The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary

Masataro Asai, Alex Fukunaga
Graduate School of Arts and Sciences
The 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 deep learning has achieved significant success in many fields, the knowledge is encoded in a subsymbolic representation which is incompatible with symbolic systems such as planners. We propose LatPlan, an unsupervised architecture combining deep learning and classical planning. Given only an unlabeled set of image pairs showing a subset of transitions allowed in the environment (training inputs), and a pair of images representing the initial and the goal states (planning inputs), LatPlan finds a plan to the goal state in a symbolic latent space and returns a visualized plan execution. The contribution of this paper is twofold: (1) State Autoencoder, which finds a propositional state representation of the environment using a Variational Autoencoder. It generates 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. (2) Action Autoencoder / Discriminator, a neural architecture which jointly finds the action symbols and the implicit action models (preconditions/effects), and provides a successor function for the implicit graph search. We evaluate LatPlan using image-based versions of 3 planning domains: 8-puzzle, Towers of Hanoi and LightsOut.

Note: The extended manuscript on Arxiv contains all details of Latplan, but please cite this AAAI version. Latplan code is available on Github.

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 (McDermott 2000), and in general, such symbolic models need to be provided by a human, either directly in a modeling language such as 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 or a generator, e.g., autonomous spacecraft exploration. In particular this first requires generating symbols from raw sensor input, i.e., the *symbol grounding problem* (Steels 2008).

Figure 1: An image-based 8-puzzle.

Recently, significant advances have been made in neural network (NN) deep learning approaches for perceptually-based cognitive tasks including image classification (Deng et al. 2009), object recognition (Ren et al. 2015), speech recognition (Deng, Hinton, and Kingsbury 2013), machine translation as well as NN-based problem-solving systems (Mnih et al. 2015; Graves et al. 2016). However, the current state-of-the-art, pure NN-based systems do not yet provide guarantees provided by symbolic planning systems, such as deterministic completeness and solution optimality.

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 subsymbolic representations and the symbolic representations required by domain-independent planners*.

Fig. 1 (left) shows a scrambled, 3x3 tiled version of the photograph on the right, i.e., an image-based instance of the 8-puzzle. Even for humans, this photograph-based task is arguably more difficult to solve than the standard 8-puzzle because of the distracting visual aspects. 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 (1) *has no prior assumptions/knowledge* (e.g., "sliding objects", "tile arrangement"), and (2) *must acquire all knowledge from the images*, is nontrivial. Such a system should not make assumptions about the image (e.g., "a grid-like structure"). The only assumption allowed about the nature of the

task is that it can be modeled as a classical planning problem (deterministic and fully observable).

We propose Latent-space Planner (LatPlan), an architecture which completely automatically generates a symbolic problem representation from the subsymbolic input, 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 images of the environment and its propositional representation, (2) an *action model acquisition* (AMA) system which grounds the action symbols and learns the action model, and (3) a symbolic planner. Given only a set of *unlabeled images* of the environment, and in an unsupervised manner, we train the SAE and AMA to generate its symbolic representation. 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, invokes a planner, then visualizes the plan execution. We evaluate LatPlan using image-based versions of the 8-puzzle, LightsOut, and Towers of Hanoi domains.

2 Background

Knowledge Acquisition Bottleneck While ideally, symbolic models like PDDL should be learned/generated by the machine itself, in practice, they must be hand-coded by a human, resulting in the so-called Knowledge Acquisition Bottleneck (Cullen and Bryman 1988), which refers to the excessive cost of human involvement in converting real-world problems into inputs for symbolic AI systems.

In order to fully automatically acquire symbolic models for Classical Planning, **Symbol Grounding** and **Action Model Acquisition** (AMA) are necessary. **Symbol Grounding** is an unsupervised process of establishing a mapping from huge, noisy, continuous, unstructured inputs to a set of compact, discrete, identifiable (structured) entities, i.e., symbols. For example, PDDL has six kinds of symbols: Objects, predicates, propositions, actions, problems and domains. Each type of symbol requires its own mechanism for grounding. For example, the large body of work in the image processing community on recognizing objects (e.g. faces) and their attributes (male, female) in images, or scenes in videos (e.g. cooking) can be viewed as corresponding to grounding the object, predicate and action symbols, respectively. In contrast, an **Action Model** is a symbolic/subsymbolic data structure representing the causality in the transitions of the world, which, in PDDL, consists of preconditions and effects. In this paper, we focus on propositional and action symbols, as well as AMA, leaving first-order symbols (predicates, objects) as future work.

Action Model Acquisition (AMA) Methods Existing methods require symbolic or near-symbolic, structured inputs. ARMS (Yang, Wu, and Jiang 2007), LOCM (Cresswell, McCluskey, and West 2013), and Mourão et al. (2012) assume the action, object, predicate symbols. Framer (Lindsay et al. 2017) parses natural language texts and emits PDDL, but requires a clear grammatical structure and word consistency.

