

Aggregation versus selection bias, and relational neural networks

Hendrik Blockeel and Maurice Bruynooghe

Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium

Abstract

Current relational learners handle sets either by aggregating over them or by selecting specific elements, but do not combine both. This imposes a significant, possibly undesirable bias on these learners. We discuss this bias, as well as some ideas on how to lift it. In the process, we introduce the notion of relational neural networks.

1 Biases of Relational Learners

Among the many approaches to relational model learning that currently exist, a distinction can be made with respect to how they handle one-to-many and many-to-many relations, or, equivalently, how they handle sets of objects.

To illustrate this, consider a database with just a single relation “Person” with attributes Mother, Father, and Sex. (Mother and Father are foreign keys to Person.) and consider the following simple concepts:

- A. people who have two children
- B. people who have a son (that is, at least one)
- C. people who have two sons

In all three cases, we want to classify persons, based on properties of (a set of) persons related to them. The parentheses around “a set of” indicate the two different kinds of approaches that we distinguish here, a distinction also mentioned by Jensen and Neville (2002).

The first kind of relational methods use aggregate functions to handle sets. The result of an aggregate function, obviously, is a property of the set as a whole, not of individual elements of the set. Among these methods we count, e.g., probabilistic relational models (PRMs) (Getoor et al., 2001), or “propositionalization” approaches that include aggregates, such as the one by Krogel and Wrobel (2001).

A second kind of relational methods handles sets by looking at properties of their elements. Typically, tests are of the form “there exists an x in the set such that $P(x)$ holds”, with P a relatively complicated condition. Most inductive logic programming (ILP) systems follow this approach.

Let us call methods of the first kind, aggregating methods; and methods of the second kind, selective methods (in the sense that they select an element from the set and investigate

properties of that single element). Referring to the example concepts above, we can then state that aggregating methods can easily express A, but not B, whereas selective methods can easily express B, but not A. Importantly, *none of the approaches mentioned can easily express concept C*, because this description contains both selection and aggregation (select all male children, and count only these).

More formally, if we express class definitions in the relational algebra and write them as $\sigma_{C_1}(\mathcal{F}(\sigma_{C_2}(R)))$ with R the result of joining the original relation with a relation it links to, then selective methods such as ILP focus on the construction of C_2 and fix C_1 and \mathcal{F} to denote existence (count > 0), whereas aggregating methods focus on constructing a good C_1 and \mathcal{F} but fix C_2 to be true.

For instance, PRMs, as defined by Getoor et al. (2001) cannot learn concept C without having separate relations for sons and daughters. Manually introducing these separate relations of course presupposes that the user is aware of the possible importance of these concepts. Alternatively, one could define a large number of aggregate functions that have appropriate selection conditions built in; in that case, a search through a space of aggregate functions is needed.

In an ILP setting, one could of course define aggregate functions as background knowledge. Then, e.g., the rule $p(X) :- \text{count}(Y, (\text{child}(X,Y), \text{male}(Y)), 2)$ expresses concept C. The main difficulty here is that the second argument of the count meta-predicate is itself a query that is the result of a search through some hypothesis space. It is not obvious how such a search should be conducted; the many results in ILP on how to search a first-order hypothesis space efficiently (Nienhuys-Cheng and De Wolf, 1997) do not consider the case where the resulting hypothesis will be used as the argument of a metapredicate.

ILP-like approaches that do not include aggregate functions, can still express concept C as, e.g., “the person has a male child x and a male child y and $x \neq y$ and there does not exist a child z such that z is male and $z \neq x$ and $z \neq y$ ”; but in practice, the length of this rule, as well as the occurrence of a negation (the scope of which is again a conjunction of multiple literals) make it difficult to learn, and of course also the comprehensibility of the result is negatively influenced.

To our knowledge no currently existing approaches can construct theories that combine aggregate functions with (reasonably complex) selections on the set to be aggregated.

2 Combining Aggregation with Selection

In databases, both aggregation and selection are very natural operations, and ideally a relational learning system should be able to combine both in the models it builds. In order to achieve this goal, it is necessary to define a search space of hypotheses that combine aggregations and selections, and find a more or less efficient way to navigate through this search space. This is currently an open problem. We here list a number of ideas that could be investigated further. We divide them into symbolic and subsymbolic approaches.

