate the approach only on relatively small data sets up till now. For the Musk experiments, we have used Musk-1, the smaller of the two available data sets. For the Financial data set, we have not been able to include the Transaction relation in the training data, because it is too large. This means that for this data set our results are not comparable to results published elsewhere.

For the Musk-1 data set, we performed a tenfold cross-validation with the same folds that were used elsewhere. Four different combinations of parameters and architectures were tried, and predictive accuracies on unseen data varied between 79.3% (19 errors) and 85.9% (13 errors). This is to be compared with 88% (11 errors) reported for "multi-instance neural networks" (MINNs) (Ramon & De Raedt, 2000) approach, which are a special case of our relational neural networks.

Relational neural networks cannot outperform MINNs on this type of problems, since the hypothesis space searched by the former, H_{RNN}, is a superset of the hypothesis space H_{MINN} searched by the latter, and any hypothesis in $H_{RNN} - H_{MINN}$ is necessarily meaningless because it violates the multi-instance assumptions. In other words, the hypothesis in H_{RNN} that best approximates the target hypothesis, must also be in H_{MINN}. The best we can hope for, is that our approach performs as well as multi-instance neural networks on multi-instance problems (while performing better outside that class). From that point of view, the obtained results are promising, but more experimentation is needed to obtain more insight into, e.g., the probability of obtaining with our more general approach an accuracy that is comparable to that of the more specialized approach.

For the Financial dataset, 2/3 of the data were used for training and 1/3 for testing, and six different configurations were tried. Obtained accuracies were between 74% and 85%. This data set has a highly skewed class distribution: the frequency of the majority class is 85%. Thus, accuracy-wise, these results are quite bad. A ROC analysis, however, shows that on average the method does perform better than random guessing, in the best case combining a true positive rate of 0.95 with a false positive rate of 0.75. As mentioned, these results are not comparable with published results because only a subset of the available information was used.

4. Conclusions

We have presented a novel, neural network based, approach to relational learning. The approach consists of constructing a neural network based on the relational database schema, where recurrent components are introduced for one-to-many relationships. The power of this approach is yet to be determined, but this initial exploration reveals that, in Perlich and Provost's terminology (Perlich & Provost, 2003), simple category 3 and category 4 concepts (as defined for our toy database) can be accurately represented and learned. An experiment on a multi-instance benchmark further indicates that the approach may work equally well on this type of problems as the more specialized MINN approach (Ramon & De Raedt, 2000). An experiment on a fully relational problem has mainly revealed the need for a more efficient implementation: the computational complexity of the approach is relatively high and currently precludes a meaningful comparison of our approach to other approaches, on this benchmark.

Many questions remain open. What is the best architecture for these networks? We have proposed two options; our Option 2 is clearly more expressive than the other, but both appear to learn well on the problems we have considered. Among the three architectures for recurrent networks, there was a tendency of the Williams-Zipfer architecture to perform somewhat better than the others in our experiments, but again further experimentation is necessary.

What subclass of Perlich and Provost's category 3-4 can be learned using this approach? We have focused here on "combining selection and aggregation", which is a special case of multidimensional aggregation for which it is intuitively easier to see that our relational neural networks should be able to represent them. Even for this subclass, formal results would be desirable, and further experiments should be conducted.

Our recurrent neural networks in principle learn from ordered sets; they might be made to learn more effectively if their parameters are somehow constrained so that the network expresses a function of an unordered set. (Randomly permuting the elements of a set, each time the set is fed to the network, is also an option, but may not be the most efficient one.)

It is clear that there are connections between this approach and some existing approaches for learning from structured data that are not directly connected with statistical relational learning. We expect that some further investigation into those approaches may advance the field of relational learning.

5. Acknowledgements

Hendrik Blockeel is a post-doctoral fellow of the Fund for Scientific Research of Flanders (FWO-Vlaanderen). We thank Martijn van Otterlo, Kristian Kersting and Jan Ramon for useful suggestions concerning this work.

References

Basilio, R., Zaverucha, G., & Barbosa, V. C. (2001). Learning logic programs with neural networks. *Proceedings of the Eleventh International Conference on Inductive Logic Programming*. Springer-Verlag.

Berka, P. (2000). Guide to the financial data set. *The ECML/PKDD 2000 Discovery Challenge*.

Botta, M., Giordana, A., & Piola, R. (1997). Fonn: Combining first order logic with connectionist learning. *Proceedings of the 14th International Conference on Machine Learning* (pp. 46–56). Morgan Kaufmann.

Dietterich, T. G., Lathrop, R. H., & Lozano-Pérez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, *89*, 31–71.

Frasconi, P., Gori, M., & Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE-NN*, *9*, 768–786.

Goller, C., & Küchler, A. (1996). Learning task-dependent distributed representations by back-propagation through structure. *Proceedings of the IEEE International Conference on Neural Networks (ICNN-96)* (pp. 347–352).

Knobbe, A., Siebes, A., & Marseille, B. (2002). Involving aggregate functions in multi-relational search. *Principles of Data Mining and Knowledge Discovery, Proceedings of the 6th European Conference* (pp. 287–298). Springer-Verlag.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming : Theory and methods. *Journal of Logic Programming*, *19,20*, 629–679.

Perlich, C., & Provost, F. (2003). Aggregation-based feature invention and relational concept classes. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 167–176). ACM Press.

Ramon, J., & De Raedt, L. (2000). Multi instance neural networks. *Proceedings of the ICML-Workshop on Attribute-Value and Relational Learning*.

Ramon, J., Driessens, K., & Demoen, B. (2002). Neural logic programs. Unpublished.

Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, *8*, 714–735.