Konidaris, Kaelbling, and Lozano-Pérez generated PDDL from semi-MDP (2014). They convert a probabilistic *model* into a propositional *model*, i.e., they do not generate a model from unstructured inputs. In fact, options ($\approx$ actions) in their semi-MDP have names assigned by a human (move/interact), and state variables are identifiable entities (x/y distances toward objects, light level, state of a switch) i.e. already symbolic.

Previous work in Learning from Observation, which could take images (unstructured input), typically assume domain-dependent hand-coded symbol extractors, such as *ellipse detectors* for tic-tac-toe board data which immediately obtains propositions (Barbu, Narayanaswamy, and Siskind 2010). Kaiser (2012) similarly assumes grids and pieces to obtain the relational structures in the board image.

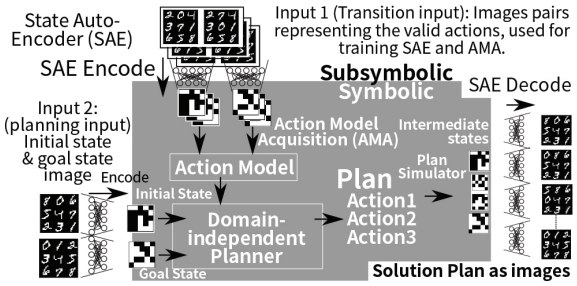

Figure 2: Classical planning in latent space: We use the learned State Autoencoder (Sec. 4) to convert pairs of images (pre, suc) first to symbolic transitions, from which the AMA component generates an action model. We also encode the initial and goal state images into symbolic initial/goal states. A classical planner finds the symbolic solution plan. Finally, intermediate states in the plan are decoded back to a human-comprehensible image sequence.

Autoencoders and Latent Representations An Autoencoder (AE) is a type of feed-forward neural network that learns an identity function whose output matches the input (Hinton and Salakhutdinov 2006). Its intermediate layer (typically smaller than the input) has a compressed, *latent representation* of the input. AEs are trained by backpropagation (BP) to minimize the reconstruction loss, the distance between the input and the output according to a distance function such as Euclidean distance. NNs, including AEs, typically have continuous activations and integrating them with propositional reasoners is not straightforward.

3 LatPlan: System Architecture

This section describes the high-level architecture of LatPlan (Fig. 2). LatPlan takes two inputs. The first input is the *transition input* Tr, a set of pairs of raw data. Each pair $tr_i = (pre_i, suc_i) \in Tr$ represents a transition of the environment before and after some action is executed. The second input is the *planning input* (i, g), a pair of raw data, which corresponds to the initial and the goal state of the environment. The output of LatPlan is a data sequence representing the plan execution that reaches g from i. While we present an image-based implementation ("data" = raw images), the architecture itself does not make such assumptions

and could be applied to the other data formats e.g. audio/text.

LatPlan works in 3 phases. In Phase 1, a *State Autoencoder* (SAE) learns a bidirectional mapping between raw data (e.g., images) and propositional states from a set of unlabeled, random snapshots of the environment. The $Encode$ function maps images to propositional states, and $Decode$ function maps the propositional states back to images. After training the SAE from $\{pre_i, suc_i \ldots\}$, we apply $Encode$ to each $tr_i \in Tr$ and obtain $(Encode(pre_i), Encode(suc_i)) = (s_i, t_i) = \overline{tr}_i \in \overline{Tr}$, the symbolic representations (latent space vectors) of the transitions.

In Phase 2, an AMA method identifies the action symbols from $\overline{Tr}$ and learns an action model, both in an unsupervised manner. We propose two approaches: AMA_1 directly generates a PDDL and AMA_2 produces a successor function (implicit model). Both methods have advantages and drawbacks. AMA_1 is a trivial AMA method designed to show the feasibility of SAE-generated propositional symbols. It does not learn/generalize from examples, instead requiring all valid state transitions. However, since AMA_1 directly produces a PDDL model, it serves as a demonstration that in principle, the approach is compatible with existing planners. AMA_2 is a novel NN architecture which jointly learns action symbols and action models from a small subset of transitions in an unsupervised manner. Unlike existing methods, AMA_2 does not require action symbols. Since it does not produce PDDL, it needs a search algorithm (such as A*) for AMA_2, or semi-declarative symbolic planners (Frances et al. 2017), instead of PDDL-based solvers.

In Phase 3, a planning problem instance is generated from the planning input (i, g). These are converted to symbolic states by the SAE, and the symbolic planner solves the problem. For example, an 8-puzzle problem instance consists of an image of the start (scrambled) configuration of the puzzle (i), and an image of the solved state (g).

Since the intermediate states comprising the plan are SAE-generated latent bit vectors, the "meaning" of each state (and thus the plan) is not necessarily clear to a human observer. However, in the final step, we obtain a step-by-step visualization of the plan execution (e.g. Fig. 4) by $Decode$'ing the latent bit vectors for each intermediate state.

