Auto-encoding Logic Programs

Sebastijan Dumančić [1] **Tias Guns** [2] **Wannes Meert** [1] **Hendrik Blockleel** [1]

1. Introduction

Although deep learning (Goodfellow et al., 2016) has achieved a tremendous progress over the last decade, the majority of the tasks addressed focus on signal data such as images, speech and text. Ubiquitous relational data, e.g. biological and social networks, has only recently started to receive attention. Learning and reasoning with such complex data has typically been in the domain of logic-based approaches to Artificial intelligence such as *Statistical relational learning* (SRL) (Getoor & Taskar, 2007; de Raedt et al., 2016). Exceptions to that are *knowledge graph embedding* (Nickel et al., 2016) and *differentiable interpreter/programmer* (Rocktäschel & Riedel, 2017; Reed & de Freitas, 2016) approaches. Both lines of research focus on developing ways to represent relational data (or programs) in Euclidean spaces or distributed amongst the weights of a neural network.

Though both directions have demonstrated great potential, they also exhibit several limitations. First, representing concepts and reasoning in Euclidean space or distributing it amongst the weights looses the interpretability of logical representation and clarity of its reasoning. Second, it is difficult to know how much of the expressive power of logic can be captured by vectorizing it. Third, these methods are often *data-hungry* and have problems generalizing beyond the training data.

Therefore, retaining predicate logic as representation language for deep learning primitives offers several benefits. First, predicate logic is a full-fledged programming language and thus provides a language that does not introduce any limitations upfront. Second, it is easily interpretable and modular which might allow for efficient learning strategies. Third, logic-based learning approaches are typically extremely *data-efficient* requiring only a few examples. Fourth, having a domain knowledge specified in the logical format allows us to pose questions about anything in a domain, and thus don't have to commit to the specific target in advance. A disadvantage of logic as the representation language is its symbolic nature, which does not allow us to leverage extensive computational infrastructure developed within deep learning. However, that complexity can often be dealt with in practice, as done within the Inductive logic programming (de Raedt, 2008) and satisfiability communities[1] (Biere et al., 2009).

This works exploits the above-outlined benefits and introduces *auto-encoding logic programs* (ALPs), which implement both the encoder and decoder functions as logic programs. We focus on auto-encoders (Hinton & Salakhutdinov, 2006) as they have proven to be an extremely versatile deep learning primitive applicable to many different settings (un-, semi- and supervised) (Bengio et al., 2006; Kingma et al., 2014; Maas et al., 2012; Vincent et al., 2008; Kingma & Welling, 2014; Bengio et al., 2006) by leveraging a simple learning principle. This versatility allows one to use the same representation learning component for a variety of SRL learning tasks.

2. Auto-encoding logic program

This section explains individual components of ALPs, illustrated in Figure 1.

Data The first question that arises is that of the data representation format. We build upon standard SRL approaches and assume the data is given as a *collection of logical atoms* $\mathcal{F}$ – a predicate symbol applied to terms, i.e. constants or variables. For example, in the atom `mother(anna,dirk)` `mother/2` is a predicate (relation) between two (`/2`) constants `anna` and `dirk`. Data vocabulary $\mathcal{V}$ consists of all predicate $\mathcal{P}$ and constant $\mathcal{C}$ symbols defined in the data. By convention, all provided atoms are true, and everything else is false.

Encoder and Decoder Encoder $\mathcal{E}$ and decoder $\mathcal{D}$ function are implemented and learned as logic programs, i.e., a set of clauses. A clause is a logical formula of the form `h :- b1,...,bn`, where `h` and $\texttt{b}_i$ are (non-ground) atoms or their negations. `h` is called *head* atom and $\texttt{b}_i$ are *body* atoms. The bodies of encoder clauses consist of predicates in $\mathcal{P}$, and the corresponding heads define a new set of *latent*

[1]Department of CS, KU Leuven, Belgium [2]Department of Business Technology and Operations, VUB, Belgium. Correspondence to: Sebastijan Dumančić <sebastijan.dumancic@cs.kuleuven.be>.

Published at the ICML workshop Neural Abstract Machines & Program Induction v2 (NAMPI) — Extended Abstract, Stockholm, Sweden, 2018.

[1] http://beyondnp.org

```
Input:      mother(anna,dirk).  female(anna).  father(tom,dirk).  male(tom).
Encoder:                                latent1(X,Y) :- mother(X,Y);father(X,Y).
                                                latent2(X) :- female(X).
Latent rep.:          latent1(anna,dirk).  latent1(tom,dirk).  latent2(anna).
Decoder:                                mother(X,Y) :- latent1(X,Y),latent2(X).
                                                female(X) :- latent2(X).
                                father(X,Y) :- latent1(X,Y),not(latent2(X)).
                                                male(X) :- not(latent2(X)).
Output:     mother(anna,dirk).  female(anna).  father(tom,dirk).  male(tom).
```

Figure 1. Illustration of an auto-encoding logic program (in Prolog format) for a family domain

predicates $\mathcal{H}$. The bodies of decoder clauses consist of predicate in $\mathcal{H}$ and heads are in $\mathcal{P}$.

