

Classical Planning in Deep Latent Space

Masataro Asai

MIT-IBM Watson AI Lab, IBM Research, Cambridge, USA

MASATARO.ASAI@IBM.COM

Hiroshi Kajino

IBM Research - Tokyo, Tokyo, Japan

KAJINO@JP.IBM.COM

Alex Fukunaga

Graduate School of Arts and Sciences

The University of Tokyo

Tokyo, Japan

FUKUNAGA@IDEA.C.U-TOKYO.AC.JP

Christian Muise

School of Computing, Queen's University, Kingston, Canada

CHRISTIAN.MUISE@QUEENSU.CA

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 sub-symbolic 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), Latplan learns a complete propositional PDDL action model of the environment. Later, when a pair of images representing the initial and the goal states (planning inputs) is given, Latplan finds a plan to the goal state in a symbolic latent space and returns a visualized plan execution. We evaluate Latplan using image-based versions of 6 planning domains: 8-puzzle, 15-Puzzle, Blocksworld, Sokoban and Two variations of LightsOut.

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 Planning Domain Definition Language (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. 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).

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 et al., 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.

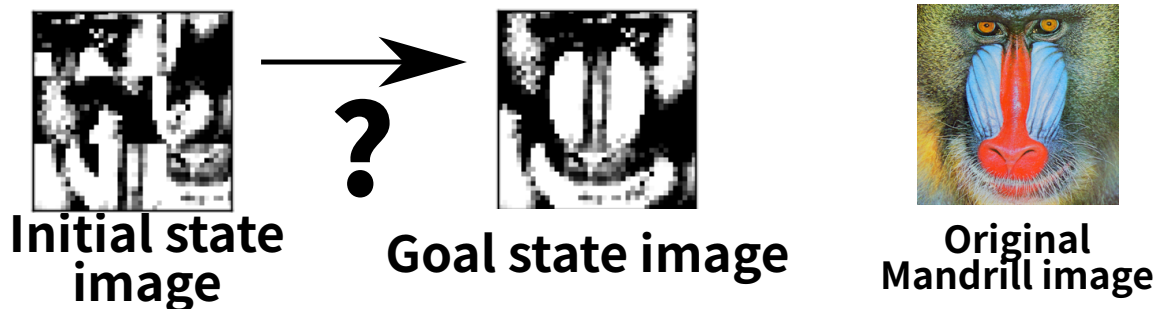


Figure 1.1: An image-based 8-puzzle.

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

Figure 1.1 (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 (Figure 1.2). 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 that automatically generates a symbolic problem representation from the subsymbolic input, that can be used as the input for a classical planner. The current implementation of Latplan contains four neural components in addition to the classical planner:

- A discrete autoencoder with multinomial binary latent variables, named *State Autoencoder* (SAE), which learns a bidirectional mapping between the raw observations of the environment and its propositional representation.
- A discrete autoencoder with a single categorical latent variable and a skip connection, named *Action Autoencoder* (AAE), which performs an unsupervised clustering over the latent transitions and generates action symbols.
- A specific decoder implementation of the AAE, named Back-To-Logit (BTL), which models the state progression / forward dynamics that directly compiles into STRIPS action effects.
- An identical BTL structure to model the state *regression* / time-inverse dynamics which directly compiles into STRIPS action preconditions.

Given only a set of *unlabeled images* of the environment, and in an unsupervised manner, we train Latplan to generate a symbolic representation. Then, given a planning problem instance as a pair of initial and goal images such as Figure 1.1, Latplan uses the SAE to map the problem to a symbolic planning instance, invokes a planner, then visualizes the plan execution by a sequence of images.

A system that generates symbols from the scratch has an advantage of being able to work on multiple domains more easily. In planning, symbolic manipulation enables the encoding of powerful domain-independent knowledge that can be easily applied to multiple tasks without training data. For example, given a problem instance from a previously unseen STRIPS planning domain D , a planning algorithm can often solve an instance of D much faster than blind search by using domain-independent heuristic functions that exploit D based purely on the symbolic structure of the action model of D (Hoffmann & Nebel, 2001; Helmert & Domshlak, 2009). The advantage of exploiting symbolic structures is a predicament of the Physical Symbol Systems Hypothesis (Newell & Simon, 1976; Newell, 1980; Nilsson, 2007), which states that “[a] physical symbol system has the necessary and sufficient means for general intelligent action.” In contrast, while current learning-based approaches to planning such as AlphaZero (Silver et al., 2017) or MuZero (Schrittwieser et al., 2019) achieve impressive performance, they require using or generating massive amounts of data in order to learn task-dependent evaluation functions and policies that result in high performance on a given task. Transferring domain-independent strategies across tasks remains a challenge for learning-based approaches, as they currently lack a convenient representation for expressing and exchanging task-independent knowledge between different systems. It is trivial in classical planning, where domain-independent heuristics are available.

The paper is organized as follows. We first begin with a review of preliminaries and background (Section 2). We next provide our high-level problem statement (Section 3). We next give an overview of the Latplan architecture (Section 4). In Section 5, we describe the SAE implemented as a Binary-Concrete Variational Auto-Encoder, which generates propositional symbols from images. We identify and define the *Symbol Stability Problem* which arises when grounding propositional symbols, and propose countermeasures to address it.

Next, in Sections 6-9, we explain our approach to action model learning. Since the action model learning is a complex problem, we introduce four increasingly sophisticated versions (AMA_1 - AMA_4^+), where each version inherits the entire model of its previous version as a component. We chose this presentation to help illustrate which aspect of the learning problem is addressed by each component. AMA_1 - AMA_4^+ can be summarized as follows: (Section 6) AMA_1 , a direct translation of image transitions to grounded actions, (Section 7) AMA_2 , which uses the AAE as a general, black-box successor function, (Section 8) AMA_3^+ , an approach which trains a *Cube-Space AE* network which jointly trains an SAE and a Back-to-Logit AAE for STRIPS domains and extracts a PDDL model compatible with off-the-shelf State-of-the-Art planners, and (Section 9) AMA_4^+ , an approach which trains a *Bidirectional Cube-Space AE* network which improves upon AMA_3^+ by using *complete state regression* semantics to learn accurate action preconditions.

We then evaluate Latplan using image-based versions of the 8-puzzle, 15-puzzle, LightsOut (two versions), Blocksworld, and Sokoban domains. Section 10 presents empirical evaluations of the accuracy and stability of the SAE, as well as the action model accuracy of AMA_3^+ and AMA_4^+ . Section 11 presents empirical evaluation of end-to-end planning with Latplan, including the effectiveness of standard planning heuristics. Section 12 surveys related work, and we conclude

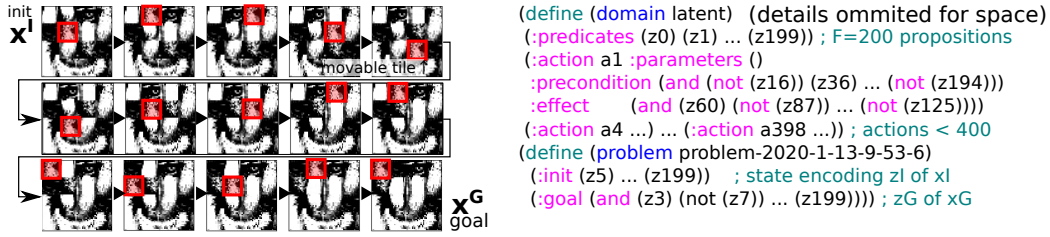


Figure 1.2: (left) A 14-step optimal plan for a 15-puzzle instance generated by our system using off-the-shelf Fast Downward with h^{LMcut} using the PDDL generated by our system. (right) The intermediate PDDL output from our NN-based system.

with a discussion of our contributions and directions for future work (Section 13). Some additional technical details, background, and data are presented in the Appendix.

Latplan is a first step in bridging the gap between symbolic and subsymbolic reasoning, therefore it currently has various limitations. For example, Latplan is evaluated in a fully-observable environment (although it is noisy). Also, Latplan’s state representation is entirely propositional and lack first-order logic generalization, thus requires a retraining when new objects are added. Latplan is limited to tasks where a single goal state is specified. Finally, Latplan requires uniform sampling from the environment, which is nontrivial in many scenarios. We discuss these limitations in detail in the discussion (Section 13).

This paper summarizes and extends the work that has appeared in (Asai & Fukunaga, 2018; Asai & Kajino, 2019; Asai & Muise, 2020). The major new technical contributions in this journal version are: (1) the improved precondition learning enabled by the regressive action modeling (Section 9), (2) theoretical justifications of the training objectives throughout the paper, and (3) thorough empirical evaluations (Sections 10-11).

2. Preliminaries and Important Background Concepts

2.1 Notations

We denote a multi-dimensional array in bold and its elements with a subscript (e.g., $\mathbf{x} \in \mathbb{R}^{N \times M}$, $\mathbf{x}_2 \in \mathbb{R}^M$). An integer range $n \leq i < m$ is represented by $n..m$. By analogy, we use dotted subscripts to denote a subarray, e.g. $\mathbf{x}_{2..5} = (\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$. $\mathbf{1}^D$ and $\mathbf{0}^D$ denote constant matrices of shape D with all elements being 1/0, respectively. $\mathbf{a}; \mathbf{b}$ denotes a concatenation of tensors \mathbf{a} and \mathbf{b} in the first axis where the rest of the dimensions are same between \mathbf{a} and \mathbf{b} . The i -th data point of a dataset is denoted by a superscript i which we may omit for clarity. These superscripts are also sometimes abbreviated by $..$ to improve the readability, e.g., $\mathbf{x}^{0..2} = (\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2)$. Functions (e.g., log, exp) are applied to arrays element-wise. Finally, we denote $\mathbb{B} = [0, 1]$.

2.2 Propositional Classical Planning

We define a grounded (propositional) STRIPS Planning problem with negative preconditions and unit costs as a 4-tuple $\langle P, A, I, G \rangle$ where P is a set of propositions, A is a set of actions, $I \subseteq P$ is the initial state, and $G \subseteq P$ is a goal condition. Each action $a \in A$ is a 4-tuple $a =$

$\langle \text{POS}(a), \text{NEG}(a), \text{ADD}(a), \text{DEL}(a) \rangle$ where $\text{POS}(a), \text{NEG}(a)$ are the positive and negative preconditions, $\text{ADD}(a), \text{DEL}(a)$ are the add-effects and delete-effects, $\text{POS}(a), \text{NEG}(a), \text{ADD}(a), \text{DEL}(a) \subseteq P$, $\text{POS}(a) \cap \text{NEG}(a) = \emptyset$, and $\text{ADD}(a) \cap \text{DEL}(a) = \emptyset$. A complete state (or just state) $s \subseteq P$ is a set of true propositions where those in $P \setminus s$ are presumed to be false. A partial state is similarly represented (i.e. a subset of P), but those propositions not mentioned may be either true or false. The initial state (I) is a complete state while the goal condition (G) and effect sets ($\text{POS}(a), \text{NEG}(a), \text{ADD}(a), \text{DEL}(a)$) are partial states. We say that a state s entails a partial state ps when $ps \subseteq s$ – intuitively, every proposition that must be true in ps is true in the state s . An action a is *applicable* when $s \supseteq \text{POS}(a)$ and $s \cap \text{NEG}(a) = \emptyset$, and applying an action a to s yields a new successor state $a(s) = (s \setminus \text{DEL}(a)) \cup \text{ADD}(a)$. The task of classical planning is to find a *plan* (a_1, \dots, a_n) which satisfies G by the repeated application of applicable actions, i.e., $G \subseteq a_n \circ \dots \circ a_1(I)$. A plan π is optimal when there are no other plans whose length is shorter than π .

2.3 Autoencoders and Variational Autoencoders

We review relevant probability theory in the appendix (Section A).

An Autoencoder (AE) is a type of neural network that learns an identity function whose output matches the input (Hinton & Salakhutdinov, 2006). Autoencoders consist of an input \mathbf{x} , a *latent vector* \mathbf{z} , an output $\hat{\mathbf{x}}$, an encoder f , and a decoder g , where $\mathbf{z} = f(\mathbf{x})$ and $\hat{\mathbf{x}} = g(\mathbf{z}) = g(f(\mathbf{x}))$. The output is also called a *reconstruction*. The dimension of \mathbf{z} is typically smaller than the input; thus \mathbf{z} is considered a compressed representation of \mathbf{x} .

The networks f, g are optimized by minimizing a *reconstruction loss* $\|\mathbf{x} - \hat{\mathbf{x}}\|$ under some norm, which is typically a Square Error / L2-norm. Which norm to use is semi-arbitrarily decided by a model designer — More explanation on the choice of the loss is given in the appendix (Section B). Assuming a 1-dimensional case, let x be a data point in the dataset, z be a certain latent value, and a probability distribution $p(x|z)$ be what the neural network (and the model designer) believes is the distribution of x given z . Typical AEs for images assume that x follows a Gaussian distribution centered around the predicted value $\hat{x} = g(z)$, i.e., $p(x|z) = \mathcal{N}(x|\hat{x}, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\hat{x})^2}{2\sigma^2}}$ for an arbitrary fixed constant σ . This leads to a *negative log likelihood* (NLL) ¹ $-\log p(x|z) = \frac{(x-\hat{x})^2}{2\sigma^2} + \log \sqrt{2\pi\sigma^2}$ which is a scaled/shifted square error / L2-norm reconstruction loss. In images, they are summed across dimensions (pixels).

A Variational Autoencoder (VAE) is an autoencoder that additionally assumes that the latent variable \mathbf{z} by default follows a certain distribution $p(\mathbf{z})$, typically called a *prior distribution*. A prior distribution is chosen arbitrarily by a model designer, such as a unit Normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$. More precisely, \mathbf{z} follows $p(\mathbf{z})$ unless the observation \mathbf{x} forces it otherwise — \mathbf{z} is meant to diverge from it based on the data through the training.

A VAE is trained by minimizing a sum of a reconstruction loss $-\log p(\mathbf{x}|\mathbf{z})$ and a KL divergence $D_{\text{KL}}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))$ between $q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$, where $q(\mathbf{z}|\mathbf{x})$ is a distribution of the latent vector \mathbf{z} obtained by the encoder at hand. $q(\mathbf{z}|\mathbf{x})$ must be in the *same family* of distributions as $p(\mathbf{z})$, e.g., if $p(\mathbf{z})$ is Gaussian, $q(\mathbf{z}|\mathbf{x})$ should also be a Gaussian, in order to obtain an analytical

1. Likelihood is a synonym of probability. “Log likelihood of \mathbf{x} ” means a logarithm of a probability of observing the data \mathbf{x} . Negative log likelihood is its negation.

form of the KL divergence that can be processed by automatic differentiation frameworks. When $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{1})$, a model engineer can then design an encoder that returns two vectors $\boldsymbol{\mu}, \boldsymbol{\sigma} = f(\mathbf{x})$ and sample \mathbf{z} as $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}) = q(\mathbf{z}|\mathbf{x})$, using $\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}$ is a noise that follows $\mathcal{N}(\mathbf{0}, \mathbf{1})$ (reparameterization trick). Then the analytical form of the KL divergence is obtained from the closed form of Gaussian distribution as follows:

$$D_{\text{KL}}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})||\mathcal{N}(\mathbf{0}, \mathbf{1})) = \sum_i \frac{1}{2}(\sigma_i + \mu_i^2 - 1 - \log \sigma_i^2). \quad (1)$$

A negative sum of the reconstruction loss and the KL divergence is called an *Evidence Lower Bound* (ELBO), a lower bound of the log likelihood $\log p(\mathbf{x})$ of observing \mathbf{x} (Kingma & Welling, 2013). Under *maximum likelihood estimation* framework, this lower bound characteristics make VAEs theoretically more appealing than AEs because maximizing $\log p(\mathbf{x})$ is the true goal of unsupervised learning: To maximize $\log p(\mathbf{x})$, we maximize its lower bound $\text{ELBO}(\mathbf{x})$, and thus we minimize the loss function $-\text{ELBO}(\mathbf{x})$. To derive a lower bound, ELBO uses the *variational method* which sees $q(\mathbf{z}|\mathbf{x})$ as an approximation of $p(\mathbf{z}|\mathbf{x})$.² The lower bound matches $\log p(\mathbf{x})$ when $q = p$. The proof is shown by using Jensen’s inequality about exchanging an expectation and a convex function (in this case, log) as follows:

$$\begin{aligned} \log p(\mathbf{x}) &= \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \log \sum_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) = \log \sum_{\mathbf{z}} p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} q(\mathbf{z}|\mathbf{x}) \\ &= \log \left(\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left\langle p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right\rangle \right) \quad (\text{definition of an expectation.}) \\ &\geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left\langle \log \left(p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right) \right\rangle \quad (\text{Jensen's inequality.}) \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \langle \log p(\mathbf{x} | \mathbf{z}) \rangle - \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left\langle \log \frac{q(\mathbf{z} | \mathbf{x})}{p(\mathbf{z})} \right\rangle \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \langle \log p(\mathbf{x} | \mathbf{z}) \rangle - D_{\text{KL}}(q(\mathbf{z} | \mathbf{x}) || p(\mathbf{z})) \quad (\text{definition of KL.}) \\ &= \text{ELBO}(\mathbf{x}). \end{aligned} \quad (2)$$

In contrast to VAEs, the loss function of an AE (= reconstruction loss $\log p(\mathbf{x}|\mathbf{z})$) does not have this lower bound property. Maximizing $\log p(\mathbf{x}|\mathbf{z})$ does not guarantee that it maximizes $\log p(\mathbf{x})$.

Finally, among popular extensions of VAEs, β -VAE (Higgins et al., 2017) uses a loss function that scales the KL divergence term with a hyperparameter $\beta \geq 1$.

$$\text{ELBO}_{\beta}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \langle \log p(\mathbf{x} | \mathbf{z}) \rangle - \beta D_{\text{KL}}(q(\mathbf{z} | \mathbf{x})||p(\mathbf{z})).$$

$\text{ELBO}(\mathbf{x}) \geq \text{ELBO}_{\beta}(\mathbf{x})$ because D_{KL} is always positive, thus $\text{ELBO}_{\beta}(\mathbf{x})$ is also a lower bound of $\log p(\mathbf{x})$. However, some literature uses $\beta < 1$ which is tuned manually by a visual inspection of the reconstructed image, which violates ELBO. In this paper, we always use $\beta \geq 1$ so that it does not violate ELBO. We will discuss the effect of different β during the empirical evaluation.

2. We say $q(\mathbf{z}|\mathbf{x})$ is a *variational distribution*. In addition, both $q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z}|\mathbf{x})$ are called *posterior distribution* because they are distributions of latent variables given observed variables. When combined, we say $q(\mathbf{z}|\mathbf{x})$ is a *variational posterior distribution*, which is a *variational approximation* of the true posterior distribution $p(\mathbf{z}|\mathbf{x})$.

2.4 Discrete Variational Autoencoders

While typically $p(z)$ is a Normal distribution $\mathcal{N}(0, 1)$ for continuous latent variables, there are multiple VAE methods for discrete latent distributions. A notable example is a method we use in this paper: Gumbel-Softmax (GS) VAE (Jang et al., 2017). Gumbel-Softmax VAE is independently discovered by Maddison, Mnih, and Teh (2017) as Concrete VAE; therefore it may be called as such in some literature. The binary special case of Gumbel Softmax VAE is called Binary-Concrete (BC) VAE (Maddison et al., 2017). These VAEs use a discrete, uniform categorical/Bernoulli(0.5) distribution as the prior $p(z)$, and further approximate it with a continuous relaxation by introducing a temperature parameter τ that is annealed down to 0. In this section, we briefly summarize the information necessary for implementing GS and BC.

For Gumbel Softmax and Binary Concrete, we denote corresponding stochastic activation functions in the latent space as $\text{GS}_\tau(\mathbf{l})$ and $\text{BC}_\tau(l)$, where $\mathbf{l} \in \mathbb{R}^C$, $l \in \mathbb{R}$, and C denotes the number of classes represented by Gumbel Softmax. \mathbf{l} and l can be an arbitrary vector/scalar produced by the encoder neural network f . Both functions use a temperature parameter τ which is annealed during the training in a fixed schedule such as an exponential schedule $\tau(t) = Ar^{-(t-t_0)}$, where t is a training epoch. Each output, $\mathbf{z} = \text{GS}_\tau(\mathbf{l})$ or $z = \text{BC}_\tau(l)$, follows Gumbel-Softmax distribution $\text{GS}(\mathbf{l}, \tau)$ and Binary-Concrete distribution $\text{BC}(l, \tau)$. In other words, the stochastic activation is equivalent to a sampling process from these distribution, and the output is a sample from these distributions. It can also be seen as a reparameterization trick for this distribution.

GS is defined as

$$\mathbf{z} = \text{GS}_\tau(\mathbf{l}) = \text{SOFTMAX}\left(\frac{\mathbf{l} + \text{GUMBEL}^C(0, 1)}{\tau}\right) \tag{3}$$

where $\text{GUMBEL}^C(0, 1) = -\log(-\log \mathbf{u})$ and $\mathbf{u} \sim \text{UNIFORM}^C(0, 1) \in \mathbb{B}^C$.

BC is a binary special case (i.e. $C = 2$) of GS. It is defined as

$$z = \text{BC}_\tau(l) = \text{SIGMOID}\left(\frac{l + \text{LOGISTIC}(0, 1)}{\tau}\right) \tag{4}$$

where $\text{LOGISTIC}(0, 1) = \log u - \log(1 - u)$ and $u \sim \text{UNIFORM}(0, 1) \in \mathbb{B}$.

Both functions converge to discrete functions at the limit $\tau \rightarrow 0$: $\text{GS}_\tau(\mathbf{l}) \rightarrow \arg \max(\mathbf{l})$ and $\text{BC}_\tau(l) \rightarrow \text{STEP}(l) = (l < 0) ? 0 : 1$. Note that we assume $\arg \max$ returns a one-hot representation rather than the index of the maximum value in order to maintain dimensional compatibility with SOFTMAX function.

In practice, GS-VAEs and BC-VAEs contain multiple latent vectors / latent scalars to model complex inputs, which we call n -way Gumbel-Softmax / Binary-Concrete. The latent space is denoted as $\mathbf{z} \in \mathbb{B}^{n \times C}$ for Gumbel-Softmax, and $\mathbf{z} \in \mathbb{B}^n$ for Binary Concrete.

For the derivation of the KL divergence of Gumbel-Softmax, refer to Section D in the Appendix. To see how Gumbel-Softmax and Binary-Concrete relate to other discrete methods, refer to Section C in the Appendix.

2.5 Notational Convention for Neural Networks

In this paper, we define each sub-network of the entire neural network so that it does not contain the last activation. This is because each network does not produce a sample of the probability distribution such as z ; the output of each network is instead a parameter of the probability distribution, such as l of a Gumbel-Softmax distribution $\mathbf{GS}(l, \tau)$ or μ, σ of a Gaussian distribution $\mathcal{N}(\mu, \sigma)$. The sampling process for $z \sim \mathbf{GS}(l, \tau)$ is placed outside each subnetwork, although it is still part of the entire network.

3. The Problem: Unsupervised Acquisition of a Classical Planning Model From Low-Level Inputs

Our goal is to build a system that acquires a classical planning model from low-level inputs (e.g., images) in a real-world (e.g., physical) environment without human involvement. In order to fully automatically acquire symbolic models for Classical Planning, we need *Symbol Grounding* (Harnad, 1990; Steels, 2008; Taddeo & Floridi, 2005) and *Action Model Acquisition* (Zhuo & Yang, 2014; Zhuo, 2015, AMA), which are one of the key challenges in achieving autonomous symbolic systems for real-world environments and in Artificial Intelligence.

Symbol Grounding is an unsupervised process of establishing a mapping between symbols and noisy, continuous and/or unstructured inputs. Action Model Acquisition (also known as Planning Operator Acquisition (Wang, 1995), or Action Model Learning (Yang et al., 2007)) is indeed a grounding process for actions, typically with a focus on descriptive action schema (preconditions and effects in the STRIPS subset of PDDL) as the referents. To clarify these high-level goals, we formalize symbol grounding and discuss its relation to PDDL construction. Following a traditional definition seen in the LISP family of programming languages, formally:

Definition 1. *A symbol is a tuple of a name and a referent, which can be uniquely looked up in a knowledge base using the name as a key. Symbol grounding is a process of assigning referents to symbols.*

For example, a *name* can be an ASCII string or an integer. A *referent* is an arbitrary representation of a meaning of the symbol, e.g., a referent of a *predicate symbol* is a boolean function, while a referent of an *action symbol* is an action schema. We do not distinguish whether referents are hand-coded or learned; thus a predicate can be a binary classifier learned by a machine learning system. A symbol *designates* a referent (Newell & Simon, 1976). A *knowledge base* is a namespace (e.g., a hash table) that defines a type of a symbol.

