Pre-Proceedings of the 12th International Workshop on Neural-Symbolic Learning and Reasoning NeSy'17

Tarek R. Besold, Artur d'Avila Garcez, Isaac Noble
(eds.)

City, University of London – London, UK – 17th & 18th of July 2017

We, as workshop organizers, want to thank the following members of the NeSy'17 program committee for their time and efforts in reviewing the submissions to the workshop and providing valuable feedback to accepted and rejected papers and abstracts alike:

- Raquel Alhama, University of Amsterdam.
- James Davidson, Google Brain.
- Richard Evans, Google DeepMind.
- Barbara Hammer, University of Bielefeld.
- Thomas Icard, Stanford University.
- Kai-Uwe Kühnberger, University of Osnabrück.
- Luis Lamb, Universidade Federal do Rio Grande do Sul.
- Thomas Lukasiewicz, University of Oxford.
- Edjard Mota, Federal University of Amazonas.
- Terrence C. Stewart, University of Waterloo.
- Serge Thill, Plymouth University.
- Son Tran, CSIRO Australia.
- Stefan Wermter, University of Hamburg.

These workshop pre-proceedings are available online from the workshop website under http://www.neural-symbolic.org/NeSy17/.

Bremen, 12th of July 2017
Tarek R. Besold, Artur S. d'Avila Garcez, and Isaac Noble.[1]

[1] Tarek R. Besold is a postdoctoral researcher at the Center for Computing and Communication Technologies (TZI) of the University of Bremen, Germany; Artur S. d'Avila Garcez is Professor of Computer Science at the Department of Computer Science, City, University of London, UK; Isaac Noble is a Software Architect at Playground Global, USA.

Contents: Contributed Papers

Contents: Contributed Abstracts

Towards Grounding Conceptual Spaces in Neural Representations

Lucas Bechberger* and Kai-Uwe Kühnberger

Institute of Cognitive Science, Osnabrück University, Osnabrück, Germany
lucas.bechberger@uni-osnabrueck.de, kai-uwe.kuehnberger@uni-osnabrueck.de

Abstract. The highly influential framework of conceptual spaces provides a geometric way of representing knowledge. It aims at bridging the gap between symbolic and subsymbolic processing. Instances are represented by points in a high-dimensional space and concepts are represented by convex regions in this space. In this paper, we present our approach towards grounding the dimensions of a conceptual space in latent spaces learned by an InfoGAN from unlabeled data.

1 Introduction

The cognitive framework of conceptual spaces [14, 15] attempts to bridge the gap between symbolic and subsymbolic AI by proposing an intermediate conceptual layer based on geometric representations. A conceptual space is a high-dimensional space spanned by a number of quality dimensions representing interpretable features. Convex regions in this space correspond to concepts. Abstract symbols can be grounded by linking them to concepts in a conceptual space whose dimensions are based on subsymbolic representations.

The framework of conceptual spaces has been highly influential in the last 15 years within cognitive science and cognitive linguistics [11, 13, 20]. It has also sparked considerable research in various subfields of artificial intelligence, ranging from robotics and computer vision [5–7] over the semantic web and ontology integration [1, 10] to plausible reasoning [9, 19].

Although this framework provides means for representing concepts, it does not consider the question of how these concepts can be learned from mostly unlabeled data. Moreover, the framework assumes that the dimensions spanning the conceptual space are already given a priori. In practical applications of the framework, they thus often need to be handcrafted by a human expert.

In this paper, we argue that by using neural networks, one can automatically extract the dimensions of a conceptual space from unlabeled data. We propose that latent spaces learned by an InfoGAN [8] (a special class of Generative Adversarial Networks [17]) can serve as domains in the conceptual spaces framework. We further propose to use a clustering algorithm in these latent spaces in order to discover meaningful concepts.

* Corresponding author, ORCID: 0000-0002-1962-1777

The remainder of this paper is structured as follows: Section 2 presents the framework of conceptual spaces and Section 3 introduces the InfoGAN framework. In Section 4, we present our idea of combining these two frameworks. Section 5 gives an illustrative example and Section 6 concludes the paper.

2 Conceptual Spaces

A conceptual space [14] is a high-dimensional space spanned by so-called "quality dimensions". Each of these dimensions represents an interpretable way in which two stimuli can be judged to be similar or different. Examples for quality dimensions include temperature, weight, time, pitch, and hue. A domain is a set of dimensions that inherently belong together. Different perceptual modalities (like color, shape, or taste) are represented by different domains. The color domain for instance can be represented by the three dimensions hue, saturation, and brightness.[1] Distance within a domain is measured by the Euclidean metric.

The overall conceptual space is defined as the product space of all dimensions. Distance within the overall conceptual space is measured by the Manhattan metric of the intra-domain distances. The similarity of two points in a conceptual space is inversely related to their distance – the closer two instances are in the conceptual space, the more similar they are considered to be.

The framework distinguishes properties like "red", "round", and "sweet" from full-fleshed concepts like "apple" or "dog": Properties are represented as regions within individual domains (e.g., color, shape, taste), whereas full-fleshed concepts span multiple domains. Reasoning within a conceptual space can be done based on geometric relationships (e.g., betweenness and similarity) and geometric operations (e.g., intersection or projection).

Recently, Balkenius & Gärdenfors [2] have argued that population coding in the human brain can give rise to conceptual spaces. They discuss the connection between neural and conceptual representations from a neuroscience/psychology perspective, whereas we take a machine learning approach in this paper.

3 Representation Learning with InfoGAN

Within the research area of neural networks, there has been some substantial work on learning compressed representations of a given feature space. Bengio et al. [3] provide a thorough overview of different approaches in the representation learning area. They define representation learning as "learning representations of the data that make it easier to extract useful information when building classifiers or other predictors". We will focus our discussion here on one specific approach that is particularly fitting to our proposal, namely InfoGAN [8]. InfoGAN is an extension of the GAN (Generative Adversarial Networks) framework [17] which has been applied to a variety of problems (e.g., [12, 18, 21–23]). We first describe the original GAN framework before moving on to InfoGAN.

[1] Of course, one can also use other color spaces, e.g., the CIE L*a*b* space.

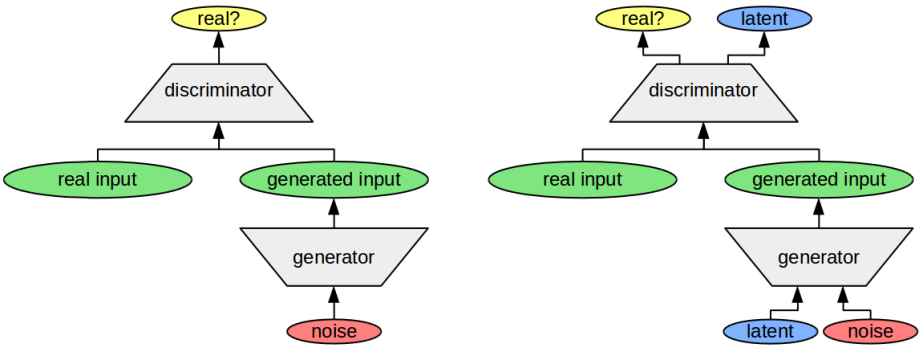

Fig. 1. Left: Illustration of a GAN. Right: Illustration of an InfoGAN.

The GAN framework (depicted in the left part of Figure 1) consists of two networks, the generator and the discriminator. The generator is fed with a low-dimensional vector of noise values. Its task is to create high-dimensional data vectors that have a similar distribution as real data vectors taken from an unlabeled training set. The discriminator receives a data vector that was either created by the generator or taken from the training set. Its task is to distinguish real inputs from generated inputs. Although the discriminator is trained on a classification task, the overall system works in an unsupervised way. The overall architecture can be interpreted as a two-player game: The generator tries to fool the discriminator by creating realistic inputs and the discriminator tries to avoid being fooled by the generator. When the GAN framework converges, the discriminator is expected to make predictions only at chance level and the generator is expected to create realistic data vectors. Although the overall framework works quite well, the dimensions of the input noise vector are usually not interpretable.

Chen et al. [8] have extended the original framework by introducing latent variables: In the InfoGAN framework (shown in the right part of Figure 1), the generator receives an additional input vector. The entries of this vector are values of latent random variables, selected based on some probability distribution that was defined a priori (e.g., uniform or Gaussian). The discriminator has the additional task to reconstruct these latent variables.[2] Chen et al. argue that this ensures that the mutual information between the latent variable vector and the generated data vector is high. They showed that after training an InfoGAN, the latent variables tend to have an interpretable meaning. For instance, in an experiment on the MNIST data set, the latent variables corresponded to type of digit, digit rotation and stroke thickness. InfoGANs can thus provide a bidirectional mapping between observable data vectors and interpretable latent dimensions: One can both extract interpretable dimensions from a given data vector and create a data vector from an interpretable latent representation.

[2] This introduces a structure similar to an autoencoder (with the latent variables as input/output and the generated data vector as hidden representation).

4 Using Representation Learning to Ground Domains

For some domains of a conceptual space, a dimensional representation is already available. For instance, the color domain can be represented by the three-dimensional HSB space. For other domains, it is however quite unclear how to represent them based on a handful of dimensions. One prominent example is the shape domain: To the best of our knowledge, there are no widely accepted dimensional models for describing shapes.

We propose to use the InfoGAN framework in order to learn such a dimensional representation based on an unlabeled data set: Each of the latent variables can be interpreted as one dimension of the given domain of interest. For instance, the latent variables learned on a data set of shapes can be interpreted as dimensions of the shape domain. Three important properties of domains in a conceptual space are the following: interpretable dimensions, a distance-based notion of similarity, and a geometric way of describing semantic betweenness. We think that the latent space of an InfoGAN is a good candidate for representing a domain of a conceptual space, because it fulfills all of the above requirements:

As described before, Chen et al. [8] found that the individual latent variables tend to have an interpretable meaning. Although this is only an empirical observation, we expect it generalize to other data sets and thus to other domains.

Moreover, the *smoothness assumption* used in representation learning (cf. [3] and [16, Ch. 15]) states that points with small distance in the input space should also have a small distance in the latent space. This means that a distance-based notion of similarity in the latent space is meaningful.

Finally, Radford et al. [18] found that linear interpolations between points in the latent space of a GAN correspond to a meaningful "morph" between generated images in the input space. This indicates that geometric betweenness in the latent space can represent semantic betweenness.

There are two important hyperparameters to the approach of grounding domains in InfoGANs: The number of latent variables (i.e., the dimensionality of the learned domain) and the type of distribution used for the latent variables (e.g., uniform vs. Gaussian). Note that one would probably aim for the lowest-dimensional representation that still describes the domain sufficiently well.

Finally, we would like to address a critical aspect of this proposal: How can one make sure that the representation learned by the neural network only represents information from the target domain (e.g., shape) and not anything related to other domains (e.g., color)? In our opinion, there are two complementary methods to "steer" the network towards the desired representation:

The first option consists of selecting only such inputs for the training set that do not exhibit major differences with respect to other domains. For instance, a training set for the shape domain should only include images of shapes that have the same color (e.g., black shape on white ground). If there is only very small variance in the data set with respect to other domains (e.g., color), the network is quite unlikely to incorporate this information into its latent representation.

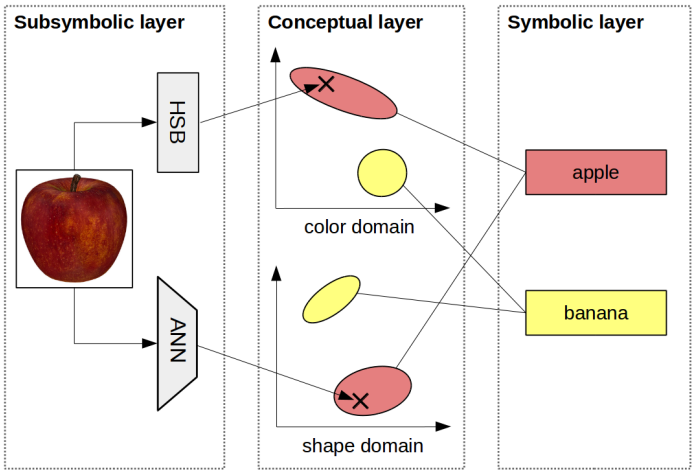

Fig. 2. Illustration of the envisioned overall system. Perceptions from the subsymbolic layer are transformed to points in different domains by feature extractors that are either handcrafted (e.g., HSB for colors) or trained neural networks (e.g., for the shape domain). Each concept is described by one region per domain.

The second option concerns modifications of the network's loss function: One could for instance introduce an additional term into the loss function which measures the correlation between the learned latent representation and dimensions from other (already defined) domains. This would cause a stronger error signal if the network starts to re-discover already known dimensions from other domains and therefore drive the network away from learning redundant representations.

A simple proof of concept implementation for the shape domain could be based on a data set of simple 2D shapes (circles, triangles, rectangles, etc.) in various orientations and locations. For a more thorough experiment, one could for instance use ShapeNet[3] [4], a data base of over 50,000 3D models for more than 50 categories of objects. One could render these 3D models from various perspectives in order to get 2D inputs (for learning to represent 2D shapes) or work on a voxelized 3D input (for learning representations of 3D shapes).

5 An Illustrative Example

Figure 2 illustrates a simplified example of our envisioned overall system. Here, we consider only two domains: color and shape. Color can be represented by the HSB space using the three dimensions hue, saturation and brightness. This is an example for a hard-coded domain. The representation of the shape domain, however, needs to be learned. The artificial neural network depicted in Figure

[3] https://www.shapenet.org/

2 corresponds to the discriminator of an InfoGAN trained on a data set of shapes.

Let us consider two example concepts: The concept of an apple can be described by the "red" region in the color domain and the "round" region in the shape domain. The concept of a banana can be represented by the "yellow" region in the color domain and the "cylindric" region in the shape domain.

If the system makes a new observation (e.g., an apple as depticted in Figure 2), it will convert this observation into a point in the conceptual space. For the color domain, this is done by a hard-coded conversion to the HSB color space. For the shape domain, the observation is fed into the discriminator and its latent representation is extracted, resulting in the coordinates for the shape domain. Now in order to classify this observation, the system needs to check whether the resulting data point is contained in any of the defined regions. If the data point is an element of the apple region in both domains (which is the case in our example), this observation should be classified as an apple. If the data point is an element of the banana region, the object should be classified as a banana.

Based on a new observation, the existing concepts can also be updated: If the observation was classified as an apple, but it is not close to the center of the apple region in one of the domains, this region might be enlarged or moved a bit, such that the observed instance is better matched by the concept description. If the observation does not match any of the given concepts at all, even a new concept might be created. This means that concepts cannot only be applied for classification, but they can also be learned and updated. Note that this can take place without explicit label information, i.e., in an unsupervised way. Our overall reserach goal is to develop a clustering algorithm that can take care of incrementally updating the regions in such a conceptual space.

Please note that the updates considered above only concern the connections between the conceptual and the symbolic layer. The connections between the subsymbolic and the conceptual layer remain fixed. The neural network thus only serves as a preprocessing step in our approach: It is trained before the overall system is used and remains unchanged afterwards. Simultaneous updates of both the neural network and the concept description might be desirable, but would probably introduce a great amount of additional complexity.

6 Conclusion and Future Work

In this paper, we outlined how neural representations can be used to ground the domains of a conceptual space in perception. This is especially useful for domains like shape, where handcrafting a dimensional representation is difficult. We argued that the latent representations learned by an InfoGAN have suitable properties for being combined with the conceptual spaces framework. In future work, we will implement the proposed idea by giving a neural grounding to the domain of simple 2D shapes. Furthermore, we will devise a clustering algorithm for discovering and updating conceptual representations in a conceptual space.

References

1. Benjamin Adams and Martin Raubal. Conceptual Space Markup Language (CSML): Towards the Cognitive Semantic Web. *2009 IEEE International Conference on Semantic Computing*, Sep 2009.
2. Christian Balkenius and Peter Gärdenfors. Spaces in the Brain: From Neurons to Meanings. *Frontiers in Psychology*, 7:1820, 2016.
3. Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.
4. Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. December 2015.
5. Antonio Chella, Haris Dindo, and Ignazio Infantino. Anchoring by Imitation Learning in Conceptual Spaces. *AI*IA 2005: Advances in Artificial Intelligence*, pages 495–506, 2005.
6. Antonio Chella, Marcello Frixione, and Salvatore Gaglio. Conceptual Spaces for Computer Vision Representations. *Artificial Intelligence Review*, 16(2):137–152, 2001.
7. Antonio Chella, Marcello Frixione, and Salvatore Gaglio. Anchoring Symbols to Conceptual Spaces: The Case of Dynamic Scenarios. *Robotics and Autonomous Systems*, 43(2-3):175–188, May 2003.
8. Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, 2016.
9. Joaquín Derrac and Steven Schockaert. Inducing Semantic Relations from Conceptual Spaces: A Data-Driven Approach to Plausible Reasoning. *Artificial Intelligence*, 228:66–94, Nov 2015.
10. Stefan Dietze and John Domingue. Exploiting Conceptual Spaces for Ontology Integration. In *Data Integration Through Semantic Technology (DIST2008) Workshop at 3rd Asian Semantic Web Conference (ASWC 2008)*, 2008.
11. Igor Douven, Lieven Decock, Richard Dietz, and Paul Égré. Vagueness: A Conceptual Spaces Approach. *Journal of Philosophical Logic*, 42(1):137–160, Nov 2011.
12. Ishan Durugkar, Ian Gemp, and Sridhar Mahadevan. Generative Multi-Adversarial Networks. November 2016.
13. Sandro R. Fiorini, Peter Gärdenfors, and Mara Abel. Representing Part-Whole Relations in Conceptual Spaces. *Cognitive Processing*, 15(2):127–142, Oct 2013.
14. Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. MIT press, 2000.
15. Peter Gärdenfors. *The Geometry of Meaning: Semantics Based on Conceptual Spaces*. MIT Press, 2014.
16. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
17. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. June 2014.
18. Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. November 2015.

19. Steven Schockaert and Henri Prade. Interpolation and Extrapolation in Conceptual Spaces: A Case Study in the Music Domain. *Lecture Notes in Computer Science*, pages 217–231, 2011.
20. Massimo Warglien, Peter Gärdenfors, and Matthijs Westera. Event Structure, Conceptual Spaces and the Semantics of Verbs. *Theoretical Linguistics*, 38(3-4), Jan 2012.
21. Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 82–90. Curran Associates, Inc., 2016.
22. Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based Generative Adversarial Network. September 2016.
23. Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. March 2017.

Propositional Rule Extraction from Neural Networks under Background Knowledge

Maryam Labaf[1,2], Pascal Hitzler[1], and Anthony B. Evans[2]

[1] Data Semantics (DaSe) Laboratory, Wright State University, Dayton, OH, USA
[2] Dept. of Math. and Stat., Wright State University, Dayton, OH, USA

Abstract. It is well-known that the input-output behaviour of a neural network can be recast in terms of a set of propositional rules, and under certain weak preconditions this is also always possible with positive (or definite) rules. Furthermore, in this case there is in fact a unique minimal (technically, *reduced*) set of such rules which perfectly captures the input-output mapping.
In this paper, we investigate to what extent these results and corresponding rule extraction algorithms can be lifted to take additional background knowledge into account. It turns out that uniqueness of the solution can then no longer be guaranteed. However, the background knowledge often makes it possible to extract simpler, and thus more easily understandable, rulesets which still perfectly capture the input-output mapping.

1 Introduction

The study of rule extraction from trained artificial neural networks [2,8,15] addresses the desire to make the learned knowledge accessible to human interpretation and formal assessment. Essentially, in the propositional case, activations of input and output nodes are discretized by introducing an arbitrary threshold. Each node is interpreted as a propositional variable, and activations above the threshold are interpreted as this variable being "true", while activations below the threshold are interpreted as this variable being "false". If $\mathcal{I}$ denotes the power set (i.e., set of all subsets) of the (finite) set $\mathcal{B}$ of all propositional variables corresponding to the nodes, then the input-output function of the network can be understood as a function $f : \mathcal{I} \to \mathcal{I}$: For $I \in \mathcal{I}$, we interpret each $p \in I$ as being "true" and all $p \notin I$ as being "false". The set $f(I)$ then contains exactly those propositional variables which are "true" (or activated) in the output layer.

In propositional rule extraction, one now seeks sets P_f of propositional rules (i.e., propositional Horn clauses) which capture or approximate the input-output mapping f. In order to obtain such sets, there exist two main lines of approaches. The first is introspective and seeks to construct rules out of the weights associated with the connections between nodes in the network, usually proceeding in a layer-by-layer fashion [8]. The second is to regard the network as a black box and to consider only the input-output function f. This was, e.g., done in [15] where it was shown, amongst other things, that a positive (or definite) ruleset can always be extracted if the mapping f is monotonic, and that there is indeed

a unique *reduced* such ruleset; we will provide sufficient technical details about these preliminary results in the next section.

However, rulesets extracted with either method are prone to be large and complex, i.e., from inspection of these rulesets it is often difficult to obtain real insights into what the network has learned. In this paper, we thus investigate rule extraction under the assumption that there is additional background knowledge which can be connected to network node activations, with the expectation that such background knowledge will make it possible to formulate simpler rulesets which still explain the input-output functions of the networks, if the background knowledge is also taken into account.

The motivation for this line of work is the fact that in recent years there has been a very significant increase in the availability of structured data on the World Wide Web, i.e., it becomes easier and easier to actually find such structured knowledge for all different kinds of application domains. That this is the case is, among other things, a result of recent developments in the field of *Semantic Web* [4,12], which is concerned with data sharing, discovery, integration and reuse, and where corresponding standards, methods and tools are being developed. E.g., structured data in the form of *knowledge graphs*, usually encoded using the W3C standards RDF [3] and OWL [11], has been made available in ever increasing quantities for over 10 years [5,17]. Other large-scale datasets include Wikidata [20] and data coming from the schema.org [9] effort which is driven by major Web search engine providers.