In this paper, we evaluate LatPlan as a high-level planner using puzzle domains such as the 8-puzzle. Mapping a high-level action to low-level actuation sequences via a motion planner is beyond the scope of this paper.

4 SAE as a Gumbel-Softmax VAE

First, note that a direct 1-to-1 mapping from images to propositions can be trivially obtained from the array of discretized pixel values or an image hash function. However, such a trivial SAE lacks the crucial properties of *generalization* – ability to describe unseen world states with the same symbols – *robustness* – two similar images that represent "the same world state" should map to the same representation – and *bijection* – ability to map symbolic states to real-world images. We need a bidirectional mapping where the symbolic representation captures the "essence" of the image, not merely the literal, raw pixel vector.

The first technical contribution of this paper is the proposal of a SAE which is implemented as a Variational Autoencoder (Kingma et al. 2014) with a Gumbel-Softmax (GS) activation (Jang, Gu, and Poole 2017) (Fig. 3).

A Variational Autoencoder (VAE) (Kingma and Welling 2013) is a type of AE that forces the *latent layer* (the most compressed layer in the AE) to follow a certain distribution (e.g., Gaussian). Since the random distribution is not differentiable (BP is not applicable), VAEs use *reparametrization tricks*, which decompose the target distribution into a differentiable and a purely random distribution (the latter does not require the gradient). For example, the Gaussian $N(\sigma, \mu)$ is decomposed to $\mu + \sigma N(1, 0)$, where μ, σ are learned. In addition to the reconstruction loss, VAE should also minimize the variational loss (the difference between the learned and the target distributions) measured by, e.g., KL divergence.

Gumbel-Softmax (GS) is a recently proposed reparametrization trick (Jang, Gu, and Poole 2017) for categorical distribution. It continuously approximates Gumbel-Max (Maddison, Tarlow, and Minka 2014), a method for drawing categorical samples. Assume the output z is a one-hot vector, e.g. if the domain is $D = \{a, b, c\}$, $\langle 0, 1, 0\rangle$ represents "b". The input is a class probability vector π, e.g. $\langle .1, .1, .8\rangle$. Gumbel-Max draws samples from D following π: $z_i \equiv [i = \arg\max_j (g_j + \log \pi_j)\ ?\ 1 : 0]$ where g_j are i.i.d samples drawn from $\text{Gumbel}(0, 1)$ (Gumbel and Lieblein 1954). Gumbel-Softmax approximates argmax with softmax to make it differentiable: $z_i = \text{Softmax}((g_i + \log \pi_i)/\tau)$. "Temperature" τ controls the magnitude of approximation, which is annealed to 0 by a certain schedule. The output of GS converges to a discrete one-hot vector when $\tau \approx 0$.

Our key observation is that these categorical variables can be used directly as propositional symbols by a symbolic reasoning system, i.e., this gives a solution to the propositional symbol grounding in our architecture. In the SAE, we use GS in the latent layer. Its input is connected to 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. We specify $M = 2$, effectively obtaining N propositional state variables. It is possible to specify different M for each variable and represent the world using multi-valued representation as in SAS+ (Bäckström and Nebel 1995), but we use $M = 2$ for all variables for simplicity. This does not affect the expressiveness because bitstrings of sufficient length can represent arbitrary integers.

The trained SAE provides a bidirectional mapping between the raw inputs (subsymbolic representation) and 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. These are lossy compression/decompression functions, so in general,

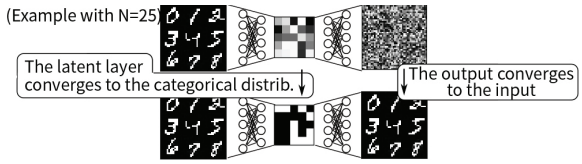

Figure 3: Step 1: Train the State Autoencoder by minimizing the sum of the reconstruction loss and the variational loss of Gumbel-Softmax. As the training continues, the output of the network converges to the input images. Also, as the Gumbel-Softmax temperature τ decreases during training, the latent values approach either 0 or 1.

$\tilde{r} = Decode(Encode(r))$ may have an acceptable amount of errors from r for visualization.

It is *not* sufficient to simply use traditional activation functions such as sigmoid or softmax and round the continuous activation values in the latent layer to obtain discrete 0/1 values. In order to map the propositional states back to images, we need a decoding network trained for 0/1 values. A rounding-based scheme would be unable to restore the images because the decoder is not trained with inputs near 0/1 values. Also, embedding the rounding operation as a layer of the network is infeasible because rounding is non-differentiable, precluding BP-based training of the network.

SAE implementation can easily and largely benefit from the progress in the image processing community. We implemented SAE as a denoising autoencoder (Vincent et al. 2008) to add noise robustness, with some techniques which improve the accuracy.