Symbols with the same name may exist in different namespaces and designate different referents. For example, many English words, such as “move”, “cut”, or “train”, can be seen as an action symbol (a verb), a predicate symbol (an adjective), or an object symbol (a noun) simultaneously. Our view is in line with historic literature, e.g., Russell, Norvig, Canny, Malik, and Edwards (1995) mentions propositional/constant(=object)/predicate/function symbols, and Vera and Simon (1993) wrote “The ‘wash’ ... is a symbol denoting the action required” (an action symbol).

PDDL has *at least* six kinds (i.e., knowledge bases) of symbols: Propositions, actions, objects, predicates, problems, and domains (Table 3.1). Each type of symbol requires its own mechanism for grounding. For example, the large body of work on recognizing objects (e.g., individual faces) and their attributes (male, female) in images, or scenes in videos (e.g., cooking), can be viewed as grounding object, predicate and action symbols, respectively. Extensions of PDDL may require

Types of symbols	
Object symbols	panel7, x₀, y₀ ...
Predicate symbols	(empty ?x ?y) (up ?y₀ ?y₁)
Propositions	empty₅ = (empty x₂ y₁)
Action symbols	(slide-up panel₇ x₀ y₁)
Problem symbols	eight-puzzle-instance1504, etc.
Domain symbols	eight-puzzle, hanoi

Table 3.1: Six types of symbols in a PDDL definition.

more types of symbols, such as numeric fluents in numeric planning (Fox & Long, 2003). Symbol grounding does not target purely syntactic constructs such as `and`, `or` or `:requirement` that do not need referents. Of six types, we aim to develop mechanisms for propositions and actions because our goal is to generate a propositional classical planning model in a single environment.

Finally, in order to fully avoid the knowledge acquisition bottleneck, a system must *generate* symbols without human input. The generation of symbols has long been recognized as an important aspect of symbol grounding (Harnad, 1990). In contrast to systems which can autonomously generate symbols, systems which does not generate symbols but only grounds symbols given by humans are characterized as *parasitic* by Taddeo and Floridi (2005), as such human-provided symbols already impose discrete categorization of the observations.

4. Latplan: Overview of the System Architecture

Latplan is a framework for *domain-independent classical planning in environments where symbolic input (PDDL models) are unavailable and only subsymbolic sensor input (e.g., images) are accessible*. Latplan operates in two phases: The training phase and the planning phase. Its abstract pipeline is described in Algorithm 1. The rest of the section describe the role of each abstract procedure, although the details may vary depending on the implementation.

Algorithm 1 An abstract pipeline of Latplan framework.

Training Phase:

Require: Dataset \mathcal{X} , untrained machine learning model M

- 1: Trained model $M' \leftarrow \text{TRAIN}(M, \mathcal{X})$
- 2: M' provides functions `ENCODE` and `DECODE`.
- 3: PDDL domain file $D \leftarrow \text{GENERATEDOMAIN}(M')$
- 4: **return** M', D

Planning Phase:

Require: M', D , initial state observation x^I , goal state observation x^G

- 1: Encode x^I, x^G into propositional states z^I, z^G
 - 2: PDDL problem file $P \leftarrow \text{GENERATEPROBLEM}(z^I, z^G)$
 - 3: Plan $\pi = (a_0, a_1, \dots) \leftarrow \text{SOLVE}(P, D)$ using a planner (e.g., Fast Downward)
 - 4: State trace $(z^I = z^0, z^1 = a_0(z^0), z^2 = a_1(z^1), \dots, z^G) \leftarrow \text{SIMULATE}(\pi, z^I, D)$ using a plan validator for PDDL, e.g., VAL (Howey & Long, 2003).
 - 5: **return** Decode an observation trace $(x^I = x^0, x^1, x^2, \dots, x^G)$.
-

4.1 Training Phase

In the training phase, it first trains a model M which learns state representations and transition rules of the environment entirely from subsymbolic state transition data \mathcal{X} (e.g., image-based observations) using deep neural networks (line 1). This is done by an abstract procedure TRAIN which obtains a trained model M' . M' provides a function ENCODE that maps subsymbolic observations into symbolic states, and a function DECODE that maps symbolic states back to subsymbolic visualization (line 2). Then we produce a PDDL domain file D using an abstract procedure GENERATEDOMAIN (line 3).

\mathcal{X} is a set of pairs of observations (e.g., raw image data) sampled from the environment. The i -th pair in the dataset $\mathbf{x}^i = (\mathbf{x}^{i,0}, \mathbf{x}^{i,1}) \in \mathcal{X}$ is a transition from an observation $\mathbf{x}^{i,0}$ to another observation $\mathbf{x}^{i,1}$ caused by an unknown high-level action. As discussed in Section 3, there are mainly three key challenges in training Latplan: **(1)** generating and grounding *propositional symbols*, **(2)** generating and grounding *action symbols*, and **(3)** acquiring a *descriptive action model*. The first challenge arises because we have no access to the state representation. The second challenge arises because we have no access to the action labels that caused the transitions. The third challenge arises because no symbolic representation of the action is available.

Challenge **(1)** is addressed by a *State AutoEncoder* (SAE) (Figure 5.1) neural network that learns a bidirectional mapping between subsymbolic raw data \mathbf{x} (e.g., images) and propositional states $\mathbf{z} \in \{0, 1\}^F$, i.e., F -dimensional bit vectors. This generates and grounds propositional symbols to feature extractors encoding complex patterns found in the image. The network consists of two functions ENCODE and DECODE, where ENCODE encodes an image \mathbf{x} to a bit vector \mathbf{z} , and DECODE decodes \mathbf{z} back to an image $\tilde{\mathbf{x}}$. These are trained so that $\tilde{\mathbf{x}} \approx \mathbf{x}$ holds.

Challenge **(2)** and **(3)** are more intricate, and we propose four variants of action model acquisition methods. The mapping ENCODE from $\{\dots \mathbf{x}^{i,0}, \mathbf{x}^{i,1} \dots\}$ to $\{\dots \mathbf{z}^{i,0}, \mathbf{z}^{i,1} \dots\}$ provides a set of propositional transitions $\mathbf{z}^i = (\mathbf{z}^{i,0}, \mathbf{z}^{i,1}) \in \mathcal{Z}$. Using \mathcal{Z} , action model acquisition methods have to learn not only action symbols but also their transition rules. The four variants, in increasing order of sophistication, are as follows.

- GENERATEDOMAIN of AMA₁ applies the SAE to pairs of image transitions to obtain propositional transition \mathbf{z}^i and directly converts each \mathbf{z}^i to a ground STRIPS action. For example, in a state space represented by 2 latent space propositions $\mathbf{z} = (z_1, z_2)$, a transition from $\mathbf{z}^{i,0} = (0, 1)$ to $\mathbf{z}^{i,1} = (1, 0)$ is translated into an action with $\text{POS}(a) = \{z_2\}$, $\text{NEG}(a) = \{z_1\}$, $\text{ADD}(a) = \{z_1\}$, $\text{DEL}(a) = \{z_2\}$. AMA₁ is developed to demonstrate the feasibility of using SAE-generated propositional symbols directly with existing planners. While it successfully demonstrates the feasibility, AMA₁ is impractical in that it does not generalize \mathcal{X} : AMA₁ only encodes the transitions (ground actions) in the training data \mathcal{X} . Thus, generating an AMA₁ model which can be used to find a path between an arbitrary start and goal state (assuming such a path exists in the domain) requires that \mathcal{X} contains the transitions necessary to form such a path – in the worst case, this may require the entire state space (i.e., all possible valid image transitions) as input. Furthermore, the size of the PDDL model is proportional to the number of transitions in the state space, slowing down the planning preprocessing and heuristic calculation at each search node. In other words, AMA₁ only addresses the challenge **(1)** (using SAE).
- AMA₂ is a neural architecture that jointly learns action symbols and action models from a small subset of transitions in an unsupervised manner. It learns a black-box successor gen-

eration function, which, given a latent space symbolic state, returns its successors. Unlike existing methods, AMA_2 does not require high-level action symbols as part of the input. While AMA_2 is a general approach to learning a successor function which does not make strong assumptions about the domain (e.g., STRIPS assumptions), this limits its applicability to classical planning because it does not produce PDDL-compatible action descriptions and it cannot implement `GENERATEDOMAIN`; it is incompatible with PDDL-based solvers and cannot leverage existing implementations of domain-independent heuristics. It requires a search algorithm implementation which does not require descriptive action models, such as novelty-based planners (Frances et al., 2017), or a best-first search algorithm with trivial heuristics which work with AMA_2 's black-box limitation, such as Goal Count heuristics (Fikes & Nilsson, 1972). In other words, AMA_2 addresses the challenge (1) and (2).

- AMA_3^+ is a neural architecture that improves AMA_2 so that the resulting neural network can be directly compiled into a descriptive action model (PDDL). AMA_3^+ builds on the previously published AMA_3 to provide an architecture and an optimization objective that are theoretically motivated by the Evidence Lower Bound (ELBO). AMA_3^+ has a neural network component that exclusively models the STRIPS/PDDL-compatible action effects and can implement `GENERATEDOMAIN`, but it requires a separate, ad-hoc process for generating preconditions. In other words, it addresses the challenge (3), but only partially.
- AMA_4^+ , which we propose in this paper, is an enhancement of AMA_3^+ with a new ability to learn and emit STRIPS/PDDL-compatible preconditions. Unlike AMA_3^+ , it does not require a separate, ad-hoc process which results in less accurate preconditions. AMA_4^+ models the state space with *complete state regression* semantics, which is inspired by SAS+ formalism (Bäckström & Nebel, 1995). This finally addresses all of challenges (1), (2), and (3).

We reemphasize that each new model inherits the entire components of its predecessor: SAE in AMA_1 is included in AMA_2 and later, AAE in AMA_2 is included in AMA_3^+ and later as a subnetwork, and finally, AMA_4^+ contains AMA_3^+ as a subnetwork. Therefore, understanding each earlier variant is important for understanding the full potential and mechanics of our most sophisticated model AMA_4^+ .

4.2 Planning Phase

In the planning phase, Latplan takes as input a *planning input* $(\mathbf{x}^I, \mathbf{x}^G)$, a pair of subsymbolic observations (raw data, images) corresponding to an initial and goal state of the environment. We first apply `ENCODE` to each observation and obtain their symbolic representations (line 1). An abstract procedure `GENERATEPROBLEM` then generates a PDDL problem file containing these symbolic initial and goal states (line 2). An abstract procedure `SOLVE` then solves the problem in the learned symbolic state space and returns a symbolic plan, using an off-the-shelf planner such as Fast Downward (Helmert, 2006) (line 3). It then obtains a propositional intermediate state trace (z^I, \dots, z^G) of the plan by applying a plan simulator (`SIMULATE`) to the symbolic initial state and the symbolic plan (line 4).

While we have a symbolic state trace at this point, they are not interpretable for a human observer, which is an issue. This is primarily because the states in the state trace are generated by SAE neural network which was trained unsupervised — each bit of the state has a “meaning” in the latent space determined by the neural network, which does not necessarily directly correspond to

human-interpretable notions. Furthermore, the “actions” in the plan trace are transitions according to the AMA, but are not necessarily directly interpretable by a human. In order to make the plan generated by Latplan understandable to a human (or other agents outside Latplan), Latplan outputs a step-by-step visualization of the plan execution, $(\mathbf{x}^I, \dots, \mathbf{x}^G)$ (e.g. Figure 1.2), by decoding the latent bit vectors for each intermediate state in $(\mathbf{z}^I, \dots, \mathbf{z}^G)$ (line 5).

While we present an image-based implementation (“data” = raw images), the architecture itself does not make such assumptions and could be applied to other types of data such as audio/text.

5. State AutoEncoder (SAE) for Propositional Symbol Grounding

This section presents a method to a propositional representation of a real-world environment on which a planning solver can perform logical reasoning. We believe such a representation must satisfy the following three properties to be practical:

- The ability to describe unseen world states using the same symbols,
- Similar images for “the same world state” should map to the same representation,
- The ability to map symbolic states back to images.

For example, one may come up with a trivial method to simply discretize the pixel values of an image array or to compute an image hash function. Such a trivial representation lacks robustness and the ability to generalize.

In the following, Section 5.1 presents a State AutoEncoder (SAE) as a framework to address propositional symbol grounding by a discrete VAE. As elaborated in Section 5.2, we find that its naive implementation is unfavorable to a planning system because of unstable symbols caused by two sources of uncertainty. Sections 5.3-5.4 present two methods to address them.

5.1 Naive State AutoEncoder

Our first technical contribution is a State AutoEncoder (SAE) implemented by a discrete VAE (Figure 5.1), which allows us to obtain a propositional representation satisfying the three requirements described above. *Our key observation is that the categorical variables modeled by discrete VAEs are compatible with symbolic reasoning systems.* For example, binary latent variables of Binary-Concrete VAE can be used as propositional symbols in STRIPS language, providing a solution to the propositional symbol grounding, i.e., generation and grounding of propositional symbols.

The trained SAE provides an approximation of a bidirectional mapping f, f^{-1} between the raw inputs (subsymbolic representation) and their symbolic representations:

- $z = f(\mathbf{x})$ maps an image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ to a boolean vector $z \in \mathbb{B}^F$.
- $\tilde{\mathbf{x}} \approx f^{-1}(z)$ maps a boolean vector z to an image $\tilde{\mathbf{x}}$.

Here, H, W, C represents the height, the width, and the number of color channels of the image. $f(\mathbf{x})$ maps a raw input \mathbf{x} to a symbolic representation by feeding the raw input to the encoder network which ends with a Binary Concrete activation, resulting in a binary vector of length F . $f^{-1}(z)$ maps a binary vector z back to an image. These are lossy compression/decompression functions, so in general, $\tilde{\mathbf{x}}$ may have an acceptable amount of errors from \mathbf{x} for visualization.

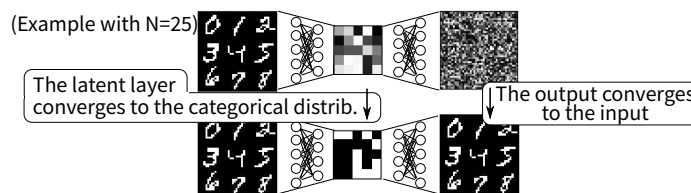


Figure 5.1: We train the State Autoencoder by minimizing the sum of the reconstruction loss and the variational loss of Binary-Concrete. As the training continues, the output of the network converges to the input images. Also, as the temperature τ decreases during training, the latent values approach either 0 or 1.

In order to map an image to a latent state with sufficient accuracy, the latent layer requires a certain capacity. The lower bound of the number of propositional variables F is the encoding length of the states, i.e., $F \geq \log_2 |S|$ for a state space S . In practice, however, obtaining the most compact representation is not only difficult but also unnecessary, and we use a hyperparameter F which could be significantly larger than $\log_2 |S|$.

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, simply using the rounding operation as a layer of the network is infeasible because rounding is non-differentiable, precluding backpropagation-based training of the network. Furthermore, as we discuss in Appendix Section C, AEs with Straight-Through step function is known to be outperformed by VAEs in terms of accuracy, and its optimization objective lacks theoretical justification as a lower bound of the likelihood.

The SAE implementation can easily and significantly benefit from progress made by the image processing community. We augmented the VAE implementation with a GaussianNoise layer to add noise robustness (Vincent et al., 2008), as well as Batch Normalization (Ioffe & Szegedy, 2015) and Dropout (Hinton et al., 2012), which helps faster training under high learning rates (see experimental details in Section 10.2).

5.2 The Symbol Stability Problem: Issues Caused by Unstable Propositional Symbols

Propositional representations learned by Binary Concrete VAEs in the original form proposed and evaluated by (Jang et al., 2017; Maddison et al., 2017) have a problematic behavior that makes them less suitable for propositional reasoning. While the SAE can reconstruct the input with high accuracy, the learned latent representations are not “stable,” i.e., some propositions may flip the value (true/false) randomly given identical or nearly identical image inputs (Figure 5.2).

The core issue with the vanilla Binary Concrete VAEs is that the class probability for the class “true” and the class “false” could be neutral (e.g., around 0.5) at some neuron, causing the value of the neuron to change frequently due to the stochasticity of the system. The source of stochasticity is twofold:

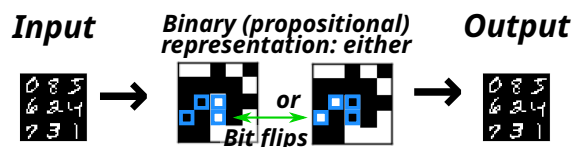


Figure 5.2: Propositions found by SAEs may contain uninformative random bits that do not affect the output.

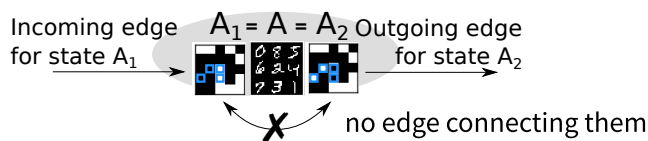


Figure 5.3: Random variations of the propositional encoding could disconnect the search space.

- *Systemic/epistemic uncertainty*: The first source is the random sampling in the network, which introduces stochasticity and causes the propositions to change values even for the exact same inputs.
- *Statistical/aleatoric uncertainty*: The second source is the stochastic observation of the environment, which corrupts the input image. When the class probabilities are almost neutral, a tiny variation in the input image may cause the activation to cross the decision boundary for each neuron, causing bit flips. In contrast, humans still regard the corrupted image as the “same” image.

These unstable symbols are harmful to symbolic reasoning because they break the identity assumption built into the recipient symbolic reasoning algorithms such as classical planners. It causes several issues: Firstly, the performance of search algorithms (e.g., A^*) that run on the state space generated by the propositional vectors are degraded by multiple propositional states which correspond to the same real-world state. As standard symbolic duplicate detection techniques would treat such spurious representations of the same state as different states, the search algorithm can unnecessarily re-expand the “same” real-world state several times, slowing down the search.

Secondly, the state space could be disconnected due to such random variations (Figure 5.3). Some states may be reachable only via a single variation of the real-world state and are not connected to another propositional variation of the same real-world state. One way to circumvent this phenomenon is *state augmentation*, which samples the propositional states at every node by decoding and encoding the current state several times. This is inefficient, and is also infeasible in the standard PDDL-based classical planners that operate completely in the propositional space.

Thirdly, in order to reduce the stochasticity of the propositions, we encounter a hyperparameter tuning problem which is costly when we train a large NN. The neurons that behave randomly for the same or perturbed input do not affect the output, i.e., they are unused and uninformative. Unused neurons appear because the network has an excessive capacity to model the entire state space, i.e., they are surplus neurons. Therefore, a straightforward solution to address this issue is to reduce the size of the latent space F . On the other hand, if F is too small, it lacks the capacity to represent the

state space, and the SAE no longer learns to reconstruct the real-world image. Thus we need to find an *ideal* value of F , a costly and difficult hyperparameter tuning problem.

These unstable propositional symbols indicate that the Binary Concrete VAE learns the mapping between images and binary representations as a many-to-many relationship. While this property has not been considered as a major issue in the machine learning community where accuracy is the primary consideration, its unstable latent representation poses a significant impediment to the reasoning ability of the planners. Unlike machine learning tasks, symbolic planning requires a mapping that abstracts many images into a single symbolic state, i.e., many-to-one mapping. To this end, it is necessary to develop an autoencoder that learns a many-to-one relationship between images and binary representations.

Fundamentally, the first two harmful effects are caused by the fact that the representation learned by a standard Binary Concrete VAE lacks a critical feature of symbols, *designation* (Newell & Simon, 1976), that each symbol uniquely designates a referent, e.g., the referents of the symbols grounded by SAEs are the truth assignments based on visual features. If the grounding (meaning) of a symbol changes frequently and unexpectedly, the entire symbolic manipulation is fruitless because the underlying symbols are not tied to any particular concept and do not represent the real-world.

Thus, for a symbol grounding procedure to produce a set of symbols for symbolic reasoning, it is not sufficient to find a set of symbols that can be mapped to real states in the environment; It should find a *stable* symbolic representation that *uniquely* represents the environment.

Definition 2. *A symbol is stable when its referents are identical for the same observation, under some equivalence relation (e.g., invariance or noise threshold on the observation).*

Example 1. *A propositional symbol points to a boolean value. The value should not change under the same real-world state and its noisy observations.*

Example 2. *A symbolic state ID points to a set of propositions whose values are true. The content of the set should be the same under the same real-world state and its observations.*

Example 3. *A symbolic action label points to a tuple containing an action definition (e.g., preconditions). It should point to the same action definition each time it observes the same state transition.*

Example 4. *Say that a real-world state s_1 transitions to another state s_2 by performing a certain symbolic action a . The representation obtained by applying the symbolic action definition of a to the symbolic representation of s_1 should be equivalent to the symbolic representation directly observed from s_2 .*

The stability of the representation obtained by a NN depends on its inherent (systemic, epistemic) stochasticity of the system during the runtime (as opposed to the training time) as well as the stochasticity of the environment (statistical, aleatoric). Thus, *any* symbol grounding system potentially suffers from the symbol stability problem. As for the stochasticity of the environment, in many real-world tasks, it is common to obtain stochastic observations due to external interference, e.g., vibrations of the camera caused by the wind. As for the stochasticity of the network, both VAEs (Kingma & Welling, 2013; Jang et al., 2017; Higgins et al., 2017) used in Latplan and GANs (Generative Adversarial Networks) (Goodfellow et al., 2014) used in Causal InfoGAN (Kurutach et al., 2018) rely on sampling processes.

The stability of the learned representation is orthogonal to the robustness of the autoencoder because unstable latent variables do not always cause bad reconstruction accuracy. The reason that

random latent variables do not harm the reconstruction accuracy is that the decoder pipeline of the network learns to ignore the random neurons by assigning a negligible weight. This indicates that *unstable* propositions are also “unused” and “uninformative.”

While this issue is similar to *posterior collapse* (Dinh & Dumoulin, 2014; Bowman et al., 2016), which is also caused by ignored latent variables, an important difference is that the latter degrades the reconstruction (makes it blurry) because there are too many ignored variables. In contrast, the symbol stability issue is significant even if the reconstruction is successful.

5.3 Addressing the Systemic Uncertainty: Removing the Run-Time Stochasticity

The first improvement we made from the original GS-VAE/BC-VAE is that we can disable the stochasticity of the network in the test time. After the training, we replace the Gumbel-softmax activation with a pure argmax of class probabilities, which makes the network fully deterministic.

$$\text{GS}_\tau(\mathbf{l}) \rightarrow \arg \max(\mathbf{l}). \quad (\mathbf{l} \in \mathbb{R}^{F \times C}) \quad (5)$$

Again note that we assume $\arg \max$ returns a one-hot representation rather than the index of the maximum value. In this form, there is no systemic uncertainty due to the Gumbel noise. In Binary Concrete, this is equivalent to using Heaviside step function:

$$\text{BC}_\tau(\mathbf{l}) \rightarrow \text{STEP}(\mathbf{l}). \quad (\mathbf{l} \in \mathbb{R}^F) \quad (6)$$

Similarly, there is no systemic uncertainty due to the Logistic noise in this form.

5.4 Addressing the Statistical Uncertainty: Selecting the Appropriate Prior

The prior distribution of the original Gumbel-Softmax VAE is a uniform random categorical distribution $\text{Cat}(\mathbf{1}/C)$, and the prior distribution of the original Binary-Concrete VAE is a uniform random Bernoulli distribution $\text{Bernoulli}(0.5)$. By optimizing the ELBO with stochastic gradient descent algorithms, the KL divergence $D_{\text{KL}}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))$ in the ELBO moves the encoder distribution $q(\mathbf{z}|\mathbf{x})$ closer to the target distribution (prior distribution) $p(\mathbf{z}) = \text{Bernoulli}(0.5)$. Observe that this objective encourages the latent representation to be *more random*, as $\text{Bernoulli}(0.5)$ is the most random distribution among $\text{Bernoulli}(p)$ for all p .

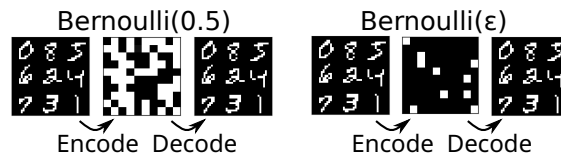


Figure 5.4: Autoencoding results of an MNIST 8-puzzle state using a vanilla BC-VAE and a proposed, non-uniform Bernoulli prior with 100 propositions. The latter obtains a sparse/compact representation with fewer true bits.