In order to motivate the rest of the paper, consider the following very simple example. Assume that the input-output mapping P of the neural network without background knowledge is

$$p_1 \wedge q \rightarrow r \qquad\qquad p_2 \wedge q \rightarrow r$$

and that we also have background knowledge K in form of the rules

$$p_1 \rightarrow p \qquad\qquad p_2 \rightarrow p.$$

We then obtain the simplified input-output mapping P_K, taking background knowledge into account, as

$$p \wedge q \rightarrow r.$$

The example already displays a key insight why background knowledge can lead to simpler extracted rulesets: In the example just given, p serves as a "more general" proposition, e.g., p_1 could stand for "is an apple" while p_2 could stand for "is a banana", while p could stand for "is a fruit". If we now also take, e.g., q to stand for "is ripe" and r to stand for "can be harvested", then we obtain a not-so-abstract toy example, where the background knowledge facilitates a simplification because it captures both apples and bananas using the more general concept "fruit".

In this paper, we will formally define the setting for which we just gave an initial example. We will furthermore investigate to what extent we can carry over results regarding positive rulesets from [15] to this new scenario with background

knowledge. We will see that the pleasing theoretical results such as uniqueness of a solution no longer hold. However, existence of solutions can still be guaranteed under the same mild conditions as in [15], and we will still be able to obtain algorithms for extracting corresponding rulesets.

The rest of the paper will be structured as follows. In Section 2 we will introduce notation as needed and recall preliminary results from [15]. In Section 3 we present the results of our investigation into adding background knowledge. In Section 4, we briefly discuss related work, and in Section 5 we conclude and discuss avenues for furture work.

2 Preliminaries

We recall notation and some results from [15] which will be central for the rest of the paper. For further background on notions concerning logic programs, cf. [13].

As laid out in the introduction, let $\mathcal{B}$ be a finite set of propositional variables, let $\mathcal{I}$ be the power set of $\mathcal{B}$, and we consider functions $f : \mathcal{I} \to \mathcal{I}$ as discretizations of input-output functions of trained neural networks. In this paper, we consider only positive (or definite) propositional rules, which are of the form $p_1 \wedge \cdots \wedge p_n \to q$, where q and all p_i are propositional variables. A set P of such rules is called a (propositional) logic program. For such a rule, we call q the *head* of the rule, and $p_1 \wedge \cdots \wedge p_n$ the *body* of the rule.

A logic program P is called *reduced* if all of the following hold.

1. For every rule $p_1 \wedge \cdots \wedge p_n \to q$ in P we have that all p_i are mutually distinct.
2. There are no two rules $p_1 \wedge \cdots \wedge p_n \to q$ and $r_1 \wedge \cdots \wedge r_m \to q$ in P with $\{p_1, \ldots, p_n\} \subseteq \{r_1, \ldots, r_m\}$.

To every propositional logic program P over $\mathcal{B}$ we can associate a semantic operator T_P, called the *immediate consequence operator*, which is the function

$$T_P : \mathcal{I} \to \mathcal{I} :$$
$$T_P(I) = \{q \mid \text{there exists } p_1 \wedge \cdots \wedge p_n \to q \text{ in } P \text{ with } \{p_1, \ldots, p_n\} \subseteq I\}.$$

This operator is well-known to be monotonic in the sense that whenever $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.

We make some additional mild assumptions: We assume that the propositional variables used to represent input and output nodes are distinct, i.e., each propositional variable gets used either to represent an input node, or an output node, but not both. Technically, this means that $\mathcal{B}$ can be partitioned into two sets $\mathcal{B}_1$ and $\mathcal{B}_2$, i.e., $\mathcal{B} = \mathcal{B}_1 \dot\cup \mathcal{B}_2$, and we obtain the corresponding power sets $\mathcal{I}_1$ and $\mathcal{I}_2$ such that $T_P : \mathcal{I}_1 \to \mathcal{I}_2$.

While the definition of the immediate consequence operator just presented is very common in the literature, we will now give a different but equivalent formalization, which will help us in this paper. For any $I = \{p_1, \ldots, p_n\} \subseteq \mathcal{B}$, let $c(I) = p_1 \wedge \cdots \wedge p_n$. In fact, whenever $I \subseteq \mathcal{B}$, in the following we will often simply write I although we may mean $c(I)$, and the context will make it clear

Algorithm 1: Reduced Definite Program Extraction

Input: A monotone mapping $f : \mathcal{I}_1 \to \mathcal{I}_2$.
Output: P, a definite logic program with $T_P(I) = f(I)$ for all $I \in \mathcal{I}_1$.

```
Initialization: P = ∅.
Choose a total linear order ≺ on I_1, such that for any I_i, I_j ∈ I_1 with i < j
  we have |I_i| < |I_j|.
for all I = {p_1, ..., p_n} ∈ I_1, chosen in ascending order according to ≺ do
    for all q ∈ f(I) do
        if there is no q_1 ∧ ··· ∧ q_n → q in P with {q_1, ..., q_n} ⊆ I then
            add the rule p_1 ∧ ··· ∧ p_n → q to P.
        end if
    end for
end for
Return P as result.
```

which notation is meant; e.g., if I appears as part of a logical formula, then we actually mean $c(I)$.

Now, given a logic program P and $I \in \mathcal{I}_1$, we obtain

$$T_P(I) = \{q \in \mathcal{B}_2 \mid I \wedge P \models q\},$$

where $\models$ denotes entailment in propositional logic. Please note that we use another common notational simplification, as $I \wedge P$ is used to denote $I \wedge \bigwedge_{R \in P} R$.

In [15], the following was shown.

Theorem 1. *Let $f : \mathcal{I}_1 \to \mathcal{I}_2$ be monotonic. Then there exists a unique reduced logic program P with $T_P = f$. Furthermore, this logic program can be obtained using Algorithm 1.*

If we drop the precondition on f to be monotonic, then Theorem 1 no longer holds, because of the fact mentioned above that immediate consequence operators are always monotonic.

We will now investigate Theorem 1 when considering additional background knowledge. It will be helpful to have the following corollary from Theorem 1 at hand.

Theorem 2. *Given a logic program P, there is always a unique reduced logic program Q with $T_P = T_Q$.*

Proof. Given P, we know that T_P is monotonic. Now apply Theorem 1.

Let us give an example for reducing a given program. Let $\mathcal{B}_1 = \{p_1, p_2, p_3\}$ and $\mathcal{B}_2 = \{q_1, q_2\}$ be input and output sets, respectively, and consider the logic program P given as

$$\begin{aligned} p_1 \wedge p_2 &\to q_1 & \qquad p_1 \wedge p_2 \wedge p_3 &\to q_1 \\ p_1 \wedge p_3 &\to q_1 & p_1 &\to q_2 \\ p_1 \wedge p_2 &\to q_2. \end{aligned}$$

Applying Algorithm 1 then yields the reduced program

$$p_1 \wedge p_2 \to q_1 \qquad p_1 \wedge p_3 \to q_1$$
$$p_1 \to q_2.$$

3 Rule extraction with Background Knowledge

We consider the following setting. Assume P is a logic program which captures the input-output function of a trained neural network according to Theorem 1. Let furthermore K be a logic program which constitutes our background knowledge, and which may use additional propositional variables, i.e., propositional variables not occurring in P. We then seek a logic program P_K such that, for all $I \in \mathcal{I}_1$, we have

$$\{q \in \mathcal{B}_2 \mid I \wedge P \models q\} = \{q \in \mathcal{B}_2 \mid I \wedge K \wedge P_K \models q\}. \tag{1}$$

In this case, we call P_K a *solution* for (P, K).

3.1 Existence of Solutions

We next make two more mild assumptions, namely (1) that no propositional variable from $\mathcal{B}_2$ appears in K, and that (2) propositional variables from $\mathcal{B}_1$ appear only in bodies of rules in K. The first is easily justified by the use case, since we want to explain the network behaviour, and the occurrence of variables from $\mathcal{B}_2$ in K would bypass the network. The second is also easily justified by the use case, which indicates that network input activations should be our starting point, i.e. the activations should not be altered by the background knowledge.

If we drop assumption (2) just stated, then existence of a solution cannot be guaranteed: Let $\mathcal{B}_1 = \{p_1, p_2\}$, let $\mathcal{B}_2 = \{q_1, q_2\}$. Then, for the given programs $P = \{p_1 \to q_1, p_2 \to q_2\}$ and $K = \{p_1 \to p_2\}$ there is no solution for (P, K). To see this, assume that P_K be a solution for (P, K). Then because $p_2 \wedge P \models q_2$ we obtain that $p_2 \wedge K \wedge P_K \models q_2$. But then $p_1 \wedge K \wedge P_K \models q_2$ although $p_1 \wedge P \not\models q_2$, i.e., P_K cannot be a solution for (P, K).

If condition (2) from above is assumed, though, a solution always exists.

Proposition 1. *Under our standing assumptions on given logic programs P and K, there always exists a solution for (P, K) which is reduced.*

Proof. Because rule heads from K never appear in P, we obtain

$$\{q \in \mathcal{B}_2 \mid I \wedge P \models q\} = \{q \in \mathcal{B}_2 \mid I \wedge K \wedge P \models q\}$$

for all $I \in \mathcal{I}_1$, i.e., P is always a solution for (P, K). Existence of a reduced solution then follows from Theorem 2.

Our interest of course lies in determining *other* solutions which are simpler than P.

Algorithm 2: Construct all reduced solutions for (P, K)

Input: Logic programs P and K with $T_P : \mathcal{I}_1 \to \mathcal{I}_2$ and $T_K : \mathcal{I}_1 \to \mathcal{I}_3$ which satisfy our standing assumptions, where $\mathcal{B}_2 = \{q_1, \ldots, q_n\}$.
Output: All the reduced solutions for (P, K).

Set $\mathcal{S} = \emptyset$ and $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_3$.
Set $\mathcal{I}$ to be the power set of $\mathcal{B}$.
Set $\mathcal{R}$ to be the power set of $\mathcal{I}$.
for all $(R_1, \ldots, R_n) \in \mathcal{R}^n$ **do**
  for all $i \in \{1, \ldots, n\}$ **do**
    $Q_i = \{c(B) \to q_i \mid B \in R_i\}$
  end for
  Set $Q = \bigcup_{i \in \{1,\ldots,n\}} Q_i$.
  if T_Q is a solution for (P, K) **then**
    Apply Algorithm 1 to T_Q to obtain a reduced program S with $T_S = T_Q$.
    if $S \notin \mathcal{S}$ **then**
      Add S to $\mathcal{S}$.
    end if
  end if
end for
Return $\mathcal{S}$ as result.

Proposition 2. *There exist logic programs P and K which satisfy our standing assumptions, such that there are two distinct reduced solutions for (P, K).*

Proof. Let $\mathcal{B}_1 = \{p_1, p_2, p_3\}$ and $\mathcal{B}_2 = \{q\}$. Then consider the programs P as

$$p_2 \wedge p_3 \to q \qquad\qquad p_1 \wedge p_3 \to q$$

and K as

$$\begin{aligned} p_2 \wedge p_3 &\to r_1 & \qquad p_1 \wedge p_3 &\to r_1 \\ p_1 &\to r_2 & \qquad p_2 &\to r_2. \end{aligned}$$

The two logic programs

$$\begin{aligned} P_{K_1} &= \{r1 \to q\} \qquad \text{and} \\ P_{K_2} &= \{p_3 \wedge r_2 \to q\} \end{aligned}$$

are then both reduced solutions for (P, K).

We will see later in the proof of Theorem 3, that the number of reduced solutions is actually worst-case exponential in the combined size of P and K.

3.2 Algorithms

We first present a naive algorithm for computing all reduced solutions for given (P, K). It is given as Algorithm 2 and it uses a brute-force approach to

check all possible logic programs which can be constructed over the given propositional variables, whether they constitute a solution for (P, K). For each such solution, it then invokes Algorithm 1 to obtain a corresponding reduced program, which is then added to the solution set. The algorithm is quite obviously correct and always terminating, and we skip a formal proof of this.

The given algorithm is of course too naive to be practically useful for anything other than toy examples. Still, it is worst-case optimal, as the following theorem shows – note that Algorithm 2 has exponential runtime because of line 4.

Theorem 3. *The problem of finding all solutions to (P, K) is worst-case exponential in the combined size of P and K.*

Proof. Let n be any positive integer. Define the logic program P_n to consist of the single rule $p_1 \wedge \cdots \wedge p_n \to q$ and let

$$K_n = \{p_i \to r_{i,1}, p_i \to r_{i,2} \mid i = 1, \ldots, n\}.$$

Then, for any function $f : \{1, \ldots, n\} \to \{1, 2\}$, the logic program

$$P_f = \{r_{1,f(1)} \wedge \cdots \wedge r_{n,f(n)} \to q\}$$

is a reduced solution for (P_n, K_n). Since there exist 2^n distinct such functions f, the number of reduced solutions in this case is 2^n, so their production is exponential in n, while the combined size of P_n and K_n grows only linearly in n.

A more efficient algorithm for obtaining only *one* reduced solution is given as Algorithm 3. It is essentially a combination of Algorithms 1 and 2.

Proposition 3. *Algorithm 3 is correct and always terminating.*

Proof. Like Algorithm 1, Algorithm 3 checks all combinations of $I \in \mathcal{I}_1$ and $q \in T_P(I)$ and makes sure that there are rules in the output program such that $I \wedge K \wedge S \models q$. The rules for the output program are checked one by one in increasing length until a suitable one is found. Note that the rule $I \to q$ is going to be checked at some stage, i.e. the algorithm will either choose this rule, or a shorter one, but in any case we will eventually have $I \wedge K \wedge S \models q$. This shows that the algorithm always terminates and that we obtain $I \wedge K \wedge S \models q$ for all $q \in T_P(I)$.

In order to demonstrate that the algorithm output S is indeed a solution for (P, K), we also need to show that for all $q \in \mathcal{B}_2$ and $H \in \mathcal{I}_1$ we have that $H \wedge K \wedge S \models q$ implies $q \in T_P(H)$. This is in fact guaranteed by line 11 of Algorithm 3, i.e. the algorithm output S is indeed a solution for (P, K).

We finally show that the output of the algorithm is reduced. Assume otherwise. Then there are $I_1 \to q$ and $J \to q$ in S with $I_1 \subsetneq J$. By our condition on the order we thus have $I_1 \prec J$ and so we know that $I_1 \to q$ was added to S earlier in the algorithm than $J \to q$. now let us look at the instance of line 12 in Algorithm 3 when the rule $J \to q$ was added to S. In this case (using notation from the algorithm description, and S denoting the current S at that

Algorithm 3: Reduced solution for (P, K)

Input: Logic programs P and K with $T_P : \mathcal{I}_1 \to \mathcal{I}_2$ and $T_K : \mathcal{I}_1 \to \mathcal{I}_3$ which satisfy our standing assumptions.
Output: A reduced solution for (P, K).

Set $S = \emptyset$ and $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_3$.
Set $\mathcal{I}$ to be the power set of $\mathcal{B}$.
Choose a total linear order $\prec$ on $\mathcal{I}$, such that for any $I_i, I_j \in \mathcal{I}$ with $i < j$ we have $|I_i| < |I_j|$.
for all $I = \{p_1, \ldots, p_m\} \in \mathcal{I}_1$, chosen in ascending order according to $\prec$ **do**
  for all $q \in T_P(I)$ **do**
    if $I \wedge K \wedge S \not\models q$ **then**
      Set endloop = false.
      Choose first $J = \{b_1, \ldots, b_n\} \in \mathcal{I}$ according to $\prec$.
      while endloop = false **do**
        if $I \wedge K \wedge S \wedge (J \to q) \models q$ **then**
          if $\{H \in \mathcal{I}_1 \mid H \wedge K \wedge S \wedge (J \to q) \models q\} \subseteq \{H \in \mathcal{I}_1 \mid q \in T_P(H)\}$ **then**
            Add the rule $J \to q$ to S and set endloop = true.
          end if
        else
          Choose next $J = \{b_1, \ldots, b_n\} \in \mathcal{I}$ according to $\prec$.
        end if
      end while
    end if
  end for
end for
Return S as a result.

moment) we know that $I \wedge K \wedge S \wedge (J \to q) \models q$ and $I \wedge K \wedge S \not\models q$. This implies $I \wedge K \wedge S \models J$, and because $I_1 \subseteq J$ we obtain $I \wedge K \wedge S \models I_1$. But we also have already observed that $I_1 \to q$ is already contained in S at this stage, and thus we obtain $I \wedge K \wedge S \models q$, which contradicts the earlier statement that $I \wedge K \wedge S \not\models q$. We thus have to reject the assumption that S is not reduced; hence S is indeed reduced. This completes the proof.

To close, we give a somewhat more complex example. Let $\mathcal{B}_1 = \{p_1, p_2, p_3\}$ and $\mathcal{B} = \{q_1, q_2, q_3, q_4\}$. Consider the program P as

$$\begin{array}{rr} p_1 \to q_1 & p_2 \to q_1 \\ p_1 \to q_2 & p_2 \to q_2 \\ p_1 \to q_3 & p_2 \to q_3 \\ p_1 \to q_4 & p_2 \wedge p_3 \to q_4 \end{array}$$

and K as

$$p_1 \rightarrow r_1 \qquad p_2 \wedge p_3 \rightarrow r_1$$
$$p_1 \rightarrow r_2 \qquad p_2 \rightarrow r_2$$
$$p_2 \rightarrow r_3.$$

Then there is only one reduced solution P_K for (P, K), which is

$$r_2 \rightarrow q_1 \qquad r_2 \rightarrow q_2$$
$$r_2 \rightarrow q_3 \qquad r_1 \rightarrow q_4.$$

Note, that P_K is simpler and shorter than P.

4 Related Work

It would be out of place to have a lengthy discussion of related work in neural-symbolic integration, or even just on the topic of rule extraction, in this brief paper. We hence limit ourselves to some key pointers including overview texts. We already discussed the rule-extraction work [15] on which our work is based, and [8] which pursues a different approach based on inspecting weights. For more extensive entry points to literature on neural-symbolic integration we refer to [2,6,7,10] and to the proceedings of the workshop series on Neural-Symbolic Learning and Reasoning.[3]

Regarding the novel aspect of this work, namely the utilization of background knowledge for rule extraction, we are not aware of any prior work which pursues this. However, concurrently the second author has worked on lifting the idea to the application level in [19], by utilizing description logics and Semantic Web background knowledge in the form of ontologies and knowledge graphs [12] together with the DL-Learner system [16] for rule extraction. The results herein, which are constrained to the propositional case, can be considered foundational for the more application-oriented work currently pursued along the lines of [19].

We are also greatful that a reviewer pointed out a possible relationship of our work with work laid out in [18] in the context of abduction in logic programming. Looked at on a very generic level, the general abduction task is very similar to our formulation in equation (1), which means that the field of abduction may indeed provide additional insights or even algorithms for our setting. On the detail level, however, [18] differs significantly. Most importantly, [18] consideres literals or atoms as abducibles, i.e., an explanation consists of a set of literals, while in our setting explanations are actually rule sets. Another difference is that [18] considers logic programs under the non-monotonic answer set semantics, i.e., logic programs with default negations, while we consider only logic programs without negation in our work – it was laid out in much detail in [15] that propositional rule extraction under negation has a significantly different dynamics. Nevertheless, the general field of abduction in propositional logic programming may provide inspiration for further developing our approach, but working out the exact relationships appears to be a more substantial investigation.

[3] `http://neural-symbolic.org/`

5 Conclusions and Further Work

We have investigated the issue of propositional rule extraction from trained neural networks under background knowledge, for the case of definite rules. We have shown that a mild assumption on the background knowledge and monotonicity of the input-output function of the network suffices to guarantee that a reduced logic program can be extracted such that the input-output function is exactly reproduced. We have also shown that the solution is not unique. Furthermore, we have provided algorithms for obtaining corresponding reduced programs.

We consider our results to be foundational for further work, rather than directly applicable in practice. Our observation that background knowledge can yield simpler extracted rulesets of course carries over to more expressive logics which extend propositional logic.

It is such extensions which we intend to pursue, which hold significant promise for practical applicability: structured information on the World Wide Web, as discussed in the Introduction, is provided in logical forms which are usually non-propositional fragments of first-order predicate logic, or closely related formalisms. In particular, description logics [1], i.e. decidable fragments of first-order predicate logic, form the foundation of the Web Ontology Language OWL. First-order rules are also commonly used [14]. This raises the question how to extract meaningful non-propositional rules from trained neural networks while taking (non-propositional) background knowledge, in a form commonly used on the World Wide Web, into account.