2.1 Symbolic Approaches

To build a concept in symbolic form, a search space has to be traversed that consists of combinations of aggregations and selections. This could be done in a hill-climbing way, but it appears that in some cases the search can be made slightly more exhaustive without increasing its computational complexity much. For instance, counting the number of children of a person takes just as much work as counting the number of sons and daughters separately, and a simple addition of these counts yields the total number of children. More generally, given a partition $\{S_1, \dots, S_n\}$ of a set S , aggregates of S can often be computed efficiently from aggregates of the S_i , and the latter can all together be computed as efficiently as computing the aggregate for S . This holds at least for the often occurring aggregate functions count, sum, average, min, max. Thus, when we search for conditions of the form $\mathcal{F}(\sigma(S))\theta c$ with \mathcal{F} an aggregate function, σ some selection, and θ some operator ($\leq, =, \dots$), a certain subspace of all possible σ 's can be searched exhaustively at very little additional computational cost, compared to considering only the condition $\mathcal{F}(S)\theta c$. This suggests a straightforward possible improvement to some of the existing approaches.

2.2 Subsymbolic Approaches

Another direction for future research that seems interesting, is that of modelling relational databases with neural networks. Neural networks are usually considered propositional learners. A number of approaches exist to extend them to the context of first order logic, but not (to our knowledge) to that of relational databases, which could in fact be simpler. One approach to do that is based on the following observation.

Any data can be modelled using only two basic data structures: tuples and sets. (The relational data model is based on just these two notions.) Propositional learning algorithms handle tuples; to make them relational, it is sufficient to add the capability to process sets. (This is consistent with De Raedt (1998), who identifies multi-instance learning as the simplest “relational” learning task; it is indeed the simplest case where a single example is described by a set of tuples.)

The input of a standard feedforward neural network is a tuple. A relational neural network should in addition have the ability to handle sets, which can have an unlimited number of unordered elements. Recurrent neural networks have this capability: by feeding the output of a layer back into the network, they can aggregate information over an indefinite number of previous inputs. They are typically used for tasks such as time series prediction, where an input at time t can

influence the output at time $t + k$ with k not bounded, but they can just as well be used for processing sets.

Thus, a relational neural network would essentially consist of “normal” and “aggregating” nodes; an aggregating node is simply a node that is fed back into a lower layer. Such a relational neural network would have the same structure as the skeletons used in PRMs. Where the PRM skeleton contains an aggregate function, the relational neural net contains one or more aggregating nodes. Relational neural nets are very similar to Ramon, Driessens and Demoen’s (2002) “neural logic programs”, with as main difference that Ramon et al. consider fixed combination functions for the different kinds of nodes and handle sets using nodes with a variable number of inputs, instead of recurrent nodes.

Relational neural networks would have as advantage over the other approaches that they can *learn* an aggregate function, without that function being pre-encoded in the network, and with selection possibly integrated in it. Thus, training the relational neural network automatically constitutes a search through aggregations and selections simultaneously. Moreover, a wider variety of aggregate functions is considered: not just sums, counts, etc. but also more exotic functions. On the other hand, the learnability of the relational neural networks we propose here, is an open problem. It is known that recurrent neural networks are harder to train than feedforward networks. Increasing the number of layers, as we do here, may further decrease learnability. We believe these issues are worth further investigation.

Acknowledgement

The first author is a postdoctoral fellow of the Fund for Scientific Research of Flanders, Belgium.

References

- [1] L. De Raedt. Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In D. Page, editor, *Proc. 8th Int’l Conf. on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 1–8. Springer, 1998.
- [2] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Džeroski and N. Lavrac, editors, *Relational Data Mining*, pages 7–34. Springer-Verlag, 2001.
- [3] D. Jensen and J. Neville. Schemas and models. In *Proc. of the ACM SIGKDD 2002 Workshop on Multi-Relational Data Mining (MRDM 2002)*, pages 56–70, 2002.
- [4] M.-A. Krogel and S. Wrobel. Transformation-based learning using multi-relational aggregation. In *Proc. 11th Int’l Conf. on Inductive Logic Programming*, pages 142–155, 2001.
- [5] S.-H. Nienhuys-Cheng and R. De Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science and Lecture Notes in Artificial Intelligence*. Springer, 1997.
- [6] J. Ramon, K. Driessens, and B. Demoen. Neural logic programs. Unpublished, 2002.