To tackle the instability of the propositional representation, *we propose a different prior distribution $p(\mathbf{z})$ for the latent random variables: $p(\mathbf{z}) = \text{Bernoulli}(0)$* (Figure 5.4). Its key insight is to penalize the latent propositions for unnecessarily becoming true while preserving the propositions that are absolutely necessary for maintaining the reconstruction accuracy. In practice, however,

since computing the KL divergence with $p(z) = \text{Bernoulli}(0)$ causes a division-by-zero error in the logarithm, we instead use $p(z) = \text{Bernoulli}(\epsilon)$ for a small ϵ (e.g., 0.1, 0.01).

We formalize the notions above as follows. Consider a case where z is a 1-dimensional scalar z . The KL divergence $D_{\text{KL}}(q(z|\mathbf{x}) \parallel p(z))$ with $p(z) = \text{Bernoulli}(0.5)$ and $q = q(z = 1|\mathbf{x}) = \text{SIGMOID}(\text{ENCODE}(\mathbf{x}))$ is as follows (Section 2.4):

$$\sum_{k \in \{0,1\}} q(z = k|\mathbf{x}) \log \frac{q(z = k|\mathbf{x})}{p(z = k)} = q \log \frac{q}{0.5} + (1 - q) \log \frac{(1 - q)}{(1 - 0.5)} = -H(q) + \log 2.$$

$H(q)$ is an *entropy* of the probability q , which models the randomness of q . Therefore, minimizing the KL divergence is equivalent to maximizing the entropy, which encourages the latent vector $z \sim q(z|\mathbf{x})$ to be more random. The KL divergence with $p(z) = \text{Bernoulli}(\epsilon)$ results in a form

$$q \log \frac{q}{\epsilon} + (1 - q) \log \frac{(1 - q)}{(1 - \epsilon)} = -H(q) + \alpha \cdot q - \log(1 - \epsilon) \tag{7}$$

for $\alpha = \log(1 - \epsilon) - \log \epsilon$. When $\alpha > 0$ ($\epsilon < \frac{1}{2}$), this form shows that minimizing the KL divergence has an additional regularization pressure to move q toward 0.

Intuitively, this optimization objective assumes a variable to be false when there is not enough “evidence” from the input image to flip a variable to true. The evidence in the data detected as a combination of complex patterns by the neural network declares some variables to be true if the evidence is strong enough. This resembles *closed-world assumption* (Reiter, 1981) where propositional variables (generated by grounding a set of first order logic formulae with terms) are assumed to be false unless it is explicitly declared or proven to be true. The difference is whether the declaration is implicit (performed by the encoder using the input data) or explicit (encoded manually by human, or proved by a proof system).

In contrast, the optimization objective made by the original prior distribution $p(z) = \text{Bernoulli}(0.5)$ makes a latent proposition highly random when there is neither the evidence for it to be true nor the evidence for it to be false. Since a uniform binary distribution $\text{Bernoulli}(0.5)$ carries no information about either the truthfulness or the falsifiability of the proposition, thus a variable having this distribution can be seen as in an “unknown” state. Again, this resembles an open-world assumption where a propositional variable that is neither declared true or false are put in an unknown state.

We discuss the difference from an ad-hoc method presented in the conference version (Asai & Kajino, 2019) in appendix Section E.

6. AMA₁: An SAE-based Translation from Image Transitions to Ground Actions

An SAE by itself is sufficient for a minimal approach to generating a PDDL definition for a grounded STRIPS planning problem from image data. Given a pair of images $\mathbf{x}^i = (\mathbf{x}^{i,0}, \mathbf{x}^{i,1})$ which represents a valid transition in the domain, we can apply the SAE to \mathbf{x}^0 and \mathbf{x}^1 to obtain $\mathbf{z}^{i,0}$ and $\mathbf{z}^{i,1}$, the latent-space propositional representations of these states.

The transition $(\mathbf{z}^{i,0}, \mathbf{z}^{i,1}) \in \mathcal{Z}$ can be directly translated to an action \mathbf{a}^i as follows: each bit $\mathbf{z}_f^{i,j}$ ($f \in 1..F, j \in 0..1$) in a boolean vector $\mathbf{z}^{i,j}$ is mapped to a proposition (z_f) when the value is 1, or

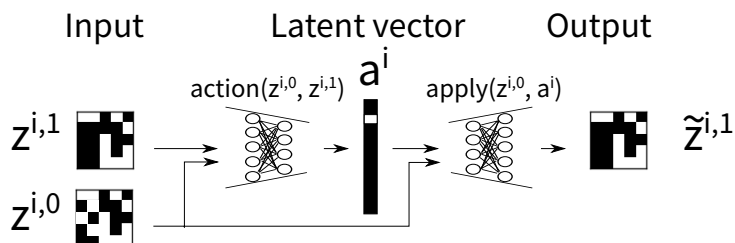


Figure 7.1: Action Autoencoder.

to its negation (`not (zf)`) when the value is 0. The elements of a bitvector $z^{i,0}$ are directly used as the preconditions of action a^i using negative precondition extension of STRIPS. The add/delete effects of the action are obtained from the bitwise differences between $z^{i,0}$ and $z^{i,1}$. For example, when $(z^{i,0}, z^{i,1}) = (0011, 0101)$, the action definition would look like

```
(:action action-0011-0101
:preconditions (and (not (z0)) (not (z1)) (z2) (z3))
:effects      (and (z1) (not (z2))))
```

Given a set of image pairs \mathcal{X} , AMA_1 directly maps each image transition to a STRIPS action as described above. The initial and the goal states are similarly created by applying the SAE to the initial and goal images and encoding them into a PDDL file, which can be input to an off-the-shelf planner. *If \mathcal{X} contains images for a sufficient portion of the state space containing a path from the initial and goal states*, then a planner can find a satisficing solution. If \mathcal{X} includes image pairs for *all* valid transitions, then an optimal solution can be found using an admissible search algorithm. However, collecting such a sufficient of transitions \mathcal{X} is impractical in most domains. Thus, AMA_1 is of limited practical utility, as it only provides the planner with a grounded model of state transitions which have already been observed – this lacks the key ability to generalize from grounded observations and predict/project *previously unseen* transitions, i.e., AMA_1 lacks the ability to generate more general action schemas.

7. AMA_2 : Action Symbol Grounding with an Action Auto-Encoder (AAE)

Next, we propose an unsupervised approach to acquiring an action model from a limited set of examples (image transition pairs). AMA_2 contains a neural network named *Action Autoencoder* (AAE, Figure 7.1). The AAE jointly learns action symbols and the corresponding effects and provides an ability to enumerate the successors of a given state, i.e., it can be used as a black-box successor function for a forward state space search algorithm. The key idea is that the successor predictor can be trained along with the action predictor by a variant of Gumbel-Softmax VAE.

- The encoder $\text{ACTION}(z^{i,0}, z^{i,1})$ samples an action label a^i for a state transition $(z^{i,0}, z^{i,1})$.
- The decoder $\text{APPLY}(a^i, z^{i,0})$ applies a^i to $z^{i,0}$ and samples a successor $\tilde{z}^{i,1}$. Thus the decoder network can be seen as modeling the *effect* of each action, which represents what change should be made to $z^{i,0}$ due to the application of the action.

- It takes a concatenated vector $z^{i,0}; z^{i,1}$ of a state transition pair $(z^{i,0}, z^{i,1})$ and returns a reconstruction $\tilde{z}^{i,1}$. The output layer is activated by a sigmoid function. The training is performed by minimizing the loss $\mathcal{L}(z^{i,1}, \tilde{z}^{i,1}) + D_{\text{KL}}$. The reconstruction loss \mathcal{L} is a Binary Cross Entropy loss because predicting a binary successor state is equivalent to multinomial classification. The KL divergence is a standard one for Gumbel-Softmax.
- Unlike typical Gumbel-Softmax VAEs, which contain multiple (n -way) one-hot vectors in the latent space, AAE has a single (1-way) one-hot vector of A classes, representing an **action label** $a^i \in \mathbb{B}^A$. A is a hyperparameter for the maximum number of action labels to be learned.

The number of labels A serves as an upper bound on the number of action symbols learned by the network. We tend to use a large value for A because (1) too few labels make AAE reconstruction loss fail to converge to zero, and (2) even if a large A is given, AAE tends to use the available set of labels frugally and leave the excess labels unused. We remove those unused labels after the training by iterating through the dataset and checking which labels were used.

Our earlier conference paper showed that AMA_2 could be used to successfully learn an action model for several image-based planning domains, and that a forward search algorithm using AMA_2 as the successor function could be used to solve planning instances in those domains (Asai & Fukunaga, 2018).

AMA_2 provides a non-descriptive, black-box neural model as the successor generator, instead of a descriptive, symbolic model (e.g., PDDL model). A black-box action model can be useful for exploration-based search algorithms such as Iterated Width (Frances et al., 2017). On the other hand, the non-descriptive, black-box nature of AMA_2 has a drawback: The lack of a descriptive model prevents the use of existing, goal-directed heuristic search techniques, which are known to further improve the performance when combined with exploration-based search (Lipovetzky, 2017).

To overcome this problem, we could try to translate/compile such a black-box model into a descriptive model usable by a standard symbolic problem solver. However, this is not trivial. Converting an AAE learned for a standard STRIPS domain to PDDL can result in a PDDL file which is several orders of magnitude larger than typical PDDL benchmarks. This is because the AAE has the ability to learn an arbitrarily complex state transition model. The traditional STRIPS progression $\text{APPLY}(s, a) = (s \setminus \text{DEL}(a)) \cup \text{ADD}(a)$ disentangles the effects from the current state s , i.e., the effect $\text{ADD}(a)$ and $\text{DEL}(a)$ are defined entirely based on action a . In contrast, the AAE’s black-box progression $\text{APPLY}(a^i, z^{i,0})$ does not offer such a separation, allowing the model to learn arbitrarily complex conditional effects. For example, we found that a straightforward logical translation of the AAE with a rule-based learner (e.g., Random Forest (Ho, 1998)) results in a PDDL that cannot be processed by modern classical planners due to the huge file size and exponential compilation of disjunctive formula (Asai, 2020).

8. $\text{AMA}_3/\text{AMA}_3^+$: Descriptive Action Model Acquisition with Cube-Space AutoEncoder

We now propose the *Cube-Space AutoEncoder* architecture (CSAE), which can generate a standard PDDL model file from a limited set of observed image transitions for domains which conform to STRIPS semantics. There are two major technical advances from AMA_2 . First, CSAE jointly learns a state representation and an action model, whereas AMA_2 learns a state representation first and learns the action model on the fixed state representation. As often pointed out in the ma-

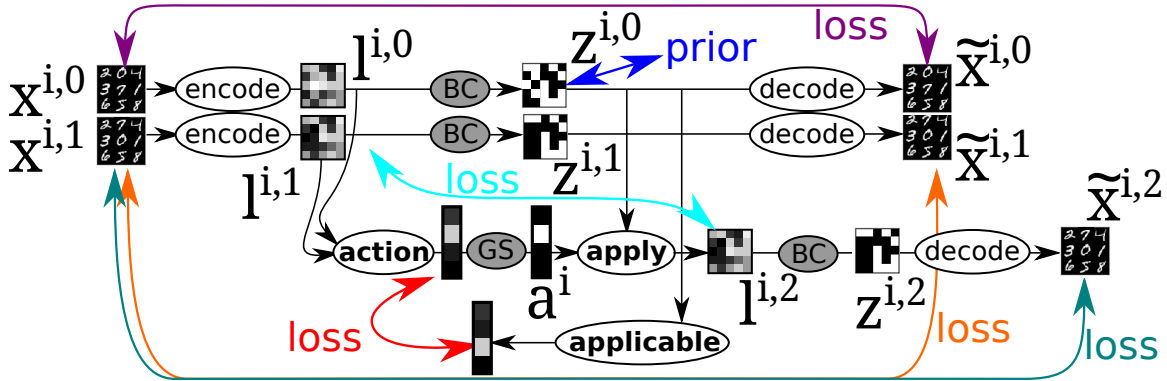


Figure 8.1: AMA_3^+ , a vanilla Space AutoEncoder architecture that we propose in this paper. The inputs for ACTION can be any deterministic value computed from x^0, x^1 (including x^0, x^1 itself), and we use l^0, l^1 because some features are supposed to be already extracted in them. Each color corresponds to each formula in Equation 9.

chine learning literature, the joint learning is expected to improve the performance. Second, CSAE constrains the action model to the STRIPS semantics. Thus we can trivially extract STRIPS/PDDL-compatible action effects from the trained CSAE, providing a grounded PDDL that is immediately usable by off-the-shelf planners. We implement the restriction as a structural prior that replaces the MLP decoder APPLY in the AAE. Due to the joint training, the state space shapes itself to satisfy the constraints of the action model.

We will refer to the previously published version of Cube-Space AE (Asai & Muise, 2020) as AMA_3 , and the updated version presented here as AMA_3^+ . The architecture and the objective function of AMA_3 were neither theoretically analyzed in (Asai & Muise, 2020), nor were designed with theoretical justification in mind. AMA_3^+ revises AMA_3 by providing a theoretically sound optimization objective and architecture.

This section proceeds as follows: In Section 8.1, we introduce Vanilla Space AutoEncoder, which performs a joint training of the state and action models in AMA_2 , but without the novel structure in CSAE that restricts the action model. We define its training objective which is theoretically justified as a lower bound of the log likelihood of observing the data. In Section 8.3, we formally introduce CSAE by inserting a structural prior to Vanilla Space AE. Section 8.3 shows that the CSAE generates a STRIPS model. Section 8.3 briefly discuss how to extract the action effects and preconditions from CSAE. In the following, we always omit the dataset index i .

8.1 Vanilla Space AutoEncoder

Vanilla Space AE (Figure 8.1) is a network representing a function of two inputs x^0, x^1 and three outputs $\tilde{x}^0, \tilde{x}^1, \tilde{x}^2$. It consists of several subnetworks: ENCODE, DECODE, ACTION, APPLY, APPLICABLE. APPLICABLE is a new subnetwork that was missing AMA_3 and is necessary for using a theoretically justifiable optimization objective (ELBO). The data pipeline from the input to the output is defined in Figure 8. APPLICABLE is not used in this main pipeline because it is used only for regularizing the network during the training, and is not used in the test time where we generate a PDDL output.

	(input)	$\mathbf{x}^0, \mathbf{x}^1$	
	(encoded logits)	$\mathbf{l}^k = \text{ENCODE}(\mathbf{x}^k).$	$k \in \{0, 1\}$
	(sampled binary representations)	$\mathbf{z}^k = \text{BC}_\tau(\mathbf{l}^k).$	$k \in \{0, 1\}$
	(logit for actions)	$\mathbf{l}^a = \text{ACTION}(\mathbf{l}^0, \mathbf{l}^1).$	
	(sampled discrete action label)	$\mathbf{a} = \text{GS}_\tau(\mathbf{l}^a).$	
	(logit for the result of progression)	$\mathbf{l}^2 = \text{APPLY}(\mathbf{z}^0, \mathbf{a})$	
	(sampled binary result of progression)	$\mathbf{z}^2 = \text{BC}_\tau(\mathbf{l}^2).$	
	(reconstructions)	$\tilde{\mathbf{x}}^k = \text{DECODE}(\mathbf{z}^k).$	$k \in \{0, 1, 2\}$ (8)

$$\begin{aligned}
 \log p(\mathbf{x}^0, \mathbf{x}^1) &\geq \text{ELBO}(\mathbf{x}^0, \mathbf{x}^1) \\
 &= \log p(\mathbf{x}^0 | \mathbf{z}^0) + \frac{1}{2} \log p(\mathbf{x}^1 | \mathbf{z}^1) + \frac{1}{2} \log p(\mathbf{x}^1 | \mathbf{z}^2) && \text{(Reconstruction losses)} \\
 &\quad - \beta_1 D_{\text{KL}}(q(\mathbf{z}^0 | \mathbf{x}^0) \| p(\mathbf{z}^0)) && \text{(Stabilize Example 1.)} \\
 &\quad - \beta_2 D_{\text{KL}}(q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) \| p(\mathbf{a} | \mathbf{z}^0)) && \text{(Action must be applicable.)} \\
 &\quad - \frac{1}{2} \beta_3 D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^2 | \mathbf{z}^0, \mathbf{a})). && \text{(Stabilize Example 4.)}
 \end{aligned} \tag{9}$$

$$\begin{aligned}
 p(\mathbf{z}^0) &= \text{Bernoulli}(\epsilon). && \text{Section 5.4.} \\
 p(\mathbf{a} | \mathbf{z}^0) &= \text{Cat}(\text{APPLICABLE}(\mathbf{z}^0)). \\
 p(\mathbf{x}^k | \mathbf{z}^k) &= \mathcal{N}(\mathbf{x}^k | \tilde{\mathbf{x}}^k, \sigma). && k \in \{0, 1\} \\
 p(\mathbf{x}^1 | \mathbf{z}^2) &= \mathcal{N}(\mathbf{x}^1 | \tilde{\mathbf{x}}^2, \sigma). \\
 p(\mathbf{z}^2 | \mathbf{z}^0, \mathbf{a}) &= \text{Bernoulli}(\mathbf{q}^2), \text{ where } \mathbf{q}^2 = \text{SIGMOID}(\mathbf{l}^2). && (10)
 \end{aligned}$$

$$\begin{aligned}
 q(\mathbf{z}^k | \mathbf{x}^k) &= \text{Bernoulli}(\mathbf{q}^k), \text{ where } \mathbf{q}^k = \text{SIGMOID}(\mathbf{l}^k). && k \in \{0, 1\} \\
 q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) &= \text{Cat}(\mathbf{q}^a), \text{ where } \mathbf{q}^a = \text{SOFTMAX}(\mathbf{l}^a). && (11)
 \end{aligned}$$

 Figure 8.2: Main data pipeline, ELBO, generative model, and variational model of AMA₃⁺.

The network is trained by maximizing $\text{ELBO}(\mathbf{x}^0, \mathbf{x}^1)$ defined in Equation 9, where each probability is obtained from the corresponding neural network as defined in the generative model (Equation 10) and the variational model (Equation 11).

Theoretical Property Unlike AMA_3 , the loss function $\text{ELBO}(\mathbf{x}^0, \mathbf{x}^1)$ is theoretically justified as the lower bound of the log likelihood $\log p(\mathbf{x}^0, \mathbf{x}^1)$ of observing a pair of images $(\mathbf{x}^0, \mathbf{x}^1)$, following Maximum Likelihood Estimation (MLE) and variational inference framework (Section 2.3). An important implication of this theoretical property is that, *in theory*, when $p(\mathbf{x}^0, \mathbf{x}^1)$ converges to the ground-truth distribution, Latplan never generates visualizations containing invalid states and invalid transitions. In the ground-truth distribution, $p(\mathbf{x}^0, \mathbf{x}^1) = 0$ if $(\mathbf{x}^0, \mathbf{x}^1)$ is an invalid transition, or if either \mathbf{x}^0 or \mathbf{x}^1 are invalid states. MLE achieves this by maximizing $p(\mathbf{x}^0, \mathbf{x}^1)$ for real data, which reduces $p(\mathbf{x}^0, \mathbf{x}^1)$ for invalid data because a probability distribution sums/integrates into 1 ($\int p(\mathbf{x}^0, \mathbf{x}^1) d(\mathbf{x}^0, \mathbf{x}^1) = 1$). Thus, following the MLE framework, which is the current dominant theoretical foundation of machine learning models, converging a lower bound (ELBO) of the log likelihood $\log p(\mathbf{x}^0, \mathbf{x}^1)$ to the ground-truth distribution guarantees the correctness of the learned results as well as the planning results (due to the soundness of the planner being used).

In practice, this convergence is not achieved and visualized plans do not have a correctness guarantee in a traditional sense. However, note that this is not necessarily a disadvantage of our paradigm, since the use case of this system is where human supervision and manual modeling are not available. Moreover, even human could write a PDDL model with a bug that results in an incorrect modeling.

Interpretation Equation 9 is not only theoretically justified, but also has a clear interpretation. The first three terms represent reconstruction losses for $\mathbf{z}^0, \mathbf{z}^1$. The remaining KL divergence terms, which regularizes the training, have the following interpretations: $D_{\text{KL}}(q(\mathbf{z}^0 | \mathbf{x}^0) \| p(\mathbf{z}^0))$ addresses the stability of propositional symbols learned by SAE due to noise, as discussed in Section 5.2, Example 1. $D_{\text{KL}}(q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) \| p(\mathbf{a} | \mathbf{z}^0))$ addresses applicability of actions (precondition learning). $D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a}))$ addresses effects of actions and the stability of propositional symbols due to effects, as discussed in Section 5.2, Example 4.

$D_{\text{KL}}(q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) \| p(\mathbf{a} | \mathbf{z}^0))$ compares $q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1)$ and $p(\mathbf{a} | \mathbf{z}^0)$, which are outputs of ACTION and APPLICABLE, respectively. $q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1)$ is a distribution of an action predicted by observing the states both before and after the transition. In contrast, $p(\mathbf{a} | \mathbf{z}^0)$ is a distribution predicted without observing the result (successor state) of the action. The latter is thus intrinsically ambiguous, and returns a distribution of *applicable* actions (hence the name of the network). The former, in contrast, can be seen as a distribution of the *actual* action that happened, having access to the consequence of the action.

By minimizing the KL divergence on the training data, two things happen: First, the distribution of the actual action ($q(\dots)$) gets closer to applicable actions ($p(\dots)$) in each transition. Second, the distribution of applicable actions ($p(\dots)$) is stretched toward various actual actions ($q(\dots)$) by multiple transitions with the same \mathbf{z}^0 and different \mathbf{z}^1 . For example, $p(\dots)$ could be 1/3 for each of three applicable actions, while $q(\dots)$ is one-hot. As a result, informally speaking, the KL divergence is forcing each actual action to be in a set of applicable actions.

$D_{\text{KL}}(q(\mathbf{z}^1 = \mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 = \mathbf{z}^2 | \mathbf{z}^0, \mathbf{a}))$ compares $q(\mathbf{z}^1 | \mathbf{x}^1)$ and $p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})$, which are both distributions of a latent successor state \mathbf{z}^1 . $p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})$ is a distribution of the result of symbolically *applying* an action to a latent current state (thus is an output of APPLY), and $q(\mathbf{z}^1 | \mathbf{x}^1)$

is a distribution of the latent successor state obtained from an observation. By matching them, we maintain the stability of the latent propositions and the correctness of the effects at the same time.

To reduce the length of the paper, more in-depth discussion on the design decision and the derivation of this loss function was moved to the appendix (Section F). Readers who are not familiar with machine learning may find the section useful because we describe not only our mathematical derivation, but also the general strategy of statistical modelling which allows one to derive a machine learning model with a principled theoretical justification.

8.2 Cube-Like Graph

The state space modeled by AMA₂, and Vanilla Space AE as its end-to-end version, are not specifically designed for generating a compact STRIPS action model, therefore its STRIPS compilation may be exponentially large (Asai, 2020). To illustrate this issue, we consider the problem of assigning action labels to each edge in the latent space based on node embedding differences caused by action effects. This yields a graph class called *Cube-Like Graph*, which inspired the name of our new architecture:

Definition 3. A cube-like graph (Payan, 1992) is a simple³ undirected graph $G(S, D) = (V, E)$ defined by sets S and D . Each node $v \in V$ is a finite subset of S , i.e., $v \subseteq S$. The set D is a family of subsets of S , and for every edge $e = (v, w) \in E$, the symmetric difference between the connected nodes belongs to D , i.e., $d = v \oplus w = (v \setminus w) \cup (w \setminus v) \in D$.

For example, a unit cube becomes a cube-like graph if we assign a set to each vertex appropriately as in Figure 8.3, i.e.,

$$\begin{aligned} S &= \{x, y, z\}, & V &= \{\emptyset, \{x\}, \dots, \{x, y, z\}\}, \\ E &= \{(\emptyset, \{x\}), \dots, (\{y, z\}, \{x, y, z\})\}, & D &= \{\{x\}, \{y\}, \{z\}\}. \end{aligned}$$

The set-based representations can alternatively be represented as bit-vectors, e.g.,

$$V = \{(0, 0, 0), (0, 0, 1), \dots, (1, 1, 1)\}.$$

We observed a similarity between STRIPS action models and cube-like graphs, with the difference being that STRIPS action models consider a directed version, i.e., for every edge $e = (v, w) \in E$, their asymmetric differences $(d^+, d^-) = (w \setminus v, v \setminus w)$ satisfy $w = (v \setminus d^-) \cup d^+$, which is typically denoted as $w = (v \setminus \text{DEL}(a)) \cup \text{ADD}(a)$.