Acknowledgements. The first two authors acknowledge support by the Ohio Federal Research Network project *Human-Centered Big Data.*

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edn. (2010)
2. Bader, S., Hitzler, P.: Dimensions of neural-symbolic integration – A structured survey. In: Artëmov, S.N., Barringer, H., d'Avila Garcez, A.S., Lamb, L.C., Woods, J. (eds.) We Will Show Them! Essays in Honour of Dov Gabbay, Volume One. pp. 167–194. College Publications (2005)
3. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1. Turtle – Terse RDF Triple Language. W3C Recommendation (25 February 2014), available at http://www.w3.org/TR/turtle/
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (May 2001)
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story So Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
6. d'Avila Garcez, A., 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: McCallum, A., Gabrilovich, E., Guha, R., Murphy, K. (eds.) Proceedings of the AAAI 2015

Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches. AAAI Press Technical Report, vol. SS-15-03. AAAI Press, Palo Alto (2015)
7. d'Avila Garcez, A.S., Lamb, L.C., Gabbay, D.M.: Neural-Symbolic Cognitive Reasoning. Cognitive Technologies, Springer (2009)
8. d'Avila Garcez, A.S., Zaverucha, G.: The connectionist inductive lerarning and logic programming system. Applied Intelligence 11(1), 59–77 (1999)
9. Guha, R.V., Brickley, D., Macbeth, S.: Schema.org: evolution of structured data on the web. Commun. ACM 59(2), 44–51 (2016)
10. Hammer, B., Hitzler, P. (eds.): Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence, vol. 77. Springer (2007)
11. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S. (eds.): OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation (11 December 2012), http://www.w3.org/TR/owl2-primer/
12. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. CRC Press/Chapman & Hall (2010)
13. Hitzler, P., Seda, A.K.: Mathematical Aspects of Logic Programming Semantics. CRC Press/Chapman and Hall (2010)
14. Krisnadhi, A., Maier, F., Hitzler, P.: OWL and rules. In: Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., Patel-Schneider, P.F. (eds.) Reasoning Web. Semantic Technologies for the Web of Data – 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures. Lecture Notes in Computer Science, vol. 6848, pp. 382–415. Springer (2011)
15. Lehmann, J., Bader, S., Hitzler, P.: Extracting reduced logic programs from artificial neural networks. Appl. Intell. 32(3), 249–266 (2010)
16. Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Machine Learning 78(1-2), 203–250 (2010)
17. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web 6(2), 167–195 (2015)
18. Lin, F., You, J.: Abduction in logic programming: A new definition and an abductive procedure based on rewriting. Artif. Intell. 140(1/2), 175–205 (2002)
19. Sarker, M.K., Xie, N., Doran, D., Raymer, M., Hitzler, P.: Explaining trained neural networks with semantic web technologies: First steps. In: Proceedings of the Twelveth International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'17, London, UK, July 2017 (2017), to appear
20. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM 57(10), 78–85 (2014)

Classical Planning in Deep Latent Space: From Unlabeled Images to PDDL (and back)

Masataro Asai, Alex Fukunaga

guicho2.71828@gmail.com
Graduate School of Arts and Sciences
University of Tokyo

Abstract. Current domain-independent, classical planners require symbolic models of the problem domain and instance as input, resulting in a knowledge acquisition bottleneck. Meanwhile, although recent work in deep learning has achieved impressive results in many fields, the knowledge is encoded in a subsymbolic representation which cannot be directly used by symbolic systems such as planners. We propose LatPlan, an integrated architecture combining deep learning and a classical planner. Given a set of unlabeled training image pairs showing allowed actions in the problem domain, and a pair of images representing the start and goal states, LatPlan uses a Variational Autoencoder to generate a discrete latent vector from the images, based on which a PDDL model can be constructed and then solved by an off-the-shelf planner. We evaluate LatPlan using image-based versions of 3 planning domains: 8-puzzle, LightsOut, and Towers of Hanoi.

1 Introduction

Recent advances in domain-independent planning have greatly enhanced their capabilities. However, planning problems need to be provided to the planner in a structured, symbolic representation such as PDDL [22], and in general, such symbolic models need to be provided by a human, either directly in PDDL, or via a compiler which transforms some other symbolic problem representation into PDDL. This results in the *knowledge-acquisition bottleneck*, where the modeling step is sometimes the bottleneck in the problem solving cycle. In addition, the requirement for symbolic input poses a significant obstacle to applying planning in *new, unforeseen* situations where no human is available to create such a model, e.g., autonomous spacecraft exploration. This first requires generating symbols from raw sensor input, i.e., the *symbol grounding problem* [30].

Recently, significant advances have been made in neural network (NN) approaches for cognitive tasks including image classification [8], object recognition [26], speech recognition [9], machine translation as well as NN-based problem-solving systems [23, 10]. However, the current state-of-the-art in pure NN-based systems do not yet provide guarantees provided by symbolic planning systems, such as deterministic completeness and solution optimality.

Fig. 1. An image-based 8-puzzle.

Using a NN-based perceptual system to *automatically* provide input models for domain-independent planners could greatly expand the applicability of planning technology and offer the benefits of both paradigms. *We consider the problem of robustly, automatically bridging the gap between such symbolic and subsymbolic representations.*

Fig. 1 (left) shows a scrambled, 3x3 tiled version of the the photograph on the right, i.e., an image-based instance of the 8-puzzle. We seek a domain-independent system which, given only a set of unlabeled images showing the valid moves for this image-based puzzle, finds an optimal solution to the puzzle. Although the 8-puzzle is trivial for symbolic planners, solving this image-based problem with a domain-independent system which *has no prior assumptions/knowledge* (e.g., "sliding objects", "tile arrangement", "a grid-like structure") is nontrivial. The only assumption allowed about the nature of the task is that it can be modeled and solved as a classical planning problem.

We propose Latent-space Planner (LatPlan), a hybrid architecture which uses NN-based image processing to completely automatically generate a propositional, symbolic problem representation which can be used as the input for a classical planner. LatPlan consists of 3 components: (1) a NN-based *State Autoencoder* (SAE), which provides a bidirectional mapping between the raw input of the world states and its symbolic/categorical representation, (2) an *action model generator* which generates a PDDL model using the symbolic representation acquired by the SAE, and (3) a symbolic planner. Given only a set of unlabeled images from the domain as input, we train (unsupervised) the SAE and use it to generate D, a PDDL representation of the image-based domain. Then, given a planning problem instance as a pair of initial and goal images such as Fig. 1, LatPlan uses the SAE to map the problem to a symbolic planning instance in D, and uses the planner to solve the problem.

2 LatPlan: System Architecture

This section describes the LatPlan architecture and the current implementation, LatPlanα. LatPlan works in 3 phases. In Phase 1 (symbol-grounding), a State AutoEncoder providing a bidirectional mapping between raw data (e.g., images) and symbols is learned (unsupervised) from a set of unlabeled images of representative states. In Phase 2 (action model generation), the operators available in the domain is generated from a set of pairs of unlabeled images, and a PDDL domain model is generated. In Phase 3 (planning), a planning problem instance

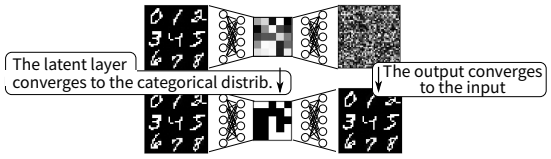

Fig. 2. Step 1: Train the State Autoencoder by minimizing the sum of the reconstruction loss (binary cross-entropy) and the variational loss of Gumbel-Softmax.

is input as a pair of images (i, g) where i shows an *initial state* and g shows a *goal state*. These are converted to symbolic form using the SAE, and the problem is solved by the symbolic planner. For example, an 8-puzzle problem instance in our system consists of an image of the start (scrambled) configuration of the puzzle (i), and an image of the solved state (g). Finally, the symbolic, latent-space plan is converted to a human-comprehensible visualization of the plan.

Symbol Grounding with a State Autoencoder The State Autoencoder (SAE) provides a bidirectional mapping between images and a symbolic representation.

First, note that a direct 1-to-1 mapping between images and discrete objects can be trivially obtained simply by using the array of discretized pixel values as a "symbol". However, such a trivial SAE lacks the crucial properties of *generalization* – ability to encode/decode unforeseen world states to symbols – and *robustness* – two similar images that represent "the same world state" should map to the same symbolic representation. Thus, we need a mapping where the symbolic representation captures the "essence" of the image, not merely the raw pixel vector. The main technical contribution of this paper is the proposal of a SAE which is implemented as a Variational Autoencoder [16] with a Gumbel-Softmax (GS) activation function [14].

Gumbel-Softmax (GS) activation is a recently proposed reparametrization trick [14] for categorical distribution. Using GS in the network in place of standard activation functions (Sigmoid, Softmax, ReLU) forces the activation to converge to a discrete one-hot vector. GS has a "temperature" parameter τ which controls the magnitude of approximation. τ is annealed by a schedule $\tau \leftarrow \max(0.1, \exp(-rt))$ where t is the current training epoch and r is an annealing ratio [14]. We chose r so that $\tau = 0.1$ when the training finishes.

In our implementation, the SAE is comprised of multilayer perceptrons combined with Dropouts and Batch Normalization in both the encoder and the decoder networks, with a GS layer in between. The input to the GS layer is the flat, last layer of the encoder network. The output is an (N, M) matrix where N is the number of categorical variables and M is the number of categories.

Our key observation is that these categorical variables can be used directly as propositional symbols by a symbolic reasoning system, i.e., this provides a solution to the symbol grounding problem in our architecture. We specify $M = 2$, effectively obtaining N propositional state variables. It is possible to specify

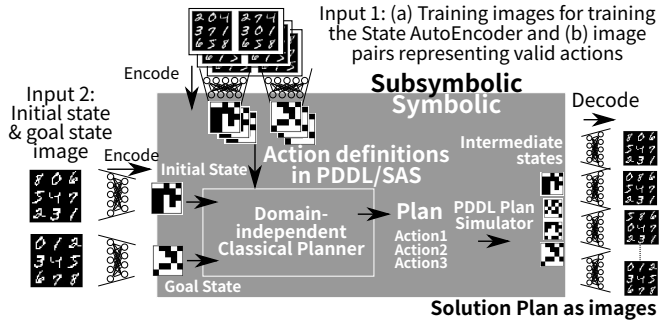

Fig. 3. Classical planning in latent space: We use the learned State AutoEncoder to convert pairs of images $(pre, post)$ first to symbolic ground actions and then to a PDDL domain. We also encode initial and goal state images into a symbolic ground actions and then a PDDL problem. A classical planner finds the symbolic solution plan. Finally, intermediate states in the plan are decoded back to a human-comprehensible image sequence.

different M for each variable and represent the world using multi-valued representation as in SAS+ [3] but we always use $M = 2$ for simplicity.

The trained SAE provides bidirectional mapping between the raw inputs (subsymbolic representation) to and from their symbolic representations:

- $b = Encode(r)$ maps an image r to a boolean vector b.
- $\tilde{r} = Decode(b)$ maps a boolean vector b to an image $\tilde{r}$.

$Encode(r)$ maps raw input r to a symbolic representation by feeding the raw input to the encoder network, extract the activation in the GS layer, and take the first row in the $N \times 2$ matrix, resulting in a binary vector of length N. Similarly, $Decode(b)$ maps a binary vector b back to an image by concatenating b and its complement $\bar{b}$ to obtain a $N \times 2$ matrix and feeding it to the decoder.

It is *not* sufficient to use traditional activation functions such as softmax and round the activation values to obtain discrete 0/1 values because we need to map the symbolic plan back to images. We need a decoding network trained for 0/1 values approximated by a smooth function, e.g., GS or similar approach such as [21]. A rounding-based scheme would be unable to restore the images from discrete values because the decoder is trained using continuous values. Also, the rounding operation cannot be part of a backpropagated network because rounding is non-differentiable.

An SAE trained on a small fraction of the possible states successfully generalizes so that it can *Encode* and *Decode* every possible state in that domain. In all our experiments below, we train the SAE using randomly selected images from the domain. For example, on the 8-puzzle, the SAE trained on 12000 randomly generated configurations out of 362880 possible configurations is used by the domain model generator to *Encode* every 8-puzzle state.

Domain Model Generation The model generator takes as input a trained

SAE, and a set R contains pairs of raw images. In each image pair $(pre_i, post_i) \in R$, pre_i and $post_i$ are images representing the state of the world before and after some action a_i is executed, respectively. In each ground action image pair, the "action" is implied by the difference between pre_i and $post_i$. The output of the model generator is a PDDL domain file for a grounded unit-cost STRIPS planning problem. For each (pre_i, $post_i$) $\in R$ we apply the learned SAE to pre_i and $post_i$ to obtain ($Encode(pre_i)$, $Encode(post_i)$), the symbolic representations (latent space vectors) of the state before and after action a_i is executed. This results in a set of symbolic ground action instances A.

Ideally, a model generation component would induce a complete action model from a limited set of symbolic ground action instances. However, *action model learning* from a limited set of action instances is a nontrivial area of active research [7, 11, 18, 24, 32, 6]. Since the focus of this paper is on the overall LatPlan architecture and the SAE, we leave model induction for future work. Instead, the current implementation LatPlanα uses a trivial, baseline strategy which generates a model based on *all* ground actions, which are supposed to be easily replaced by existing off-the-shelf action model learner. In this baseline method, R contains image pairs representing all ground actions that are possible in this domain, so $A = \{Encode(r) | r \in R\}$ contains all symbolic ground actions possible in the domain. In Sec. 5, we further discuss the implication and the impact of this model. In the experiments (Sec. 3), we generate image pairs for all ground actions using an external image generator. It is important to note that while R contains all possible actions, R is not used for training the SAE. As explained before, the SAE is trained using at most 12000 images while the entire state space is much larger.

LatPlanα compiles A directly into a PDDL model as follows. For each action ($Encode(pre_i)$, $Encode(post_i)$) $\in A$, each bit $b_j (1 \leq j \leq N)$ in these boolean vectors is mapped to propositions `(b`$_j$`-true)` and `(b`$_j$`-false)` when the encoded value is 1 and 0 (resp.). $Encode(pre_i)$ is directly used as the preconditions of action a_i. The add/delete effects of action i are computed by taking the bit-wise difference between $Encode(pre_i)$ and $Encode(post_i)$. For example, when b_j changes from 1 to 0, it compiles into `(and (b`$_j$`-false) (not (b`$_j$`-true)))`. The initial and the goal states are similarly created by applying the SAE to the initial and goal images.

Planning with an Off-the-Shelf Planner The PDDL instance generated in the previous step can be solved by an off-the-shelf planner. LatPlanα uses the Fast Downward planner [12]. However, on the models generated by LatPlanα, the invariant detection routines in the Fast Downward PDDL-SAS converter became a bottleneck, so we wrote a trivial, replacement PDDL-SAS converter without the invariant detection.

LatPlan inherits all of the search-related properties of the planner which is used. For example, if the planner is complete and optimal, LatPlan will find an optimal plan for the given problem (if one exists), with respect to the portion of the state-space graph captured by the acquired model. Domain-independent heuristics developed in the planning literature are designed to exploit structure

in the domain model. Although the structure in models acquired by LatPlan may not directly correspond to those in hand-coded models, intuitively, there should be some exploitable structure. The search results in Sec. 3 suggest that the domain-independent heuristics can reduce the search effort.

Visualizing/Executing the Plans Since the actions comprising the plan are SAE-generated latent bit vectors, the "meaning" of each symbol (and thus the plan) is not necessarily clear to a human observer. However, we can obtain a step-by-step visualization of the world (images) as the plan is executed (e.g. Fig. 4) by starting with the latent state representation of the initial state, applying (simulating) actions step-by-step (according to the PDDL model acquired above) and *Decode*'ing the latent bit vectors for each intermediate state to images using the SAE. In this paper, a "mental image" of the solution (i.e., the image sequence visualization) is sufficient. In a less simplified setting, mapping the actions found by LatPlan (transitions between latent bit vector pairs) to lower-level actuation would be necessary (future work).

3 Experimental Evaluation

All of the SAE networks used in the evaluation have the same network topology except the input layer which should fit the size of the input images. The network consists of the following layers: [Input, GaussianNoise(0.1), fc(4000), relu, bn, dropout(0.4), fc(4000), relu, bn, dropout(0.4), fc(49x2), GumbelSoftmax, dropout(0.4), fc(4000), relu, bn, dropout(0.4), fc(4000), relu, bn, dropout(0.4), fc(*input*), sigmoid]. Here, fc = fully connected layer, bn = Batch Normalization, and tensors are reshaped accordingly. The last layers can be replaced with [fc($input \times 2$), GumbelSoftmax, TakeFirstRow] for better reconstruction when we can assume that the input image is binarized. The network is trained using Adam optimizer (lr:0.001) for 1000 epochs.

The latent layer has 49 bits, which sufficiently covers the total number of states in any of the problems that are used in the following experiments. This could be reduced for each domain (made more compact) with further engineering.

MNIST 8-puzzle This is an image-based version of the 8-puzzle, where tiles contain hand-written digits (0-9) from the MNIST database [20]. Each digit is shrunk to 14x14 pixels, so each state of the puzzle is a 42x42 image. Valid moves in this domain swap the "0" tile with a neighboring tile, i.e., the "0" serves as the "blank" tile in the classic 8-puzzle. The entire state space consists of 362880 states (9!). Note that the same image is used for each digit in all states, e.g., the "1" digit is the same image in all states.

Out of 362880 images, 12000 randomly selected images are used for training the SAE. This set is further divided into a training set (11000) and a validation set (1000). Training takes 40 minutes/1000 epochs on a NVIDIA GTX-1070.

Scrambled Photograph 8-puzzle The above MNIST 8-puzzle described above consists of images where each digit is cleanly separated from the black region. To show that LatPlan does not rely on cleanly separated objects, we solve 8-puzzles generated by cutting and scrambling real photographs (similar to

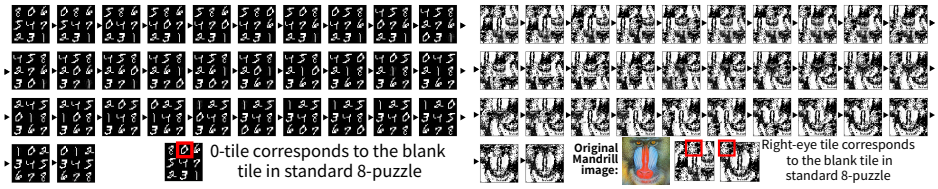

Fig. 4. (Left) Output of solving the MNIST 8-puzzle instance with the longest (31 steps) optimal plan. [Reinefeld 1993] (Right) Output of solving a photograph-based 8-puzzle (Mandrill). We emphasize that LatPlan has no built-in notion of "sliding object", or "tile arrangement"; furthermore, the SAE is being trained completely from scratch when LatPlan is applied to this scrambled photograph puzzle – there is no transfer/reuse of knowledge from the SAE learned for the MNIST 8-puzzle.

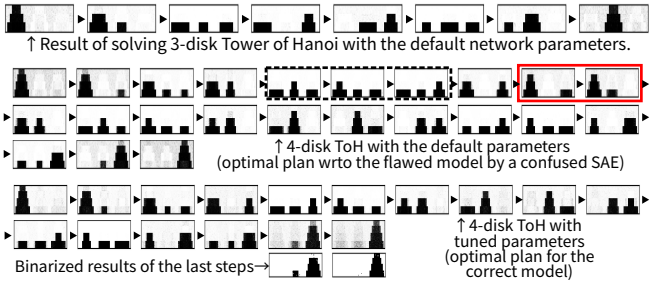

Fig. 5. Output of solving ToH with 3 and 4 disks. The third picture is the result of SAE with different parameters.

sliding tile puzzle toys sold in stores). We used the "Mandrill" image, a standard benchmark in the image processing literature. The image was first converted to greyscale and then rounded to black/white (0/1) values. The same number of images as in the MNIST-8puzzle experiments are used.

Towers of Hanoi (ToH) Disks of various sizes must be moved from one peg to another, with the constraint that a larger disk can never be placed on top of a smaller disk. Due to the smaller number of states (3^d states for d disks), we used images of all states as the set of images for training SAE. This is further divided into the training set (90%) and the validation set (10%), and we verified that the network has learned a generalized model without overfitting.

3-disk ToH is solved successfully and optimally using the default hyperparameters (Fig. 5, top). However, on 4-disks, the SAE trained with the default hyperparameters (Fig. 5, middle) is confused, resulting in a flawed model which causes the planner to choose suboptimal moves (dashed box). Sometimes, the size/existence of disks is confused (red box). Tuning the hyperparameters to reduce the SAE loss corrects this problem. After increasing the training epochs (10000) and tuning the network shape (fc(6000), $N = 29$), the SAE generated a correct model, resulting in the optimal 15-step plan (Fig. 5, bottom).

Fig. 6. Output of solving 4x4 LightsOut (left) and its binarized result (right). Although the goal state shows two blurred switches, they have low values (around 0.3) and disappear after rounding.

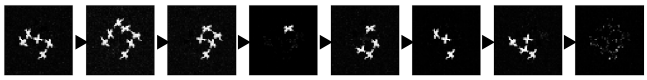

Fig. 7. Output of solving 3x3 Twisted LightsOut.

LightsOut A video game where a grid of lights is in some on/off configuration (+: On), and pressing a light toggles its state (On/Off) as well as the state of all of its neighbors. The goal is all lights Off. Unlike the 8-puzzle where each move affects only two adjacent tiles, a single operator in 4x4 LightsOut can simultaneously flip 5/16 locations. Also, unlike 8-puzzle and ToH, the LightsOut game allows some "objects" (lights) to disappear. This demonstrates that LatPlan is not limited to domains with highly local effects and static objects.

Twisted LightsOut In all of the above domains, the "objects" correspond to rectangles. To show that LatPlan does not rely on rectangular regions, we demonstrate its result on "Twisted LightsOut", a distorted version of the game where the original LightsOut image is twisted around the center. Unlike previous domains, the input images are not binarized.

Robustness to Noisy Input We show the robustness of the system against the input noise. We corrupted the initial/goal state inputs by adding Gaussian or salt noise, as shown in Fig. 8. The system is robust enough to successfully solve the problem, because our SAE is a Denoising Autoencoder [31] which has an internal *GaussianNoise layer* which adds a Gaussian noise to the inputs (only during training) and learn to reconstruct the original image from a corrupted version of the image.

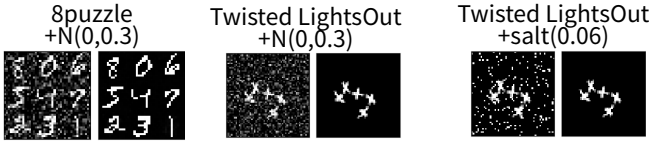

Fig. 8. SAE robustness vs noise: Corrupted initial state image r and its reconstruction $Decode(Encode(r))$ by SAE on MNIST 8-puzzle and Twisted LightsOut. Images are corrupted by Gaussian noise of σ up to 0.3 for both problems, and by salt noise up to $p = 0.06$ for Twisted LightsOut. LatPlanα successfully solved the problems. The SAE maps the noisy image to the correct symbolic vector $b = Encode(r)$, conduct planning, then map b back to the de-noised image $Decode(b)$.