5 AMA$_1$: Oracular PDDL Generator

In AMA$_1$, our first AMA method, the output is a PDDL definition for a grounded unit-cost STRIPS planning problem. AMA$_1$ is a trivial, *oracular* strategy which generates a model based on *all* transitions, i.e., Tr contains image pairs representing all transitions that are possible in this domain, and $\overline{Tr}$ contains all corresponding symbolic transitions. The images are generated by an external, domain-specific image generator. It is important to note that while Tr for AMA$_1$ contains all transitions, the SAE is trained using only a subset of state images. Although ideally an AMA component should induce a complete action model from a limited set of transitions, AMA$_1$ is intended to demonstrate the overall feasibility of SAE-produced propositions and the overall LatPlan architecture.

AMA$_1$ compiles $\overline{Tr}$ directly into a PDDL model as follows. Each transition $\overline{tr}_i \in \overline{Tr}$ directly maps to an action a_i. Each bit $b_j (1 \leq j \leq N)$ in boolean vectors s_i and t_i is mapped to propositions `(b`$_j$`-true)` and `(b`$_j$`-false)` when the encoded value is 1 and 0 (resp.). s_i is directly used as the preconditions of action a_i. The add/delete effects of action i are computed by taking the bitwise difference between s_i and t_i. For example, when b_j changes from 1 to 0, the effect compiles to `(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.

The PDDL instance output by AMA$_1$ can be solved by an off-the-shelf planner. We use a modified version of Fast Downward (Helmert 2006). 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 Action Model.

5.1 Evaluating AMA$_1$ on Various Puzzles

We evaluated LatPlan with AMA$_1$ on several puzzle domains. Resulting plans are shown in Fig. 4. See the extended Arxiv version for further details of the network, training and inputs.

MNIST 8-puzzle is an image-based version of the 8-puzzle, where tiles contain hand-written digits (0-9) from the MNIST database (LeCun et al. 1998). 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 **Scrambled Photograph 8-puzzle (Mandrill, Spider)** cuts and scrambles real photographs, similar to the puzzles sold in stores). These differ from the MNIST 8-puzzle in that "tiles" are *not* cleanly separated by black regions (we re-emphasize that LatPlan has no built-in notion of square or movable region). In **Towers of Hanoi (ToH)**, we generated the 4 disks instances. 4-disk ToH resulted in a 15-step optimal plan. **LightsOut** is a video game where a grid of lights is in some on/off configuration (+: On), and pressing a light toggles its state as well as the states of its neighbors. The goal is all lights Off. Unlike previous puzzles, a single operator can flip 5/16 locations at once and removes some "objects" (lights). This demonstrates that LatPlan is not limited to domains with highly local effects and static objects. **Twisted LightsOut** distorts the original LightsOut game image by a swirl effect, showing that LatPlan is not limited to handling rectangular "objects"/regions.

Robustness to Noisy Input Fig. 4 (*Bottom, Right*) demonstrates the robustness of the system vs. input noise. We corrupted the initial/goal state inputs by adding Gaussian or salt/pepper noise. The system is robust enough to successfully solve the problem because of the Denoising AE (Vincent et al. 2008).

6 AMA$_2$: Action Symbol Grounding

LatPlan + AMA$_1$ shows that (1) the SAE can robustly learn image $\leftrightarrow$ propositional vector mappings from examples, and that (2) if all valid image-image transitions (i.e., the entire state space) is given, LatPlan can correctly generate optimal plans. However, AMA$_1$ is clearly not practical due to the requirement that it uses the entire state space as input, and lacks the ability to learn/generalize an action model from a small subset of valid action transitions (image pairs). Next, we propose AMA$_2$, a novel neural architecture which jointly grounds the action symbols and acquires the action model from the subset of examples, in an unsupervised manner.

Acquiring a descriptive action model (e.g., PDDL) from a set of unlabeled propositional state transitions consists of three steps. (Step 1) Identify the "types" of transitions,