Consider the number of actions required in two node embeddings in Figure 8.3. The first embedding (left and center) requires 3 labels (6 labels if directed), where each label is assigned to 4 parallel edges which share the same node embedding differences. The set of node embedding differences corresponds to the set D , and each element of D represents an action, such as a whose difference is $(1, 0, 0)$ in the figure.

In contrast, the graph on the right has node embeddings that are randomly shuffled. Despite having the same topology and the same embedding size, this graph lacks the patterns we saw on the left, thus it needs more action labels, i.e., it lacks a compact STRIPS action model. For example, action a_1 represents a difference $(1, 1, 0) - (0, 0, 1) = (1, 1, -1)$ and action a_2 represents $(1, 0, 0)$, requiring different effects. While some edges may share the effects, such as the action b with difference $(-1, 0, 0)$, there are 9 differences in this graph, thus it needs 9 actions (18 if directed).

3. No duplicated edges between the same pair of nodes

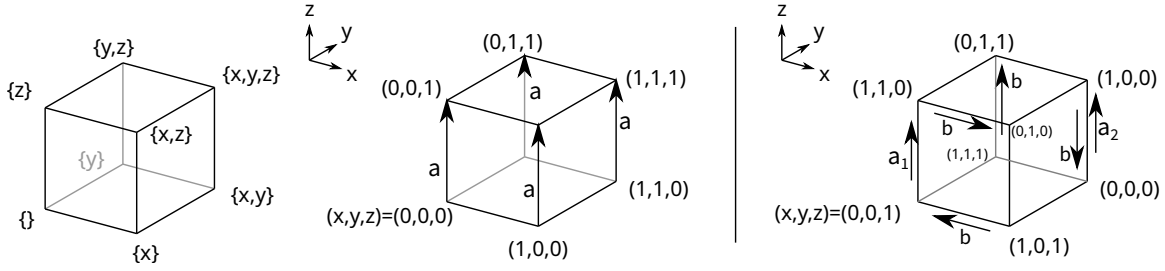


Figure 8.3: (Left) A graph representing a 3-dimensional cube can be made into a cube-like graph by assigning a set to each vertex. (Center) Its bit-vector representation. (Right) The same graph with a randomly shuffled node embedding.

8.3 Cube-Space AE (CSAE)

Based on this intuition, we named our new architecture Cube-Space AE. Cube-Space AE modifies the APPLY network so that it directly predicts the effects without taking the current state as the input and logically computes the successor state based on the predicted effect and the current state.

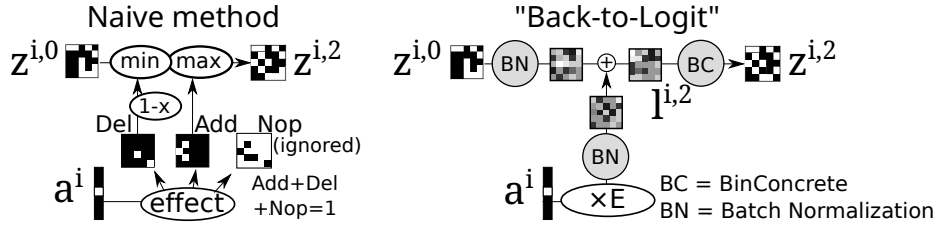


Figure 8.4: A naive and a Back-to-Logit implementation of the APPLY module of the CSAE.

Naive implementation: To illustrate the idea, we first present an ad-hoc, naive and impractical implementation of such a network in Figure 8.4 (left), which directly simulates STRIPS action application with vector operations. The EFFECT network predicts a binary tensor of shape $F \times 3$ using F -way Gumbel-Softmax of 3 categories. Each Gumbel Softmax corresponds to one bit in the F -bit latent space and 3 classes correspond to the add effect, delete effect, and NOP, only one of which is selected by the one-hot vector. The effects are applied to the current state by max/min operations. Formally, the naive CSAE is formulated as follows:

$$\mathbb{B}^F \ni z^2 = \max(\min(z^0, 1 - \text{DEL}(\mathbf{a})), \text{ADD}(\mathbf{a})), \quad \text{where} \quad (12)$$

$$\begin{aligned} \text{EFFECT}(\mathbf{a}) &= \text{GS}_\tau(\text{MLP}(\mathbf{a})) \in \mathbb{B}^{F \times 3}, & \text{ADD}(\mathbf{a}) &= \text{EFFECT}(\mathbf{a})_0, \\ \text{DEL}(\mathbf{a}) &= \text{EFFECT}(\mathbf{a})_1, & \text{NOP}(\mathbf{a}) &= \text{EFFECT}(\mathbf{a})_2. \end{aligned} \quad (13)$$

While intuitive, we found these naive implementations extremely difficult to train in AMA₃. Moreover, z^2 in these models is not produced by Binary Concrete using a logit l^2 , making it impossible to compute the KL divergence $D_{\text{KL}}(q(z^1 | \mathbf{x}^1) \| p(z^2 | z^0, \mathbf{a}))$ where $q(z^1 | \mathbf{x}^1) =$

SIGMOID(\mathbf{l}^1) and $p(\mathbf{z}^2 | \mathbf{z}^0, \mathbf{a}) = \text{SIGMOID}(\mathbf{l}^2)$, which is required by the theoretically motivated ELBO objective of AMA₃[†]. We thus abandoned this idea in favor of a theoretically justifiable alternative.

Back-to-Logit: Our contribution to the architecture is *Back-to-Logit* (BTL, Figure 8.4, right), a generic approach that computes a logical operation in the continuous logit space. We re-encode a logical, binary vector back to a continuous, logit representation by an element-wise monotonic function m . This monotonicity preserves the order between true (1) and false (0) even after transformed into real numbers. We then apply the given action by adding a *continuous effect vector* to the continuous current state. The effect vector is produced by applying an MLP named EFFECT to the action vector \mathbf{a} . After adding the continuous vectors, we re-discretize the result with Binary Concrete. Formally,

$$\mathbf{z}^2 = \text{BC}_\tau(\text{APPLY}(\mathbf{z}^0, \mathbf{a})) = \text{BC}_\tau(m(\mathbf{z}^0) + \text{EFFECT}(\mathbf{a})). \quad (14)$$

Batch Normalization as m : We found that an easy and successful way to implement m is *Batch Normalization* (Ioffe & Szegedy, 2015), a method that was originally developed for addressing *covariate shift* in deep neural networks. For simplicity, we consider a scalar operation, which can be applied to vectors element-wise. During the batched training of the neural network, Batch Normalization layer $\text{BN}(x)$ takes a minibatch input $B = \{x^1 \dots x^{|B|}\}$, computes the mean μ_B and the variance σ_B^2 of B , then shifts and scales each x^i so that the resulting batch has a mean of 0 and a variance of 1. It then shifts and scales the results by two trainable scalars γ and β , which are shared across different batches. Formally,

$$\forall x^i \in B; \text{BN}(x^i) = \frac{x^i - \mu_B}{\sigma_B} \gamma + \beta. \quad (15)$$

After the training, BN behaves deterministically. At the end of the training, μ_B and σ_B are set to the statistics of the entire training dataset \mathcal{X} , i.e., $\mu_{\mathcal{X}}, \sigma_{\mathcal{X}}$.

Final definition: Furthermore, since \mathbf{a} is a probability vector over A action ids and \mathbf{a} eventually converges to a one-hot vector due to Gumbel-Softmax annealing, the additional MLP can be merely a linear embedding, i.e., $\text{EFFECT}(\mathbf{a}) = \mathbf{E}\mathbf{a}$, where $\mathbf{E} \in \mathbb{R}^{F \times A}$. It also helps the training if we apply batch normalization on the effect vector. Therefore, a recommended implementation is:

$$\mathbf{z}^2 = \text{BC}_\tau(\text{APPLY}(\mathbf{z}^0, \mathbf{a})) = \text{BC}_\tau(\text{BN}(\mathbf{z}^0) + \text{BN}(\mathbf{E}\mathbf{a})). \quad (16)$$

Similar to the original Vanilla Space AE, we can interpret it from a Bayesian generative modeling perspective as well. We discuss the interpretation in Appendix Section G.

Equivalence to STRIPS: Consider an ideal training result where $\forall i; \mathbf{z}^{i,1} \equiv \mathbf{z}^{i,2}$, i.e.,

$$D_{\text{KL}}(q(\mathbf{z}^1 = \mathbf{z}^1 | \mathbf{x}^1) || p(\mathbf{z}^1 = \mathbf{z}^2 | \mathbf{z}^0, \mathbf{a})) = 0.$$

States learned by BTL have the following property:

Theorem 1. For state transitions $\mathbf{z}^{i,0}, \mathbf{z}^{i,1}$ whose action is \mathbf{a} , the transitions satisfy the following conditions. For each bit j , it is either:

$$(add:) \forall i; (\mathbf{z}_j^{i,0}, \mathbf{z}_j^{i,1}) \in \{(0, 1), (1, 1)\}, \quad (17)$$

$$\text{or } (del:) \forall i; (\mathbf{z}_j^{i,0}, \mathbf{z}_j^{i,1}) \in \{(1, 0), (0, 0)\}, \quad (18)$$

$$\text{or } (nop:) \forall i; (\mathbf{z}_j^{i,0}, \mathbf{z}_j^{i,1}) \in \{(0, 0), (1, 1)\}. \quad (19)$$

In other words, they are bitwise monotonic ($z_j^{i,0} \leq z_j^{i,1}$, $z_j^{i,0} = z_j^{i,1}$, or $z_j^{i,0} \geq z_j^{i,1}$), deterministic ($z_j^{i,0} \mapsto z_j^{i,1}$ is a function), and these three modes never mix up for the same action. The proof is straightforward from the monotonicity of m and Binary Concrete (See Appendix Section H).

This theorem guarantees that each action deterministically sets a certain bit on and off in the binary latent space. Therefore, the actions and the transitions satisfy the STRIPS state transition rule $s' = (s \setminus \text{DEL}(a)) \cup \text{ADD}(a)$, thus enabling a direct translation from neural network weights to PDDL modeling language, as depicted in Figure 1.2.

Effect extraction: To extract the effects of an action a from CSAE, we compute $\text{ADD}(a) = \text{APPLY}(a, \mathbf{0})$ and $\text{DEL}(a) = 1 - \text{APPLY}(a, \mathbf{1})$ for each action a , where $\mathbf{0}, \mathbf{1} \in \mathbb{B}^F$ are vectors filled by zeros/ones and has the same size as the binary embedding. Since APPLY deterministically sets values to 0 or 1, feeding these vectors is sufficient to see which bit it turns on and off. For each j -th bit that is 1 in each result, a corresponding proposition is added to the add/delete-effect, respectively.

There is one difference between theory and practice. As mentioned in Equation 15, batch normalization has a trainable parameter γ for each dimension. This γ must be positive in order for a batch normalization layer to be monotonic. The value of γ is typically initialized by a positive constant (e.g., 1 in Keras) and typically stays positive during the training. However, in rare cases, it is possible that γ turns negative in some bits, which requires special handling.

We call this monotonicity violation as XOR semantics, as the resulting bitwise value follows a pattern $\{(0, 1), (1, 0)\}$ that is not allowed under STRIPS. Detecting XOR semantics from the extracted effects is easy because these problematic bits have both add- and delete-effects in the same action. We compile them away by splitting the action into two versions, appending appropriate preconditions. While the compilation is exponential to the number of violations, violations are usually rare. We will report the number of violations in the empirical evaluation.

Precondition extraction: To extract the preconditions of an action a , we propose a simple ad-hoc method that is similar to (Wang, 1994) and (Amado et al., 2018). It looks for bits that always have the same value when a is used. Let $\mathcal{Z}^0(a)$ be the set of states that we can observe the usage of an action a in the dataset, i.e., $\{z^{i,0} \mid (z^{i,0}, z^{i,1}) \in \mathcal{Z}; \arg \max(\text{ACTION}(z^{i,0}, z^{i,1})) = a\}$. Then the positive and the negative preconditions of an action a are defined as follows:

$$\text{POS}(a) = \{f \mid \forall z^{i,0} \in \mathcal{Z}^0(a); z_f^{i,0} = 1\}, \text{NEG}(a) = \{f \mid \forall z^{i,0} \in \mathcal{Z}^0(a); z_f^{i,0} = 0\}. \quad (20)$$

This ad-hoc method suffers from a serious lack of accuracy because the propositions whose values differ between different samples in $\mathcal{Z}^0(a)$ are always treated as “don’t care” even if they could consist of several patterns. Suppose the ground-truth generator of $\mathcal{Z}^0(a)$ is a random vector $\mathbf{z} = [0, a, \neg b, \neg b, b, 1, c]$ of independent Bernoulli(0.5) random variables a, b, c and constants 0, 1. The random vector can be alternatively represented by a logical formula $\neg \mathbf{z}_0 \wedge \mathbf{z}_5 \wedge ((\neg \mathbf{z}_2 \wedge \neg \mathbf{z}_3 \wedge \mathbf{z}_4) \vee (\mathbf{z}_2 \wedge \mathbf{z}_3 \wedge \neg \mathbf{z}_4))$. The ad-hoc method will only recognize $\neg \mathbf{z}_0 \wedge \mathbf{z}_5$, failing to detect the disjunctions.

While we could develop a divide-and-conquer method which recursively extracts disjunctive conditions, DSAMA (Asai, 2020) showed that it is impractical. It learned them with decision trees applied to $\mathcal{Z}^0(a)$ obtained from AMA2, then compiled the tree into a disjunctive precondition in PDDL. Disjunctive conditions are typically compiled away in modern planners at the cost of exponential blowup in the number of ground actions, which a deeper and more accurate tree suffers

from. On the other hand, if we stop the recursion at a certain depth to suppress the explosion, such a shallow decision tree suffers from the limited accuracy.

These issues are naturally caused by the fact that the network and the state representation are not conditioned to learn a compact action model with conjunctive preconditions. To address these issues, we propose AMA_4^+ , an improved architecture which also restricts the actions to have fully conjunctive preconditions.

9. AMA_4^+ : Learning Preconditions as Effects Backward in Time

While AMA_3^+ managed to model the STRIPS action effects, it requires an ad-hoc precondition extraction with limited accuracy because the CSAE lacks an explicit mechanism for action preconditions. We address this problem in a *Bidirectional Cube-Space AE* (BiCSAE): a neural network that casts precondition learning as *effect learning for regression planning*. To our knowledge, no existing work has addressed precondition learning in this manner. Modeling the precondition learning as a type of effect learning requires us to treat the problem in a time-symmetric manner. However, it causes a unique issue in that we have an expectation that regressed states are complete.

9.1 Complete State Regression Semantics

Regression planning is a group of search methods that solve a classical planning problem by looking for the initial state backward from the goal (Bonet & Geffner, 2001; Alcázar et al., 2013). It typically operates on a partial state / condition as a search node, which specifies values only for a subset of state variables. For a classical planning problem $\langle P, A, I, G \rangle$, it starts from a node that contains the goal condition G , unlike usual forward search planners, which starts from the initial state I . State transitions on partial states are performed by *regression*, which infers the properties of a state prior to applying an action.

In general, regression produces partial states. This is so, *even if we regress a complete state*. Because of the nature of our learned latent representation, we must maintain this complete state property. Consider Table 9.1: a list of possible transitions of a propositional variable $p \in \{0, 1\}$ when a predecessor state s transitioned to a successor state t using a STRIPS action. In the precondition / effect columns, + indicates a positive precondition and an add-effect, - indicates a negative precondition and a delete effect, 0 indicates that the action does not contain either effects, * indicates that the action does not contain either preconditions. When the precondition is not specified (*) for a bit that is modified by an effect, there is ambiguity in the source state, indicated by “?” (line 3, 6 in Table 9.1) — the previous value could be either 0 or 1. In other words, regressing such an action is non-deterministic, and the default STRIPS semantics for unspecified precondition is disjunctive (true or false).

One way to address this uncertainty / non-determinism / disjunctiveness in complete state is to reformulate the action model using *prevail condition* introduced in SAS+ formalism (Bäckström & Nebel, 1995), a condition that is satisfied when the value is *unchanged*. In this form, a precondition for a propositional variable is either +, -, or 0 (unchanged), instead of +, -, * (don’t-care). This modification rules out the ambiguous columns as seen in Table 9.1 (right), making regression deterministic, and making preconditions conjunctive. Note that certain combinations of preconditions and effects are equivalent. For example, lines 0, 1, 3, as well as lines 6, 8, 9 in Table 9.1 are redundant. We refer to this restricted form of regression as *complete state regression*.

	s	precondition	effect	t		s	precondition	effect	t
0	$p = 1$	+	+	$p = 1$	0	$p = 1$	+	+	$p = 1$
1	$p = 1$	+	0	$p = 1$	1	$p = 1$	+	0	$p = 1$
2	$p = 1$	+	-	$p = 0$	2	$p = 1$	+	-	$p = 0$
3	$p = ?$	*	+	$p = 1$	3	$p = 1$	0	+	$p = 1$
4	$p = 0$	*	0	$p = 0$	4	$p = 0$	0	0	$p = 0$
5	$p = 1$	*	0	$p = 1$	5	$p = 1$	0	0	$p = 1$
6	$p = ?$	*	-	$p = 0$	6	$p = 0$	0	-	$p = 0$
7	$p = 0$	-	+	$p = 1$	7	$p = 0$	-	+	$p = 1$
8	$p = 0$	-	0	$p = 0$	8	$p = 0$	-	0	$p = 0$
9	$p = 0$	-	-	$p = 0$	9	$p = 0$	-	-	$p = 0$

Table 9.1: (Left) Possible transitions of a propositional value with a STRIPS action. (Right) Possible transitions of a propositional value with a STRIPS action, modeled by a prevail condition (precondition=0).

The semantics of preconditions with positive, negative, and prevail conditions exactly mirrors the semantics of effects (add, delete, NOP), with the only difference being the direction of application. For example, BTL is able to model NOP, which does not change the value of a propositional variable. In complete state regression, this corresponds to a prevail condition.

9.2 Learning with Complete State Regression in Bidirectional Cube-Space AE (BiCSAE)

Complete state regression semantics provides a theoretical background for modeling a precondition learning problem as an effect learning problem. Based on this underlying semantics, we now propose *Bidirectional Cube-Space AE* (BiCSAE), a neural network that can learn the effects and the preconditions at the same time. Since the semantics of complete state regression and STRIPS effects are equivalent, we can apply the same Back-to-Logit technique to learn a complete state regression model. As a result, the resulting network contains a symmetric copy of Cube-Space AE (AMA_3^+).

We build BiCSAE by augmenting CSAE with a network $\text{REGRESS}(z^{i,1}, \mathbf{a}^i)$ that uses the same BTL mechanism for predicting the outcome of action regression, i.e., predict the current state $z^{i,3}$ from a successor state $z^{i,1}$ and a one-hot action vector \mathbf{a}^i . A BiCSAE also has $\text{REGRESSABLE}(z^{i,1})$, a symmetric counterpart of $\text{APPLICABLE}(z^{i,0})$ which returns a probability over actions and is used for regularizing \mathbf{a}^i via KL divergence. The generative model is also a symmetric copy of AMA_3^+ . The main data pipeline excluding the networks for regularization (REGRESSABLE , APPLICABLE) is formalized in Figure 9.1. The full network is depicted in Figure 9.2.

Since \mathbf{a}^i is learned unsupervised, action replication happens automatically during the training. REGRESS is formalized with a trainable matrix \mathbf{P} (for preconditions) as:

$$z^{i,3} = \text{BC}_\tau(\text{REGRESS}(z^{i,1}, \mathbf{a}^i)) = \text{BC}_\tau(\text{BN}(z^{i,1}) + \text{BN}(\mathbf{P}\mathbf{a}^i)). \quad (21)$$

In REGRESS , add-effects and delete-effects now correspond to *positive preconditions* and *negative preconditions*. While negative preconditions are (strictly speaking) out of STRIPS formalism, it is commonly supported by modern classical planners that participate in the recent competitions.

	(input)	$\mathbf{x}^{i,0}, \mathbf{x}^{i,1}$
	(encoder and encoded logits)	$\mathbf{l}^{i,0}, \mathbf{l}^{i,1} = \text{ENCODE}(\mathbf{x}^{i,0}), \text{ENCODE}(\mathbf{x}^{i,1})$
	(sampled binary representations)	$\mathbf{z}^{i,0}, \mathbf{z}^{i,1} = \text{BC}_\tau(\mathbf{l}^{i,0}), \text{BC}_\tau(\mathbf{l}^{i,1})$
	(action assignment)	$\mathbf{a}^i = \text{GS}_\tau(\text{ACTION}(\mathbf{l}^{i,0}, \mathbf{l}^{i,1}))$
	(logits for progression / forward dynamics)	$\mathbf{l}^{i,2} = \text{APPLY}(\mathbf{z}^{i,0}, \mathbf{a}^i)$
	(sampled binary representation for progression)	$\mathbf{z}^{i,2} = \text{BC}_\tau(\mathbf{l}^{i,2})$
	(logits for regression / backward dynamics)	$\mathbf{l}^{i,3} = \text{REGRESS}(\mathbf{z}^{i,1}, \mathbf{a}^i)$
	(sampled binary representation for regression)	$\mathbf{z}^{i,3} = \text{BC}_\tau(\mathbf{l}^{i,3})$
	(reconstructions)	$\tilde{\mathbf{x}}^{i,0}, \tilde{\mathbf{x}}^{i,1} = \text{DECODE}(\mathbf{z}^{i,0}), \text{DECODE}(\mathbf{z}^{i,1})$
	(reconstruction based on forward dynamics)	$\tilde{\mathbf{x}}^{i,2} = \text{DECODE}(\mathbf{z}^{i,2})$
	(reconstruction based on backward dynamics)	$\tilde{\mathbf{x}}^{i,3} = \text{DECODE}(\mathbf{z}^{i,3})$

Figure 9.1: Main data pipeline of AMA_4^+ .

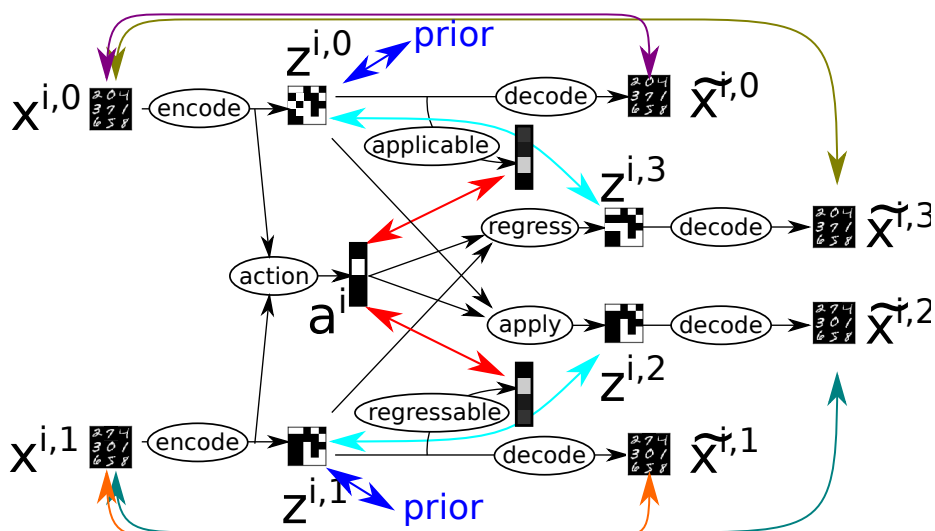


Figure 9.2: An illustration of Bidirectional Cube-Space AE for AMA_4^+ .

BiCSAE restricts the preconditions to be strictly conjunctive. Note that STRIPS planning assumes deterministic environments, thus its effects are conjunctive: For example, if an action contained a disjunctive effect such as “flipping a coin makes it tail *or* head”, then the state transition is non-deterministic. As seen in Theorem 1, this does not happen in CSAE. By using the same mechanism for learning the regression (preconditions), preconditions learned by BiCSAE do not contain disjunctions, its regression is deterministic, and it achieve complete regression semantics, as promised.

