Addressing the Symbol Grounding Problem with Constraints in Neuro-Symbolic Planning

Aymeric Barbin[1,2,*], Federico Cerutti[2,3] and Alfonso Emilio Gerevini[2]

[1]Sapienza Università di Roma, Italy
[2]Università degli Studi di Brescia, Italy
[3]Cardiff University, UK

Abstract

In this paper, we address the Symbol Grounding Problem (SGP) in the context of neuro-symbolic planning, where the categorical vectors learned to represent high dimensional inputs suffer from instability, which poses a problem of efficiency during the planning phase. One way to alleviate the SGP is to enforce constraints — among the latent variables — by expressing them in the loss function during the learning process. Combining an existing tool for invariant search and ideas from Logic Tensor Networks (fuzzy logic), we propose to automatize the process of finding and enforcing relevant constraints. We apply our idea to LatPlan, a domain independent, image-based classical planner.

Keywords

Neuro-Symbolic Planning, Symbol Grounding Problem, Action Model Learning

1. Introduction

The core interest of Domain-Independent Planning [1] is developing general-purpose algorithms and systems that can solve planning problems independently of any specific knowledge of the latter. A planning problem can be specified using the Planning Domain Description Language (PDDL) [2] in terms of a symbolic description of the states and actions composing the domain, and initial state, and a goal.

When the states of the problem are only available as sub-symbolic data (e.g., images), generating a PDDL description necessarily needs to address the Symbol Grounding Problem (SGP) [3] which refers to the case where two different inputs (e.g., images) grounding the same symbol (e.g., digit "1") have different vector representations, revealing a lack of generalisation power.

LatPlan [4] is a neuro-symbolic architecture for planning proposed to address the bottleneck of PDDL construction from raw input data. It leverages Deep Learning to learn the PDDL description from a set of unlabeled pairs of transition images. More specifically, it uses variational autoencoders (VAE) [5] to learn and generate categorical vector representations of the images (states) and of the transitions between them (actions), which are then used to generate the

Italian Workshop on Planning and Scheduling (IPS-2022, 10th edition)
*Corresponding author.
aymeric.barbin@uniroma1.it (A. Barbin); federico.cerutti@unibs.it (F. Cerutti); alfonso.gerevini@unibs.it (A. E. Gerevini)

PDDL. LatPlan is also endowed with tools that alleviate the SGP of these representations, which can be assessed by measuring their variance on perturbed input.

Another way to address the SGP is to enforce constraints on the categorical vectors. If we consider a vector v as an ideally grounded symbol and its unstable version v', then v' contains some noise, i.e. variables with undesired values. If we know appropriate constraints to apply during training, the noisy representation corresponding to v' should converge to v.

In this paper, we propose to use an automatic tool to look for invariants in the PDDL generated by LatPlan and to test their incidence on the learning by expressing them as an auxiliary loss term. To do so, we borrow ideas from Logic Tensor Networks (LTN) [6], where this additional loss is created using the *t-norm fuzzy logic* [7]. In Section 2, we provide background on LatPlan, invariants, and fuzzy logic, then, in Section 3, we discuss our proposed solution to integrate the search for constraints into the training loop.

2. Background

2.1. LatPlan

LatPlan is a neuro-symbolic architecture that receives pairs of images representing transitions and learns categorical vector representations from them. Once trained, it can generate a PDDL representation of the states and actions of the problem, which can be input to a planner.

For our work, we are interested in improving the learning of two components of LatPlan, the State AutoEncoder network (SAE), which learns to represent images as categorical vectors, and the Action Model Acquisition network (AMA), which learns to represent actions as categorical vectors representing preconditions and effects. These two models are learned end-to-end from a dataset of pairs of images, each pair representing a valid transition occurring in the sub-symbolic world (e.g. the switching of a tile in the case of 8-puzzle).

The SAE learns a bi-directionnal mapping between sub-symbolic raw data x and propositional states $z \in \{0,1\}^F$ (with F being the number of variables in the categorical vector). Concretely, it consists of the encoder and the decoder of a VAE that learns $\mathsf{DECODE}(\mathsf{ENCODE}(x)) = x$.

The AMA model consists of three networks: ACTION, APPLY and REGRESS. ACTION learns to associate two consecutive states (returned by the SAE) to an action (expressed as a one-hot vector). APPLY learns to predict the next categorical state from the previous one and an action. REGRESS is symmetric of APPLY and learns to predict backwards the previous state from the next one and from the action.

The SAE, APPLY and REGRESS networks all output binary categorical vectors of the same size as the SAE's output. Each element of these vectors is interpreted as a binary variable and is represented by a unary predicate in the PDDL files generated by LatPlan. This representation is negatively affected by the SGP, as it can break the identity assumption inherent to symbolic reasoning algorithms — in which a state must not change — and can cause disconnections during the search process, i.e. if two states exist that should represent the same state, an action might lead to one of them but not to the other, the latter would then be a dead end.