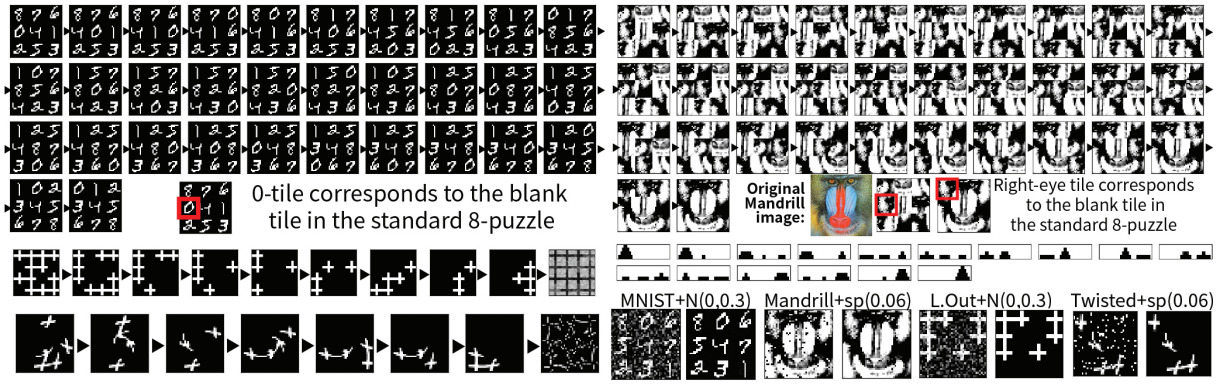

Figure 4: (*Top*) Output of LatPlan + AMA_1 solving the MNIST/Mandrill 8-puzzle instance with the longest (31 steps) optimal plan (Reinefeld 1993). This shows that LatPlan finds an optimal solution given a correct model by AMA_1 and an admissible search algorithm. LatPlan has no notion of "slide" or "tiles", making MNIST, Mandrill and Spider entirely distinct domains. (*Bottom, Left*) Output of solving 4x4 LightsOut and Twisted LightsOut. The blurs in the goal states are the noise that was normalized and enhanced by the plotting library. (*Middle, Right*) Output of solving ToH with 4 disks. (*Bottom, Right*) SAE robustness vs noise: Corrupted initial state image r and its reconstruction $Decode(Encode(r))$. r are corrupted by Gaussian noise of σ up to 0.3 and by salt/pepper noise up to $p = 0.06$. LatPlan successfully solved the problems. The SAE maps the noisy image to the correct propositional vector, finds a plan, then maps the plan back to the denoised images.

where each "type" is an identifiable, *action symbol*. For example, a hand-coded "slide-up-8-at-1-2" in 8-puzzle is an example of action symbols, but note that an AMA system should ground anonymous symbols without human-provided labels. While they are not lifted/parameterized, they still provide abstraction. For example, the same "slide-up-8-at-1-2" action, which slides the tile 8 at position $(x, y) = (1, 2)$ upward, applies to many states (each state being a permutation of tiles 1-7). (Step 2) Identify the preconditions and the effects of each action and store the information in an action model. (Step 3) Represent the model in a modeling language (e.g., PDDL).

Addressing this entire process is a daunting task. Existing AMA methods typically handle only Steps 2 and 3, skipping Step 1. Without step 1, however, an agent lacks the ability to learn in an unknown environment where it does not know *what is even possible*. Note that even if the agent has the full knowledge of its low-level actuator capabilities, it does not know its own high-level capabilities e.g. sliding a tile. Note that AMA_1 handles only Step 3.

On the other hand, search on a state space graph in an unknown environment is *feasible* even if Step 3 is missing. PDDL provides two elements, a *successor function* and its *description*. While ideally both are available, the description is not the *essential* requirement. The description may increase the explainability of the system in a language such as PDDL, but such explainability may be lost anyway when the propositional symbols are identified by SAE, as the meanings of such propositions are unclear to humans (Sec. 3). The description is also useful for constructing the heuristic functions, but the recent success of simulator-based planning (Frances et al. 2017) shows that, in some application, efficient search is possible without action descriptions.

The new method, AMA_2, thus focuses on Steps 1 and 2. It grounds the action symbols (Step 1) and finds a successor function that can be used for forward state space search (Step 2), but maintains its implicit representation. AMA_2 comprises two networks: an *Action Autoencoder* (AAE) and an *Action Discriminator* (AD). The AAE jointly learns the action symbols and the action effects, and provides the ability to enumerate the candidates of the successors of a given state. The AD learns which transitions are valid, i.e. preconditions. Using the enumeration & filtering approach, the AAE and the AD provides a successor function that returns a list of valid successors of the current state. Both networks are trained unsupervised, and operate in the symbolic latent space, i.e. both the input and output are SAE-generated bitvectors. This keeps the network small and easy to train.

6.1 Action Autoencoder

Consider a simple, linear search space with no branches. In this case, grounding the action symbol is not necessary and the AMA task reduces to predicting the next state t from the current state s. A NN a' could be trained for a successor function $a(s) = t$, minimizing the loss $|t - a'(s)|$. This applies to much of the work on scene prediction from videos such as (Srivastava, Mansimov, and Salakhudinov 2015).

However, when the current state has multiple successors, as in planning problems, such a network cannot be applied. One might consider training a separate NN for each action, but (1) it is unknown how many types of transitions are available, (2) the number of transitions depends on the current state, and (3) it does not know which transition belongs to which action. Although a single NN could learn a multimodal distribution, it lacks the ability to *enumerate* the successors, a crucial requirement for a search algorithm.