To extract the preconditions from the network, we apply the same method used for extracting the effects from the progressive/forward dynamics with two modifications. First, when an action contains a prevail condition for a bit j and either an add-effect or a delete-effect for j , we must convert a prevail condition for j into a positive / negative precondition for j , respectively. Otherwise, it underspecifies a set of transitions. Second, when both the effect and the precondition of a bit j have the XOR semantics, we should duplicate the action only once (resulting in 2 copies of actions), not separately (resulting in 4 copies of actions).

Given a BiCSAE network as shown in Figure 9.2, we must define its optimization objective as a variational lower bound similar to that of CSAE/AMA₃⁺. This is straightforward given the ELBO of CSAE, because the backward network is a symmetric copy of CSAE. We obtain an identical loss function for the forward and the backward network, then optimize the average of these objectives.

As seen in AMA₃⁺, each uni-directional model already individually captures both the preconditions and the effects by itself. We therefore have duplicate components in the forward and the backward model: Preconditions are learned by both APPLICABLE and REGRESS, effects are learned by both APPLY and REGRESSABLE. Two models are individually theoretically justified and we merely trained them at once. Due to this decoupling, the training objectives in the forward and the backward direction do not need any special term designed for bidirectionality. Note that, while two models are trained at once, they share ENCODE, DECODE and ACTION, therefore they use the same state encoding and the same set of actions. In other words, this weight sharing ties two uni-directional models together.

Having duplicate components does not affect the theoretical validity of the model. This is because, at the limit of convergence where the lower-bound (ELBO) matches the ground truth ($\log p(\mathbf{x}^0, \mathbf{x}^1)$) in both forward/backward models, the variational distributions (distributions using q) matches the ground-truth generative distributions (distributions using p), as explained in Section 2.3. Therefore the forward model eventually matches the backward model and vice versa.

10. Training Performance

In this section, we empirically and thoroughly analyze the networks proposed in this paper. We compare several metrics (overall accuracy, latent state stability, successor prediction accuracy, monotonicity violation, precondition accuracy) between different latent space priors, AMA₃⁺ and AMA₄⁺, and various hyperparameters. The primary objective of the comparisons made in this section is to verify the following factors:

1. The effect of hyperparameter tuning on the overall accuracy, especially the impact of tuning $\beta_{1..3}$.
2. The effect of non-standard prior Bernoulli(ϵ), $\epsilon = 0.1$ proposed in Section 5.4, compared to the standard uniform Bernoulli prior $\epsilon = 0.5$.

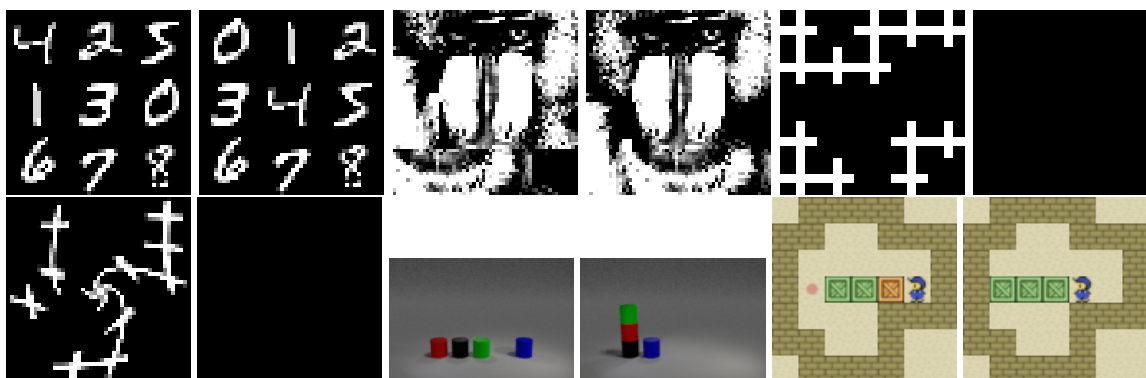


Figure 10.1: An example of problem instances (initial and goal images) for each domain. In Blocks-world and Sokoban, the images shown are high-resolution images, while the actual training and planning experiments are performed on smaller resized images.

3. The successor prediction accuracy of Back-to-Logit proposed in Section 5.4.
4. The number of monotonicity violations described in Section 8.3 in the learned results.

In addition, we provide a training curve plot in Section I in the appendix to facilitate the readers’ replication of our result.

We first describe the experimental setup including dataset generation Section 10.1 and training/architecture details Section 10.2.

10.1 Experimental Domains

We evaluate Latplan on 6 domains. Example visualizations are shown in Figure 10.1.

- **MNIST 8-Puzzle** (Asai & Fukunaga, 2018) is a 42x42 pixel, monochrome image-based version of the 8-Puzzle. Tiles contain hand-written digits (0-9) from the MNIST database (LeCun et al., 1998), which are shrunk to 14x14 pixels so that each state of the puzzle is a 42x42 image. Valid moves swap the “0” tile with a neighboring tile, i.e., the “0” serves as the “blank” tile in the classic 8-Puzzle. 8-Puzzle has 362880(= 9!) states and 967680 transitions, while half of the states are unreachable. The longest shortest path in 8-puzzle contains 31 moves (Reinefeld, 1993). From any specific goal state, the reachable number of states is 181440 (9!/2). Note that the same image is used for each digit in all states, e.g., the tile for the “1” digit is the same image in all states.
- The **Scrambled Photograph (Mandrill) 15-puzzle** cuts and scrambles a real photograph, similar to the puzzles sold in stores. We used a “Mandrill” image taken from the USC-SIPI benchmark set (Weber, 1997). 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). Unlike 8-puzzle, this domain can increase the state space easily by dissecting the image 4x4 instead of 3x3, generating 15-puzzle instances whose state space is significantly larger than that of 8-puzzle. The image consists of 56x56 pixels. It was converted to greyscale, and then histogram-normalization and contrast enhancement was applied.

- A recreation of **LightsOut** (Anderson & Feil, 1998) by Tiger Electronics in 1995, or sometimes also known as “Magic Square” game mode of *Merlin The Electronic Wizard*. It is an electronic puzzle 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 to turn all lights Off. Unlike previous puzzles, a single operator can flip 5 locations at once and removes some “objects” (lights). This demonstrates that Latplan is not limited to domains with highly local effects and static objects. $n \times n$ LightsOut has 2^{n^2} states and $n^2 2^{n^2}$ transitions. In this paper, we used 5x5 configuration (33554432 states) whose images consist of 45x45 pixels.
- **Twisted LightsOut** distorts images in LightsOut by a swirl effect available in scikit-image package, showing that Latplan is not limited to handling rectangular “objects”/regions. It has the same image dimensions as the original LightsOut. The effect is applied to the center, with strength=3, linear interpolation, and radius equal to 0.75 times the dimension of the image.
- **Photo-realistic Blocksworld** (Asai, 2018) is a dataset that consists of 100x150 RGB images rendered by Blender 3D engine. The image contains complex visual elements such as reflections from neighboring blocks. We modified the rendering parameter to simplify the dataset: We limited the object shapes to be cylinders and removed the jitters (small noise) in the object placements. The final images are resized to 30x45 pixels. As shown in the figures, Blocksworld contains 4 objects and maximum 5 towers, which results in $4! \times 5^4 = 15000$ logical states and each state has a branching factor from 4 (where all blocks are stacked in a single tower) to 20 (where all blocks are on the table). For each logical state, there are infinite number of images due to different random jitter and raytracing rendering noise.
- **Sokoban** (Culberson, 1998; Junghanns & Schaeffer, 2001) is a PSPACE-hard puzzle domain whose 112x112 pixel visualizations are obtained from the PDDLgym library (Silver & Chitnis, 2020) and are resized into 28x28 pixels. Unlike other domains, this domain is irreversible and dead-ends exist. We used p006-microban-sequential instance as the visualization source. p006-microban-sequential has 23363 states reachable from the initial state. States have varying branching factors from 1 to 4. The dataset is generated by generating all reachable states with an exhaustive Dijkstra search and their corresponding transitions, then sampling 20000 valid transitions.

In 8-Puzzle and two Lightsout domains, we sampled 5000 transitions / 10000 states. In 15-Puzzle, Blocksworld, and Sokoban, we sampled 20000 transitions / 40000 states because they are visually challenging domains that contain more objects, more noise, and more complex modes of state transitions. All datasets are generated by domain-specific generators. Except sokoban, the generators generate a pair of states at a time: One uniformly sampled configuration of the environment and its uniformly sampled successor state. In Sokoban, the dataset is uniformly sampled from the whole state space enumerated by an exhaustive search. These datasets are divided into 90%,5%,5% for the training set, validation/tuning set, and testing set, respectively. During the development of the network models, we look at the validation set to tune the network while only in the final report we evaluate the network on the test set and report the results. While the number of states (images) used for visually challenging domains may appear to be reaching the interpolation rather than generalization, note that there are much more transitions (multiplied by the branching factor) in the environment that the system must generalize to.

The pixel values in the image datasets are normalized to mean 0, variance 1 for each pixel and channel, i.e., for an image pair $(\mathbf{x}^{i,0}, \mathbf{x}^{i,1})$ where $\mathbf{x}^{i,j} \in \mathbb{R}^{H,W,C}$ (with H, W, C being height, width, and color channel respectively), $\forall k, l, m; \mathbb{E}_{i,j}[\mathbf{x}_{klm}^{i,j}] = 0$ and $\text{Var}_{i,j}[\mathbf{x}_{klm}^{i,j}] = 1$.

All experiments are performed on a distributed compute cluster equipped with nVidia Tesla K80 / V100 / A100 GPUs and Xeon E5-2600 v4. With V100 (a Volta generation GPU), training a single instance of AMA_3^+ model on 8-Puzzle takes 2 hours, and AMA_4^+ model takes 3 hours due to additional networks. 15-Puzzle, Blocksworld, and Sokoban take more runtime varying from 5 hours to 15 hours due to more data points and larger images. The system is implemented on top of Keras library (Chollet et al., 2015).

10.2 Network Specifications

We trained the AMA_3^+ and the AMA_4^+ models as discussed in Sections 8-9. We do not include training results of previous models $\text{AMA}_{1..3}$ due to the lack of STRIPS export capability and their ad-hoc natures. The ad-hoc nature indeed prevents us from defining an effective metric that is independent from the implementation of the neural network. For example, while we can directly compare the loss values of the AMA_3^+ and the AMA_4^+ because both values are lower bounds of the likelihood $p(\mathbf{x}^0, \mathbf{x}^1)$, we cannot derive a meaningful conclusion by comparing the loss values of $\text{AMA}_{1..3}$.

Detailed specifications of each network component of AMA_3^+ and AMA_4^+ are listed in Table 10.1. The encoder and the decoders consist of three convolutional layers (Fukushima, 1980; LeCun et al., 1989) each, while action-related networks primarily consist of fully connected networks. We use Rectified Linear Unit ($\text{RELU}(x) = \max(0, x)$) as the activation function of each layer, which performs better in deep networks because it avoids gradient vanishing compared to a SIGMOID function. Each activation is typically followed by Batch Normalization (BN) (Ioffe & Szegedy, 2015) and Dropout (Hinton et al., 2012) layers, both of which are generally known to improve the performance and allow for training the network with a higher learning rate. A layer that is immediately followed by a BN layer does not need the bias weights because BN has its own bias weights. Following the theoretical and empirical finding by He, Zhang, Ren, and Sun (2015), all layers that are immediately followed by a ReLU activation are initialized by Kaimin He’s uniform weight initialization (He et al., 2015), as opposed to other layers that are initialized by the Glorot Xavier’s uniform weight initialization (Glorot & Bengio, 2010), which is the default in Keras library. In addition, the input is augmented by a GaussianNoise layer which adds a fixed amount of Gaussian noise, making the network an instance of Denoising AutoEncoder (Vincent et al., 2008).

For each dataset, we evaluated 30 hyperparameter configurations using grid search, varying the latent space size F and the β_1, β_3 coefficients of KL divergence terms (β_2 was kept at 1). Parameter ranges are listed in Table 10.2. We tested numerous hyperparameters during the development (using validation set) to narrow down the range of hyperparameters to those listed in this table. Naturally, a single hyperparameter does not work for all domains because, for example, a more complex domain (e.g., 15 Puzzle) requires a larger latent space than a simpler domain (e.g., 8 Puzzle). The values in this table have a sufficient range for covering all domains we tested.

10.3 Accuracy Measured as a Lower Bound of Likelihood

We first evaluate the ELBO, the lower bound of the likelihood $\log p(\mathbf{x}^0, \mathbf{x}^1)$ of observing the entire dataset (the higher, the better). The summary of the results is shown in Table 10.3. We report the

<i>Subnetworks</i>	<i>Layers</i>
ENCODE	GaussianNoise(0.2), BN, Conv(5,32), ReLU, BN, Dropout(0.2), Conv(5,32), ReLU, BN, Dropout(0.2), Conv(5,32)
DECODE	BN, Conv(5,32), ReLU, BN, Dropout(0.2), Conv(5,32), ReLU, BN, Dropout(0.2), Conv(5,32)
ACTION	SIGMOID, fc(1000), ReLU, BN, Dropout(0.2), fc(6000)
APPLY, REGRESS	Equation 16
APPLICABLE, REGRESSABLE	fc(6000)

Table 10.1: Detailed overview of the subnetworks of AMA_3^+ and AMA_4^+ . “Conv(k, c)” denotes a convolutional layer with kernel size k and channel width c , and “fc(w)” denotes a fully-connected layer with width w . We omit reshaping operators such as “flatten.” To maintain the notational consistency with Figure 8.1, this table does not include BC_τ activation in ENCODE and GS_τ activation in ACTION.

<i>Training parameters</i>	
Optimizer	Rectified Adam (Liu et al., 2019)
Training Epochs	2000
Batch size	400
Learning rate	10^{-3}
Gradient norm clipping	0.1
<i>Gumbel Softmax / Binary Concrete annealing parameters</i>	
Initial annealing temperature τ_{\max}	5
Final annealing temperature τ_{\min}	0.5
Annealing schedule $\tau(t)$ for epoch t	$\tau_{\max} \left(\frac{\tau_{\min}}{\tau_{\max}} \right)^{\frac{\min(t, 1000)}{1000}}$
<i>Network shape parameters</i>	
Latent space dimension F	$F \in \{50, 100, 300\}$
Maximum number of actions A	$A = 6000$
<i>Loss and Regularization Parameters</i>	
σ for all reconstruction losses (cf. Section 2.3)	0.1
β_1 for $D_{\text{KL}}(q(\mathbf{z}^{i,0} \mathbf{x}^{i,0}) p(\mathbf{z}^{i,0}))$	$\beta_1 \in \{1, 10\}$
β_2 for $D_{\text{KL}}(q(\mathbf{a}^i \mathbf{x}^{i,0}, \mathbf{x}^{i,1}) p(\mathbf{a}^i \mathbf{z}^{i,0}))$	1
β_3 for $D_{\text{KL}}(q(\mathbf{z}^{i,1} \mathbf{x}^{i,1}) p(\mathbf{z}^{i,2} \mathbf{z}^{i,0}, \mathbf{a}^i))$	$\beta_3 \in \{1, 10, 100, 1000, 10000\}$

Table 10.2: List of hyperparameters.

ELBO values obtained by setting $\beta_{1..3}$ to 1 after the training. The original maximization objectives with $\beta_{1..3} \geq 1$ used for training are lower bounds of the ELBO with $\beta_{1..3} = 1$ because all KL divergences have a negative sign (Equation 47) and KL divergences themselves are always positive (Equation 2.3). Resetting $\beta_{1..3}$ to 1 improves / increases the lower bound while still being the lower bound of the likelihood. Since ELBO is a model-agnostic metric, we can directly compare the values of different models.

The actual numbers we report are loss values -ELBO (negative value of ELBO) that are minimized (lower the better). In addition, the numbers do not include a constant term $C_2 = \log \sqrt{2\pi\sigma^2}$ with $\sigma = 0.1$ multiplied by the number of pixels (cf. Section 2.3). While the constant should be added to obtain the true value of -ELBO, and -ELBO+Constant has an intuitive interpretation as a sum of scaled squared errors and KL divergences. The constant does not affect the training either. The drawback is that the -ELBO we report should not be compared between different domains because different datasets have different numbers of pixels, resulting in a different amount of constant offsets.

Note that the best accuracy/ELBO alone is *not* the sole factor that improves the likelihood of success in solving visual planning problems. It is affected by multiple factors, including all metrics we evaluate in the following sections. For example, even if the overall accuracy (-ELBO, which is a sum of reconstruction and KL losses) is good, the planning is not likely to succeed if its action model does not predict the latent successor state accurately because it overly focuses on improving the reconstruction accuracy and sacrifices successor prediction accuracy modeled by $\beta_3 D_{\text{KL}}(q(z^{i,1} | \mathbf{x}^{i,1}) || p(z^{i,2} | z^{i,0}, \mathbf{a}^i))$.

In Table 10.3, we first assess the effectiveness of β -VAE objective with tuned $\beta_{1..3}$. Comparing the best hyperparameters (“kltune”) and the baseline that uses $\beta_{1..3} = 1$ during the training (“no-tune”), we observed that tuning these parameters is important for obtaining accurate models. Note that the latent space size F is still tuned in both cases.

We also plotted the results with all matching hyperparameters (same F) in Figure 10.2. The scatter plot provides richer information on how these hyperparameters affect the ELBO. It emphasizes the importance of hyperparameter tuning: Bad hyperparameter of $\beta_{1..3}$ could quite negatively affect the result while tuning the value appropriately yields a better result.

We next compare the results obtained by training the networks with a standard Bernoulli($\epsilon = 0.5$) prior and with a proposed Bernoulli($\epsilon = 0.1$) prior for the latent vector z . As we already discussed in Section 5.2, we observed comparable results; therefore it does not affect the *accuracy*. For the scatter plot, please refer to Figure 10.3. This is not surprising, as we are merely training the network with two different underlying assumptions: Bernoulli($\epsilon = 0.5$) assumes an *open-world assumption*, while $\lim_{\epsilon \rightarrow 0} \text{Bernoulli}(\epsilon)$ assumes a *closed-world assumption* (Section 5.4).

We next compare AMA_3^+ and AMA_4^+ models by their ELBO. Table 10.3 and Figure 10.4 show that the accuracy of AMA_3^+ and AMA_4^+ are comparable.

10.4 Stability of Propositional Symbols Measured as a State Variance

For each hyperparameter which resulted in the best ELBO in each domain and configuration, we next measured other effects of using $p(z^0) = \text{Bernoulli}(\epsilon = 0.1)$ (“kltune”) against the default

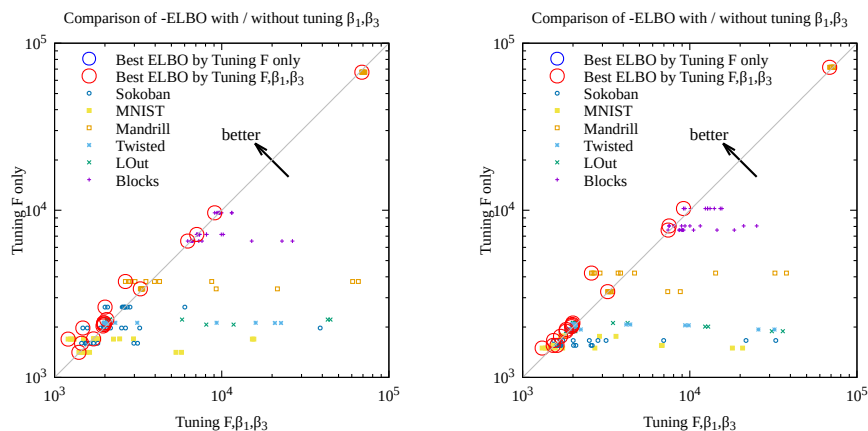


Figure 10.2: (Left: AMA_3^+ , Right: AMA_4^+ .) Comparing the effect of tuning $\beta_{1..3}$ on -ELBO. Each point represents a pair of configurations with the same F , where the y -axis always represents the result from $\beta_1 = \beta_3 = 1$, while the x -axis represents the results from various β_1, β_3 . For each F , we highlighted the best configuration of β_1, β_3 with blue circles. While bad parameters will significantly degrade the accuracy (below the diagonal), tuning the value appropriately will improve the accuracy (above the diagonal).

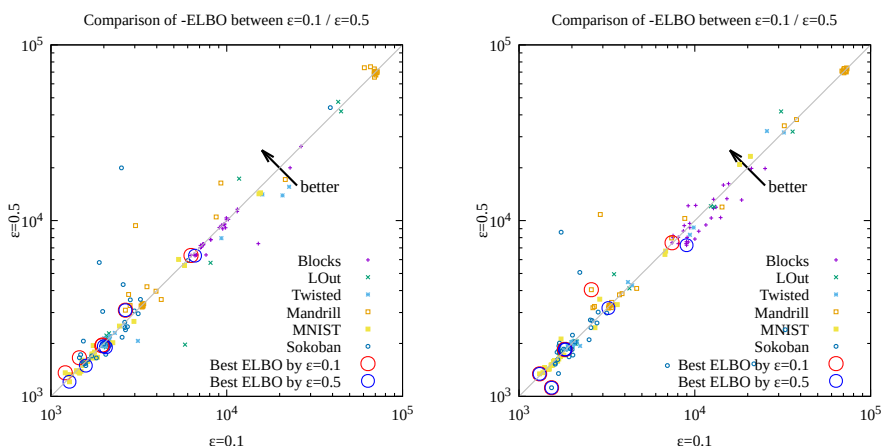


Figure 10.3: (Left: AMA_3^+ , Right: AMA_4^+ .) Comparing the effect of the prior distribution on -ELBO. Each point represents a pair of configurations with the same F, β_1, β_3 , where the x -axis represents a result from $\epsilon = 0.1$, while the y -axis represents a result from $\epsilon = 0.5$. As expected, we do not see a significant difference of ELBO between these two configurations.

domain	kltune $\epsilon = 0.1$				notune $\epsilon = 0.1, \beta_{1..3} = 1$				default $\epsilon = 0.5$			
	-ELBO	β_1	β_3	F	-ELBO	β_1	β_3	F	-ELBO	β_1	β_3	F
AMA ₃ ⁺												
Blocks	6.26E+03	10	1	300	6.55E+03	1	1	300	<i>6.32E+03</i>	1	100	300
LOut	1.97E+03	1	1000	50	2.07E+03	1	1	100	1.90E+03	1	1000	300
Twisted	1.95E+03	10	1000	50	2.03E+03	1	1	50	1.93E+03	10	1000	300
Mandrill	2.65E+03	10	10	300	3.39E+03	1	1	100	<i>3.09E+03</i>	10	10	300
MNIST	1.21E+03	10	10	300	1.41E+03	1	1	100	<i>1.21E+03</i>	1	10	300
Sokoban	1.45E+03	10	1	50	1.60E+03	1	1	50	1.50E+03	1	1	50
AMA ₄ ⁺												
Blocks	7.43E+03	1	1	100	7.61E+03	1	1	100	7.25E+03	10	10	100
LOut	1.82E+03	10	10	300	1.89E+03	1	1	300	<i>1.85E+03</i>	10	1	300
Twisted	1.82E+03	10	1	300	1.93E+03	1	1	300	<i>1.85E+03</i>	10	1	300
Mandrill	2.58E+03	10	100	300	3.26E+03	1	1	100	3.18E+03	10	100	100
MNIST	1.31E+03	10	1	300	1.50E+03	1	1	300	1.34E+03	10	1	300
Sokoban	1.53E+03	10	10	300	1.55E+03	1	1	100	1.12E+03	10	10	300

Table 10.3: Best negative ELBO (lower the better) of three different configurations (kltune, notune, default) of AMA₃⁺ and AMA₄⁺ models and the hyperparameters which achieved it. The best results among three configurations are highlighted in **bold**. Also, the better result among AMA₃⁺ and AMA₄⁺ are highlighted in *italic*. Overall, in terms of -ELBO, which represents training accuracy, **kltune is better than notune, kltune and default are comparable, and AMA₄⁺ and AMA₃⁺ are comparable.**

$p(\mathbf{z}^0) = \text{Bernoulli}(\epsilon = 0.5)$ configuration (“default”). We focus on the stability of a latent vector $\mathbf{z}^{i,0}$ on perturbed inputs, whose role in a symbol grounding process was discussed in Section 5.2.