To address this issue, the authors of LatPlan propose two solutions. First, at test time, they replace the sampling of the categorical vector with an *argmax* layer, therefore removing stochasticity. Second, during training, they select a version of the prior distribution for the

sampling (of the categorical vector) that favours sparsity of truthiness among the binary variables. This leads to more stable latent state vectors, and showed improved performances of the model in terms of next-state prediction accuracy (APPLY network) and planning performances. In our work, we are mainly interested in stabilizing the latent states vectors, generated by the SAE network; to assess it, we use the same metric as in LatPlan paper, i.e. the State Variance. To compute it, we use the state variance for noisy input, i.e., the variance of the latent vectors :

$$z^{i,0} = ENCODE(x^{i,0} + n)$$

where:

- $n \sim N(\mu = 0, \sigma = 0.3)$
- $x^{i,0}$: 1^{st} image of the i^{th} transition pair of the dataset

The State Variance is computed by iterating over 10 random vectors, then averaged over *F* bits in the latent space and over the dataset indexed by *i*. Formally:

$$\mathbb{E}_{f \in 0...F}\, \mathbb{E}_i\ Var_{j \in 0..10}[\mathsf{ENCODE}(x^{i,0}, n^j)_f] \tag{1}$$

2.2. Invariants and constraints in planning

In classical planning, an invariant is defined as a logical formula over variables of the domain which is true in any reachable state [8]. Invariants can be seen as hidden but logical properties of the domain, and can also be termed as *state constraints*. In LatPlan, we can enforce them in learning the SAE and AMA networks.

An important part of the research in classical planning focuses on discovering invariants in planning domains. Today, automatic tools [9] [10] already exist that can find a variety of invariants, such as predicate domain invariants (i.e., in the effect of an action), static invariants on predicates (ones that are unaffected by any operator), simple implicative invariants (e.g., if $z_1 \rightarrow z_2$), mutually exclusive invariant (e.g., $\neg z_1 \vee \neg z_2$), etc.

2.3. Logic Tensor Networks and Fuzzy logic

Extensive work on integrating logical constraints *during the training* of neural networks have been conducted in the last few years. Notably, one can express constraints among neural networks outputs taken as predicates thanks to *t-norm* operations and the fuzzy generalisation of First Order Logic (FOL) [11] [7].

For example, if we want to express the truth value of $\neg z_1 \vee \neg z_2$, we can compute it by its average over the dataset, i.e.,

$$\frac{1}{|Z|} \sum_{z_1, z_2 \in Z} (1 - P(z_1)) + (1 - P(z_2)) \tag{2}$$

where $P(z_1)$ is the (continuous) value of truth of one grounding of "z_1 *is true*", and Z is the set of all the groundings. The inverse of this value can be directly appended as a penalty to the loss function, which eventually forces the network to re-adapt its weights to this new constraint.

3. Automatically finding and enforcing relevant invariant candidates

We first discuss how we intend to search for invariants of interest among LatPlan binary variables with the help of an automatic tool; then, we discuss how to enforce these invariants during training and we give details about integrating the search in the training loop.

3.1. Searching for invariants of interest

Automatic tools, like Fast Downward (FD) [10] and DISCOPLAN [9], can find invariants in the PDDL representation outputted by LatPlan using invariant synthesis . But, since LatPlan learns the PDDL by statistical inference, there is no guarantee that any of the invariants found in the PDDL maps to the ground truth invariants, i.e., to invariants of the (unknown) ground truth PDDL domain. For example if $z_1 \rightarrow z_2$ is returned by FD as an invariant, but z_1 is true only once in the whole dataset, it's possible that $z_1 \rightarrow z_2$ would be false if we had more data with z_1 being true. Thus, in our work we consider the invariants computed by an invariants generation tool (on LatPlan PDDL) as a set of *probable invariants* that we intend to test.

3.2. Applying the constraints

In Latplan, internal categorical representations are continuous vectors from which elements converge to binary values during training. At each batch, we can compute the truth value — thanks to fuzzy logic — of any constraint, expressed as a logical forumula, over these binary variables, for instance "z_1 *is true*." Then, by taking the inverse of this value and multiplying it by a normalizing factor, we obtain an additional loss that we can append to the total loss. This way, the network will adapt its weights through backpropagation to comply with the constraint.

3.3. Augmenting the training loop with a search over invariants

Our idea is to integrate an automatic invariant finder in the learning process of LatPlan. More precisely, we want to perform a search — similar to a hyperparameter search — on the invariants. Further details can be find in Appendix A. The implementation of this loop builds upon the t-norm like functions that already exist in Tensorflow [12] and the possibility to customize the training loop in Keras [13], LatPlan being coded with both of them.