Are Domain-Independent Heuristics Effective in Latent Space? We compare the numbers of nodes expanded by a search using a greedy merging PDB [28] and blind heuristics (i.e., breadth-first search) in Fast Downward:

- MNIST 8-puzzle (6 instances, mean(StdDev)): Blind 176658(25226), PDB **77811**(32978)
- Mandrill 8-puzzle (1 instance with 31-step optimal solution, corresponding to the 8-puzzle instance [25]): Blind 335378, PDB **88851**
- ToH (4 disks, 1 instance): Blind 55, PDB **17**,
- 4x4 LightsOut (1 instance): Blind 952, PDB **27**,
- 3x3 Twisted LightsOut (1 instance): Blind 522, PDB **214**

The domain-independent PDB heuristic significantly reduced node expansions. Search times ($<$ 3 seconds for all instances) were also faster for all instances with the PDB. Although total runtimes including heuristic initialization is slightly slower than blind search, in domains where goal states and operators are the same for all instances (e.g., 8-puzzle) PDBs can be reused [19], and PDB generation time can be amortized across many instances. Although these results show that existing heuristics for classical planning are able to reduce search effort compared to blind search, much more work is required in order to understand how the features in latent space interact with existing heuristics.

4 Related Work

[18] propose a method for generating PDDL from a low-level, sensor actuator space of an agent characterized as a semi-MDP. The inputs to their system are 33 variables representing accurate structured input (e.g., x/y distances) or categorical states (the on/off state of a button etc.) while LatPlan takes noisy unstructured images (e.g., for 8-puzzle, 42x42=1764-dimensional arrays).

Compared to learning from observation (LfO) in the robotics literature [2], (1) LatPlan is trained based on image pairs showing individual actions, not plan executions (sequence of actions); (2) LatPlan focuses on PDDL for high-level (puzzle-like) tasks, not on motion planning tasks. This significantly affects the data collection scheme: While LfO has *action segmentation* issue because it does not know when an action starts/ends in the plan traces (e.g. video clip), LatPlan does not, because it assumes that a robot can explore the world by itself, initiating/terminating its own action and taking pictures by a camera. The robot can perform a random walk under physical constraints and supervision, which ensure the legal moves (e.g., the physical tile in 8-puzzle). If we further assume that it can "reset" the world (e.g., into a random configuration), then, the robot could eventually obtain images of the entire state space.

A closely related line of work in LfO is learning of board game play from videos [4, 15, 17]. Unlike LatPlan, these works make relatively strong assumptions about the environment, e.g., that there is a grid-like environment.

There is a large body of previous work using neural networks to directly solve combinatorial tasks, such as TSP [13] or Tower of Hanoi [5]. Although they use

NNs to solve search problems, they assume a fully symbolic representation of the problem as input. Other line of hybrid systems embed NNs *inside* a search algorithm to provide search control knowledge [29, 1, 27]. In contrast, we use a NN-based SAE for symbol grounding, not for search control.

Deep Reinforcement Learning (DRL) has solved complex image-based problems [23]. For unit-action-cost planning, LatPlan does not require a reinforcement signal (reward function). Also, it can provide guarantees of completeness and solution cost optimality.

5 Discussion and Conclusion

We proposed LatPlan, an integrated architecture for planning which, given only a set of unlabeled images and no prior knowledge, generates a classical planning problem model, solves it with a symbolic planner, and presents the resulting plan as a human-comprehensible sequence of images. We demonstrated its feasibility using image-based versions of planning/state-space-search problems (8-puzzle, Towers of Hanoi, Lights Out). *The key technical contribution is the SAE, which leverages the Gumbel-Softmax reparametrization technique [14] and learns (unsupervised) a bidirectional mapping between raw images and a propositional representation usable by symbolic planners.* Aside from the key assumptions about the deterministic environment and the sufficient training images, we avoid assumptions about the input domain. Thus, we have shown that domains with different characteristics can all be solved by the same system. In other words, *LatPlan is a domain-independent, image-based classical planner.*

To our knowledge, LatPlan is the first completely automated system of the kind. However, as a proof-of-concept, it has significant limitations to be addressed in future work. In particular, the domain model generator in LatPlanα *does not perform action model learning* from a small set of sample actions because the focus of this paper is not on action learning. Thus the current generator requires the entire set of latent states, transitions and in turn images. While this is obviously impractical, *this is* ***not*** *a fundamental limitation of the LatPlan* ***architecture***. The primitive generator is merely a placeholder for investigating the overall feasibility of an SAE-based end-to-end planning system (our major contribution) and is supposed to be easily replaced by the more sophisticated ones [7, 18, 24, 32]. To our knowledge, all previous domain learning methods require the structured (e.g., propositional) representations of states.

A related topic is how to specify a partial goal specification for LatPlan as in IPC domains (e.g. "having tiles 0,1,2 in the correct places is the goal" in a 8-puzzle), rather than assuming a single goal state, is an interesting future work.

Finally, we do *not* claim that the specific implementation of SAE in this paper works robustly on all images. Making a robust autoencoder is *not* a problem unique to LatPlan, but rather, a fundamental problem in deep learning. *Out contribution is the demonstration that it is possible to leverage some existing deep learning techniques quite effectively in an planning system*, and future work will continue leveraging further improvements in image processing techniques.

References

1. Arfaee, S.J., Zilles, S., Holte, R.C.: Learning Heuristic Functions for Large State Spaces. Artificial Intelligence 175(16-17), 2075–2098 (2011), http://dx.doi.org/10.1016/j.artint.2011.08.001;http://dblp.uni-trier.de/rec/bib/journals/ai/ArfaeeZH11
2. Argall, B., Chernova, S., Veloso, M.M., Browning, B.: A Survey of Robot Learning from Demonstration. Robotics and Autonomous Systems 57(5), 469–483 (2009), http://dx.doi.org/10.1016/j.robot.2008.10.024
3. Bäckström, C., Nebel, B.: Complexity Results for SAS+ Planning. Computational Intelligence 11(4), 625–655 (1995)
4. Barbu, A., Narayanaswamy, S., Siskind, J.M.: Learning Physically-Instantiated Game Play through Visual Observation. In: ICRA. pp. 1879–1886 (2010), http://dx.doi.org/10.1109/ROBOT.2010.5509925
5. Bieszczad, A., Kuchar, S.: Neurosolver Learning to Solve Towers of Hanoi Puzzles. In: IJCCI. vol. 3, pp. 28–38 (2015)
6. Celorrio, S.J., de la Rosa, T., Fernández, S., Fernández, F., Borrajo, D.: A Review of Machine Learning for Automated Planning. Knowledge Eng. Review 27(4), 433–467 (2012), http://dx.doi.org/10.1017/S026988891200001X
7. Cresswell, S., McCluskey, T.L., West, M.M.: Acquiring planning domain models using *LOCM*. Knowledge Eng. Review 28(2), 195–213 (2013)
8. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR. pp. 248–255. IEEE (2009)
9. Deng, L., Hinton, G., Kingsbury, B.: New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview. In: ICASSP. pp. 8599–8603. IEEE (2013)
10. Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J., et al.: Hybrid Computing using a Neural Network with Dynamic External Memory. Nature 538(7626), 471–476 (2016)
11. Gregory, P., Cresswell, S.: Domain model acquisition in the presence of static relations in the LOP system. In: ICAPS. pp. 97–105 (2015), http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10621
12. Helmert, M.: The Fast Downward Planning System. J. Artif. Intell. Res.(JAIR) 26, 191–246 (2006), http://www.aaai.org/Papers/JAIR/Vol26/JAIR-2606.pdf
13. Hopfield, J.J., Tank, D.W.: "Neural" Computation of Decisions in Optimization Problems. Biological cybernetics 52(3), 141–152 (1985)
14. Jang, E., Gu, S., Poole, B.: Categorical Reparameterization with Gumbel-Softmax. In: ICLR (2017)
15. Kaiser, L.: Learning Games from Videos Guided by Descriptive Complexity. In: AAAI (2012), http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5091;http://dblp.uni-trier.de/rec/bib/conf/aaai/Kaiser12
16. Kingma, D.P., Mohamed, S., Rezende, D.J., Welling, M.: Semi-supervised learning with deep generative models. In: NIPS. pp. 3581–3589 (2014)
17. Kirk, J.R., Laird, J.E.: Learning General and Efficient Representations of Novel Games Through Interactive Instruction. Advances in Cognitive Systems 4 (2016)
18. Konidaris, G., Kaelbling, L.P., Lozano-Pérez, T.: Constructing Symbolic Representations for High-Level Planning. In: AAAI. pp. 1932–1938 (2014), http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8424

19. Korf, R.E., Felner, A.: Disjoint Pattern Database Heuristics. Artificial Intelligence 134(1-2), 9–22 (2002), http://dx.doi.org/10.1016/S0004-3702(01)00092-3;http://dblp.uni-trier.de/rec/bib/journals/ai/KorfF02
20. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-Based Learning Applied to Document Recognition. Proc. of the IEEE 86(11), 2278–2324 (1998)
21. Maddison, C.J., Mnih, A., Teh, Y.W.: The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In: ICLR (2017)
22. McDermott, D.V.: The 1998 AI Planning Systems Competition. AI Magazine 21(2), 35–55 (2000), http://www.aaai.org/ojs/index.php/aimagazine/article/view/1506
23. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-Level Control through Deep Reinforcement Learning. Nature 518(7540), 529–533 (2015)
24. Mourão, K., Zettlemoyer, L.S., Petrick, R.P.A., Steedman, M.: Learning STRIPS Operators from Noisy and Incomplete Observations. In: UAI. pp. 614–623 (2012), https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2322&proceeding_id=28
25. Reinefeld, A.: Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA. In: IJCAI. pp. 248–253 (1993), http://ijcai.org/Proceedings/93-1/Papers/035.pdf
26. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In: NIPS. pp. 91–99 (2015)
27. Satzger, B., Kramer, O.: Goal Distance Estimation for Automated Planning using Neural Networks and Support Vector Machines. Natural Computing 12(1), 87–100 (2013), http://dx.doi.org/10.1007/s11047-012-9332-y
28. Sievers, S., Ortlieb, M., Helmert, M.: Efficient Implementation of Pattern Database Heuristics for Classical Planning. In: SOCS (2012)
29. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature 529(7587), 484–489 (2016)
30. Steels, L.: The Symbol Grounding Problem has been solved. So what's next? In: de Vega, M., Glenberg, A., Graesser, A. (eds.) Symbols and Embodiment. Oxford University Press (2008)
31. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and Composing Robust Features with Denoising Autoencoders. In: ICML. pp. 1096–1103. ACM (2008)
32. Yang, Q., Wu, K., Jiang, Y.: Learning Action Models from Plan Examples using Weighted MAX-SAT. Artificial Intelligence 171(2-3), 107–143 (2007), http://dx.doi.org/10.1016/j.artint.2006.11.005;http://dblp.uni-trier.de/rec/bib/journals/ai/YangWJ07

Explaining Trained Neural Networks with Semantic Web Technologies: First Steps

Md Kamruzzaman Sarker, Ning Xie, Derek Doran, Michael Raymer, and Pascal Hitzler

Data Science and Security Cluster, Wright State University, Dayton, OH, USA

Abstract. The ever increasing prevalence of publicly available structured data on the World Wide Web enables new applications in a variety of domains. In this paper, we provide a conceptual approach that leverages such data in order to explain the input-output behavior of trained artificial neural networks. We apply existing Semantic Web technologies in order to provide an experimental proof of concept.

1 Introduction

Trained neural networks are usually imagined as black boxes, in that they do not give any direct indications why an output (e.g., a prediction) was made by the network. The reason for this lies in the distributed nature of the information encoded in the weighted connections of the network. Of course, for applications, e.g., safety-critical ones, this is an unsatisfactory situation. Methods are therefore sought to explain how the output of trained neural networks are reached.

This topic of explaining trained neural networks is not a new one, in fact there is already quite a bit of tradition and literature on the topic of rule extraction from such networks (see, e.g., [2,9,16]), which pursued very similar goals. Rule extraction, however, utilized propositional rules as target logic for generating explanations, and as such remained very limited in terms of explanations which are human-understandable. Novel deep learning architectures attempt to retrieve explanations as well, but often the use-case is only for computer vision tasks like object or scene recognition. Moreover, explanations in this context actually encode greater details about the images provided as input, rather than explaining why or how the neural network was able to recognize a particular object or scene.

Semantic Web [4,12] is concerned with data sharing, discovery, integration, and reuse. As field, it does not only target data on the World Wide Web, but its methods are also applicable to knowledge management and other tasks off the Web. Central to the field is the use of knowledge graphs (usually expressed using the W3C standard Resource Description Framework RDF [3]) and type logics attached to these graphs, which are called *ontologies* and are usually expressed using the W3C standard Web Ontology Language OWL [11].

This paper introduces a new paradigm for explaining neural network behavior. It goes beyond the limited propositional paradigm, and directly targets the problem of explaining neural network activity rather than the qualities of

the input. The paradigm leverages advances in knowledge representation on the World Wide Web, more precisely from the field of Semantic Web technologies. It in particular utilizes the fact that methods, tool, and structured data in the mentioned formats are now widely available, and that the amount of such structured data on the Web is in fact constantly growing [5,18]. Prominent examples of large-scale datasets include Wikidata [22] and data coming from the schema.org [10] effort which is driven by major Web search engine providers. We will utilize this available data as background knowledge, on the hypothesis that background knowledge will make it possible to obtain more concise explanations. This addresses the issue in propositional rule extraction that extracted rulesets are often large and complex, and due to their sizes difficult to understand for humans. While the paper only attempts to explain input-output behavior, the authors are actively exploring ways to also explain internal node activations.

An illustrative example

Let us consider the following very simple example which is taken from [14]. Assume that the input-output mapping P of the neural network without background knowledge could be extracted as

$$p_1 \wedge q \rightarrow r \qquad p_2 \wedge q \rightarrow r.$$

Now assume furthermore that we also have background knowledge K in form of the rules

$$p_1 \rightarrow p \qquad p_2 \rightarrow p.$$

The background knowledge then makes it possible to obtain the simplified input-output mapping P_K, as

$$p \wedge q \rightarrow r.$$

The simplification through the background knowledge is caused by p acting as a "generalization" of both p_1 and p_2. For the rest of the paper it may be beneficial to think of p, p_1 and p_2 as classes or concepts, which are hierarchically related, e.g., p_1 being "oak," p_2 being "maple," and p being "tree."

Yet this example is confined to propositional logic.[1] In the following, we show how we can bring structured (non-propositional) Semantic Web background knowledge to bear on the problem of explanation generation for trained neural networks, and how we can utilize Semantic Web technologies in order to generate non-propositional explanations. This work is at a very early stage, i.e., we will only present the conceptual architecture of the approach and minimal experimental results which are encouraging for continuing the effort.

The rest of the paper is structured as follows. In Section 2 we introduce notation as needed, in particular regarding description logics which underly the OWL standard, and briefly introduce the DL-Learner tool which features prominently in our approach. In Section 3 we present the conceptual and experimental setup

[1] How to go beyond the propositional paradigm in neural-symbolic integration is one of the major challenges in the field [8].

for our approach, and report on some first experiments. In Section 4 we conclude and discuss avenues for future work.

2 Preliminaries

We describe a minimum of preliminary notions and information needed in order to keep this paper relatively self-contained. *Description logics* [1,12] are a major paradigm in knowledge representation as a subfield of artificial intelligence. At the same time, they play a very prominent role in the Semantic Web field since they are the foundation for one of the central Semantic Web standards, namely the W3C Web Ontology Language OWL [11,12].

Technically speaking, a description logic is a decidable fragment of first-order predicate logic (sometimes with equality or other extensions) using only unary and binary predicates. The unary predicates are called *atomic classes*,[2] while the binary ones are refered to as *roles*,[3] and constants are refered to as *individuals*. In the following, we formally define the fundamental description logic known as $\mathcal{ALC}$, which will suffice for this paper. OWL is a proper superset of $\mathcal{ALC}$.

Desciption logics allow for a simplified syntax (compared to first-order predicate logic), and we will introduce $\mathcal{ALC}$ in this simplified syntax. A translation into first-order predicate logic will be provided further below.

Let $\mathcal{C}$ be a finite set of atomic classes, $\mathcal{R}$ be a finite set of roles, and $\mathcal{N}$ be a finite set of individuals. Then *class expressions* (or simply, *classes*) are defined recursively using the following grammar, where A denotes atomic classes from $\mathcal{A}$ and R denotes roles from $\mathcal{R}$. The symbols $\sqcap$ and $\sqcup$ denote conjunction and disjunction, respectively.

$$C, D ::= A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$$

A *TBox* is a set of statements, called (*general class inclusion*) *axioms*, of the form $C \sqsubseteq D$, where C and D are class expressions – the symbol $\sqsubseteq$ can be understood as a type of subset inclusion, or alternatively, as a logical implication. An *ABox* is a set of statements of the forms $A(a)$ or $R(a, b)$, where A is an atomic class, R is a role, and a, b are individuals. A description logic *knowledge base* consists of a TBox and an ABox. The notion of *ontology* is used in different ways in the literature; sometimes it is used as equivalent to TBox, sometimes as equivalent to knowledge base. We will adopt the latter usage.

We characterize the semantics of $\mathcal{ALC}$ knowledge bases by giving a translation into first-order predicate logic. If α is a TBox axiom of the form $C \sqsubseteq D$, then $\pi(\alpha)$ is defined inductively as in Figure 1, where A is a class name. ABox axioms remain unchanged.

DL-Learner [6,17] is a machine learning system inspired by inductive logic programming [20]. Given a knowledge base and two sets of individuals from the knowledge base – called positive respectively negative examples – DL-Learner

[2] or *atomic concepts*
[3] or *properties*

$$\begin{aligned}
\pi(C \sqsubseteq D) &= (\forall x_0)(\pi_{x_0}(C) \rightarrow \pi_{x_0}(D)) \\
\pi_{x_i}(A) &= A(x_i) \\
\pi_{x_i}(\neg C) &= \neg \pi_{x_i}(C) \\
\pi_{x_i}(C \sqcap D) &= \pi_{x_i}(C) \wedge \pi_{x_i}(D) \\
\pi_{x_i}(C \sqcup D) &= \pi_{x_i}(C) \vee \pi_{x_i}(D) \\
\pi_{x_i}(\forall R.C) &= (\forall x_{i+1})(R(x_i, x_{i+1}) \rightarrow \pi_{x_{i+1}}(C)) \\
\pi_{x_i}(\exists R.C) &= (\exists x_{i+1})(R(x_i, x_{i+1}) \wedge \pi_{x_{i+1}}(C))
\end{aligned}$$

Fig. 1. Translating TBox axioms into first-order predicate logic. We use auxiliary functions π_{x_i}, where the x_i are variables. The axiom $A \sqsubseteq \exists R.\exists S.B$, for example, would be translated to $(\forall x_0)((A(x_0)) \rightarrow (\exists x_1)(R(x_0, x_1) \wedge (\exists x_2)(S(x_1, x_2) \wedge B(x_2))))$.

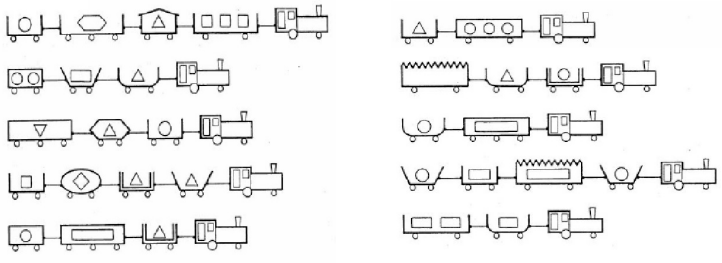

Fig. 2. Michalski's trains, picture from [15]. Positive examples on the left, negative ones on the right.

attempts to construct class expressions such that all the positive examples are contained in each of the class expressions, while none of the negative examples is. DL-Learner gives preference to shorter solutions, and in the standard setting returns approximate solutions if no fully correct solution is found. The inner workings of DL-Learner will not matter for this paper, and we refer to [6,17] for details. However, we exemplify its functionality by looking at Michalski's trains as an example, which is a symbolic machine learning task from [15], and which was presented also in [17].

For purposes of illustrating DL-Learner, Figure 2 shows two sets of trains, the positive examples are on the left, the negative ones are on the right. Following [17], we use a simple encoding of the trains as a knowledge base: Each train is an individual, and has cars attached to it using the hasCar property, and each car then falls into different categories, e.g., the top leftmost car would fall into the classes Open, Rectangular and Short, and would also have information attached to it regarding symbol carried (in this case, square), and how many of them (in this case, one). Given these examples and knowledge base, DL-Learner comes

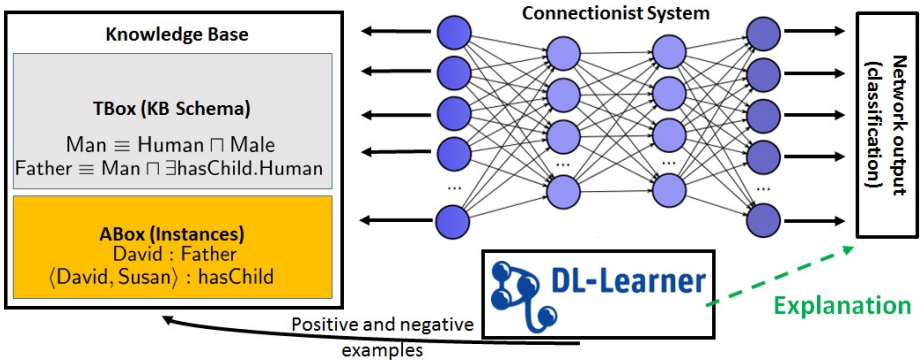

Fig. 3. Conceptual architecture – see text for explanations.

up with the class

$$\exists \text{hasCar}.(\text{Closed} \sqcap \text{Short})$$

which indeed is a simple class expression such that all positive examples fall under it, while no negative example does.