The stability is measured by the *state variance* for the noisy input, i.e., the variance of the latent vectors $\mathbf{z}^{i,0} = \text{BC}_\tau(\text{ENCODE}(\mathbf{x}^{i,0} + \mathbf{n}))$ where \mathbf{n} is a noise following a Gaussian distribution $\mathcal{N}(\mu = 0, \sigma = 0.3)$. We compute the variance by iterating over 10 random vectors, then average the results over F bits in the latent space and the dataset index i . Formally,

$$\text{State Variance} = \mathbb{E}_{f \in 0..F} \mathbb{E}_i \text{Var}_{j \in 0..10} [\text{BC}_\tau(\text{ENCODE}(\mathbf{x}^{i,0} + \mathbf{n}^j))_f].$$

As we discussed in Section 5.2 about the potential cause of unstable symbols, this value is likely to increase when the latent space has an excessive capacity (larger F) and could also be affected by other hyperparameters. Therefore, we made a scatter plot where each point represents a result of evaluating $\epsilon = 0.1$ and $\epsilon = 0.5$ configurations with the same hyperparameter. In Figure 10.5, we observed that the network trained with Bernoulli($\epsilon = 0.1$) has a lower state variance, confirming that the proposed prior makes the resulting symbolic propositional states more stable against aleatoric uncertainty.

Another related metric we can evaluate is how many “effective bits” each model uses in the latent space. An effective bit is defined as follows: For each bit f in the latent vector $\mathbf{z}^{i,0} = \text{BC}_\tau(\text{ENCODE}(\mathbf{x}^{i,0}))$ where $\mathbf{x}^{i,0}$ is from a test dataset, we check if $\mathbf{z}_f^{i,0}$ ever changes its value when we iterated across different i in the dataset. If it changes, it is an “effective bit.” Otherwise, the bit

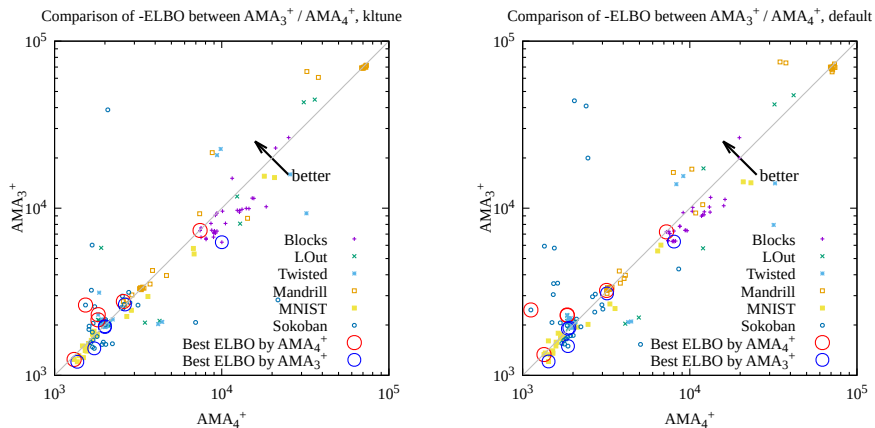


Figure 10.4: Comparing the accuracy (-ELBO) between AMA_3^+ and AMA_4^+ (Left: $\epsilon = 0.1$, Right: $\epsilon = 0.5$). Each point represents a pair of configurations with the same hyperparameter tuple (F, β_1, β_3) , where the x -axis represents a result from AMA_3^+ , while the y -axis represents a result from AMA_4^+ . We do not see a significant difference of ELBO between them.

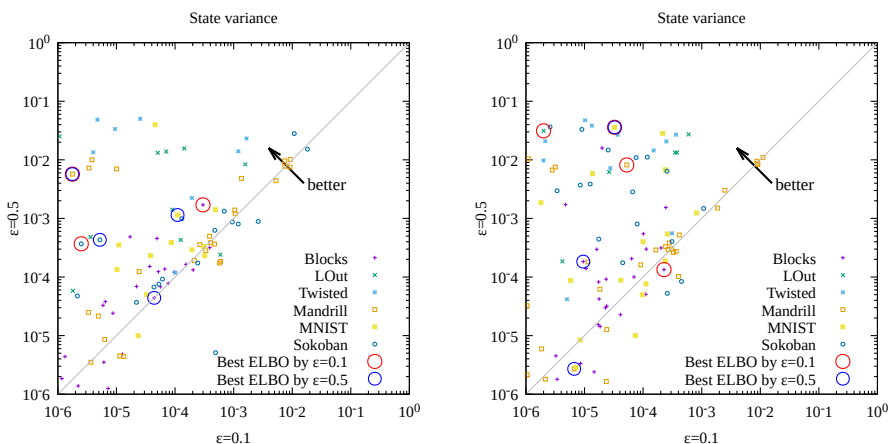


Figure 10.5: Each subfigure corresponds to AMA_3^+ (left) and AMA_4^+ (right). Results showing the variance of the latent vectors when the input image is corrupted by Gaussian noise, where the x -axis corresponds to the value in $\epsilon = 0.1$ configuration and the y -axis shows the value in $\epsilon = 0.5$ configuration, while the rest of the hyperparameters are the same. Additionally, we highlighted each configuration in a circle when it achieved the best ELBO in each domain.

is a constant and is not used for encoding the information in an image. In Figure 10.6, we observed that our proposed configuration (“kltune”) tends to have fewer effective bits; thus each bit of the latent vectors changes its value less frequently across the dataset.

Furthermore, we confirmed that those static bits tend to be 0 rather than 1. We extracted the bits whose values are always 0 regardless of the input by iterating over the test dataset. Figure 10.7 shows that the modified prior induces more constant 0 bits in the latent space.

Note that better stability does not by itself imply the overall “quality” of the resulting network and should always be considered in conjunction with other metrics. For example, a failed training result that exhibits *posterior collapse* may have a latent space where all bits are always 0, and may fail to reconstruct an input image at all. While such a latent space shows a perfect “stability,” the network is not useful for planning because it does not map the input image to an informative latent vector.

10.5 Evaluating the Accuracy of Action Models: Successor State Prediction

We next measured the accuracy of the progression prediction (effect). In this experiment, we measured $\mathbb{E}_{i,f} |z_f^{i,1} - z_f^{i,2}|$, i.e., the absolute error between $z^{i,1}$ (encoded directly from the image) and $z^{i,2}$ (predicted by the BTL mechanism) averaged over latent bits and the test dataset. We show the absolute error instead of the corresponding KL divergence loss $D_{\text{KL}}(q(z^{i,1} | \mathbf{x}^{i,1}) \| p(z^{i,2} | z^{i,0}, \mathbf{a}^i))$ which was used for training because the average absolute error gives an intuitive understanding of “how often it predicts a wrong bit.”

Figure 10.8 shows these metrics for the best hyperparameter configuration that achieved the best ELBO. For those configurations, AMA_3^+ and AMA_4^+ models obtained comparable accuracy. If we compare AMA_3^+ and AMA_4^+ having the same hyperparameter, the result seems slightly in favor of AMA_4^+ .

Comparing the results of different priors, $\epsilon = 0.1$ configurations (Figure 10.8, left) tend to have better successor prediction accuracy than $\epsilon = 0.5$ (Figure 10.8, right). This is presumably because the state instability causes a non-deterministic state transition that cannot be captured well by the BTL mechanism because BTL assumes STRIPS-compatible state transitions.

Again, note that neither the best ELBO or the best successor prediction accuracy alone determine the likelihood of success in solving visual planning problems. For example, in a collapsed latent space, successor prediction is quite easy because not a single latent space bit will change its value. Also, even if the overall accuracy (ELBO) is good, the planning is not likely to succeed if the network sacrifices successor prediction accuracy for reconstruction accuracy.

10.6 Violation of Monotonicity in BTL

As discussed in Section 8.3, Batch Normalization used as a “continualizer” m in BTL may violate the monotonicity of m . This causes an action with *XOR semantics* which always flips the value, which is not directly expressible in the STRIPS semantics.

To quantify the harmful effect of these bits with XOR semantics, we first counted the number of such violations averaged over all actions. As we see in Table 10.4, the number of such bits is small compared to the entire representation size. We also compared the number of actions before and after compiling these XOR semantics away. While the increase of STRIPS action is exponential to the number of violations in the worst case, the empirical increase tends to be marginal because the number of violations was small for each action, except for one instance of Sokoban in AMA_4^+ .

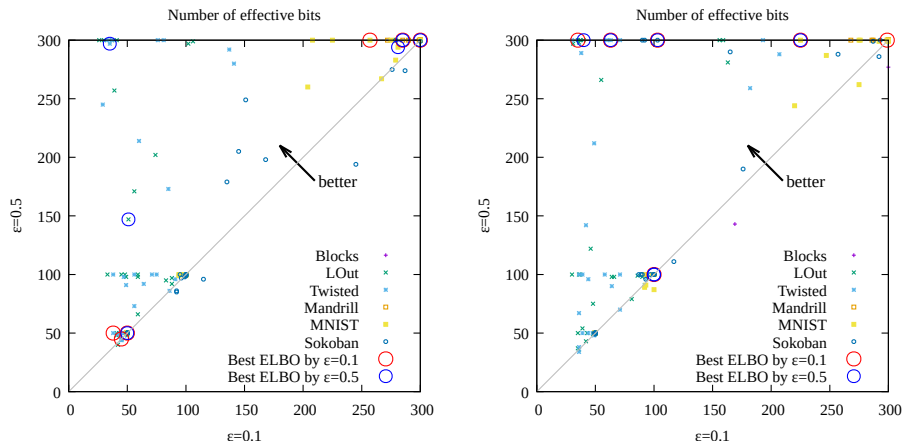


Figure 10.6: Each subfigure corresponds to AMA_3^+ (left) and AMA_4^+ (right). Results showing the number of effective bits in the test dataset where the x -axis corresponds to the value in $\epsilon = 0.1$ configuration and the y -axis shows the value in $\epsilon = 0.5$ configuration, while the rest of the hyperparameters are the same. Additionally, we highlighted each configuration in a circle when it achieved the best ELBO in each domain. $\epsilon = 0.1$ has a significantly fewer number of effective bits.

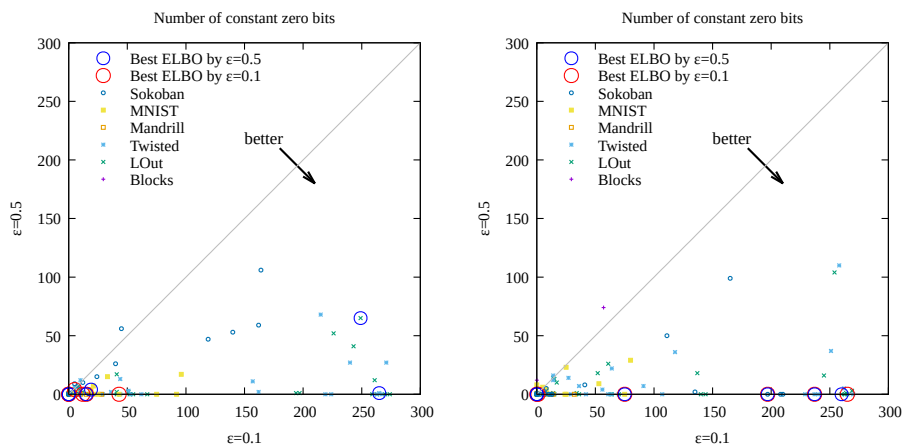


Figure 10.7: Each subfigure corresponds to AMA_3^+ (left) and AMA_4^+ (right). Results showing the number of constant zero bits in the test dataset where the x -axis corresponds to the value in $\epsilon = 0.1$ configuration and the y -axis shows the value in $\epsilon = 0.5$ configuration, while the rest of the hyperparameters are the same. Additionally, we highlighted each configuration in a circle when it achieved the best ELBO in each domain. $\epsilon = 0.1$ has significantly more constant bits.

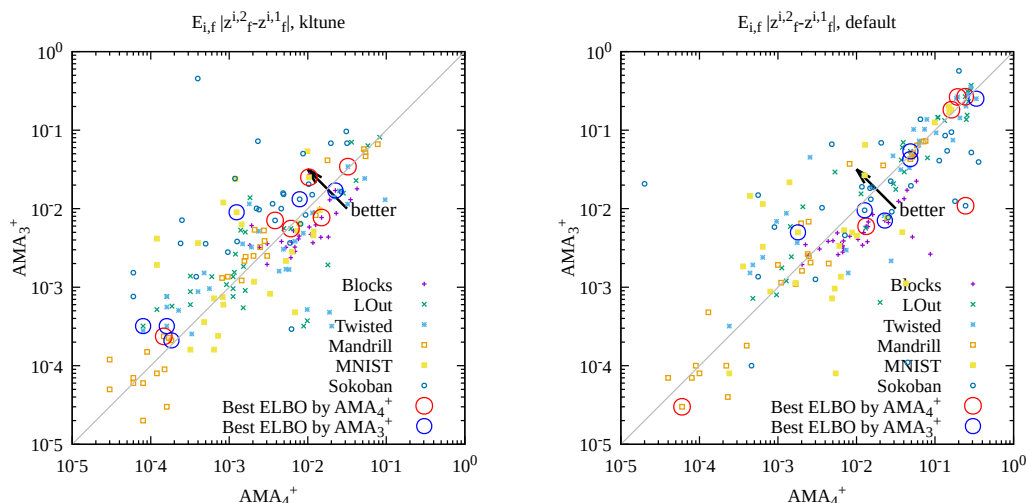


Figure 10.8: Both axes represent the absolute error $|z^{i,2} - z^{i,1}|$ between $z^{i,1}$ obtained from the encoder distribution $q(z^{i,1}|\mathbf{x}^{i,1})$ and $z^{i,2}$ obtained from the AAE distribution $p(z^{i,2}|z^{i,0}, \mathbf{a}^i)$ (applying an action to the current state using a BTL mechanism). The values are averaged over the test dataset index i and the latent dimension f . Each point (x, y) represents a pair of $(\text{AMA}_4^+, \text{AMA}_3^+)$ configurations with the same hyperparameter tuple (F, β_1, β_3) , where the x -axis represents a result from AMA_4^+ . The left figure is a result from networks trained with $\epsilon = 0.1$, and the right figure is a result with $\epsilon = 0.5$. With $\epsilon = 0.1$, AMA_4^+ tends to have a slightly better successor prediction accuracy. With $\epsilon = 0.5$, we do not observe a significant difference between AMA_3^+ and AMA_4^+ . Comparing $\epsilon = 0.1$ (left) and $\epsilon = 0.5$ (right), we noticed that the entire point cloud is moved toward the top right in $\epsilon = 0.5$, indicating that the networks trained with $\epsilon = 0.1$ tend to be more accurate.

While the Sokoban result from the best ELBO hyperparameter of AMA_4^+ resulted in a relatively large number of monotonicity violations, this is not always the case across hyperparameters. Table 10.4 (Right) plots ELBO and the compiled number of actions, A_2 , which varies significantly between hyperparameters despite having a similar ELBO. This is another case demonstrating that the ELBO alone does not necessarily characterize the detrimental effect on search performance.

Note that the increase depends on whether the violations are concentrated in one action. For example, if 5 violations are found in one action, the action is compiled into $2^5 = 32$ variants. However, if 1 violation is found in 5 separate actions, it only adds 5 more actions.

10.7 Summary Statistics of the Learned PDDL Representation

Finally, Table 10.5 summarizes the statistics of the PDDL obtained from our final model, AMA_4^+ , with Bernoulli($\epsilon = 0.1$) prior. This table shows the number of effective bits (Section 10.4), the number of compiled actions (Section 10.6), average state differences between current and successor states, and the average number of delete/add effects, positive/negative preconditions.

domain	XOR bits		F	Actions	
	effects	precondition		A_1	A_2
AMA ₃ ⁺					
Blocks	0.00	-	300	1908	1917
LOut	0.00	-	50	877	877
Twisted	0.00	-	50	919	919
Mandrill	0.00	-	300	3265	3265
MNIST	0.09	-	300	974	1065
Sokoban	0.29	-	50	737	1028
AMA ₄ ⁺					
Blocks	0.00	0.02	100	2317	2364
LOut	0.00	0.00	300	941	941
Twisted	0.00	0.00	300	1069	1069
Mandrill	0.00	0.00	300	3450	3469
MNIST	0.19	0.10	300	1263	1664
Sokoban	2.70	2.81	300	1115	47451

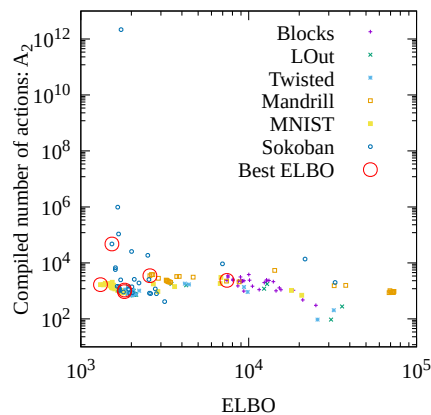


Table 10.4: The XOR columns show the number of bits with XOR semantics averaged across actions for a hyperparameter configuration that achieved the best ELBO in each domain. A_1 column shows the number of actions before compiling XOR bits away, and A_2 shows the number after the compilation. Note that $A_1 \leq A = 6000$, where A is the hyperparameter that specifies the length of one-hot vector \mathbf{a}^i , thus serves as the maximum number of action labels that can be learned by the neural network. In AMA₃⁺, we observed that the number of monotonicity violations is small, and thus the empirical increase of the number of actions due to the compilation is marginal. In AMA₄⁺, this is also the case except for the Sokoban domain that resulted in a larger number of XOR bits. However, note that this is not always the case in all hyperparameters; In the figure on the right, we plotted the number of compiled actions in the y -axis against the ELBO in the x -axis. The number of compiled actions varies significantly for a similar ELBO.

State differences are the average number of different bits between the current and the successor states over the dataset, i.e., $\mathbb{E}_{i,f} |z_f^{i,0} - z_f^{i,1}|$. It explains why the PDDL files generally contain larger numbers of delete effects and negative preconditions: The system recognized static predicates that take false values and added it to the preconditions/effects.

domain	Propositions (effective bits)	Actions (xor-compiled)	State differences	Effects		Preconditions	
				ADD	DEL	POS	NEG
Blocks	100	2366	23.88	20.62	21.44	19.96	22.04
LOut	35	941	0.86	2.67	146.50	2.74	146.49
Twisted	63	1069	2.08	10.03	145.35	10.06	145.33
Mandrill	299	3469	8.90	4.79	16.44	4.82	16.43
MNIST	225	1665	7.46	5.85	53.08	5.83	53.63
Sokoban	103	47495	2.84	5.00	219.29	4.93	220.03

Table 10.5: A summary of the statistics of the PDDL files obtained from the AMA_4^+ model for each domain (hyperparameter with the best ELBO and with Bernoulli($\epsilon = 0.1$) prior).

11. Planning Performance

We next evaluate the PDDL models produced by the networks using randomly generated problem instances for each domain. The objectives of the comparisons made in this section are threefold:

1. Verifying the effectiveness of the preconditions generated by Bidirectional Cube-Space AE (AMA_4^+) over the ad-hoc preconditions generated by Cube-Space AE (AMA_3^+).
2. Verifying the effect of state-of-the-art heuristics on the search performance in the latent space generated by $\text{AMA}_3^+/\text{AMA}_4^+$.
3. Verifying the effect of improved symbol stability on the planning performance,

11.1 Experimental Setup

We ran the off-the-shelf planner Fast Downward on the PDDL generated by our system. To evaluate the quality of the domains, we similarly generate multiple problems by encoding an initial and a goal state image ($\mathbf{x}^I, \mathbf{x}^G$) which are randomly generated using domain-specific code. The images are normalized to mean-0, variance-1 using the statistics of the training dataset, i.e., they are shifted and scaled by the mean and the variance of the training dataset. Each normalized image is encoded by the encoder, then is converted into a PDDL encoding of the initial state and the goal state. Formally, when $\mathbf{x}^I \in [0, 255]^{H,W,C}$ is an initial state image of width H , width W and color channel $C \in \{1, 3\}$, and $\boldsymbol{\mu} = \mathbb{E}_{i,j}[\mathbf{x}^{i,j}]$ and $\boldsymbol{\sigma}^2 = \text{Var}_{i,j}[\mathbf{x}^{i,j}]$ are the mean and the variance of the training dataset, the latent propositional initial state vector is $\mathbf{z}^I = \text{BC}_\tau(\text{ENCODE}(\frac{\mathbf{x}^I - \boldsymbol{\mu}}{\boldsymbol{\sigma}}))$, just as all training is performed on the normalized dataset.

In 8-Puzzle, 15-Puzzle, LightsOut and Twisted, we used a randomly sampled initial state image \mathbf{x}^I and a fixed complete goal state \mathbf{x}^G . The goal states are those states that are normally considered to solve the puzzles, e.g., for a Mandrill 15-Puzzle, the goal state is a state where all tiles are

ordered correctly so that the whole configuration recovers the original photograph of a Mandrill. Initial states x^I are sampled from the frontier of a Dijkstra search, which was run backward from the goal. The search stops when it exhausts the plateau at a specific g -value, at which point the shortest path length from the goal state is obtained for each of those states. We then randomly select a state from this plateau. In each domain, we generated 20 instances for $g = 7$ and $g = 14$, resulting in 40 instances per domain.⁴

In 8-Puzzle, we also added instances whose goal states are randomly generated using domain-specific code, and the initial states are sampled with $g = 7$ and $g = 14$ thresholds. The purpose of evaluating them is to see whether the system is generalized over the goals.

In Sokoban, the goal-based sampling approach described above does not work because the problem is not reversible. We instead sampled goal states from the initial state of p006-microban-sequential instance with $g = 7$ and $g = 14$ thresholds. These goal states do not correspond to the goal states of p006-microban-sequential which solve the puzzle.

In Blocksworld, we randomly generated an initial state and performed a 7-step or 14-step random walk to generate a goal state. The number of steps does not correspond to the optimal plan length.

We tested A^* with blind heuristics, Landmark-Cut (Helmert & Domshlak, 2009, LMcut), Merge-and-Shrink (M&S) (Helmert et al., 2014), and the first iteration of the satisficing planner LAMA (Richter & Westphal, 2010). Experiments are run with the 10 minutes time limit and 8GB memory limit. We tested the PDDL generated by AMA_3^+ and AMA_4^+ models, each trained with different latent space priors Bernoulli($\epsilon = 0.5$) and Bernoulli($\epsilon = 0.1$).

We counted the number of instances in which a solution was **found**. However, the solution found by our system is not always correct because the correctness of the plan is guaranteed only with respect to the PDDL model — if the symbolic mapping produced by the neural network is incorrect, the resulting visualization of the symbolic plan does not correspond to a realistically correct visual plan. To address this issue, we wrote a domain-specific plan validator for each domain which heuristically examines the visualized results and we counted the number of plans which are empirically **valid**. Finally, we are also interested in how often the solution is **optimal** out of the valid plans. In domains where we generated the instances with Dijkstra-based sampling method, we compared the solution length with the g threshold which was used to generate the instance.

In the comparisons made in the following sections, we select the hyperparameter configuration which maximizes the metric of interest (e.g., the number of valid solutions or the number of optimal solutions). This selection is performed on a per domain basis.

11.2 Bidirectional Models Outperform Unidirectional Models

We evaluated each hyperparameter configuration of AMA_3^+ and AMA_4^+ on the problem instances we generated. We verified the solutions and Table 11.1 shows the numbers obtained from configurations that achieved the highest number of valid solutions. The result shows that AMA_4^+ outperforms AMA_3^+ , indicating that the precondition generated by the complete state regression in AMA_4^+ is more effective than those generated by an ad-hoc method in AMA_3^+ . All results are based on Bernoulli($\epsilon = 0.1$) prior for the latent representation (i.e., ϵ is not considered as a hyperparameter).

4. g is the measure of distance from the initial state to a node in the search graph.

In particular, we note the significant jump in (1) coverage on arguably the most visually challenging domains (Blocks and Sokoban); and (2) the ratio of valid plans that are also optimal (AMA_4^+ struggled with only MNIST in this regard).

	Blind			LAMA			LMCut			M&S		
	found	valid	optimal	found	valid	optimal	found	valid	optimal	found	valid	optimal
AMA_3^+												
Blocks	1	1	-	1	1	-	1	1	-	1	1	-
LOut	40	40	35	40	39	0	20	20	18	40	40	35
Twisted	40	40	36	40	40	0	23	20	18	40	40	16
Mandrill	38	38	20	38	38	9	38	38	18	40	40	26
MNIST	39	39	5	39	39	4	39	39	5	39	39	5
Random MNIST	39	39	4	39	36	4	39	39	4	39	39	4
Sokoban	14	14	12	14	14	12	14	14	12	14	14	12
Total	211	211	112	211	207	29	174	171	75	213	213	98
AMA_4^+												
Blocks	33	32	-	19	19	-	34	34	-	34	33	-
LOut	40	40	40	40	40	1	20	20	20	40	40	40
Twisted	40	40	40	40	40	1	20	20	20	40	40	40
Mandrill	25	23	23	20	15	11	38	30	30	40	32	32
MNIST	40	39	6	35	35	16	40	39	6	40	39	6
Random MNIST	36	34	11	32	32	5	35	32	11	36	33	11
Sokoban	40	39	38	40	31	21	40	38	37	40	39	38
Total	254	247	158	226	212	55	227	213	124	270	256	167

Table 11.1: Planning results. We highlight the numbers in **bold** when AMA_4^+ and AMA_3^+ outperforms each other, except ties. The results indicate that the regression-based precondition generation method in AMA_4^+ is effective.