To solve this, we propose an Action Autoencoder (AAE, Fig. 5). The key idea of AAE is to reformulate the transitions as $apply(a, s) = t$, which lifts the action symbol and makes it trainable, and to realize that s is the *background in-*

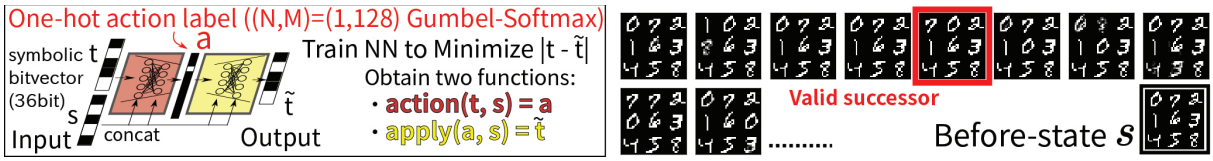

Figure 5: *(Left)* Action Autoencoder. *(Right)* The first 10 successors of a state s, generated by actions identified by AAE.

formation of the state transition function. The AAE has s, t as inputs and reconstructs t as $\tilde{t}$ whose error $|t - \tilde{t}|$ is minimized. The main difference from a typical AE is: (1) The latent layer is a Gumbel-Softmax one-hot vector indicating the **action label** a. (2) Every layer is concatenated with s. Thus s conditions the entire network and this makes the 128 action labels (7bit) represent only the *conditional information* (difference) necessary to "reconstruct t *given* s", unlike typical AEs which encode the *entire* information of the input. As a result, the AAE learns the bidirectional mapping between t and a, both conditioned by s:

- $action(t, s) = a$ returns the action label from t.
- $apply(a, s) = \tilde{t}$ applies a to s and returns a successor $\tilde{t}$.

The number of labels serves as the upper bound on the number of action symbols learned by the network. Too few labels make AAE reconstruction loss fail to converge to zero. After training, some labels may not be mapped to by any of the example transitions. In the later phases of LatPlan, these unused labels are ignored. Since we obtain a limited number of action labels, we can enumerate the candidates of the successor states of the given current state in constant time. Without AAE, all 2^N states would be enumerated as the potential successors, which is clearly impractical.

6.2 Action Discriminator

An AAE identifies the number of actions and learns their effects, but does not address the applicability (preconditions). Preconditions are necessary to avoid invalid moves (e.g. swapping 3 tiles at once) or invalid states (e.g. having duplicated tiles), as shown in Fig. 5. We address this by an *Action Discriminator* (AD) which learns the 0/1 mapping for each transition indicating whether it is valid. This is a binary classification function which takes s, t as inputs and returns a probability that (s, t) is valid.

One technical problem in training the AD is that explicit *invalid* transitions are unavailable. This is not just a matter of insufficient data, but rather a fundamental constraint in the physical environment: Invalid transitions which violate the laws of physics (e.g. teleportation) are *never* observed. We then might consider "imagining/generating" the negative examples, like a thought experiment, but it is impossible due to the lack of specification of *what* is invalid.

To overcome this issue, we use the PU-Learning framework (Elkan and Noto 2008), which can learn a positive/negative classifier from the positive and *mixed* examples that may contain both positive and negative examples. We used $\overline{Tr}$ as the positive examples (they are all valid). The mixed, i.e. possibly invalid, examples are generated by applying each action a (except unused ones) on each before-state s in $\overline{Tr}$.

State Discriminator As a performance improvement, we also trained a State Discriminator (SD) which is a binary classifier for a single state s and detects the invalid states, e.g. states with duplicated tiles in 8-puzzles. Again, we use PU-learning. Positive examples are the before/after states in $\overline{Tr}$ (all valid). Mixed examples are generated from the random bit vectors ρ (may be invalid): Many of the images $Decode$'d from ρ are blurry and do not represent autoencodable, meaningful real-world images. However, when they are repeatedly encoded/decoded, they converge to the clear, autoencodable invalid states because of the denoising AE (Vincent et al. 2008), and we used the results as the mixed examples. We use the SD to prune some mixed action examples for the AD training so that they contain only the valid successors. This improves the AD accuracy significantly.

6.3 Evaluating LatPlan using AMA_2

In the case of AMA_2, we can not use an off-the-shelf PDDL-based planner because the action model is embedded in the AAE, AD, and SD neural networks. However, they allow us to implement a successor function which can be used in any symbolic, forward state space search algorithm such as A^* (Hart, Nilsson, and Raphael 1968). The AAE generates the (potentially invalid) successors and the AD and SD filter the invalid states/transitions:

$$Succ(s) = \{t = apply(a, s) \mid a \in \{0 \dots 127\} \setminus unused, \\ AD(s, t) \geq 0.5 \wedge SD(t) \geq 0.5\}$$

We implemented A* in which states are latent-space (propositional) vectors, and the above $Succ$ function is used to generate successors of states. A simple goal-count heuristic is used. As the goal-count heuristic is inadmissible, the results could be suboptimal. However, the purpose of implementing this planner is to see the feasibility of the action model.