3 Approach and Experiments

In this paper, we follow the lead of the propositional rule extraction work mentioned in the introduction, with the intent of improving on it in several ways.

1. We generalize the approach by going significantly beyond the propositional rule paradigm, by utilizing description logics.
2. We include significantly sized and publicly available background knowledge in our approach in order to arrive at explanations which are more concise.

More concretely, we use DL-Learner as the key tool to arrive at the explanations. Figure 3 depicts our conceptual architecture: The trained artificial neural network (connectionist system) acts as a classifier. Its inputs are mapped to a background knowledge base and according to the networks' classification, positive and negative examples are distinguished. DL-Learner is then run on the example sets and provides explanations for the classifications based on the background knowledge.

In the following, we report on preliminary experiments we have conducted using our approach. Their sole purpose is to provide first and very preliminary insights into the feasibility of the proposed method. All experimental data is available from `http://daselab.org/projects/human-centered-big-data`.

We utilize the ADE20K dataset [23,24]. It contains 20,000 images of scenes which have been pre-classified regarding scenes depicted, i.e., we assume that the

Fig. 4. Test images. Positive examples p_1, p_2, p_3 on the left (from top), negative examples n_1, n_2, n_3 on the right (from top).

classification is done by a trained neural network.[4] For our initial test, we used six images, three of which have been classified as "outdoor warehouse" scenes (our positive examples), and three of which have not been classified as such (our negative examples). In fact, for simplicity, we took the negative examples from among the images which had been classified as "indoor warehouse" scenes. The images are shown in Figure 4.

The ADE20K dataset furthermore provides annotations for each image which identify information about objects which have been identified in the image. The annotations are in fact richer than that and also talk about the number of objects, whether they are occluded, and some more, but for our initial experiment we only used presence or absence of an object. To keep the initial experiment simple, we furthermore only used those detected objects which could easily be mapped to our chosen background knowledge, the Suggested Upper Merged Ontology (SUMO).[5] Table 1 shows, for each image, the objects we kept. The Suggested Upper Merged Ontology was chosen because it contains many, namely about 25,000 common terms which cover a wide range of domains. At the same

[4] Strictly speaking, this is not true for the training subset of the ADE20K dataset, but that doesn't really matter for our demonstration.

[5] `http://www.adampease.org/OP/`

image p_1: road, window, door, wheel, sidewalk, truck, box, building
image p_2: tree, road, window, timber, building, lumber
image p_3: hand, sidewalk, clock, steps, door, face, building, window, road
image n_1: shelf, ceiling, floor
image n_2: box, floor, wall, ceiling, product
image n_3: ceiling, wall, shelf, floor, product

Table 1. Objects recorded for each image.

time, the ontology arguably structures the terms in a relatively straightforward manner which seemed to simplify matters for our initial experiment.

In order to connect the annotations to SUMO, we used a single role called "contains." Each image was made an individual in the knowledge base. Furthermore, for each of the object identifying terms in Table 1, we either identified a corresponding matching SUMO class, or created one and added it to SUMO by inserting it at an appropriate place within SUMO's class hierarchy. We furthermore created individuals for each of the object identifying terms, including duplicates, in Table 1, and added them to the knowledge base by typing them with the corresponding class. Finally, we related each image individual to each corresponding object individual via the "contains" role.

To exemplify – for the image p_1 we added individuals road1, window1, door1, wheel1, sidewalk1, truck1, box1, building1, declared Road(road1), Window(window1), etc., and finally added the ABox statements contains(p_1, road1), contains(p_1, window1), etc., to the knowledge base. For the image p_2, we added contains(p_2, tree2), contains(p_2, road2), etc. as well as the corresponding type declarations Tree(tree2), Road(road2), etc.

The mapping of the image annotations to SUMO is of course very simple, and this was done deliberately in order to show that a straightforward approach already yields interesting results. As our work progresses, we do of course anticipate that we will utilize more complex knowledge bases and will need to generate more complex mappings from picture annotations (or features) to the background knowledge.

Finally, we ran DL-Learner on the knowledge base, with the positive and negative examples as indicated. DL-Learner returns 10 solutions, which are listed in Figure 5. Of these, some are straightforward from the image annotations, such as (1), (5), (8, (9) and (10). Others, such as (2), (4), (6), (7) are much more interesting as they provide solutions in terms of the background knowledge without using any of the terms from the original annotation. Solution (3) looks odd at first sight, but is meaningful in the context of the SUMO ontology: SelfConnectedObject is an abstract class which is a direct child of the class Object in SUMO's class hierarchy. Its natural language definition is given as "A SelfConnectedObject is any Object that does not consist of two or more disconnected parts." As such, the class is a superclass of the class Road, which explains why (3) is indeed a solution in terms of the SUMO ontology.

∃contains.Window	(1)	∃contains.LandTransitway	(6)
∃contains.Transitway	(2)	∃contains.LandArea	(7)
∃contains.SelfConnectedObject	(3)	∃contains.Building	(8)
∃contains.Roadway	(4)	∀contains.¬Floor	(9)
∃contains.Road	(5)	∀contains.¬Ceiling	(10)

Fig. 5. Solutions produced by DL-Learner for the warehouse test.

We have conducted four additional experiments along the same lines as described above. We briefly describe them below – the full raw data and results are available from `http://daselab.org/projects/human-centered-big-data`.

In the second experiment, we chose four workroom pictures as positive examples, and eight warehouse pictures (indoors and outdoors) as negative examples. An example explanation DL-Learner came up with is

$$\exists \text{contains.}(\text{DurableGood} \sqcap \neg \text{ForestProduct}).$$

On of the outdoor warehouse pictures indeed shows timber. DurableGoods in SUMO include furniture, machinery, and appliances.

In the third experiment, we chose the same four workroom pictures as negative examples, and the same eight warehouse pictures (indoors and outdoors) as positve examples. An example explanation DL-Learner came up with is

$$\forall \text{contains.}(\neg \text{Furniture} \sqcap \neg \text{IndustrialSupply}),$$

i.e., "contains neither furniture nor industrial supply". IndustrialSupply in SUMO includes machinery. Indeed it turns out that furniture alone is insufficient for distingushing between the positive and negative exaples, because "shelf" is not classified as funiture in SUMO. This shows the dependency of the explanations on the conceptualizations encoded in the background knowledge.

In the fourth experiment, we chose eight market pictures (indoors and outdoors) as positive examples, and eight warehouse pictures (indoors and outdoors) as well as four workroom pictures as negative examples. An example explanation DL-Learner came up with is

$$\exists \text{contains.SentientAgent},$$

And indeed it turns out that people are shown on all the market pictures. There is actually also a man shown on one of the warehouse pictures, driving a forklift, however "man" or "person" was not among the annotations used for the picture. This example indicates how our approach could be utilized: A human monitor inquiring with an interactive system about the reasons for a certain classification may notice that the man was missed by the software on that particular picture, and can opt to interfere with the decision and attempt to correct it.

In the fifth experiment, we chose four mountain pictures as positive examples, and eight warehouse pictures (indoors and outdoors) as well as four workroom

pictures as negative examples. An example explanation DL-Learner came up with is

$$\exists \text{contains.BodyOfWater}.$$

Indeed, it turns out that all mountain pictures in the example set show either a river or a lake. Similar to the previous example, a human monitor may be able to catch that some misclassifications may occur because presence of a body of water is not always indicative of presence of a mountain.

4 Conclusions and Further Work

We have laid out a conceptual sketch how to approach the issue of explaining artificial neural networks' classification behaviour using Semantic Web background knowledge and technologies, in a non-propositional setting. We have also reported on some very preliminary experiments to support our concepts.

The sketch already indicates where to go from here: We will need to incorporate more complex and more comprehensive background knowledge, and if readily available structured knowledge turns out to be insufficient, then we foresee using state of the art knowledge graph generation and ontology learning methods [13,19] to obtain suitable background knowledge. We will need to use automatic methods for mapping network input features to the background knowledge [7,21], while the features to be mapped may have to be generated from the input in the first place, e.g. using object recognition software in the case of images. And finally, we also intend to apply the approach to sets of hidden neurons in order to understand what their activations indicate.

Acknowledgements. This work was supported by the Ohio Federal Research Network project *Human-Centered Big Data*.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edn. (2010)
2. Bader, S., Hitzler, P.: Dimensions of neural-symbolic integration – A structured survey. In: Artëmov, S.N., Barringer, H., d'Avila Garcez, A.S., Lamb, L.C., Woods, J. (eds.) We Will Show Them! Essays in Honour of Dov Gabbay, Volume One. pp. 167–194. College Publications (2005)
3. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1. Turtle – Terse RDF Triple Language. W3C Recommendation (25 February 2014), available at http://www.w3.org/TR/turtle/
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (May 2001)
5. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story So Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
6. Bühmann, L., Lehmann, J., Westphal, P.: DL-Learner – A framework for inductive learning on the semantic web. Journal of Web Semantics 39, 15–24 (2016)

7. Euzenat, J., Shvaiko, P.: Ontology Matching, Second Edition. Springer (2013)
8. Garcez, A., Besold, T., de Raedt, L., Földiak, P., Hitzler, P., Icard, T., Kühnberger, K.U., Lamb, L., Miikkulainen, R., Silver, D.: Neural-symbolic learning and reasoning: Contributions and challenges. In: Gabrilovich, E., Guha, R., McCallum, A., Murphy, K. (eds.) Proceedings of the AAAI 2015 Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches. Technical Report, vol. SS-15-03. AAAI Press, Palo Alto, CA (2015)
9. d'Avila Garcez, A.S., Zaverucha, G.: The connectionist inductive lerarning and logic programming system. Applied Intelligence 11(1), 59–77 (1999)
10. Guha, R.V., Brickley, D., Macbeth, S.: Schema.org: evolution of structured data on the web. Commun. ACM 59(2), 44–51 (2016)
11. Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S. (eds.): OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation (11 December 2012), http://www.w3.org/TR/owl2-primer/
12. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. CRC Press/Chapman & Hall (2010)
13. Ji, H., Grishman, R.: Knowledge base population: Successful approaches and challenges. In: Lin, D., Matsumoto, Y., Mihalcea, R. (eds.) The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA. pp. 1148–1158. The Association for Computer Linguistics (2011)
14. Labaf, M., Hitzler, P., Evans, A.B.: Propositional rule extraction from neural networks under background knowledge. In: Proceedings of the Twelfth International Workshop on Neural-Symbolic Learning and Reasoning, NeSy'17, London, UK, July 2017 (2017), to appear
15. Larson, J., Michalski, R.S.: Inductive inference of VL decision rules. SIGART Newsletter 63, 38–44 (1977)
16. Lehmann, J., Bader, S., Hitzler, P.: Extracting reduced logic programs from artificial neural networks. Applied Intelligence 32(3), 249–266 (2010)
17. Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Machine Learning 78(1-2), 203–250 (2010)
18. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web 6(2), 167–195 (2015)
19. Lehmann, J., Völker, J.: Perspectives on Ontology Learning, Studies on the Semantic Web, vol. 18. IOS Press (2014)
20. Muggleton, S., Raedt, L.D.: Inductive logic programming: Theory and methods. Journal of Logic Programming 19/20, 629–679 (1994)
21. Uren, V.S., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F.: Semantic annotation for knowledge management: Requirements and a survey of the state of the art. J. Web Sem. 4(1), 14–28 (2006)
22. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM 57(10), 78–85 (2014)
23. Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Semantic understanding of scenes through the ADE20K dataset. arXiv preprint arXiv:1608.05442 (2016)
24. Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Scene parsing through ADE20K dataset. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2017)

Inductive Learning in Shared Neural Multi-Spaces

Edjard de Souza Mota[1,2], Jacob M. Howe[1], and Artur S. d'Avila Garcez[1]

[1] City, University of London, London, EC1V 0HB, UK
{edjard.de-souza-mota,j.m.howe,a.garcez}@city.ac.uk,
[2] Universidade Federal do Amazonas,
Instituto de Computação, Campus Setor Norte
Coroado - Manaus - AM - Brasil CEP: 69080-900

Abstract. The learning of rules from examples is of continuing interest to machine learning since it allows generalization from fewer training examples. Inductive Logic Programming (ILP) generates hypothetical rules (clauses) from a knowledge base augmented with (positive and negative) examples. A successful hypothesis entails all positive examples and does not entail any negative example. The Shared Neural Multi-Space (Shared NeMuS) structure encodes first order expressions in a graph suitable for ILP-style learning. This paper explores the NeMuS structure and its relationship with the Herbrand Base of a knowledge-base to generate hypotheses inductively. It is demonstrated that inductive learning driven by the knowledge-base structure can be implementated successfully in the Amao cognitive agent framework, including the learning of recursive hypotheses.

1 Introduction

There is renewed interest in inductive logic programming as a framework for machine learning owing to its human like ability to infer new knowledge from background knowledge together with a small number of examples. It also comes with strong mathematical and logical underpinnings. This paper builds on [9], where a data structure (Shared NeMuS) for the representation of first-order logic was introduced. It revisits inductive logic programming and demonstrates that Shared NeMuS provides a structure that can be used to build an inductive logic programming system.

A part of the Shared NeMuS structure is weightings on individual elements (atom, predicate, function). The purpose of these weightings is to provide a guide to the use of these elements, for example in theorem proving. One of the key challenges in inductive logic programming is to find good heuristics to search the hypothesis space; the long-term goal of this work is to learn weights in a training phase that can in turn be used to guide the search of the hypothesis space and improve the efficiency and success of inductive learning.

The main purpose of this work is to present a new approach for Clausal Inductive Learning (CIL) using Shared NeMuS in which the search mechanism

uses the Herbrand Base (HB) to build up hypothesis candidates using inverse unification. This generalization of ground expressions to universally quantified ones is supported by the idea of *regions of concepts* that was explored in [9] to find refutation patterns. Here, weights are not explicitly used, but the intuitive use of them is explored to define *linkage patterns* between predicates of the HB and the occurrences of ground terms in positive examples. Meaningless hypotheses are pruned away as a result of *inductive momentum* between predicates connected to positive and negative examples. This paper makes the following contributions: it is demonstrated that the Shared NeMuS data structure can be used as a suitable structure for inductive logic programming; the Herbrand Base is used to build candidate hypotheses; chaining and abstraction for rules, including recursive rules, are given; these rules help keep the number of hypotheses small. NeMuS is designed for extension with machine learning tactics.

The remainder of this paper is structured as follows: section 2 gives some brief background on inductive logic programming and the Shared NeMuS data structure, sections 3 and 4 describe the implementation of inductive learning in Amao using the Shared NeMuS data structure, then section 5 describes some related work and section 6 discusses the work presented.

2 Background

2.1 Inductive Logic Programming (ILP)

The goal of inductive logic programming (introduced in [10]) is to learn logical formulae that describe a target concept, based on a set of examples. In a typical set up there is a knowledge base of predicates, called background knowledge (BK), along with a set of examples that the target concept should prove (positive examples, e^+) and a set of examples that the target concept should not prove (negative examples, e^-). The inductive logic programming problem is to search for a logical description (a hypothesis, H) of the target concept based on the knowledge and the examples so that the knowledge base plus the hypothesis entails the positive examples, whilst does not entail the negative examples.

Inductive logic programming systems implement search strategies over the space of possible hypotheses. A good heuristic is one that will a) arrive at a successful hypothesis, b) find this hypothesis quickly and c) find a succinct hypothesis. In order to achieve this, the efficiency of hypothesis searching mechanisms depend on partial order of θ-subsumption [13], or on a total ordering over the Herbrand Base to constrain deductive, abductive and inductive operations [12].

The approach presented in this paper for CIL takes a totally different approach. Search considers separated spaces for constant terms, predicates and clauses, and elements are indexed by their unique identification codes. The spaces are interconnected through weighted bindings pointing to the target space in which an occurrence of an element appears. This creates a network of shared spaces because the elements are shared via bindings. As the weights are not used here, the networks shall be referred to as multi-spaces.

2.2 Shared NeMuS

Shared Neural Multi-Space (Shared NeMuS) [9] takes inspiration from [2] to give a shared multi-space representation for a portion of first-order logic designed for use with machine learning and neural network methods. The structure incorporates a relative degree of importance for each element according to the element's attributes and uses an architecture that leads to a fast implementation.

Shared NeMuS uses a Smarandache multi-space [8], a union of n spaces $A_1, ..., A_n$ in which each A_i is the space of a distinct observed characteristic of the overall space. For each A_i there is a different metric to describe a different side of the "major" side. There will be a space for each component of a first-order language: atomic constants (of the Herbrand Universe, space 0), functions (space 1), predicates with literal instances (space 2), and clauses (space 3). Variables are used to refer to sets of atomic terms via quantification, and they belong to the same space of atoms. In this work, the function space is suppressed, since it is not being dealt with for relational learning. In what follows vectors are written $\boldsymbol{v}$, and $\boldsymbol{v}[i]$ or $\boldsymbol{v}_i$ is used to refer to an element of a vector at position i.

Each logical element is described by a T-Node, and in particular each element is described by a unique integer code within its space. In addition, a T-Node identifies the lexicographic occurrence of the element, and (when appropriate) an attribute position.

Definition 1 (T-Node and Binding) *Let $c, a, i \in \mathbb{Z}$ and $h \in \{0, 1, 2, 3\}$. A* T-Node (target node) *is a quadruple (h, c, i, a) that identifies an object at space h, with code c and occurrence i, at attribute position a (when it applies, otherwise 1). If p is a T-Node, then $n_h(p) = h$, $n_c(p) = c$, $n_a(p) = a$ and $n_i(p) = i$. A NeMuS* Binding *is a pair $(p, w)_k$, which represents the influence w of object k over occurrence $n_i(p)$ of object $n_c(p)$ at space $n_h(p)$ in position $n_a(p)$.*

The elements of the subject space represent all of the occurrences of atoms and variables in an expression.

Definition 2 (Subject Space) *Let $\boldsymbol{C} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m]$ and $\boldsymbol{V} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n]$, where each $\boldsymbol{x}_i, \boldsymbol{y}_i$ is a vector of bindings.* Subject Space *refers to the pair $(\boldsymbol{C}, \boldsymbol{V})$ for constants and variables, respectively.*

The function β maps a constant i to the vector of its bindings $\boldsymbol{x_i}$, as above.

Higher spaces are made of structured elements, such as predicates and clauses. Such objects have attributes uniquely identified by T-Nodes referring to objects in the spaces below and their bindings of the objects on which they exert influence.

Definition 3 (Compound) *Let $\boldsymbol{x}_a^i = [c_1, \ldots, c_m]$ be a vector of T-Nodes for each attribute instance of compound i. Let $\boldsymbol{w}_i$ be a vector of NeMuS bindings. Then a NeMuS* Compound *is the pair $(\boldsymbol{x}_a^i, \boldsymbol{w}_i)$. A NeMuS* Compound Space (C-Space) *is a vector of NeMuS Compounds.*

For each literal there is a C-Space to represent it. Since a literal is an instance of a predicate the predicate space is a vector of C-Spaces. As predicates have positive and negative instances, there are two vector regions for a predicate space. Clauses' attributes are the literals that they are made of, and as they exert no influence upon spaces above for simplicity the bindings of a clause shall be empty.

Definition 4 (Predicate and Clause Spaces) *Let* $\boldsymbol{C}_p^+$ *and* $\boldsymbol{C}_p^-$ *be two vectors of C-spaces. The pair* $(\boldsymbol{C}_p^+, \boldsymbol{C}_p^-)$ *is called the NeMuS* Predicate Space*. A NeMuS* Clause Space *is a vector of C-spaces such that every pair in the vector shall be* $(\boldsymbol{x}_a^i, [])$*.*

A Shared NeMuS for a coded first-order expression is a triple $\langle \mathcal{S}, \mathcal{P}, \mathcal{C} \rangle$, in which $\mathcal{S}$ is the subject space, $\mathcal{P}$ is the predicate space and $\mathcal{C}$ is the clause space.

Example 1. Assume the following symbolic BK (adapted from [3]):

```
mother(pam,ann).
wife(ann,bob).
wife(eve,wallie).
brother(wallie, ann).
```

This translates into the Shared NeMuS below (with weights at default value of 0). The constant region of the subject space, the predicate space and the clause space are each given with some commentary. The labels and the numbers before the column are just to emphasise structure.

```
Subject Space:   Constant region: {
  1:[((2,1,1,1),0)]
  2:[(2,1,1,2),0),(2,2,1,1),0),((2,3,1,2),0)]
  3:[((2,2,1,2),0)]
  4:[((2,2,2,1),0)]
  5:[((2,2,2,2),0),((2,3,1,1),0)]  }
```

The subject space encodes the occurrences of the constants. For example, the first entry gives the binding for `pam`, with code 1, stating that this atom occurs in the first occurrence of the first predicate (`mother`) as the first attribute. The second entry gives the list of bindings for the three occurrences of `ann`.

```
Predicate Space:
    //      pam        ann
   {+1:[([(0,1,1,1),(0,2,1,2)],[((3,1,1,1),0)])],  -1: []}
    //      ann        bob
   {+2:[([(0,2,2,1),(0,3,1,2)],[((3,2,1,1),0)]),
        ([(0,4,1,1),(0,5,1,2)],[((3,3,1,1),0)])],  -2: []}
   {+3:[([(0,5,2,1),(0,2,3,2)],[((3,4,1,1),0)])],  -3: []}
```

The predicate space encodes each predicate. For example, the first entry links to the bindings of the two constants occurring in the only clause in which it occurs. The second entry details the two clauses for `wife`.

```
Clause Space:
   {([(2,1,2,1)],[()]), ([(2,2,2,1)],[()]),
    ([(2,2,3,1)],[()]), ([(2,3,2,1)],[()])}
```

Here, the clauses link back to the predicates that define them. For example, the first entry says that the first clause is built from the first predicate.