11.3 Classical Planning Heuristics Reduce the Search Effort in the Latent Space

Domain-independent heuristics in the planning literature are traditionally evaluated on hand-coded benchmarks, e.g., International Planning Competition instances (McDermott, 2000), which contain a wide variety of domains and are regarded as a representative subset of the real-world tasks. As a result, they assume a certain general class of structures that are commonly found in domains made by humans, such as serializable subgoals and abstraction (Korf, 1985). However, the latent space domains derived from raw pixels and neural networks may have completely different characteristics that could render them useless. In fact, blind heuristic can sometimes outperform well-known heuristics on some IPC domains (e.g., floortile) specifically designed to defeat their efficacy (Edelkamp, 2012). Also, even though the image-based 8-puzzle domains we evaluate *correspond* to the 8-puzzle domain used in the combinatorial search and classical planning literature, the latent representations of the 8-puzzle states may be unlike any standard PDDL encoding of the 8-puzzle written by humans. While the symbolic representation acquired by Latplan captures the state space graph of the domain, the propositions in the latent space do not necessarily correspond to conceptual propositions in a natural, hand-coded PDDL model. Thus, there is little *a priori* reason to believe that the standard heuristics will be effective until we evaluate them.

We evaluated the effect of state-of-the-art heuristic functions in the latent space generated by our neural networks. Figure 11.1 compares node evaluations and search time between blind search

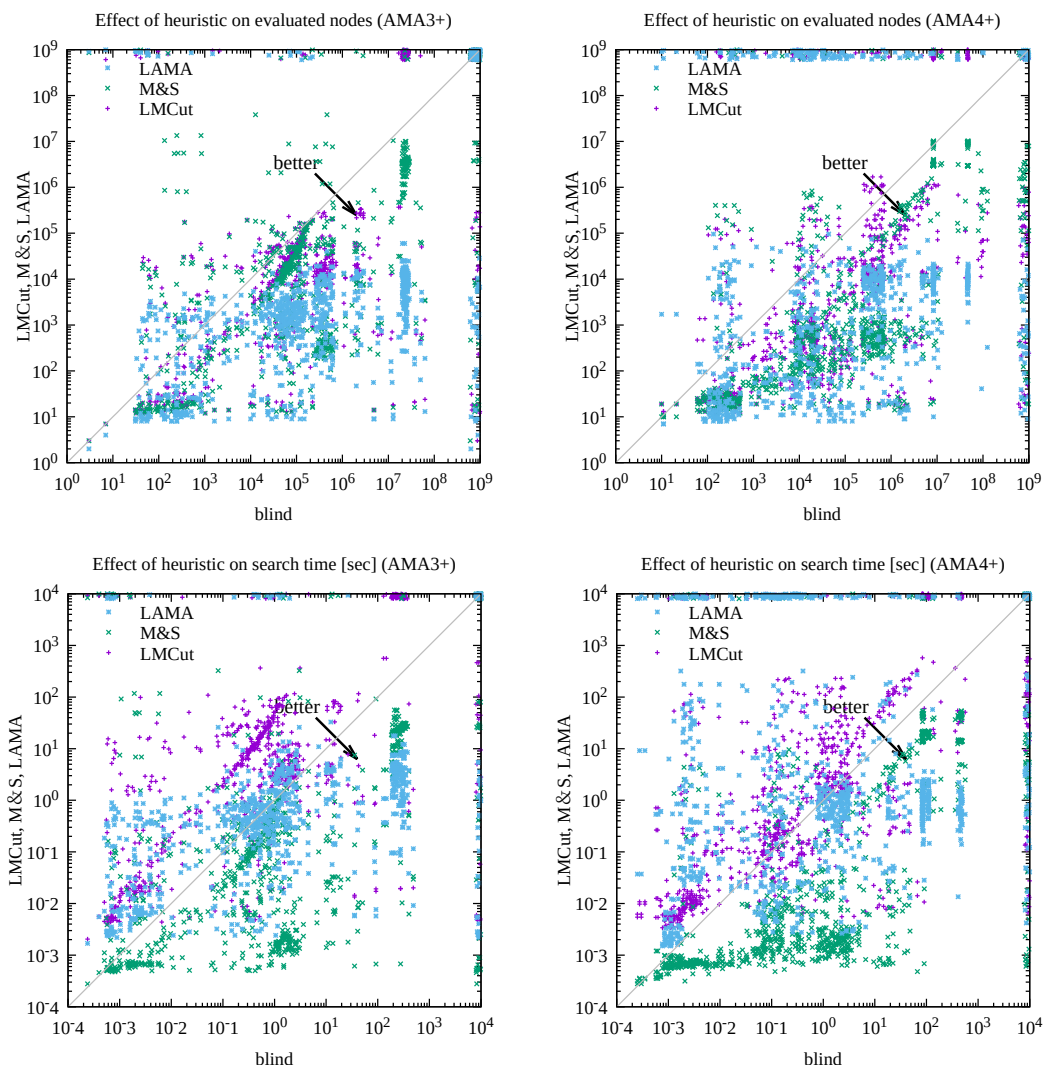


Figure 11.1: The left column shows AMA_3^+ and the right column shows AMA_4^+ . The top row shows the number of node evaluations for h^{blind} heuristics (x -axis) and h^{LMcut} , $h^{M\&S}$ heuristics as well as LAMA (y -axis), for instances solved and validated by both. Unsolved instances are placed on the border. State-of-the-Art heuristics developed in classical planning literature significantly reduces the search effort. The bottom row shows the search time. M&S still outperforms blind. Due to the higher cost-per-node, LMCut tends to consume more search time compared to blind. Similarly, LAMA seems to struggle in easy instances. However, it tends to behave better in more difficult instances. For a plot colored by domains rather than heuristics, see appendix (Section J).

and LMcut, M&S heuristics as well as LAMA. For each domain, we select the hyperparameter configuration with the largest number of valid solutions. Results are based on Bernoulli($\epsilon = 0.1$).

Node evaluation plots confirm the overall reduction in the search effort. In contrast, the results on search time are mixed. M&S reduces the search time by an order of magnitude, while LMcut tends to be slow due to the high evaluation cost per node. LAMA tends to be faster than a blind search in more difficult instances, though a blind search is faster in easier instances that require less search (< 10 seconds).

Despite the increased runtime, the reduced node evaluation strongly indicates that either the models we are inferring contain similar properties as the human-designed PDDL models, or that powerful, domain-independent planning heuristics can induce effective search guidance even in less human-like PDDL models.

11.4 Stable Symbols Contributes to the Better Performance

Section 5.2 discussed three harmful effects of unstable symbols: **(1)** disconnected search space, **(2)** having many variations of latent states for a single real-world state, **(3)** hyperparameter tuning for improving symbol stability. We already evaluated the issue **(3)** in the previous sections, where we verified that Bernoulli($\epsilon = 0.1$) automatically finds a latent space with smaller effective bits when a large F is given. In this section, we proceed to verify the effect of Bernoulli($\epsilon = 0.1$) on the first and the second issue.

Issue **(1)** (disconnected search space) would result in more *exhausted instances*, i.e., the number of instances where Fast Downward exhausted the state space without finding a solution. It would also result in a lower success ratio, given that we set a certain time limit on the planner and not all unsolvable instances are detected.

Issue **(2)** (many variations of latent states) would result in more *suboptimal visualized plans that are optimal in the latent space / PDDL encoding*. There are several possible reasons that a visualized solution becomes suboptimal.

The first and most obvious one is that satisficing planner configuration (LAMA) could return a suboptimal latent space solution, which results in a suboptimal visual solution. Since this is trivial, we do not include LAMA during optimality evaluation.

Second, if a certain action is not correctly learned by the neural network (e.g., missing or having unnecessary preconditions and effects), the optimal solution with regard to the PDDL model may not be in fact optimal when visualized: The planner may not be able to take certain otherwise valid transitions, thus resulting in a longer solution; Or an action effect may alter the values of some bits that do not affect the image reconstruction, and the planner tries to fix it with additional actions. As discussed in Example 4, accurate successor prediction is a form of symbol stability. In Section 10.5, we already observed that $\epsilon = 0.1$ tend to predict successors better than $\epsilon = 0.5$ due to its stable state representation.

The third case is unstable init/goal states: When symbol instability is significant, a noisy goal state image x^G may map to multiple different latent states z^G, z'^G depending on the input noise. The encoded initial state z^I may be closer to z'^G than to z^G specified in PDDL, thus reaching z^G may require an extra number of steps, resulting in a suboptimal visual solution. The same mechanism applies to the initial state too.

To investigate the effect of symbol stability, we additionally evaluated our system on a set of initial/goal state images that are corrupted by a Gaussian noise $\mathcal{N}(0, 1)$. We added the noise to

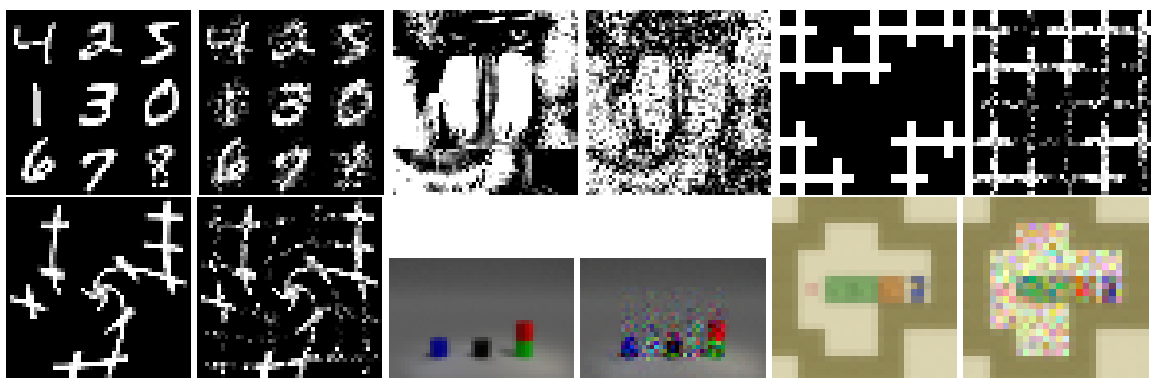


Figure 11.2: Examples of initial state images before and after being corrupted by Gaussian noise. The amount of noise is noticeable but is not as strong as making it difficult to understand the scene. Note that unlike Figure 10.1, we used low-resolution images for Blocksworld and Sokoban that were used for training and testing.

the images that are already normalized to mean-0, variance-1 using the statistics from the training dataset (see Section 11.1). As a result, when de-normalized for visualization in Figure 11.2, the noise appears missing in the static regions, which have little variations across the dataset.

We first investigated the effect of (1). We evaluated AMA_3^+ and AMA_4^+ trained with different Bernoulli(ϵ) priors. Figure 11.3 plots the number of instances where Fast Downward exhaustively searched the state space and failed to find a solution. The PDDL is generated by AMA_4^+ , and we compare the numbers between Bernoulli($\epsilon = 0.1$) prior and the Bernoulli($\epsilon = 0.5$) prior. We do not show AMA_3^+ results because its preconditions are less accurate and ad-hoc, and the precondition accuracy is crucial in this evaluation. The figure clearly shows that more instances have disconnected search spaces when encoded by $\epsilon = 0.5$ prior.

Next, to address the issue of insufficient runtime to prove the unsolvability, Table 11.2 shows the number of valid solutions found. For each domain, heuristic, and $\epsilon \in \{0.1, 0.5\}$, we selected the hyperparameter which resulted in the largest number of valid solutions. $\epsilon = 0.1$ configuration outperforms $\epsilon = 0.5$ on both clean and corrupted input images.

Recall that networks trained with different Bernoulli(ϵ) priors had comparable results in terms of ELBO — thus, ELBO (accuracy) alone is not the sole factor that determines the likelihood of success in finding a solution. In fact, $\epsilon = 0.1$ configuration solves more instances than $\epsilon = 0.5$ configuration does in LightsOut (AMA_3^+), Blocks, and Sokoban (AMA_4^+) where $\epsilon = 0.5$ had the better ELBO (Table 10.3).

We next evaluate the effect of (2), predicted to result in more suboptimal plans. For each domain, heuristics, and prior distribution ($\epsilon \in \{0.1, 0.5\}$), we selected the hyperparameter configuration which found the most valid optimal solutions. Note that the absolute number of optimal plans is not a proper metric for evaluating the effect of (1) because if a configuration finds fewer valid plans, the number of optimal plans found is also expected to decrease. Instead, we measure the ratio between the number of optimal plans over the number of valid plans. The results are shown in Table 11.3. AMA_4^+ results show that $\epsilon = 0.1$ is less affected by the noise than $\epsilon = 0.5$ is. While AMA_3^+ results are mixed, we attribute the mixed results to the lower optimality ratio of $\epsilon = 0.5$ in MNIST (around

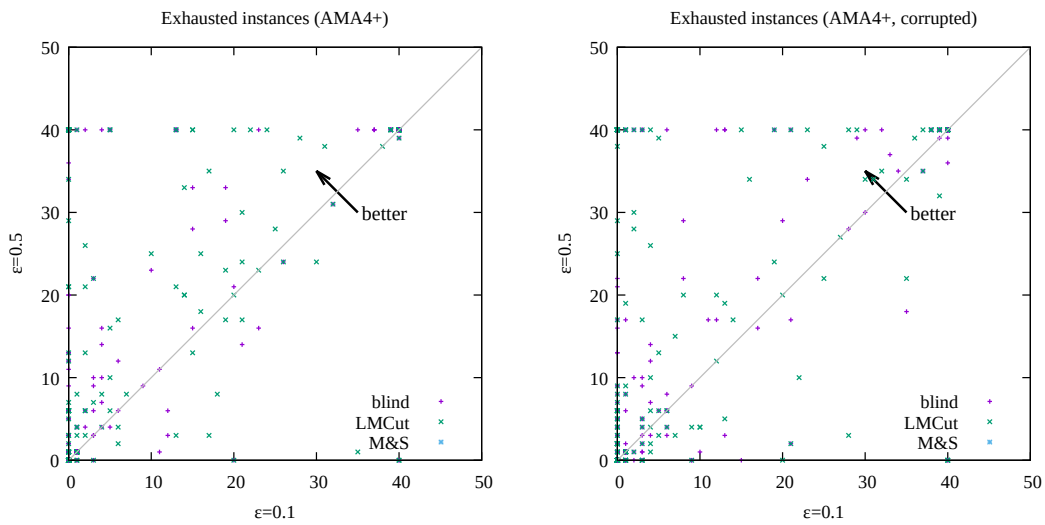


Figure 11.3: The left figure shows the noiseless instances, and the right figure shows the result from corrupted input images. Each point represents a tuple $(\text{domain}, \text{heuristic}, F, \beta_1, \beta_3)$. We evaluated different heuristics because they have different performances in terms of exhaustively searching the space within the time limit. For each point, each axis represents the number of unsolvable instances among the 40 instances in the domain, which are discovered by Fast Downward by exhaustively searching the state space. x -axis represents AMA_4^+ networks with $\epsilon = 0.1$ prior, and y -axis the $\epsilon = 0.5$ prior. $\epsilon = 0.1$ resulted in significantly fewer unsolvable instances. For a plot colored by domains rather than heuristics, see appendix (Section K).

$\epsilon =$	Clean input images						Corrupted input images									
	Blind		LAMA		LMCut		M&S		Blind		LAMA		LMCut		M&S	
	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5
	AMA ₃ ⁺															
Blocks	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0
LOut	40	27	39	40	20	20	40	28	40	30	39	33	20	19	40	31
Twisted	40	40	40	40	20	20	40	40	40	40	40	40	20	20	40	40
Mandrill	38	39	38	40	38	38	40	40	39	37	39	33	37	35	39	38
MNIST	39	40	39	40	39	40	39	40	36	40	36	40	36	40	36	40
Random MNIST	39	38	36	38	39	38	39	38	36	36	33	36	36	36	36	36
Sokoban	14	1	14	1	14	1	14	1	13	1	13	1	13	1	13	1
total	211	186	207	200	171	158	213	188	205	184	201	183	163	151	205	186
	AMA ₄ ⁺															
Blocks	32	31	19	23	34	29	33	31	25	27	10	12	24	26	24	27
LOut	40	40	40	40	20	20	40	40	40	40	40	40	20	23	40	40
Twisted	40	40	40	40	20	20	40	40	40	40	40	40	20	20	40	40
Mandrill	23	24	15	10	30	33	32	36	19	17	7	10	27	28	26	30
MNIST	39	37	35	26	39	36	39	34	36	37	26	21	36	36	36	34
Random MNIST	34	37	32	19	32	35	33	35	34	36	28	19	32	34	33	34
Sokoban	39	33	31	34	38	31	39	32	39	32	31	33	38	31	39	31
total	247	242	212	192	213	204	256	248	233	229	182	175	197	198	238	236

Table 11.2: The number of instances where a valid solution is found. Each number is the largest number among different hyperparameter configurations (F, β_1, β_3) in a single domain. Networks trained with Bernoulli($\epsilon = 0.1$) latent space prior outperformed the networks trained with Bernoulli($\epsilon = 0.5$). Numbers are highlighted in **bold** for the better latent space prior, except ties.

$\epsilon =$	Clean input images						Corrupted input images						Difference due to noise					
	Blind		LMCut		M&S		Blind		LMCut		M&S		Blind		LMCut		M&S	
	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5
	AMA ₃ ⁺																	
LOut	.88	.93	.95	1.00	.88	.93	.88	.90	.95	1.00	.88	.90	.000	-.026	.000	.000	.000	-.025
Twisted	.90	.88	.90	.85	.90	.88	.90	.88	.90	.84	.90	.88	.000	.000	.000	-.008	.000	.000
Mandrill	.81	.77	.83	.79	.79	.75	.79	.70	.84	.74	.79	.68	-.016	-.067	.005	-.047	.005	-.066
MNIST	.43	.20	.48	.20	.43	.20	.43	.21	.48	.21	.43	.21	.000	.007	.000	.007	.000	.007
Random MNIST	.24	.28	.24	.28	.24	.28	.29	.28	.28	.28	.28	.28	.056	.002	.045	.002	.045	.002
Sokoban	.86	1.00	.86	1.00	.86	1.00	.85	1.00	.85	1.00	.85	1.00	-.011	.000	-.011	.000	-.011	.000
	AMA ₄ ⁺																	
LOut	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	.000	.000	.000	.000	.000	.000
Twisted	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	.000	.000	.000	.000	.000	.000
Mandrill	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	.96	1.00	1.00	1.00	.000	.000	-.037	.000	.000	.000
MNIST	1.00	.78	1.00	.81	1.00	.79	1.00	.78	1.00	.81	1.00	.79	.000	.000	.000	.000	.000	.000
Random MNIST	.91	.92	.94	.91	.94	.91	.95	.92	1.00	.91	.95	.91	.041	-.002	.065	-.003	.017	-.003
Sokoban	.97	.79	.97	.80	.97	.78	.97	.78	.97	.77	.97	.77	.000	-.007	.000	-.026	.000	-.007

Table 11.3: We calculated the rate of the number of optimal solutions found over the number of valid solutions found. We then obtained the rate difference due to the input noise and compared the results between $\epsilon = 0.1$ and $\epsilon = 0.5$. The results suggest that the input noise more negatively affects the optimality rate of $\epsilon = 0.5$ than that of $\epsilon = 0.1$.

0.2, compared to around 0.43-0.48 in AMA_4^+) as well as the fact that AMA_3^+ finds only a single valid instance of Sokoban, which was optimal (thus the optimality rate is 1).

11.5 The Effect of Planner Preprocessing

One may be interested in whether the planners are by themselves able to prune more propositions/actions in the translation phase. We compared the number of “relevant propositions/actions” as measured by Fast Downward translator (translator variables / translator operators) against total number of propositions and actions. For each domain, we selected an AMA_4^+ model that returned the largest number of valid solutions with M&S (we selected M&S because it achieved the best overall performance).

As the total number of propositions, we used the number of effective bits discussed in Figure 10.6, rather than the number of propositions in a PDDL file, because we do not filter propositions based on their effectiveness when writing them into a PDDL file. As the total number of actions, we directly used the number of actions in a PDDL file because we already filter actions that is never mapped to by ACTION in the training dataset (see AMA_2 section). The number of actions in a PDDL file corresponds to A_2 after XOR compilation (Section 8.3, Section 10.6).

Results showed that Fast Downward did not reduce propositions/actions significantly. For actions, over 95% of original actions are retained. The ratio $\frac{[\text{translator operators}]}{[\text{actions}]}$ was blocks:99.8%, lightsout:98.6%, twisted:98.8%, mandrill:96.2%, mnist:95.4%, sokoban:99.7%.

The number of effective bits and the number of translator variables also closely matched: The ratio $\frac{[\text{translator variables}]}{[\text{effective bits}]}$ was blocks:100.0%, lightsout:103.2%, twisted:103.4%, mandrill:100.0%, mnist:100.0%, sokoban:100.0%. The numbers can exceed 100% because effective bits can be underestimated: It is measured by encoding a dataset, which contains only a subset of samples in the entire state space.

There were no variations in the translator variables / operators across problem instances except instances where no solution was found. This is unlike normal PDDL instances where these numbers differ due to different goal conditions. In instances where no solution was found, typically all actions and variables are detected as irrelevant and then removed.

12. Related Work

In this section, we cover a wide range of related work from purely connectionist approaches to purely symbolic approaches. In between, there is a recent body of work which are collectively called Neural/Neuro-Symbolic hybrid approaches.

12.1 Previous Work on Action Model Acquisition and Symbol Grounding

Table 12.1 summarizes a variety of action model acquisition approaches that have been proposed already (Jiménez et al., 2012; Arora et al., 2018). They can be understood from three major axes:

Input: Traditionally, symbolic action learners tend to require a certain type of human domain knowledge and have been situating itself merely as an additional assistance tool for humans, rather than a system that builds knowledge from the scratch, e.g., from unstructured images. Existing systems (Yang et al., 2007; Mourão et al., 2012; Zhuo & Kambhampati, 2013; Zhuo et al., 2019; Cresswell et al., 2009; Cresswell & Gregory, 2011; Cresswell et al., 2013; Gregory & Cresswell,

	Required Input						Symbol Generation			Symbol Grounding			Optimization Method for Likelihood Maximization	Target language
	state ID	prop.	pred.	act.	obs.	noise	prop.	pred.	act.	prop.	pred.	act.		
(Yang et al., 2007)	Y	Y	Y	Y	full	N	N	N	N	N	Y	MAXSAT	PDDL	
(Mourão et al., 2012)	Y	Y	Y	Y	partial	Y	N	N	N	N	Y	SVM	PDDL	
(Zhuo & Kambhampati, 2013)	Y	Y	Y	Y	full	Y	N	N	N	N	Y	MAXSAT	PDDL	
(Zhuo et al., 2019)	Y	Y	Y	Y	dis.	Y	N	N	N	N	Y	MAXSAT	PDDL	
(Cresswell et al., 2009)	Y	Y	Y	Y	full	N	N	N	N	N	Y	FSM	PDDL	
(Cresswell & Gregory, 2011)	Y	Y	Y	Y	full	N	N	N	N	N	Y	FSM	PDDL	
(Cresswell et al., 2013)	Y	Y	Y	Y	full	N	N	N	N	N	Y	FSM	PDDL	
(Gregory & Cresswell, 2015)	Y	Y	Y	Y	full	N	N	N	N	N	Y	FSM	PDDL	
(Aineto et al., 2018)	Y	Y	Y	N	partial	N	N	Y	N	N	Y	Classical Planning	PDDL	
(Bonet & Geffner, 2020)	Y	N	N	Y	N/A	Y	Y	N	Y	Y	Y	MAXSAT	PDDL	
(Rodriguez et al., 2021)	Y	N	N	Y	N/A	Y	Y	N	Y	Y	Y	ASP	PDDL	
(Konidaris et al., 2014)	N	N	N	Y	full	N	Y	N	N	Y	Y	C4.5	PDDL	
(Konidaris et al., 2015)	N	N	N	Y	full	Y	Y	N	N	Y	Y	mixed	PPDDL	
(Andersen & Konidaris, 2017)	N	N	N	Y	full	Y	Y	N	N	Y	Y	mixed	PPDDL	
(Konidaris et al., 2018)	N	N	N	Y	full	Y	Y	N	N	Y	Y	mixed	PPDDL	
(Barbu et al., 2010)	N	Y	Y	Y	N/A	Y	N	N	N	N	Y	ILP	Prolog	
(Kaiser, 2012)	N	Y	Y	Y	N/A	Y	N	N	N	N	Y	ILP	GGP	
(Ugur & Piater, 2015)	N	N	N	Y	full	Y	Y	Y*	N	N	Y	mixed	PDDL	
(Ahmetoglu et al., 2020)	N	N	N	Y	full	Y	Y	Y*	N	N	Y	mixed	PPDDL	
(Lindsay et al., 2017)	natural language						N	N	N	N	N	Y	k-medoid clustering	PDDL
(Miglani & Yorke-Smith, 2020)	natural language						N	N	N	N	N	Y	RL+LOCM2	PDDL
Latplan	N	N	N	N	full	Y	Y	N	Y	Y	N	Y	Deep Generative Model	PDDL

Table 12.1: Comparison of various symbol grounding & action model acquisition approaches. Synopsis: *prop.*=propositional symbols, *pred.*=predicate symbols, *act.*=action symbols, *obs.*=observability, *dis.*=disordered observation. (*) These systems obtain predicates, but they are limited to types (static predicates).