Reconstruction We define the symbolic reconstruction loss $s(\mathcal{F}, \mathcal{E}, \mathcal{D})$ as a sum of the number of non-reconstructed atoms in $\mathcal{D}$ and the number of *falsely* reconstructed atoms.

Constraints The key ingredient of traditional auto-encoders is imposing *bottleneck* constraints on the latent representation. We do the same by imposing the constraint that the number of facts in the latent representation has to be smaller than the number of the provided facts. We also impose different constraints on interaction between encoder and decoder clauses.

Learning ALPs The task of learning ALPs as then *finding the set of encoder and decoder clauses the minimizes the reconstruction loss* $s(\cdot, \cdot, \cdot)$.

2.1. Learning as constraint optimization

To find the encoder and decoder clauses, we cast learning ALPs as a *constraint optimization* problem (Rossi et al., 2006). Intuitively, we construct possible encoder and decoder clause candidates and pick a small subset of those that minimizes the reconstruction loss. To find candidate clauses, we rely on the notion of *language bias* (Blockeel, 2017) given as a set of syntactic constraints on candidate clauses, e.g., all conjunctive formulas containing at most 3 literals and at most 2, existentially quantified, variables.

Decision variables are Boolean variables indicating whether a certain encoder/decoder clause is selected or not.

Soft constraints indicate how facts can be reconstructed. For instance, if fact f is reconstructed by decoder clauses dc_{N-M}, we introduce the constraints $\bigvee_{i=N}^{M} dc_i \Leftrightarrow r_f$, saying that r_f is true if at least of the the decoder clauses dc_{N-M} is selected.

Objective function is then formulated as *minimizing the number of unsatisfied soft constraints and the number of soft constraints associated with falsely reconstructed atoms*.

Hard constraints are used to make the latent representation *compressive* and enforce certain properties of latent representation, such as requiring that all $\mathcal{P}$ are reconstructable, stating that if two decoder clauses reconstruct identical facts not both can be selected at the same time, and so on.

3. Preliminary experiments

We have conduced preliminary experiments in order to see whether learning from latent representation created by ALPs is beneficial. We focus on learning generative modelling and evaluate whether the model learned on the latent data representation explains data better than the model learned on the original data representation. Therefore, the baseline model simply learns a generative *Markov logic Network* (MLN) (Richardson & Domingos, 2006) on the original data representation, while the latent MLN learns a generative model on the ALP-induced latent representation and uses the decoder to decode latent facts back to the original space.

We learn ALPs by limiting the length of clauses to 2 or 3 atoms; if more complexity is needed, it can be achieved by *stacking* several layers of ALPs. The number of latent facts is limited to $x\%$ of the number of facts in the original data, with $x \in \{0.5, 0.7\}$. We also impose constraints stating that (1) if a clause is a *refinement* of another clause, they cannot both be selected, (2) if two decoder clauses cover the same set of fact, they cannot both be a part of the solution, (3) every predicate from $\mathcal{P}$ should have at least one decoder clause, and (4) if one decoder clause reconstructs a subset of facts of another decoder clause, at most one can be a part of the solution.

We compared the AUC-PR scores of the baseline and latent MLN on two standard SRL datasets - WebKB and Cora entity resolution (Singla & Domingos, 2006). The results indicate that latent representation created this way captures useful patterns that allow to better capture domain knowledge, resulting in better AUC-PR scores of the latent model. Even the most limiting version of ALP using only the clauses with 2 atoms is enough the improve the

performance.

References

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pp. 153–160, Cambridge, MA, USA, 2006. MIT Press.

Biere, A., Biere, A., Heule, M., van Maaren, H., and Walsh, T. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009.

Blockeel, H. Bias specification language. In *Encyclopedia of Machine Learning and Data Mining*, pp. 125–128. 2017. doi: 10.1007/978-1-4899-7687-1_73.

de Raedt, L. *Logical and Relational Learning: From ILP to MRDM (Cognitive Technologies)*. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 3540200401.

de Raedt, L., Kersting, K., Natarajan, S., and Poole, D. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2016.

Getoor, L. and Taskar, B. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007. ISBN 0262072882.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 0036-8075. doi: 10.1126/science.1127647.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*, April 2014.

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., and Welling, M. Semi-supervised learning with deep generative models. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3581–3589. Curran Associates, Inc., 2014.

Maas, A., Le, Q. V., O'Neil, T. M., Vinyals, O., Nguyen, P., and Ng, A. Y. Recurrent neural networks for noise reduction in robust asr. In *INTERSPEECH*, 2012.

Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, Jan 2016. ISSN 0018-9219. doi: 10.1109/JPROC.2015.2483592.

OscaR Team. OscaR: Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.

Reed, S. and de Freitas, N. Neural programmer-interpreters. In *International Conference on Learning Representations (ICLR)*, 2016.

Richardson, M. and Domingos, P. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, February 2006. ISSN 0885-6125.

Rocktäschel, T. and Riedel, S. End-to-end differentiable proving. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 3788–3800. Curran Associates, Inc., 2017.

Rossi, F., Beek, P. v., and Walsh, T. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. ISBN 0444527265.

Singla, P. and Domingos, P. Entity resolution with markov logic. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pp. 572–582, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2701-9. doi: 10.1109/ICDM.2006.65. URL `https://doi.org/10.1109/ICDM.2006.65`.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 1096–1103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294.