This simple example shows how easy it is to navigate across a shared NeMuS structure to induce new rules from relational data or sets of literals.

3 Inductive Learning with Shared NeMuS

This section describes, with the aid of running examples, how inductive learning is performed in Amao, using the Shared NeMuS structure. Amao[3] a cognitive artificial agent which originally performed just symbolic reasoning on structured clauses via Linear Resolution [16]. It was extended in [9] to generate a shared NeMuS representation, during the compilation of symbolic representation, for neural-symbolic reasoning purposes.

Inverse unification from HB is *plausible* since all ground expressions of a given concept p will be at the same region as far as weighted grounds are concerned. For instance, $p(a, b)$, $p(c, d)$ and $p(b, e)$ all belong to the region of p. So, $p(X, Y)$ is a sound generalization of such instances. However, if there are other concepts involving the elements of the Herbrand Universe, say $q(a, d)$ and $r(d)$ then $p(X, Y)$ is not a straight generalization without taking into account the combination of the regions for r and q since their ground atoms have occurrences of constants appearing in all three concepts.

The induction algorithm proposed is guided by kinds of *linkage patterns* (sections 3.1 and 3.2) and using only those in which there is an *inductive momentum* (section 3.3). The explanation is that positive examples bring the ground expressions "close" to the hypothesis to be generated, while the negative ones pull apart those which are likely to generate inconsistent hypothesis.

3.1 Linear Linkage Patterns

One form of the Amao operation for performing inductive learning with target predicate `p/n` is:

```
consider induction on p(X1,...,Xn) knowing p(t1,...,tn).
```

That is, `p/n` is not in the BK and Amao will attempt to find a hypothesis H such that the positive example(s) `p(t1,...,tn)` can be deduced from $BK \cup H$. In what follows, the symbol representation will be used to mean the Shared NeMuS code of each logical element and recall that β maps code symbols to their bindings.

[3] Amao is the name of a deity that taught people of Camanaos tribe, who lived on the margins of the Negro River in the Brazilian part of the Amazon rainforest, the process of making mandioca powder and beiju biscuit for their diet.

The BK is that used in Example 1. Suppose that the target predicate is `motherInLaw/2`, with positive knowledge `motherInLaw(pam,bob)` then when Amao is asked to generate hypotheses with

```
consider induction on motherInLaw(X,Y) knowing motherInLaw(pam,bob).
```

amongst the successful hypotheses should be:

$$motherInLaw(X,Y) \leftarrow mother(X,Z) \wedge wife(Z,Y)$$

Parsing the positive example against the Shared NeMuS representation of the BK gives that in the constant region `pam` has code 1 and `bob` has code 3. From the constant region of the subject space the vectors of bindings $\beta(1)$ and $\beta(3)$ are found:

$\beta(1) = [(1,1,1)]$: in the first predicate, its first instance, as first attribute
$\beta(3) = [(2,1,2)]$: in the second predicate, its first instance, as second attribute

For each element of the vector of bindings, $\beta(i)$, the vector of attributes of the predicate in which it occurs is found, written $x_a(\beta(i)_j)$, where j is an index. Here,

$$x_a(\beta(1)_1) = [(0,1,1,1),(0,2,1,2)] \text{ and } x_a(\beta(3)_1) = [(0,2,2,1),(0,3,1,2)]$$

that is, `mother(pam, ann)` and `wife(ann, bob)`.

The intersection between $x_a(\beta(pam)_1)$ and $x_a(\beta(bob)_1)$ is non-empty since `ann` (code 2) occurs in both. Hence predicate codes 1 (`mother`) and 2 (`wife`) are used in the hypothesis. These are ground atoms from the HB of Example 1, and from them a new clause is built with head (positive) literal as the target anti-unified, and the negative literals are those found above. Inverse unification will incrementally build anti-substitution θ^{-1} at each step by adding literals to the body with constants substituted by variables `X` and `Y` from the targeted head. As `X` appears as the first attribute of the first predicate (`mother`), and `Y` as the second attribute of the second predicate (`wife`), then the linkage term shall be `Z_0`. Call `Z_0` the *hook* between both literals and the final θ^{-1} is {`pam/X,bob/Y,ann/Z_0`}. The terms which are not the hook term are called *attribute-mates*. Thus the hypothesis is the following clause in Amao notation:

```
motherInLaw(X,Y); ~mother(X,Z_0); ~wife(Z_0,Y)
```

This rule is added to the KB and its Shared NeMuS is updated accordingly.

In general this hook chain can be longer and involve more *linkage predicates*. Depending on the position where the linkage is formed from one to another there may be different sorts of *linkage pattern*. Besides, there can be intermediate predicates that should not be part of the hypothesis since they may deduce negative examples. Consider the following example, and this time for the sake of readability the space information will be suppressed from the bindings and the attribute position from x_a.

Example 2. Consider the following BK.

1. `parent(pam,bob).`	6. `parent(pat,jim).`	1. `female(pam).`
2. `parent(tom,bob).`	7. `parent(ann,eve).`	2. `female(liz).`
3. `parent(tom,liz).`	1. `male(tom).`	3. `female(ann).`
4. `parent(bob,ann).`	2. `male(bob).`	4. `female(pat).`
5. `parent(bob,pat).`	3. `male(jim).`	5. `female(eve).`

Codes for the logical elements in the order they are read or scanned are:

parent	male	female	pam	bob	tom	liz	ann	pat	jim	eve
1	2	3	1	2	3	4	5	6	7	8

The induction Amao is requested to perform is

```
consider induction on hasDaughter(X)
                      knowing hasDaughter(ann) ~hasDaughter(pat).
```

First find the bindings associated with the positive and negative examples.

$$+\beta(ann) = [(1,4,2),(1,7,1),(3,3,1)] \text{ and } -\beta(pat) = [(1,5,2),(1,6,1),(3,4,1)]$$

$n_c(\beta(ann)_1) = n_c(\beta(pat)_1) = 1$ and their positions are the same in the predicate attributes, given by $n_a(\beta(ann)_1) = n_a(\beta(pat)_1) = 2$. As both appear along with the same constant *bob* (2).

$$x_a(\beta(ann)_1) = [(0,2,3),(0,5,1)] \text{ and } x_a(\beta(pat)_1) = [(0,2,4),(0,6,1)]$$

This path will give hypotheses which make `hasDaughter(pat)` deducible, which is not desirable since `hasDaughter(pat)`$\in e^-$. Call this an *inconsistent path*, and the instance is dropped and another selected.

$n_c(\beta(ann)_2) = n_c(\beta(pat)_2) = 1$ and $n_a(\beta(ann)_2) = n_a(\beta(pat)_2) = 1$. This time their attribute-mates are different, *eve* (8) for *ann* and *jim* (7) for *pat*

$$x_a(\beta(ann)_2) = [(0,5,2),(0,8,1)] \text{ and } x_a(\beta(pat)_2) = [(0,6,2),(0,7,1)].$$

This splits the path into two branches and it cannot be said, at this stage, if both will lead to atomic sentences belonging to the Herbrand Base, thus deducible. Call this a *plausible path*. From this point the first literal for the body of the hypothesis can be considered as a generalization of `parent(ann, eve)`, i.e. `parent(X, `Z_0`)` (where $\{ann/X, eve/Z_0\}$ is the inverse or anti-unification of terms) has to be confirmed by pruning away any possible predicate found in the bindings of *jim* onwards.

$$+\beta(eve) = [(1,7,2),(3,5,1)] \text{ and } -\beta(jim) = [(1,6,2),(2,3,1)]$$

Their first bindings fail in the same inconsistent path as in the case of *ann* and *pat*, and so they must be dropped. However, their occurrences happen at different predicates as shown by

$$n_c(\beta(eve)_2) = 3 \text{ (for female)} \quad n_c(\beta(jim)_2) = 2 \text{ (for male)}$$

This means that the path has reached a state of positive path only, meaning that predicate 2 ($male$) can be dropped and $female(eve)$ ends the search for this branch. Add to the body the general formula $female(Z_0)$. The search for a hypothesis might stop here and Amao would learn

$$hasDaughter(X) \leftarrow parent(X, Z_0) \wedge female(Z_0)$$

This hypothesis meets the desired definition. If search is continued, further hypotheses might be found such as

$$hasDaughter(X) \leftarrow parent(X, Z_0) \wedge female(Z_0) \wedge female(X)$$

which is also consistent with BK, e^+ and e^-.

Algorithm 1 LinkagePattern(p_k, p_{k1} are T-Nodes)

```
1: if n_c(p_k) = n_c(p_k1) then                          ▷ possible recursive pattern
2:     if x_a(p_k) ∩ x_a(p_k1) ≠ ∅ then
3:         if n_a(p_k) < n_a(p_k1) then      ▷ (position of a_k is less than position of a_k1)
4:             return linear_and_recursive
5:         else if n_a(p_k) = n_a(p_k1) then
6:             return sink_hook
7:         else
8:             return side_hook_pattern
9:     else
10:        if n_a(p_k) < n_a(p_k1) then      ▷ (position of a_k is less than position of a_k1)
11:            return linear_or_recursive
12:        else if n_a(p_k) = n_a(p_k1) then
13:            return deep_sink_hook
14:        else
15:            return long_side_hook
16: else
17:    if x_a(p_k) ∩ x_a(p_k1) = ∅ then
18:        return unknown_hook
19:    else
20:        return short_linear_hook
```

3.2 Recursive Linkage Pattern

Suppose Amao is asked to generate hypotheses with `ancestor(X,Y)` with positive background knowledge `ancestor(pam,jim)`.

```
consider induction on ancestor(X,Y) knowing ancestor(pam,jim).
```

Although there is a long linear linkage from `pam` until `jim`, the successful expected hypotheses should be recursive having $\sim$`parent(X,Y)` as base. Amao generates

these as learned hypotheses by following the same steps as for linear linkage, except that the first pair of p_k and p_{k1} from $\beta(a_k)$ and $\beta(a_{k1})$ is checked for equality. In other words, if $n_c(p_k) \neq n_c(p_{k1})$, then the linkage pattern is linear and proceed as above. Otherwise it is possible that there is a *recursive pattern*. Algorithm 1 diagnoses which linkage pattern to apply (including some not discussed here as they are intended to be used within a neural network learning extension of the current method for large background knowledge). In Algorithm 1 $n_a(p_k)$ and $n_a(p_{k1})$ are the attribute positions for a_k and a_{k1}, respectively.

3.3 Inductive Momentum

The definition of ILP and all implementations use negative examples e^- more as a testing case to check whether a generated hypothesis H plus BK will entail e^-. If so, then the cause of the undesired deduction is detected, fixed and a new hypothesis generated. The approach in this work is different: negative example can be used to prune away candidates to generalized body literals if they have been reached from the bindings of terms from e^- (call them negative terms).

The *inductive momentum* between two T-Nodes a^+ of $\boldsymbol{x}_a^+$ and a^- of $\boldsymbol{x}_a^-$ given partial θ^{-1} is given by Algorithm 2.

Algorithm 2 InductiveMomentum($\boldsymbol{x}_a^+$,a^+,$\boldsymbol{x}_a^-$,a^-)

```
1: if no attribute of x_a^+ has an anti-substitution in θ^-1 then
2:     return useless_path.
3: if n_c(a^+) = n_c(a^-) then
4:     if n_a(a^+) = n_a(a^-) then
5:         if both attribute-mates a^+ of x_a^+ and a^- of x_a^- are equal then
6:             return inconsistent_path
7:         else
8:             return plausible_path
9:     else
10:        plausible_path
11: return positive_path_only
```

4 The Inductive Learning Algorithm

The induction algorithm presented in Algorithm 3 works as a search guided by the linkage pattern (Algorithm 1) and inductive momentum (Algorithm 2). Algorithm 3 is given a shared NeMuS $\mathcal{N}$, a target P with code p/n, and coded positive and negative examples e^+ and e^- with their respective coded terms being a_k^+ (possibly a_{k1}^+), a_k^- (possibly a_{k1}^-) respectively.

Note that step 8 guarantees that ground atoms common to the e^+ and e^- derivation chain are left out of the partial clause hypothesis building process.

Algorithm 3 InductiveLearn($\mathcal{N}$,p_k, p_{k1} are T-Nodes)

1: $\theta_T \leftarrow \theta^{-1}(a_k, a_{k1})$
2: $\beta_k^+ \leftarrow \beta(a_k^+)$ (and $\beta_{k1}^+ \leftarrow \beta(a_{k1}^+)$ if exists a_{k1}^+) from $\mathcal{N}$
3: $\beta_k^- \leftarrow \beta(a_k^-)$ (and $\beta_{k1}^- \leftarrow \beta(a_{k1}^-)$ if exists a_{k1}^-) from $\mathcal{N}$
4: **while** $\beta_k \neq \varnothing$ **do**
5: $H_p \leftarrow \theta_T P$
6: select p_k^+ (p_{k1}^+) p_k^- (p_{k1}^-) from β_k^+ (β_{k1}^+) and β_k^- (β_{k1}^-)
7: **if** There exists a useful LinkagePattern(p_k^+, p_{k1}^+) **then**
8: **if** InductiveMomentum($x_a(n_c(p_k^+)), n_a(p_k^+)), x_a(n_c(p_k^-)), n_a(p_k^-)$) **then**
9: **if** linkage is `linear_and_recursive` **then**
10: $\theta_b \leftarrow \theta^{-1}(x_a(n_c(p_k^+))) \cup clone(\theta_T)$
11: $H_b \leftarrow \theta_b(clone(H_p) \cup \sim p_k)$
12: $\theta_r \leftarrow \theta^{-1}(x_a(n_c(p_{k1}^+))) \cup \theta_T$
13: $H_r \leftarrow \theta_r(clone(H_p) \cup \sim p_{k1})$
14: **else**
15: $\theta_1 \leftarrow \theta^{-1}(clone(\theta_T), x_a(n_c(p_k^+)))$
16: $\theta_2 \leftarrow \theta^{-1}(\theta_1, x_a(n_c(p_{k1}^+)))$
17: $H_p \leftarrow \theta_2(clone(H_p) \cup \sim p_k \cup \sim p_{k1})$
18: **if** $\theta_1 = \theta_2$ **then**
19: save hypotheses generated
20: update βs with bindings from attribute-mates(linkage terms).

This avoids inconsistency by not allowing the generation of hypotheses that would satisfy e^- along with BK.

Consider again the example with `ancestor/2` to give an intuitive idea of how negative examples are used as an *inductive momentum*. Everything else is left out of the hypothesis since it will not be part of the HB that satisfies the positive examples e^+ plus BK and hypothesis. The process builds the hypothesis by selecting from the intersection those that meet one of the linkage patterns, and are not eliminated as a result of an inductive momentum.

```
>> consider induction on ancestor(X,Y) knowing ancestor(pam,jim).
--> Consider using these hypotheses...
ancestor(X,Y); ~parent(X,Y).
ancestor(X,Y); ~parent(X,Z0); ~ancestor(Z0,Y).
```

5 Related Work

Inductive logic programming has a large literature, from its antecedents in inductive generalization [14], through work on search strategies, the logical framework that the learning sits in, as well as the building of systems and application of these to specific problems. Inductive learning continues to be of interest in a wide range of contexts and applications as recently surveyed in [5].

Of particular relevance is the work in [4] that investigates path based algorithms to generate relationships and [15] that uses inductive logic programming

concepts in an early instance of theory repair (that is, revising a theory that is incorrect so that the counter-examples are no longer such). Additionally [6], that investigates variations on the standard anti-unification algorithm and how these impact on the efficiency of hypothesis search, is of interest in the current context. More recently, in [1] boolean constraints are used to describe undesirable areas of the hypothesis search space and solving these to prune the search space achieves significant speed ups over older inductive logic programming systems on a range of examples, whilst retaining accuracy.

The higher-order approach taken in [11, 12] uses a meta-interpreter with iterative deepening to build Metagol. Metagol has had success on a range of examples, including learning a subclass of context-free grammars from examples and inferring relationships in sports data (whilst also uncovering an error in the data representation). This includes predicate invention, a topic that these papers suggest has not been paid due attention in inductive learning.

6 Discussion and Future Work

This paper has shown how the Amao Shared NeMuS data structure can be used to build an inductive logic programming system which has been successfully applied on some small trial examples.

The results on inductive learning in Amao show that using its shared structure leads to reliable hypothesis generation in the sense that the minimally correct ones are generated. However, it still generates additional hypothesis, logically sound and correct with respect to the Herbrand base derivation. Most important is the size of the set of hypotheses generated which is small in comparison with the literature, e.g. [3].

Future work will focus on two areas. First, the power of the shared structure to allow a fast implementation of inductive inference. Second, the weights incorporated in the Shared NeMuS structure (not used in the current paper) will be used to play an important role in providing heuristics. In [9] it is shown how these weights can be updated in a manner inspired by self-organising maps [7]. The propagation across the network of nodes in the structure allows the weights to capture patterns of refutation. It should be possible to capture negative examples in inductive logic programming in the network in this way, guiding search away from these unfruitful regions to fine tune to a small set of generated hypotheses. Alongside improved hypothesis search the use of the weighted structure to drive predicate invention – to add to the knowledge base additional inferred predicates contained in neither it nor the target predicate – will be investigated.

Acknowledgement

The authors would like to thank Gerson Zaverucha for fruitful discussion and guidance through emails about Inductive Logic Programming.

References

1. Ahlgren, J., Yuen, S.Y.: Efficient Program Synthesis Using Constraint Satisfaction in Inductive Logic Programming. Journal of Machine Learning Research 14, 3649–3681 (2013)
2. Boyer, R.S., Moore, J.S.: The Sharing of Structure in Theorem-Proving Programs. In: Machine Intelligence 7. pp. 101–116. Edinburgh University Press (1972)
3. Bratko, I.: Prolog Programming for Artificial Intelligence, 4th edition. Addison Wesley (2011)
4. Bunescu, R.C., Mooney, R.J.: A Shortest Path Dependency Kernel for Relation Extraction. In: Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing. pp. 724–731. Association for Computational Linguistics (2005)
5. Gulwani, S., Hernández-Orallo, J., Kitzelmann, E., Muggleton, S.H., Schmid, U., Zorn, B.G.: Inductive Programming Meets the Real World. Communications of the ACM 58(11), 90–99 (2015)
6. Idestam-Almquist, P.: Generalization under Implication by Recursive Anti-unification. In: International Conference on Machine Learning. pp. 151–158. Morgan-Kaufmann (1993)
7. Kohonen, T.: Self-Organizing Maps. Springer, 3rd edn. (2001)
8. Mao, L.: An introduction to Smarandache multi-spaces and mathematical combinatorics. Scientia Magna 3(1), 54–80 (2007)
9. Mota, E.d.S., Diniz, Y.B.: Shared Multi-Space Representation for Neural-Symbolic Reasoning. In: Besold, T.R., Lamb, L., Serafini, L., Tabor, W. (eds.) NeSy 2016. vol. 1768. CEUR Workshop Proceedings (July 2016)
10. Muggleton, S.H.: Inductive Logic Programming. New Generation Computing 8(4), 295–318 (1991)
11. Muggleton, S.H., Lin, D., Pahlavi, N., Tamaddoni-Nezhad, A.: Meta-interpretive learning: application to grammatical inference. Machine Learning 94(1), 25–49 (2014)
12. Muggleton, S.H., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. Machine Learning 100(1), 49–73 (2015)
13. Nienhuys-Cheng, S.H., De Wolf, R.: Foundations of Inductive Logic Programming, Lecture Notes in Artificial Intelligence, vol. 1228. Springer (1997)
14. Plotkin, G.D.: A Note on Inductive Generalization. In: Machine Intelligence 5. pp. 153–164. Edinburgh University Press (1969)
15. Richards, B.L., Mooney, R.J.: Automated Refinement of First-Order Horn-Clause Domain Theories. Machine Learning 19(2), 95–131 (1995)
16. Robinson, A.: A machine-oriented logic based on the resolution principle. Journal of the ACM 12(1), 23–42 (1965)

A Comparison between Deep Q-Networks and Deep Symbolic Reinforcement Learning

Aimoré R. R. Dutra[1] and Artur S. d'Avila Garcez[1]

[1] City, University of London, London, EC1V 0HB, UK
aimorerrd@hotmail.com, a.garcez@city.ac.uk

Abstract. Deep Reinforcement Learning (DRL) has had several breakthroughs, from helicopter controlling and Atari games to the Alpha-Go success. Despite their success, DRL still lacks several important features of human intelligence, such as transfer learning, planning and interpretability. We compare two DRL approaches at learning and generalization: Deep Q-Networks and Deep Symbolic Reinforcement Learning. We implement simplified versions of these algorithms and propose two simple problems. Results indicate that although the symbolic approach is promising at generalizing and faster learning in one of the problems, it can fail systematically in the other, very similar problem. **Keywords:** Deep Reinforcement Learning, Deep Q-Networks, Neural-Symbolic Integration.

1 Introduction

The combination of classical Reinforcement Learning with Deep Neural Networks achieved human level capabilities at solving some difficult problems, especially in games with Deep Q-Networks (DQNs) [3]. There is no doubt that Deep Reinforcement Learning (DRL) has offered new perspectives for the areas of automation and AI. But why are these methods so successful? And why are they still unable to solve many problems that seem so simple for humans? Despite their success, DRL has several drawbacks. First, they need large training sets and hence learn slowly. Second, they are very task specific - a trained network that performs well on one task often performs very poorly on another, even very similar task. Third, they are difficult to extract a human-comprehensible chain of reasons for the action choices that the system makes.