We evaluate the feasibility of the action symbols and the action models learned by AAE and AD. We tested 8-puzzle (mnist, mandrill, spider), LightsOut (+ Twisted). We generated 100 instances for each domain and for each noise type (std, gaussian noise, salt/pepper noise) by 7-step (benchmark A)or 14-step (benchmark B) self-avoiding random walks from the goal state, and evaluated the planner with the 180 sec. time limit. We verified the resulting image plans with domain-specific validators. Table 1 shows that the LatPlan achieves a high success rate. The failures are due to timeouts (the successor function requires many calls to the feedforward neural nets, resulting in a very slow node generation).

domain	A:step=7			B:step=14			SD error (%)		AD error (in %)			
	std	G	s/p	std	G	s/p	type1	type2	type1	type2	2/SD	2/V
MNIST	72	64	64	6	4	3	0.09	<0.01	1.55	14.9	6.15	6.20
Mandrill	100	100	100	9	14	14	<0.01	<0.01	1.10	16.6	2.93	2.94
Spider	94	99	98	29	36	38	<0.01	<0.01	1.22	17.7	4.97	4.91
L. Out	100	99	100	59	60	51	<0.01	N/A	0.03	1.64	1.64	1.64
Twisted	96	65	98	75	68	72	<0.01	N/A	0.02	1.82	1.82	1.82

Table 1: AMA_2 results: (*left*) Number of solved instances out of 100 within 3 min. time limit. The 2nd/3rd columns show the results when the input is corrupted by G(aussian) or s(alt)/p(epper) noise. In benchmark A (created with 7-step random walks), LatPlan solved the majority of instances even under the input noise. In the harder instances (benchmark B: 14-steps), many instances were still solved. (*right*) Misclassification by SD and AD in %, measured as: (SD type-1) Generate all valid states and count the states misclassified as invalid. (type-2) Generate reconstructable states, remove the valid states (w/ validator), sample 30k states, and count the states misclassified as valid. N/A means all reconstructable states were valid. (AD type-1) Generate all valid transitions and count the number of misclassification. (type-2) For 1000 randomly selected valid states, generate all successors, remove the valid transitions (w/ validator), then count the transitions misclassified as valid. (2/SD, 2/V) Same as Type-2, but ignore the transitions whose successors are invalid according to SD or the validator. Relatively large AD errors explain the increased number of failures in MNIST 8-puzzles.

We next examine the accuracy of the AD and SD (Table 1). We measured the type-1/2 errors for the valid and invalid transitions (AD) and states (SD). Low errors show that our networks successfully learned the action models.

7 Related Work

Compared to the work by Konidaris, Kaelbling, and Lozano-Pérez (2014), the inputs to LatPlan are unstructured (42x42=1764-dimensional arrays for 8-puzzle); each pixel does not carry a meaning and the boundary between "identifiable entities" is unknown. Also, AMA_2 automatically grounds action symbols, while they rely on human-assigned symbols (move, interact). Furthermore, they do not explicitly deal with robustness to noisy input, while we implemented SAE as a denoising AE. However, effects/preconditions in AMA_2 is implicit in the network, and their approach could be utilized to extract PDDL from AAE/AD (future work).

There is a large body of work using NNs to directly solve combinatorial tasks, starting with the well-known TSP solver (Hopfield and Tank 1985). Neurosolver represents a search state as a node in NN and solved ToH (Bieszczad and Kuchar 2015). However, they assume a symbolic input.

Previous work combining symbolic search and NNs embedded NNs *inside* a search to provide the search control knowledge, e.g., domain-specific heuristic functions for the sliding-tile puzzle and Rubik's Cube (Arfaee, Zilles, and Holte 2011), classical planning (Satzger and Kramer 2013), or the game of Go (Silver et al. 2016). Deep Reinforcement Learning (DRL) has solved complex problems, including video games where it communicates to a simulator through images (Mnih et al. 2015, DQN). In contrast, LatPlan only requires a set of unlabeled image pairs (transitions), and does not require a reward function for unit-action-cost planning, nor expert solution traces (AlphaGo), nor a simulator (DQN), nor predetermined action symbols ("hands", control levers/buttons). Extending LatPlan to symbolic POMDP planning is an interesting avenue for future work.

A significant difference between LatPlan and learning from observation (LfO) in the robotics literature (Argall et al. 2009) is that LatPlan is trained based on individual transitions while LfO work is largely based on the longer sequence of transitions (e.g. videos) and should identify the start/end of actions (*action segmentation*). Action segmentation would not be an issue in an implementation of autonomous LatPlan-based agent because it has the full control over its low-level actuators and initiates/terminates its own action for the data collection.

8 Discussion and Conclusion

We proposed LatPlan, an integrated architecture for learning and planning which, given only a set of unlabeled images and no prior knowledge, generates a classical planning problem, solves it with a symbolic planner, and presents the 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). Our key technical contributions are (1) *SAE, which leverages the Gumbel-Softmax to learn a bidirectional mapping between raw images and propositional symbols compatible to symbolic planners*. On 8-puzzle, the "gist" of 42x42 training images are robustly compressed into propositions, capturing the essence of the images. (2) *AMA_2, which jointly grounds action symbols and learns the preconditions/effects.* It identifies which transitions are "same" wrto the state changes and when they are allowed.