4. Conclusions and Future Work

We discussed the idea of combining an automatic invariants finder with fuzzy logic to progress in solving the SGP that affects the latent representations of a neuro-symbolic architecture (LatPlan). More precisely, the invariants found by the automatic tool are *probable* invariants - since the PDDL they are issued from is learned by statistical inference - and we want to enforce them during training. If they bring a lower state variance, it is probable that they correspond to *ground true invariants*. The goal is to automatically find invariants that are responsible for more state stability. We are now implementing the search loop introduced in Section 3.

References

[1] D. E. Wilkins, Domain-independent planning representation and plan generation, Artificial Intelligence 22 (1984) 269–301.

[2] P. Haslum, N. Lipovetzky, D. Magazzeni, C. Muise, An introduction to the planning domain definition language, Synthesis Lectures on Artificial Intelligence and Machine Learning 13 (2019) 1–187.

[3] M. Taddeo, L. Floridi, Solving the symbol grounding problem: a critical review of fifteen years of research, Journal of Experimental & Theoretical Artificial Intelligence 17 (2005) 419–445.

[4] M. Asai, A. Fukunaga, Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary, in: Proceedings of the aaai conference on artificial intelligence, volume 32, 2018.

[5] D. P. Kingma, M. Welling, et al., An introduction to variational autoencoders, Foundations and Trends® in Machine Learning 12 (2019) 307–392.

[6] S. Badreddine, A. d. Garcez, L. Serafini, M. Spranger, Logic tensor networks, Artificial Intelligence 303 (2022) 103649.

[7] L. A. Zadeh, Fuzzy logic, Computer 21 (1988) 83–93.

[8] V. Alcázar, A. Torralba, A reminder about the importance of computing and exploiting invariants in planning, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 25, 2015, pp. 2–6.

[9] A. Gerevini, L. K. Schubert, Discovering state constraints in discoplan: Some new results, in: AAAI/IAAI, 2000, pp. 761–767.

[10] M. Helmert, Concise finite-domain representations for pddl planning tasks, Artificial Intelligence 173 (2009) 503–535.

[11] P. Hájek, Metamathematics of fuzzy logic, volume 4, Springer Science & Business Media, 2013.

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., {TensorFlow}: a system for {Large-Scale} machine learning, in: 12th USENIX symposium on operating systems design and implementation (OSDI 16), 2016, pp. 265–283.

[13] F. Chollet, et al., Keras, https://keras.io, 2015.

A. The proposed augmented training loop

This augmented training loop, GreedyConstraintTraining, is illustrated in Algorithm 1, which uses the following auxiliary functions:

- LatPlan, which is a function that embeds the training process of LatPlan: it receives as input a training set, a set of propositional formulae to embed in the loss function, and outputs the learned functions and action descriptions as binary vectors;
- Vecs2Pddl transforms the action descriptions learned by LatPlan into PDDL format;
- Pddl2Inv is a function that returns a set of invariants of a planning problem in the PDDL format provided in the input. An example is the invariant finder of Fast Downward;

- MetricEval receives as input the learned functions and action descriptions of LatPlan and outputs a score value: the higher the score value, the better. In our case, it is the inverse of the State Variance given in Formula 1 (in this case only the SAE function of LatPlan is needed).
- PickAnInvariant is a heuristic that identifies the most promising invariant in a set; for instance an invariant with the less number of variables (because it has a higher probability to be a ground true invariant compared to one with more variables).

Algorithm 1 Our proposed GreedyConstraintTraining approach, which receives as input a training set $tSet$, and returns Ω, the invariants which improve on a chosen metric

$\Omega \leftarrow \emptyset$
$LPModel \leftarrow \mathsf{LatPlan}(tSet, \emptyset)$
$\Phi \leftarrow \mathsf{Pddl2Inv}(\mathsf{Vecs2Pddl}(LPModel))$
$score \leftarrow \mathsf{MetricEval}(LPModel)$
while $\Phi \neq \emptyset$ **do**
    $\phi \leftarrow \mathsf{PickAnInvariant}(\Phi)$
    $\Phi \leftarrow \Phi \setminus \{\phi\}$
    $LPModel' \leftarrow \mathsf{LatPlan}(tSet, \Omega \cup \{\phi\})$
    if $\mathsf{MetricEval}(LPModel') > score$ **then**
        $\Omega \leftarrow \Omega \cup \{\phi\}$
        $\Phi \leftarrow \Phi \cup \mathsf{Pddl2Inv}(\mathsf{Vecs2Pddl}(LPModel'))$
        $score \leftarrow \mathsf{MetricEval}(LPModel')$
    end if
end while
return Ω