Some authors have been trying to solve some of the above shortcomings by adding prior knowledge to the system, using model-based architectures and other AI concepts [2]. One claims to have designed an architecture that solves at once all these shortcomings by combining neural-network learning with aspects of symbolic AI, called Deep Symbolic Reinforcement Learning (DSRL) [1]. In this paper, in an attempt to understand better the advantages of a symbolic approach to Reinforcement Learning, we implement and compare two simplified versions of DQN and DSRL at learning a simple video game policy.

2 The Video Game

The Deep Q-Network (DQN) was reduced to a simple Q-Learning algorithm by removing its convolutional and function approximation layers. These layers do not seem to play a major role in how an agent makes its decisions. They basically reduce the dimensionality of the states. In the Deep Symbolic Reinforcement Learning (DSRL), we ignored the first low-level extraction part. In our implementation, we skip this first part by sending the location and type of each object directly to the agent. In addition, only a spatial representation is considered, since there is no complex dynamics relating to time in the game. The simplified versions of DQN and DSRL were implemented in Python 3.5.

Fig. 1 shows three initial configurations of the proposed game. The star-shaped object is the Agent, the negative sign denotes a Trap, and the positive sign is the Goal. The agent can move up, left, right and down, and it stays at the same place when it tries to move into the wall. The reward is increased by 1 and decreased by 10 whenever the Agent's position is the same as the Goal and the Trap, respectively. The game only restarts if the Agent's position is the Goal. The environment is fully-observable, sequential, static, discrete, unknown, infinite, stationary and deterministic. Two toy examples are proposed to evaluate how DQN and DSRL apply their learned knowledge in a new, similar situation, namely, training in configuration 1 and testing in 2 (c.f. Fig. 1), and training in 2 and testing in 3.

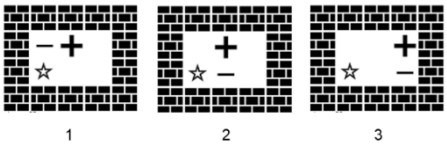

Fig. 1. Three initial game configurations

3 Results and Discussion

Fig. 2 shows that both algorithms (DQN and DSRL) learn well during the training phase, but in the test phase, while DQN has a behavior similar to random, DSRL always falls into the Trap before reaching the Goal. This shows that, while DQN could not learn from conf. 1 what to do in conf. 2; and DSRL learned something completely wrong for conf. 2 (always move to the right). It is as if any prior knowledge in DSRL had to be undefeasible, which is an unrealistic constraint. DQN, by contrast, had never seen the states in the test case during training; thus, it assumed a random policy. The reason why DSRL has very low reward is because the Goal's location did not change from training to test. Thus, our DSRL Agent

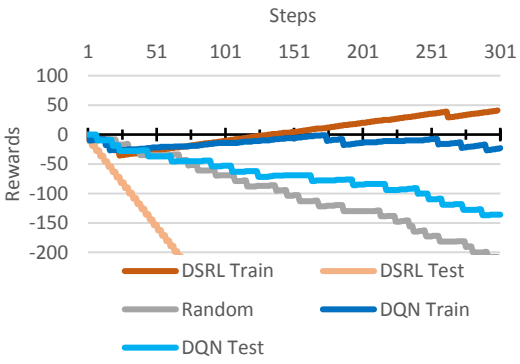

Fig. 2. Trained in conf. 1 and tested in conf. 2

assumed that the best action should remain the same (move right). The position of the Trap did not have any influence in the Agent's decision because the algorithm treats different types of objects independently. In other words, the DSRL Agent does not know what rewards to expect from a Trap in a new location.

In the second example (trained in conf. 2 and tested in 3), the situation is quite different, as Fig. 3 shows. DSRL learns how to make the right decision, and thus has good performance during testing. DQN flat lines as a result of not knowing the states in the test phase. It is interesting noting that DSRL avoided the Trap during testing because it has learned how to translate from conf. 2 to 3 (but not how to reflect from conf. 1 to 2 (c.f. Fig. 2), or to rotate a configuration, which should produce similar results as Fig. 2 for obvious reasons). Such an ability to generalize to new situations is very important, as it allows an agent to learn from similar states without having to experience them all. In the case of DSRL, generalizations bring faster learning, but seem limited to translations of configurations.

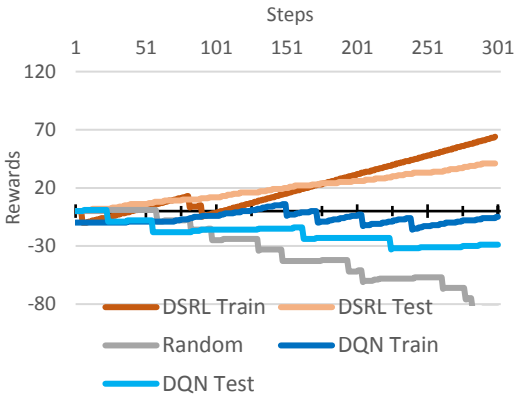

Fig. 3. Trained in conf. 2 and tested in conf. 3

4 Conclusion

We have compared two model-free RL approaches, DRL and DSRL, on their generalization capacity using two toy examples. Both have limitations at learning "the rules of the game" for succeeding in different configurations. One key finding is that transforming pixels into symbols can become a channel not only for reducing the state-space, but to enable rules between objects to be created. These rules offer a way of generalizing states, and could guide an agent during exploration. Assisted by high level rules, an agent should learn faster by exploring its environment more efficiently. Thus, as future work, we shall consider the combination of model-free and model-based approaches with symbolic rules being used for faster and hopefully more effective learning.

References

[1] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Confidence Values and Compact Rule Extraction From Probabilistic Neural Networks

Simon Odense and Artur d'Avila Garcez

City, University of London, UK
simon.odense@city.ac.uk
a.garcez@city.ac.uk,

One of the key challenges of extracting rules from neural networks is accommodation of the inherent flexibility of knowledge representation in neural networks to more rigid rule based systems. Neural networks are often seen as having 'soft constraints' as opposed to the 'hard constraints' of rule based systems. This distinction has been identified as one of the key differences between the connectionist approach and more traditional symbolic AI [1]. For deterministic networks, the distinction becomes somewhat fuzzy as every input/output relationship of a network can be encoded in a set of propositional rules with arbitrary precision, however, representing a neural network this way will be at the very least incomprehensible and in most cases intractable. Thus the issue of flexibility is tied to the issue of compactness of representation. For probabilistic networks, the issue of flexibility becomes even more of a challenge. Here we go over results showing that a previous rule extraction method applied to Restricted Boltzmann machines(RBMs) [5] can be improved by considering more compact rules called M of N rules. We also consider an example highlighting the advantage that these rules have in terms of minimizing the incidents of 'false negatives' over traditional conjunctive rules. Finally we look at the notion of 'confidence values', numeric values we associate with a rule meant to represent our degree of belief in the rule, and show that a more refined notion of confidence may be helpful when considering extracted rules from RBMs and other probabilistic networks.

Tran and Garcez developed a rule extraction algorithm for RBMs (and DBNs built from RBMs) which associates confidence values with extracted rules by looking at the average weight of the literals in the rule [2]. The extraction algorithm works by starting with the conjunction of every literal in the rule and iteratively updating the confidence value and pruning literals with small enough weights until equilibrium is achieved. The extracted rules are then composed in a deep belief network along with an inference rule in order to calculate confidence values for the output given a (partial) set of confidence values for the input. When looking at RBMs in isolation, the extracted rules can be thought of as biconditionals, however, the following example shows that when looking at RBMs a high confidence value does not necessarily correspond to a high probability of the rule being true. First, when we say that an extracted rule has a certain probability in the network we mean that, given that the visible units are uniformly distributed (if the visible distribution is defined by the network it can be shown that local rule extraction preserving the probabilities is impos-

sible assuming some basic conditions on the network), the rule has a certain probability of being true in the distribution of the network. For example, if a network has a probability distribution P, for two hidden units in a network, h_1 and h_2, $h_1 \vee h_2$ is given a probability $P(h_1) + P(h_2)$. Given a biconditional $h \leftrightarrow x_1, ..., x_k, \neg x_{k+1} ... \neg x_n$, where each x_i represents a visible unit, we will consider the probability of the biconditional being true in an RBM. For brevity we will denote the antecedent of the biconditional as ANT, the probability of this biconditional in an RBM is then $P(h = 1, ANT) + P(h = 0, \neg ANT)$ Where the distribution on the set of literals in the antecedent is uniform (since they represent visible units). We will consider an example of a rule extracted using the algorithm mentioned above to show that the associated confidence doesn't reflect the probability of the biconditional in the network. Define a network with a single hidden neuron with k identical weights W and bias 0, the antecedent of the extracted rule is the conjunction of all the literals and the confidence is W. This means that the antecedent is satisfied only when all k literals are satisfied. Using some algebra, the probability of the biconditional being true in the network can be written as

$$P(h = 1|ANT)P(ANT) + \sum_{i=0}^{k-1} \binom{k}{i} (1 - P(h = 1|ANT^i))P(ANT^i)$$

Where $P(h = 1|ANT)$ is the probability of the hidden neuron being on when the antecedent is satisfied and $P(h = 1|ANT^i)$ is the probability of the hidden neuron being on when exactly i literals of the rule are satisfied, since all the weights are the same this does not depend on which specific literals are not satisfied. Furthermore we are assuming that this visible units are taken from a uniform distribution so we have $P(ANT) = P(ANT^i) = \frac{1}{2^k}$. This gives us

$$\frac{1}{2^k} \left(\sigma(Wk) + \sum_{i=1}^{k-1} \binom{k-1}{i} (1 - \sigma(iW)) \right)$$

Since W and k are arbitrary we can take them to be as large as possible, in which case the limit of the right term goes to $1 - \sigma(0) = 0.5$ and the left hand term goes to 1 so as $k \to \infty$ the whole thing goes to 0. This shows we can extract rules with arbitrarily high confidence but arbitrarily low probability.

The issue with this example is that the extracted rule give many false negatives. There are many cases where the rule should be giving an output of 1 but is failing to since not every literal is satisfied. Rather than requiring every literal in the antecedent be satisfied in order to predict 1 we really only need one of them. It's difficult to extract a single conjunctive rule which can accurately capture the behaviour of a probabilistic network and by extracting many different rules you lose compactness. In order to find a compact way to more faithfully capture the behaviour of an RBM we relax the condition that every literal in the antecedent needs to be satisfied. This give us the so called M of N rules. In an

M of N rule the antecedent is satisfied if only M of the N literals are. In the previous example the correct rule would be 1 of the set of literals. By first applying the rule extraction algorithm and selecting M by looking for the minimum value of M for which $M \cdot c$ (where c is the confidence given to the rule) is greater than a predetermined threshold (in our case the minimum input to the hidden node) we can convert the purely conjunctive rules into M of N ones. If we cannot find an appropriate M we add new literals until there either is an appropriate M or we run out of literals. The rules produced by this algorithm perform much better than the purely conjunctive rules when tested with a variety of small datasets [6].

Assigning values to logical sentences to measure degrees of belief has been done before. The most relevant examples for us are penalty logic [4] and Markov logic networks [3], in both cases 'weights' were given to logical sentences which were then translated into weights of a network (Hopfield networks and Markov random fields respectively). A similar philosophy was used to define confidence values for deep belief networks by using the weights of the RBM in the previous algorithm. The above example shows that the extracted confidence really does not accurately reflect the underlying probability of structure of the constituent RBMs and that the extracted rules are perhaps better considered in the feed forward context rather rather than biconditionals. Extending this algorithm to M of N relieves some of the problems by loosening the requirements for the rule to be satisfied but it remains to be seen whether the confidence values extracted with M of N rules more accurately reflect the probability structure of the RBM. One possible avenue of research is, rather than look simply at the weights attached to the literals to derive confidence, look at both the minimum input to a node when the rule is satisfied and the maximum input to the node when the rule is not satisfied. Ultimately the M of N rule is a promising way of representing knowledge in a neural network with more possibilities to imbue it with more flexibility by exploring various notions of confidence values

References

[1] Smolensky, P..: On the Proper Treatment of Connectionism. Behavioral and Brain Sciences. 11(1), 1–23 (1988)

[2] Son, T., d'Avila Garcez, A.: Deep Logic Network: Inserting and Extracting Knowledge from Deep Belief Networks. IEEE Transactions on Neural Networks and Learning Systems, pp(99), 1–13(2016)

[3] Richardson, M., Domingos, P.: Markov Logic Networks. Machine Learning. 62(1), 107–136(2006)

[4] Pinkas, G.: Reasoning, Connectionist Nonmonoticity and Learning in Networks that Capture Propositional Knowledge. Artificial Intelligence. 77, 203–247

[5] Smolensky, P.: Information Processing in Dynamical Systems: Foundations of Harmony Theory. Parallel Distributed Processing: Volume 1: Foundations. MIT Press, Cambridge, 194–281 (1986) Comm. Pure Appl. Math. 33, 609–633 (1980)

[6] Odense, S., d'Avila Garcez, A.: Extracting M of N Rules From Restricted Boltzmann Machines. The 26th International Conference on Artificial Neural Networks. to appear, (2017)

Learning with Knowledge Graphs

Volker Tresp[1,2], Yunpu Ma[1,2], Stephan Baier[2]

[1] Siemens AG, Corporate Technology, Munich, Germany
[2] Ludwig-Maximilians-Universität München, Munich, Germany

Abstract. In recent years a number of large-scale triple-oriented knowledge graphs have been generated. They are being used in research and in applications to support search, text understanding and question answering. Knowledge graphs pose new challenges for machine learning, and research groups have developed novel statistical models that can be used to compress knowledge graphs, to derive implicit facts, and to detect errors in the knowledge graph. In this paper we decribe the concept of triple-oriented knowledge graphs and corresponding learning approaches. We also discuss episodic knowledge graphs which are able to represent temporal data; learning with episodic data can be the basis for decision support systems, e.g. in a clinical context. Finally we discuss how knowledge graphs can support perception, by mapping subsymbolic sensory inputs, such as images, to semantic triples. A particular feature of our approach would be that perception, episodic memory and semantic memory are highly interconnected and that, in a cognitive interpretation, all rely on the same brain structures.

1 Semantic Knowledge Graphs

A technical realization of a semantic memory is a knowledge graph (KG) which is a triple-oriented knowledge representation: A labelled link implies a *(subject, predicate, object)* statement where subject and object are entities that are represented as the nodes in the graph and where the predicate labels the link from subject to object. Large KGs have been developed that support search, text understanding and question answering [8]. A KG can be represented as a tensor which maps indices to true or false

$$s, p, o \mapsto Q$$

with $Q \in \{\mathtt{T}, \mathtt{F}\}$, and where $s \in 1, \ldots, N$ and $o \in 1, \ldots, N$ are indices for the N entities used as subject and object, and where $p \in 1, \ldots, R$ is the index for the predicate.

A statistical model for a KG can be obtained by a tensor model of the form

$$s, p, o \mapsto \mathbf{a}_{e(s)}, \mathbf{a}_p, \mathbf{a}_{e(o)} \mapsto P. \tag{1}$$

Here $e(s)$ and $e(o)$ are the entities associated with subject and object, respectively. The indices are first mapped to their latent representations $\mathbf{a}_{e(s)}, \mathbf{a}_p, \mathbf{a}_{e(o)}$

which are then mapped to a probability $P \in [0,1]$. $P((s,p,o) = \mathtt{T}|\mathbf{a}_{e(s)}, \mathbf{a}_p, \mathbf{a}_{e(o)})$ represents the Bernoulli probability that the triple (s,p,o) is true, and, when normalized across all triples, $P(s,p,o|\mathbf{a}_{e(s)}, \mathbf{a}_p, \mathbf{a}_{e(o)})$ stands for the categorical probability that the triple (s,p,o) is selected as an answer in a query process. A number of mathematical models have been developed for the mapping in Equation 1 (see [7]). A representative example is the RESCAL model [6], which is a constraint Tucker2 tensor model.

2 Episodic Knowledge Graphs

Whereas a semantic KG model reflects the state of the world, e.g, of a clinic and its patients, observations and actions describe factual knowledge about discrete events. Generalizing the semantic KG, an episodic KG can be represented as a 4-way tensor with time index t as the map

$$s, p, o, t \mapsto Q.$$

A statistical model for a KG can be obtained by a 4-way tensor model of the form

$$s, p, o, t \mapsto \mathbf{a}_{e(s)}, \mathbf{a}_p, \mathbf{a}_{e(o)}, \mathbf{a}_t \mapsto P \tag{2}$$

where $\mathbf{a}_t$ is the latent representation for time index t.

The basis for the tight link between different memory functions is the "unique representation hypothesis", which states that an entity has a unique latent representation in a technical application, but maybe also in the human brain [9].

As discussed in [11, 5] both the episodic KG and the semantic KG might rely on the same representations, i.e., it was proposed that the semantic KG can be derived from the episodic KG by a marginalization operation. Thus an episodic fact might represent that "Jack, wasDiagnosed, Diabetes, on Jan 15", the derived semantic fact might be "Jack, hasDisease, Diabetes". In [3, 4] medical decision systems are described that combine semantic and episodic tensor representations of data with recurrent neural network predictive models.

3 Perception

The tensor models permit generalization, i.e., the prediction of the probability of triples which were not known to be true in the data. This is especially important in perception, which we propose can be thought off as the mapping of subsymbolic sensory inputs to a semantic description in the form of a set of triples, describing and explaining the sensory inputs. These triples then becomes part of episodic memory.

Let $u_{t,1}, \ldots, u_{t,c}, \ldots, u_{t,C}$ be the content of the sensory buffers at time t. We propose that this sensory input can predict the latent representation for time in the form of a map

$$u_{t,1}, \ldots, u_{t,c}, \ldots, u_{t,C} \mapsto \mathbf{a}_t.$$

This map $\mathbf{a}_t(\mathbf{u}_t, \mathbf{w})$ might be modelled by a deep neural network with weights $\mathbf{w}$. Perceptual decoding then produces likely triples from the probability distribution (generalized nonlinear model) using

$$P(s, p, o; \mathbf{a}_{e(s)}, \mathbf{a}_p, \mathbf{a}_{e(o)}, \mathbf{a}_t(\mathbf{u}_t, \mathbf{w})).$$

An episodic memory would simply store $\mathbf{a}_t$, and memorizing simply means the restoring of a past $\mathbf{a}_t$, which then can be decoded as described [9, 10]. A semantic memory uses the marginalizing approach describes in Section 2.

As another approach, there is the option to use $P(s, p, o)$ or $P(s, p, o, t)$ as a semantic prior in sensory decoding. This was the basis for approaches to extract triples from Web sources [2] and for the extraction of triples from images [1].

References

[1] Stephan Baier, Yunpu Ma, and Volker Tresp. Improving visual relationship detection using semantic modeling of scene descriptions. In *ISWC*, 2017.

[2] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *KDD*, 2014.

[3] Cristóbal Esteban, Danilo Schmidt, Denis Krompaß, and Volker Tresp. Predicting sequences of clinical events by using a personalized temporal latent embedding model. In *Healthcare Informatics (ICHI), 2015 International Conference on*, 2015.

[4] Yinchong Jang, Volker Tresp, and Peter Fasching. Predictive modeling of therapy decisions in metastatic breast cancer with recurrent neural network encoder and multinomial hierarchical regression decoder. In *ICHI*, 2017.

[5] Yunpu Ma, Volker Tresp, and Erik Daxberger. Embedding models for episodic memory. In *submitted*, 2017.

[6] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A Three-Way Model for Collective Learning. In *ICML*, 2011.

[7] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs: From multi-relational link prediction to automated knowledge graph construction. *Proceedings of the IEEE*, 2015.

[8] Amit Singhal. Introducing the Knowledge Graph: things, not strings. *Official Google Blog*, 2012.

[9] Volker Tresp, Cristóbal Esteban, Yinchong Yang, Stephan Baier, and Denis Krompaß. Learning with memory embeddings. *NIPS 2015 Workshop (extended TR); arXiv:1511.07972*, 2015.

[10] Volker Tresp, Yunpu Ma, and Stephan Baier. Tensor memories. In *CCN*, 2017.

[11] Volker Tresp, Yunpu Ma, Stephan Baier, and Yinchong Yang. Embedding learning for declarative memories. In *ESWC*, 2017.

Learning about Actions and Events in Shared NeMuS

Milena Rodrigues Tenório[1], Edjard de Souza Mota[1]
Jacob M. Howe[2], Artur S. d'Avila Garcez[2]

[1] Universidade Federal do Amazonas,
Instituto de Computação, Campus Setor Norte
Coroado - Manaus - AM - Brasil CEP: 69080-900
{mrt,edjard}@icomp.ufam.edu.br,
[2] City, University of London, London, EC1V 0HB, UK
{J.M.Howe,a.garcez}@city.ac.uk

1 Introduction

The categorization process of information from pure data or learned in unsupervised artificial neural networks is still manual, especially in the labeling phase. Such a process is fundamental to knowledge representation [6], especially for symbol-based systems like logic, natural language processing and textual information retrieval. Unfortunately, applying categorization theory in large volumes of data does not lead to good results mainly because there is no generic and systematic way of categorizing such data processed by artificial neural networks and joining investigated conceptual structures.

Connectionist approaches are capable of extracting information from artificial neural networks, but categorizing them as symbolic knowledge have been little explored. The obstacle lies on the difficulty to find logical justification from response patterns of these networks [2]. This gets worse when considering inductive learning from dynamic data which is very important to Cognitive Sciences that considers categorization as a mental operation of classifying objects, actions and events [1].

We shall address the discoveries of our on-going investigation on the problem of inductively learning (IL) from dynamic data by applying a novel framework for neural-symbolic representation and reasoning called share Neural Multi-Space (NeMuS) used in the Amao system[4]. Instead of woking like traditional approaches for ILP, e.g. [5], Amao uses a shared NeMuS of a give background knowledge (BK) and uses inverse unification as the generalization mechanism of a set of logically connected expressions from the Herbrand Base (HB) of BK that defines positive examples.