The only key assumptions about the input domain we make are that (1) it is fully observable and deterministic and (2) NNs can learn from the available data. Thus, we have shown that different domains can all be solved by the same system, without modifying any code or the NN architecture. In other words, *LatPlan is a domain-independent, image-based classical planner*. To our knowledge, this is the first system which completely automatically constructs a logical representation *directly usable by a symbolic planner* from a set of unlabeled images for a diverse set of problems.

We demonstrated the feasibility of leveraging deep learning in order to enable symbolic planning using classical search algorithms such as A*, when only image pairs rep-

resenting action start/end states are available, and there is no simulator, no expert solution traces, and no reward function. Although much work is required to determine the applicability and scalability of this approach, we believe this is an important first step in bridging the gap between symbolic and subsymbolic reasoning and opens many avenues for future research.

Acknowledgments

This research was supported by a JSPS Grant-in-Aid for JSPS Fellows and a JSPS KAKENHI grant.

References

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning Heuristic Functions for Large State Spaces. *Artificial Intelligence* 175(16-17):2075–2098.

Argall, B.; Chernova, S.; Veloso, M. M.; and Browning, B. 2009. A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems* 57(5):469–483.

Bäckström, C., and Nebel, B. 1995. Complexity Results for SAS+ Planning. *Computational Intelligence* 11(4):625–655.

Barbu, A.; Narayanaswamy, S.; and Siskind, J. M. 2010. Learning Physically-Instantiated Game Play through Visual Observation. In *ICRA*, 1879–1886.

Bieszczad, A., and Kuchar, S. 2015. Neurosolver Learning to Solve Towers of Hanoi Puzzles. In *IJCCI*, volume 3, 28–38. IEEE.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using *LOCM*. *Knowledge Eng. Review* 28(2):195–213.

Cullen, J., and Bryman, A. 1988. The knowledge acquisition bottleneck: Time for reassessment? *Expert Systems* 5(3).

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 248–255. IEEE.

Deng, L.; Hinton, G.; and Kingsbury, B. 2013. New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview. In *ICASSP*, 8599–8603. IEEE.

Elkan, C., and Noto, K. 2008. Learning Classifiers from Only Positive and Unlabeled Data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 213–220. ACM.

Frances, G.; Ramırez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In *IJCAI*, 4294–4301.

Graves, A.; Wayne, G.; Reynolds, M.; Harley, T.; Danihelka, I.; Grabska-Barwińska, A.; Colmenarejo, S. G.; Grefenstette, E.; Ramalho, T.; Agapiou, J.; et al. 2016. Hybrid Computing using a Neural Network with Dynamic External Memory. *Nature* 538(7626):471–476.

Gumbel, E. J., and Lieblein, J. 1954. Statistical theory of extreme values and some practical applications: a series of lectures.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)* 26:191–246.

Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313(5786):504–507.

Hopfield, J. J., and Tank, D. W. 1985. "Neural" Computation of Decisions in Optimization Problems. *Biological Cybernetics* 52(3):141–152.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *ICLR*.

Kaiser, L. 2012. Learning Games from Videos Guided by Descriptive Complexity. In *AAAI*.

Kingma, D. P., and Welling, M. 2013. Auto-Encoding Variational Bayes. In *ICLR*.

Kingma, D. P.; Mohamed, S.; Rezende, D. J.; and Welling, M. 2014. Semi-Supervised Learning with Deep Generative Models. In *NIPS*, 3581–3589.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Pérez, T. 2014. Constructing Symbolic Representations for High-Level Planning. In *AAAI*, 1932–1938.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-Based Learning Applied to Document Recognition. *Proc. of the IEEE* 86(11):2278–2324.

Lindsay, A.; Read, J.; Ferreira, J. F.; Hayton, T.; Porteous, J.; and Gregory, P. J. 2017. Framer: Planning Models from Natural Language Action Descriptions. In *ICAPS*.

Maddison, C. J.; Tarlow, D.; and Minka, T. 2014. A* sampling. In *NIPS*, 3086–3094.

McDermott, D. V. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 21(2):35–55.

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. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature* 518(7540):529–533.

Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *UAI*, 614–623.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *NIPS*, 91–99.

Satzger, B., and Kramer, O. 2013. Goal Distance Estimation for Automated Planning using Neural Networks and Support Vector Machines. *Natural Computing* 12(1):87–100.

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. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529(7587):484–489.

Srivastava, N.; Mansimov, E.; and Salakhudinov, R. 2015. Unsupervised Learning of Video Representations using LSTMs. In *ICML*, 843–852.

Steels, L. 2008. The Symbol Grounding Problem has been Solved. So What's Next? In de Vega, M.; Glenberg, A.; and Graesser, A., eds., *Symbols and Embodiment*. Oxford University Press.

Vincent, P.; Larochelle, H.; Bengio, Y.; and Manzagol, P.-A. 2008. Extracting and Composing Robust Features with Denoising Autoencoders. In *ICML*, 1096–1103. ACM.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning Action Models from Plan Examples using Weighted MAX-SAT. *Artificial Intelligence* 171(2-3):107–143.