2 Discussion on Inductive Learning about Dynamic Data

Our scenario on the realm of Driving offences considering a small portion of it in a usual traffic light. Consider four cars `c1`, `c2`, `c3` and `c4`, and a traffic light `tl1` on a street `s1`, as depicted in Figure 1.

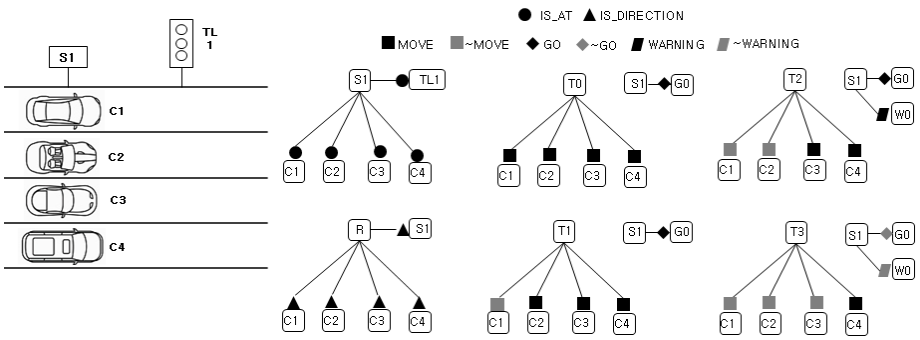

Fig. 1. BK representation of example.

The logical entities considered are based on dynamic nature of the data[3], like a historical database about a traffic light. No external action is assumed to influence this scenario and the passage of time is based on actions on it.

```
car(c1).                is_at(c1,s1).          is_direction(c1,r).
car(c2).                is_at(c2,s1).          is_direction(c2,r).
car(c3).                is_at(c3,s1).          is_direction(c3,r).
car(c4).                is_at(c4,s1).          is_direction(c4,r).
street(s1).             is_at(tl1,s1).         is_direction(s1,r).
traffic_light(tl1).
```

There are four moments of interest.**M1**: All cars are moving; **M2**: `c1` stopped on street; **M3**: `tl1` signals warning, so `c2` has stopped; and **M4**: `tl1` stops blinking yellow and goes to stop sign (red), then `c3` has stopped.

M1	**M2**	**M3**	**M4**
			`~warning(tl1,w1).`
`go(tl1,g0).`			`~go(tl1,g1).`
`move(c1,t0).`	`~move(c1,t1).`	`~move(c1,t2).`	`~move(c1,t3).`
`move(c2,t0).`	`move(c2,t1).`	`~move(c2,t2).`	`~move(c2,t3).`
`move(c3,t0).`	`move(c3,t1).`	`move(c3,t2).`	`~move(c3,t3).`
`move(c4,t0).`	`move(c4,t1).`	`move(c4,t2).`	`move(c4,t3)..`

```
during(t0,t2,g0).          during(t2,t2,w0).
during(t3,t3,g1).          during(t3,t3,w1).
```

Change through time of an entity (car or traffic light) is represented by a change from positive to negative literal of `move`, `warning` and `go`. The goal is to identify which cars have committed a traffic light or driving offense. Traffic regulations would point that `c1` and `c4` violated such laws. `c1` is blocking (it stopped) traffic on `s1` while `tl1` indicates to go, and `c4` is moving when `tl1` is indicating to stop. For lack of space, we consider just the target `block_offence(C)` with positive example `block_offence(c1)` and negative `~block_offence(c2)`, and the target `driving_offence(C)` with positive example `driving_offence(c4)` and negative `~driving_offence(c3)`. The following (unsual long) hypothesis should be generated.

```
block_offence(C); car(C); traffic_light(TL); is_at(C,S); is_at(TL,S);
    move(C,TA); go(TL,GA); ~move(C,TB); warning(TL,W1);
    during(TA,TB,GA); during(TX,TY,W1).
driving_offence(C); car(C); traffic_light(TL); is_at(C,S); is_at(TL,S);
   move(C,TA); go(TL,GA); move(C,TB); ~go(TL,GB);
   during(TA,TX,GA); during(TY,TB,GB).
```

Shared NeMuS codes allow us to know that (1) the car and traffic light are on the same street, and (2) the car has the same direction of the street. Predicate code 2 can be ignored inasmuch as it is unnecessary information, because it is a rule that involves carriage movement and obedience to traffic light. Amao gets the bindings relations (occurrences) the given objects as positive and negative examples. As `c1`, `c2`, `c3`, and `c4` are cars they belong to the same region of predicate codes. `is_at()` relating to `s1`, both have `is_direction()` with `r` (right), and have a single object different that also relates through `is_at ()` with `s1`, the `tl1`. The difference between the positive and negative examples lies in their *action predicates*, in which `c1` performs an action before any change occurs in `t1`. We, not in Amao a automatic mechanism, also noticed that the predicate `is_direction ()` and `r` do not relate in any way to another object or to *action predicates*. Thus, we can ignore this information in the construction of a hypothesis. By inverse unification, Amao finds a linkage pattern between *is_at(C,S)* and *is_at(TL,S)*, and thus connecting the car and the traffic light.

3 Concluding Remarks

The example explored here is small, and yet we have a long rule. An example with more information, such as velocity, position of the car and people on the street,their relations we will have many ways to find hypotheses. We aim to overcome this challenge by using shared NeMuS weight to to group the predicates that form an intermediate concept, an abstraction, so that we can add only the predicates needed for the rules.

References

1. Cohen, H., Lefebvre, C.: Handbook of categorization in cognitive science. Elsevier (2005)
2. d'Avila Garcez, A.S., Broda, K., Gabbay, D.: Neural-Symbolic Learning Systems: Foundations and Applications, Perspectives in Neural Computing. Springer-Verlag (2002)
3. Gardenfors, P.: Concept learning and non-monotonic reasoning. In: Handbook of Categorization (2005)
4. Mota, E.d.S., Howe, J., Garcez, A.: In: Besold, T.R., d'Avila Garcez, A., Noble, I. (eds.) To appear NeSy 2017 Neural-Symbolic Learning and Reasoning (July)
5. Muggleton, S.H.: Inductive Logic Programming. New Generation Computing 8(4), 295–318 (1991)
6. Sowa, J.F., et al.: Knowledge representation: logical, philosophical, and computational foundations, vol. 13. MIT Press (2000)

Category-based Inductive Learning in Shared NeMuS

Ana Carolina Melik Schramm[1], Edjard de Souza Mota[1],
Jacob M. Howe[2], and Artur S. d'Avila Garcez[2]

[1] Universidade Federal do Amazonas,
Instituto de Computação, Campus Setor Norte
Coroado - Manaus - AM - Brasil CEP: 69080-900
{acms, edjard}@icomp.ufam.edu.br,
[2] City, University of London, London, EC1V 0HB, UK
{J.M.Howe,a.garcez}@city.ac.uk

1 Introduction

One of the main objectives of cognitive science is to use abstraction to create models that represent accurately the cognitive processes that constitute learning, such as categorisation. Relational knowledge is important in this task, since through the reasoning processes of induction and analogy over relations that the mind "creates" categories (it later estabilishes causal relations between them by using induction and abduction), and analogies exemplify crucial properties of relational processing, like structure-consistent mapping[2].

Given the complexity of the task, no model today has accomplished it completely. The associacionist/connectionist approach represents those processes through associations between different informations. That is done by using artificial neural networks. However, it faces a great obstacle: the idea (called propositional fixation) that neural networks could not represent relational knowledge. A recent attempt to tackle the symbolic extraction from artificial neural networks was proposed in [1]

The cognitive agent Amao uses a shared Neural Multi-Space (Shared NeMuS) of coded first-order expressions to model the various aspects of logical formulae as separate spaces, with importance vectors of different sizes. Amao [4] uses inverse unification as the generalization mechanism for learning from a set of logically connected expressions of the Herbrand Base (HB). Here we present an experiment to use such learning mechanism to model a simple version of train set from Michalski's train problem[3].

2 Shared NeMuS Approach to Train Problem

In Michalski's train problem, there are 10 trains: 5 eastbound and 5 westbound. Whether a train is going east or west is determined by its properties. Using these trains, a simple base has been created, taking into account the size of the train wagons (short or not) and whether these wagons are closed or not. The number

of wheels, wagon format and other attributes have been ignored in order to make the base simpler.

All the eastbound trains have at least one wagon which is both short and closed. That is what determines whether a train is eastbound or westbound. The idea is to use the shared NeMuS structure to induce the rule eastbound knowing that `t1` (the first train) is going east. Having that information, we can directly get all predicate instances, called as *bindings*, which have `t1` is an attribute. They are the following:

```
train(t1).
car(t1, c1_t1).
car(t1, c2_t1).
car(t1, c3_t1).
car(t1, c4_t1).
```

```
short(c1_t1).
closed(c1_t1).
```

The predicate `car` links `t1` to all its wagons (or carriages), so `car(t1, c1_t1)` means that `c1_t1` is a wagon that belongs to `t1`. Taking the first instance of the predicate `car`, we now know that `t1` has a wagon named `c1_t1`. Amao, through its shared NeMuS, accesses `c1_t1`'s bindings and using a polynomial search, finds both occurrences of `c1_t1` in `short` and `closed`, as seen above. This mechanism is called linkage pattern in Amao's learning mechanism.

At this point `t1` is a train that has `c1_t1` as a wagon, and this wagon is not closed. Amao also has the linkage predicate connecting both `c1_t1` and `t1`. Thus, a candidate hypothesis generated would look like `eastbound(X)` $\leftarrow$ `car(X, Y)` $\wedge \sim$ `short(Y)` $\wedge \sim$ `closed(Y)`. However, this may not be the only possible hypothesis, so the other wagons being carried by `t1` need to be considered.

```
short(c2_t1).          ∼short(c3_t1).          short(c4_t1).
closed(c2_t1.          ∼closed(c3_t1).         ∼closed(c4_t1).
```

Among the possible hypotheses that may define a train as being eastbound, we have:

```
eastbound(X) ← car(X, Y) ∧ ∼short(Y) ∧ ∼closed(Y).
 eastbound(X) ← car(X, Y) ∧ short(Y) ∧ closed(Y).
eastbound(X) ← car(X, Y) ∧ short(Y) ∧ ∼closed(Y).
```

Adding negative examples, we can reduce the number of possible hypotheses. In this case, the simplest way to do that is to use the 10th train `t10` as a negative example. Using the same method as explained above, the structure can select all predicates that have `t10` as an attribute:

```
car(t10, c1_t10).
car(t10, c2_t10).
```

Then, all the predicates that have `t10s` wagons as attributes:

```
short(c1_t10).                          ∼short(c2_t10).
∼closed(c1_t10).                        ∼closed(c2_t10).
```

Thus, the hypotheses that definitely do not define a train as being eastbound are:

$$\texttt{eastbound(X)} \leftarrow \texttt{car(X, Y)} \wedge \texttt{short(Y)} \wedge \sim\texttt{closed(Y).}$$
$$\texttt{eastbound(X)} \leftarrow \texttt{car(X, Y)} \wedge \sim\texttt{short(Y)} \wedge \sim\texttt{closed(Y).}$$

Both hypotheses are among the possible options defined above. Excluding them, the correct option remains. The target `eastbound(X)` can be defined by:

$$\texttt{eastbound(X)} \leftarrow \texttt{car(X, Y)} \wedge \texttt{short(Y)} \wedge \texttt{closed(Y).}$$

Formalizing what was explained above:

1. With the positive example (`t1`), get all predicates (bindings) that have `t1` as an attribute;
2. Access bindings of attributes linked to t1 using polynomial search (linkage pattern)
   - in this case, the attributes are `c1_t1`, `c2_t1` and `c3_t1`
3. repeat the first two steps for the negative example (`t10`)]
   - in this case, the attributes linked to `t10` are `c1_t10` and `c2_t10`
4. if there are hypotheses generated by using the positive example that are repeated in the negative example, they will not be in the list of possible hypotheses.
   - some of the hypotheses generated by using only the positive example are:

     $$\texttt{eastbound(X)} \leftarrow \texttt{car(X, Y)} \wedge \sim\texttt{short(Y)} \wedge \sim\texttt{closed(Y).}$$
     $$\texttt{eastbound(X)} \leftarrow \texttt{car(X, Y)} \wedge \texttt{short(Y)} \wedge \texttt{closed(Y).}$$
     $$\texttt{eastbound(X)} \leftarrow \texttt{car(X, Y)} \wedge \texttt{short(Y)} \wedge \sim\texttt{closed(Y).}$$

     However, using only the negative example, the first and third hypotheses would also be generated. By using both examples, these two don't make it into the list of possible hypotheses, and the correct one, which is $\texttt{eastbound(X)} \leftarrow \texttt{car(X, Y)} \wedge \texttt{short(Y)} \wedge \texttt{closed(Y)}$, remains.

3 Concluding Remarks

The knowledge base created is only a simplification of the original train problem. As explained before, many attributes such as number of wheels, wagon format, load shape and roof shape have been ignored. Had they been included, more hypotheses could have been generated through Amao's inductive learning mechanism over the shared NeMuS. One current limitation is not being able to deal with predicate invention, that would allow to automatically create categories by means of abstraction/new predicates.

One possible road to explore is to take advantage of shared NeMuS weights to integrate a neural network classification method to help identify categories. In the train set, we know which trains are eastbound, but whatever rule defines the eastbound category is not known before using Amao to define it. Understanding what makes a train eastbound or not can help us categorize any train that might be added to the set in the future.

Another goal we aim to pursue is to make use of weights to implement neural mechanisms. We expect to envisage more efficient heuristics to guide hypotheses generation, improving Amao's learning mechanism.

References

1. França, M.V.M., D'Avila Garcez, A.S., Zaverucha, G.: Relational knowledge extraction from neural networks. In: Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches - Volume 1583. pp. 146–154. COCO'15, CEUR-WS.org, Aachen, Germany, Germany (2015), http://dl.acm.org/citation.cfm?id=2996831.2996849
2. Halford, G.S., Wilson, W.H., Phillips, S.: Relational knowledge: the foundation of higher cognition 14, 597–505 (2010)
3. Larson, J.B., Michalski, R.S.: Inductive Inference of VL Decision Rules 14, 16–20 (1977)
4. Mota, E.d.S., Howe, J., Garcez, A.: In: Besold, T.R., d'Avila Garcez, A., Noble, I. (eds.) To appear NeSy 2017 Neural-Symbolic Learning and Reasoning (July)

Q-SATyrus: Mapping Neuro-symbolic Reasoning into an Adiabatic Quantum Computer

Priscila M. V. Lima[1][0000-0002-8515-9904]

[1] Tercio Pacitti Institute, Federal University of Rio de Janeiro, Rio de Janeiro 21941-916, Brazil
priscilamvl@gmail.com

Abstract. Much has been promised about quantum computing accelerators, but few actual commercial technologies have been made available so far. The D-Wave Computers Series constitutes one family of adiabatic quantum computers, based on energy minimization techniques that are considered suitable for solving discrete optimization problems. This work shows a path to explore these machines in order to perform neuro-symbolic reasoning, by specifying it a a set of pseudo-Boolean constraints and associating their satisfiability to energy minimization. Also introduced is the platform Q-SATyrus, a spin-off of the original project SATyrus. Q-SATyrus is under development in order to systematically address such mappings.

Keywords: neuro-symbolic reasoning, adiabatic quantum computing, artificial symmetric neural networks

1 Q-SATyrus : Considering Adiabatic Quantum Computing for Neuro-symbolic Reasoning

Based on the adiabatic theorem, adiabatic quantum computing performs some calculations that some consider being a kind of quantum computing [1]. The Canadian company D-Wave Systems, founded in 1999, has developed a family of adiabatic computers, the newest one, the D-Wave 2000Q™ system, with 2000 qbits [2][3][4].

In D-WAVE systems, there are binary variables, named qubits q_i in$\{0, 1\}$. Each qubit may have an associated *weight* a_i (same as the threshold of Artificial Neural Networks [5]) and a pair of qubits q_i and q_j have their mutual influence named *coupler* (same as the binary weight of Artificial Neural Networks [5]) and represented by b_{ij} [6]. The general specification for the problem solved by a D-WAVE system is given by equation (1), which represents the objective function to be minimized. Is is also worth noting that the same equation (1) represents an artificial neural network with symmetric binary connections [5].

$$min\ O(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{q}) = \sum a_i\, q_j + \sum b_{ij}\, q_j\, q_j \tag{1}$$

By converting propositional satisfiability into energy minimization [7], some works specified limited depth proofs, among them it is possible to cite [8], [9], [10] and [11].

Works [9], [10] and [11] led to the construction of the SATyrus platform other more traditional optimization problems as well as some of their combinations were also mapped to SATyrus [12], [13], [14], [15], [16]. It should be pointed out that the mappings issued by SATyrus do not generate only binary connections energy equations. However, it is possible to convert higher-order connections into a set of binary ones together with additional units [17]. Q-SATyrus will provide the necessary intermediate conversion of energy minimization with higher-order connections to the one with corresponding global minima with binary connections. Also, although the works on binders [18], [19] and [20] were implemented in conventional computing, it is possible to map their solution to adiabatic computing.

Acknowlegements

The author wishes to thank Professor Alberto Ferreira De Souza for suggesting considering D-Wave in connection with the SATyrus platform. This work was partially sponsored by FAPERJ BBP grant E-26/201.444/2014 (coordinator: Professor Valmir Barbosa).

References

1. McGeoch, C. C., Wang, C.: Experimental Evaluation of an Adiabatic Quantum System for Combinatorial Optimization. In: Proceedings of the CF'13. Ischia (2013).
2. D–WAVE Homepage, https://www.dwavesys.com, last accessed 2017/06/18.
3. D–WAVE (D-Wave 2000Q™ System) Homepage, https://www.dwavesys.com/d-wave-two-system, last accessed 2017/06/18.
4. D–WAVE (Press Release) Homepage, https://www.dwavesys.com/press-releases/d-wave%C2%A0announces%C2%A0d-wave-2000q-quantum-computer-and-first-system-order, last accessed 2017/06/18.
5. Haykin, S.: Neural Networks: A Comprehensive Foundation, New Jersey, NJ, USA: PrenticeHall Internattional (1999).
6. D–WAVE (Programming) Homepage, https://www.dwavesys.com/sites/default/files/Map%20Coloring%20WP2.pdf, last accessed 2017/06/18.
7. Pinkas, G.: Symmetric neural networks and propositional logic satisfiability. Neural Computation, 3, 282–291 (1991a).
8. Pinkas, G.: Constructing syntactic proofs in symmetric networks. In Proceedings of Advances in Neural Information Processing Systems (NIPS-91), pp. 217–224 (1991b).
9. Lima, P. M. V.: A Neural Propositional Reasoner that is Goal-Driven and Works Without Pre-Compiled Knowledge, Proc. of the 6th Brazilian Symposium on Neural Networks. IEEE Computer Society Press, 1. pp 261-266. Rio de Janeiro (2000).
10. Lima, P. M. V.: Resolution-based Inference with Artificial Neural Networks, PhD Thesis. Imperial College London (2000).
11. Lima, P. M. V.: A Goal-Driven Neural Propositional Interpreter. International Journal of Neural Systems, 11(3), pp 311-322 (2001).
12. Lima, P. M. V., Pereira, G. C., Morveli-Espinosa, M. M. M., França, F. M. G., Lavor. C. C.: Mapping Molecular Geometry Problems into Pseudo-Boolean Constraints. In: Proceed-

ings of International Workshop on Genomic Databases - IWGD´05, 1. Rio de Janeiro (2005).
13. Lima, P. M. V., Pereira, G. C., Morveli-Espinosa, M. M. M., França, F. M. G.: Mapping and Combining Combinatorial Problems into Energy Landscapes via Pseudo-Boolean Constraints. Lecture Notes in Computer Science. , vol. 3704, pp 308 – 317. Springer (2005).
14. Lima, P. M. V., Pereira, G. C., Morveli-Espinosa, M. M. M., França, F. M. G.: (2005) SATyrus: A SAT-based Neuro-Symbolic Architecture for Constraint, Proc. of HIS'05: Fifth International Conference on Hybrid Intelligent Systems. Los Alamitos, CA, USA: IEEE Computer Society Press, 2005. p.137 – 142.
15. Lima, P. M. V., Pereira, G. C., Morveli-Espinosa, M. M. M., Ferreira, T. O., França, F. M. G.: Logical Reasoning via Satisfiability Mapped into Energy Functions. International Journal of Pattern Recognition and Artificial Intelligence, 33(5), pp 1031–1043 (2008)
16. Silva, E. F., Lima, P. M. V., Diacovo, R., França, F. M. G.: Aggregating energy scenarios using the SATyrus neuro-symbolic tool. In: Proceedings of the 19th International Symposium on Mathematical Programming, pp 146–146. Rio de Janeiro (2006).
17. Venkatesh, S., Baldi, P.: Programmed Interactions in Higher-Order Neural Networks: Maximal Capacity. Journal of Complexity, 7, pp 316–337 (1991).
18. Pinkas, G., Lima, P. M. V., Cohen, S.: Compact Crossbar Variable Binding for Neuro-Symbolic Computation In: NeSy'11 CEUR Workshop Proceedings, 764, pp 14-18 (2011).
19. Pinkas, G., Lima.P. M. V., Cohen, S.: A Dynamic Binding Mechanism for Retrieving and Unifying Complex Predicate-Logic Knowledge. In: ICANN'2012 Proceedings, 1, pp 482–490. Lausanne (2012).
20. Pinkas, G., Lima, P. M. V., Cohen, S.: Representing, binding, retrieving and unifying relational knowledge using pools of neural binders. Biologically Inspired Cognitive Architectures. , 6, 87 – 95 (2013).