2015; Aineto et al., 2018; Bonet & Geffner, 2020; Rodriguez et al., 2021; Konidaris et al., 2014, 2015; Andersen & Konidaris, 2017; Konidaris et al., 2018) require some type of symbolic inputs as defined in Section 3 and exploit the categorical information and/or the structures provided by the input, as shown in the Table 12.1. The input format could vary, e.g., sequence of ground actions (Yang et al., 2007; Zhuo & Kambhampati, 2013; Zhuo et al., 2019; Cresswell et al., 2009; Cresswell & Gregory, 2011; Cresswell et al., 2013; Gregory & Cresswell, 2015), sequence of actions and intermediate states (Mourão et al., 2012; Konidaris et al., 2014, 2015, 2018; Andersen & Konidaris, 2017), initial/goal states (and optionally plans and partial action models) (Aineto et al., 2018), or a graph of atomic states (state IDs) and actions (Bonet & Geffner, 2020; Rodriguez et al., 2021). Some systems accept varying amount of noise and partial observability (Mourão et al., 2012; Zhuo & Kambhampati, 2013; Zhuo et al., 2019; Aineto et al., 2018; Rodriguez et al., 2021). Approaches such as Framer (Lindsay et al., 2017) and cEASDRL (Miglani & Yorke-Smith, 2020) obtain action models from natural language corpus. They reuse the symbols found in the corpus, thus are grounding but not generating symbols, though it contains a type of noise specific to natural language, e.g., case inflection and ambiguity.

Output: While existing systems do not generate all types of symbols, some systems generate some symbols. For example, (Bonet & Geffner, 2020; Rodriguez et al., 2021) generates predicate symbols from state IDs and action symbols, (Aineto et al., 2018) generates action symbols from predicate symbols, and (Konidaris et al., 2014, 2015, 2018; Andersen & Konidaris, 2017) generates propositional symbols from action symbols.

Optimization Method for Likelihood Maximization: Although it is not explicitly acknowledged in some of the previous work, machine learning problems can be cast as optimization problems through Maximum-Likelihood Estimation framework (Section 2.3), thus they are implicitly operating under the same framework as our paper, with the difference being the implementation, differentiability, and joint training of the optimizer.

Each system tackles a learning problem using different optimization approaches. The learning problem is encoded into Maximum Satisfiability (Vazirani, 2013, MAX-SAT) in (Yang et al., 2007; Zhuo & Kambhampati, 2013; Zhuo et al., 2019; Bonet & Geffner, 2020), classical planning in (Aineto et al., 2018), or Answer Set Programming (Gebser et al., 2011, ASP) in (Rodriguez et al., 2021). Approaches based on Finite State Machines (FSM) (Cresswell et al., 2009; Cresswell & Gregory, 2011; Cresswell et al., 2013; Gregory & Cresswell, 2015) tackle the learning problem by generating a large number of candidate FSMs and pruning those which contradict data, effectively maximizing the number of data points that follow the set of FSMs, i.e., $\arg \max_{\text{FSMs}} p(x)$.

Many approaches (Konidaris et al., 2014, 2015, 2018; Andersen & Konidaris, 2017; Ugur & Piater, 2015; Ahmetoglu et al., 2020; Lindsay et al., 2017; Miglani & Yorke-Smith, 2020) combine several machine learning subprocedures, where each module individually optimizes conditional likelihood, which is similar to how AMA₂ learns a state representation and an action model separately. Examples include C4.5 Decision Tree in (Konidaris et al., 2014; Ugur & Piater, 2015), Support Vector Machine, DBSCAN, and Kernel density estimation in (Konidaris et al., 2015), Bayesian sparse Dirichlet-categorical model and Bayesian Hierarchical Clustering in (Andersen & Konidaris, 2017), sparse Principal Component Analysis in the robotic task using point clouds in (Konidaris et al., 2018), Binary-Concrete VAE (also used in our paper in AMA₁ and AMA₂) (Ahmetoglu et al., 2020), CoreNLP language model (Lindsay et al., 2017), or Reinforcement Learning and LOCM in (Miglani & Yorke-Smith, 2020). While they maximize individual conditional likelihoods (e.g., $q(z | x)$) separately, they are not formally motivated as such, nor are they guaranteed to maximize the likelihood of observation. In contrast, Latplan AMA₃⁺ and AMA₄⁺ jointly maximize the likelihood of observation through differentiable neural network.

12.2 Neural Networks as Search Guidance, World Models

Existing work (Arfaee et al., 2010, 2011; Thayer et al., 2011; Satzger & Kramer, 2013; Yoon et al., 2006, 2008; Ferber et al., 2020; Toyer et al., 2018; Shen et al., 2020) has combined symbolic search and machine learning by learning a function that provides search control knowledge, e.g., domain-specific heuristic functions for the sliding-tile puzzle and Rubik’s Cube (Arfaee et al., 2011), classical planning (Satzger & Kramer, 2013), or the game of Go (Silver et al., 2016). Some approaches use supervised learning from the precomputed target values (e.g., optimal solution cost) or expert traces, while others use Reinforcement Learning (Sutton & Barto, 2018, RL).

Model-free reinforcement learning has solved complex problems, including video games in Arcade Learning Environment (Bellemare et al., 2013, ALE) where the agent communicates with a simulator through images (Mnih et al., 2015, DQN). The system avoids symbol grounding by using the subsymbolic states directly and assuming a set of action symbols that are given *a priori* by a simulator. For example, the ALE provides agents its discrete action labels such as 8-directional joysticks and buttons. Another limitation of RL approaches is that they require a carefully designed dense reward function to perform efficient exploration. When the available reward function

is sparse, RL approaches suffer from sample efficiency issues where the agents require millions or billions of interactions with the simulator to learn meaningful behavior.

Latplan does not learn domain-control knowledge / search heuristics from data, but instead offloads the derivation of heuristic functions to the underlying off-the-shelf classical planner such as Fast Downward by passing a learned PDDL model. This framework enables the direct application of the large body of work on symbolically derived heuristics, e.g., delete-relaxation heuristics including h^{LMcut} (Hoffmann & Nebel, 2001; Helmert & Domshlak, 2009) and abstraction heuristics including $h^{\text{M\&S}}$ (Helmert et al., 2014). It has an advantage over policy-learning approaches because our system maintains all of the theoretical characteristics (optimality, completeness, etc.) of the underlying off-the-shelf planner, with respect to the learned state space. In contrast, learned policies typically do not guarantee admissibility, and only guarantees the convergence to the optimal policy in the limit. Greedy algorithms typically used by RL agents during the evaluation also contribute to the lack of reliability. Such a lack of the completeness/optimality guarantee is problematic for critical real-world applications. Moreover, while systems that learn search control knowledge require supervised signals, reward signals, or simulators that provides action symbols, Latplan only requires a set of unlabeled image pairs (transitions), and does not require a reward function, expert solution traces, simulators, or predetermined action symbols.

Model-based RL (Schrittwieser et al., 2019; Kaiser et al., 2020) additionally learns a transition function in the state space to facilitate efficient exploration. World-model literature (Ha & Schmidhuber, 2018) learns latent representations of states and their black-box transition functions, i.e., a representation learning portion of model-based RL. The learned transition function is typically a black-box function that is not compatible with symbolic model analysis that are necessary for deriving heuristic functions. AMA_2 (Section 7) and Vanilla Space AE (Section 8.1) can be seen as an instance of such a black-box world model. As another example, Causal InfoGAN (Kurutach et al., 2018), which was published after the conference version of this paper (Asai & Fukunaga, 2018), learns binary latent representations and transition functions similar to Latplan but with Generative Adversarial Networks (Goodfellow et al., 2014, GANs) augmented with Mutual Information maximization (Chen et al., 2016, InfoGAN). A significant limitation of this approach is that their transition function lacks the concept of actions: The successor generation relies on sampling from a transition function which is expected to be multi-modal. Unlike symbolic transition models, this does not guarantee that all logically plausible successors are enumerated by a finite number of samples from a successor function, making the search process potentially incomplete.

More recently, several papers proposed ideas that resembles Cube-Space AE, although outside the context of STRIPS model learning. Alchemy (Wang et al., 2021) is a benchmark environment and an analysis toolkit for meta-reinforcement learning, where the environment contains multiple stones which change shapes, colors, and sizes etc. due to actions. Environment observations are provided either in a form of 3D rendering or in a symbolic encoding. Its underlying dynamics assumes that each action has a consistent effect that resembles STRIPS actions. Bisimulation metrics (Zhang et al., 2021) proposed to improve the quality of the latent space learned by model-based RL. The metric also resembles the assumptions in STRIPS, although the additional loss function as well as the entire loss for representation learning are not justified as a lower bound of the likelihood.

12.3 Neural Networks that Directly Models the Problems

There is a large body of work using NNs to directly solve combinatorial tasks by modeling the problem as the network itself, starting with the well-known TSP solver by Hopfield and Tank (1985). Neurosolver represented a search state as a node in NN and solved Tower of Hanoi (Bieszczad & Kuchar, 2015). However, they assume a symbolic input that is converted to the nodes in a network. The neural network is merely used as a vehicle that carries out optimization.

12.4 Novelty-Based Planning without Action Description

While there are recent efforts in handling a complex state space without having its action description (Frances et al., 2017), action models could be used for other purposes such as Goal Recognition (Ramírez & Geffner, 2009), macro-actions (Botea & Braghin, 2015; Chrupa et al., 2015), or plan optimization (Chrupa & Siddiqui, 2015). Moreover, goal-directed heuristics based on descriptive action models are complementary to novelty-based search techniques, and has their combination has been shown to achieve state-of-the-art results on IPC domains (Lipovetzky, 2017).

13. Discussion and Conclusion

We proposed Latplan, an integrated architecture for learning and planning which, given only a set of unlabeled image pairs 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 empirically demonstrated its feasibility using image-based versions of planning/state-space-search problems (Blocksworld, 8-puzzle, 15-puzzle, Lights Out, Sokoban), provided a theoretical justification for the training objectives from the maximum-likelihood standpoint, and analyzed the model complexity of STRIPS action models from the perspective of graph coloring and Cube-Like Graphs.

Our technical contributions are (1) *State Auto-Encoder, which leverages the Binary-Concrete technique to learn a bidirectional mapping between raw images and propositional symbols compatible with symbolic planners.* For example, on the 8-puzzle, the SAE can robustly compress the “gist” of a training image into a propositional vector representing the essential information (puzzle configuration) presented in the images. (2) *Non-standard prior distribution Bernoulli($\epsilon = 0.1$) for Binary Concrete, which improves the stability of propositional symbols and helps symbolic search algorithms operate on latent representations.* The non-standard prior encodes a closed-world assumption, which assumes that propositions are False by default, unlike the standard prior Bernoulli($\epsilon = 0.5$) which assigns random, stochastic bits to unused dimensions in the latent space, i.e., “unknown,” much like in open-world assumption. (3) *Action Auto-Encoder, which grounds action symbols, i.e., identifies which transitions are “same” with regard to the state changes.* By having a finite set of action symbols, agents can enumerate successor states efficiently during planning, instead of exhaustively enumerating the entire state space or sampling successor states. (4) *Back-to-Logit, which enables learning and extracting descriptive, STRIPS-compatible action effects using a neural network.* It restricts the hypothesis space of the action model with its structural prior, and bounds the complexity of the action model through graph coloring on cube-like graphs. (5) *Complete State Regression Semantics, in which preconditions can be modeled as effects backward in time.* – preconditions are now modeled with prevail-conditions. The network enables efficient and accurate extraction of preconditions from the trained network weights, which resulted in better

planning performance. (6) *Formalization of the training process under a sound likelihood maximization / variational inference framework*. This results in a training objective that is less susceptible to overfitting due to various regularization terms and the lower bounding characteristics.

The only key assumption we make about the input domain is that it is fully observable and deterministic, i.e., that it is in fact a classical planning domain. 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 that completely automatically constructs a logical representation *directly usable by a symbolic planner* from a set of unlabeled image pairs 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 representing 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. In the following, we discuss several key limitations that Latplan must address in the future.

Latplan requires uniform sampling from the environment, which is nontrivial in many scenarios. Automatic data collection via exploration and active learning (Settles, 2012) is a major component of future work.

Next, 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. Physically “executing” the plan is not necessary, as finding the solution to the puzzles is the objective, so a “mental image” of the solution (i.e., the image sequence visualization) is sufficient. However, in domains where actions have effects in the world, it will be necessary to consider how actions found by Latplan (transitions between latent bit vector pairs) can be mapped to actuations. One direction of future work could be learning a continuous-discrete hybrid representation that is more suitable for physical dynamical systems such as robotic tasks.

Latplan was evaluated in a noisy but fully observable and deterministic environment. Representing states and state transitions with probabilistic belief states for partial observations and stochastic state transitions would enable probabilistic planners to reason in such environments.

Latplan is also currently limited to tasks where a single goal state is specified. Developing a method for specifying a set of goal states with a partial goal specification as in IPC domains is an interesting topic for future work. For example, one may want to tell the planner “the goal states must have tiles 0,1,2 in the correct places” in a MNIST 8-puzzle instance.

Although we have shown that the Latplan architecture can be successfully applied to image-based versions of several relatively large standard puzzle domains (15-puzzle, Sokoban), some seemingly simple image-based domains may pose challenges for the current implementation of Latplan. For example, in Section M we discuss the result of applying Latplan to 4-disk, 4-tower Towers of Hanoi domain where the success ratio is quite low despite the fact that the domain has only 256 valid states, and thus is much “simpler” than domains such as the 15-puzzle. We listed several potential reasons for failure in this domain, one of which is that some features are “rare”, e.g., the smallest disk rarely appears in the top region of the image, resulting in an imbalanced dataset. Such features are hard for the current SAE implementations to learn and generalize correctly, leading to incorrect plans. Thus, methods for training a SAE on similar domains where images containing key features are very rare is a direction for future work.

Latplan’s state representation is entirely propositional and lacks first-order logic concepts such as predicates and parameters. Historically, action model acquisition literature focused on such representations, which provides generalization to environments with different numbers of objects. As a result, adding an object to a Blocksworld environment currently requires training the model from the scratch. One promising direction is using object-based representation, such as an ad-hoc approach pursued in (Harman & Simoens, 2020), in a more principled probabilistic manner.

Also, although we demonstrated that the SAE is somewhat robust to variations/noise, it is not able to, for example, solve an instance (initial state image) of a sliding-tile puzzle instance scrawled on a napkin by an arbitrary person. Latplan will fail if, for example, some of the numbers in the initial state image for the 8-puzzle were rotated or translated, or the appearance of the digits differed significantly from the those in the training data. An ideal system would be robust enough to solve an instance (initial state image) of a sliding-tile puzzle instance scrawled on a napkin by an arbitrary person. Achieving this level of robustness will require improving the state encoder to the point that it is robust to styles, rotation, translation, etc.

Another direction for modifying the SAE is to combine the techniques in the autoencoders for the other types of inputs: e.g., unstructured text (Li et al., 2015) and audio data (Deng et al., 2010). Applying Gumbel-Softmax to these techniques, it may be possible for Latplan to perform language-based or voice-based reasoning.

In addition to the technical contributions, this paper provides several conceptual contributions. First and foremost, *we provide the first demonstration that it is possible to leverage deep learning quite effectively for classical planning, which “has been central to AI research since its inception.”* (Russell et al., 1995, p396) We bridged the gap between connectionist and symbolic approaches by using the former as a perception system generating the symbols for the latter resulting in a *neuro-symbolic system*.

Second, based on our observations of the problematic behavior of unstable propositions, we defined the general *Symbol Stability Problem* (SSP), a subproblem of symbol grounding. We identified two sources of stochasticity which can introduce the instability: (1) the inherent stochasticity of the network, and (2) the external stochasticity from the observations. This suggests that SSP is an important problem that applies to any modern NN-based symbol grounding process that operates on the noisy real-world inputs and performs a sampling-based, stochastic process (e.g. VAEs, GANs) that are gaining popularity in the literature. Thus, characterizing the aspect of SSP would help the process of designing a planning system operating on real-world input. We demonstrated the importance of addressing the instability by a thorough empirical analysis of the impact of instability on the planning performance. An interesting avenue for future work is to extend our approach to InfoGAN-based discrete representation of the environment (Kurutach et al., 2018).

Finally, we demonstrated that state-of-the-art domain-independent search heuristics provide effective search guidance in the automatically learned state spaces. These domain-independent functions, which have been a central focus of the planning community in the last two decades, provide search guidance without learning. This is in contrast to popular reinforcement learning approaches that suffer from poor sample efficiency, domain-dependence, and the lack of formal guarantees on admissibility. We believe this finding stimulates further research into heuristic search as well as reinforcement learning.

Appendix A. Probability Theory

A probability distribution $P(x)$ of a random variable x defined on a certain domain X is a function from a value $x \in X$ to some non-negative real $P(x = x)$. The sum / integral over X (when X is discrete / continuous) is equal to 1, i.e., $\sum_{x \in X} P(x = x) = 1$ (discrete domain) or $\int_{x \in X} P(x = x) dx = 1$ (continuous domain), respectively. The function is also called a probability mass function (PMF) or a probability density function (PDF), respectively. Typically, each random variable is given a certain meaning, thus two notations $P(x)$ and $P(y)$ denote different PMFs/PDFs and the letter P does not designate a function by itself, unlike normal mathematical functions where $f(x)$ and $f(y)$ are equivalent under variable substitution. For example, if x is a boolean variable for getting a cancer and y is a boolean variable for smoking cigarettes, then $P(x)$ and $P(y)$ denote completely different PMFs, and we could write $P(x) = f(x)$, $P(y) = g(y)$, and $f \neq g$ to make it explicit. When a value $x \in X$ is given, we obtain an actual value $P(x = x) = f(x)$, which is sometimes abbreviated as $P(x)$. To denote two different distributions for the same random variable, an alternative letter such as $Q(x)$ is used.

A joint probability $P(x = x, y = y)$ is a function of two arguments which returns a probability of observing x, y at once. Conditional probability $P(x = x | y = y)$ represents a probability of observing x when y was already observed, and is defined as $P(x | y) = \frac{P(x, y)}{P(y)}$. Therefore $P(x = x) = \sum_{y \in Y} P(x = x | y = y) P(y = y)$ holds.

For two probability distributions $Q(x)$ and $P(x)$ for a random variable x , a *Kullback-Leibler (KL) divergence* $D_{\text{KL}}(Q(x) || P(x))$ is an expectation of their log ratio over $Q(x = x \in X)$:

$$D_{\text{KL}}(Q(x) || P(x)) = \mathbb{E}_{Q(x=x)} \left\langle \log \frac{Q(x=x)}{P(x=x)} \right\rangle.$$

This is $\sum_{x \in X} Q(x) \log \frac{Q(x)}{P(x)}$ for discrete distributions and $\int_{x \in X} Q(x) \log \frac{Q(x)}{P(x)} dx$ for continuous distributions. KL divergence is always non-negative, and equals to 0 when $P = Q$. Conceptually it resembles a distance between distributions, but it is not a distance because it does not satisfy the triangular inequality.

Appendix B. Why do we Minimize L2-Norm / Square Errors ? Why not the Mean Square Errors?

While ad hoc approaches have been successful in many machine learning systems, a more principled approach to implementing machine learning algorithms is facilitated by a probabilistic interpretation (Murphy, 2012).

The networks f, g of an AE are optimized by minimizing a *reconstruction loss* $\|x - \hat{x}\|$ under some norm, which is typically a (Mean) Square Error / L2-norm. Which norm to use is determined by the distribution of x assumed by the model designer. Assuming a 1-dimensional case, let x be a data point in the dataset, z be a certain latent value, and a probability distribution $p(x|z)$ be what the neural network (and the model designer) believes is the distribution of x given z . Typical AEs for images assume that x follows a Gaussian distribution centered around the predicted value $\hat{x} = g(z)$, i.e., $p(x|z) = \mathcal{N}(x|\hat{x}, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\hat{x})^2}{2\sigma^2}}$ for an arbitrary fixed constant σ . This leads

to an analytical form of the negative log likelihood (NLL) $-\log p(x|z)$:

$$-\log \mathcal{N}(x|\hat{x}, \sigma) = -\log \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\hat{x})^2}{2\sigma^2}} \right] = \frac{(x-\hat{x})^2}{2\sigma^2} + \log \sqrt{2\pi\sigma^2} = C_1(x-\hat{x})^2 + C_2 \tag{22}$$

for some constant $C_1 > 0$ and C_2 , which is a scaled/shifted square error / L2-norm reconstruction loss.

For a multi-dimensional case, by assuming that individual outputs (e.g., pixels) x_i, x_j are independent for $i \neq j$, we sum up Equation 22 across output dimensions i , because $-\log p(\mathbf{x}|z) = -\log \prod_i p(x_i|z) = \sum_i -\log p(x_i|z)$. Finally, a *mean* square loss ($C_1 = 1/D$, where D is the number of output dimensions, e.g., pixels) is obtained by arbitrarily setting $\sigma = \sqrt{D/2}$.

By minimizing the reconstruction loss $-\log p(\mathbf{x}|z)$, the training maximizes $p(\mathbf{x}|z)$, the likelihood of observing \mathbf{x} — the higher the probability, the more likely we get an output \hat{x} closer to the real data \mathbf{x} . The general framework that casts a machine learning task as a likelihood maximization task is called *maximum likelihood estimation*.

By assuming a different distribution on \mathbf{x} , we obtain a different loss function. For example, a Laplace distribution $\frac{1}{2b} \exp(-\frac{|x-\hat{x}|}{b})$ results in an absolute error loss $|x-\hat{x}|$. Further, in many cases, a metric $d(x, \hat{x})$ automatically maps to a probability distribution through $-\log p(x|\hat{x}) \propto d(x, \hat{x}) \Leftrightarrow p(x|\hat{x}) = C_1 \exp(-C_2 d(x, \hat{x}))$, where C_1, C_2 are constants for maintaining $\int_x p(x|\hat{x}) = 1$. While the choice of the loss function (thus the distribution of \mathbf{x}) is arbitrary, we typically choose Gaussian distribution because it is the *maximum entropy distribution* among continuous distributions on \mathbb{R} with the same variance and the mean, i.e., the Gaussian distribution has the highest entropy $\int -p(x) \log p(x) dx$, thus is most random, therefore assumes the least about the distribution.

In many AE/VAE implementations, C_2 is ignored, and $C_1 = 1/D$ by assuming $\sigma = \sqrt{D/2}$. Since this is an arbitrary value, it is sometimes necessary to tune σ manually. Alternatively, Bayesian NN methods (Nix & Weigend, 1994) learn to predict both σ and the mean \hat{x} of the Gaussian distribution by doubling the size of the output of the network. It trains the network by optimizing Equation 22 without omitting (now non-constant) C_2 .

Appendix C. Other Discrete Variational Methods

Other discrete VAE methods include VQVAE (van den Oord et al., 2017), DVAE++ (Vahdat et al., 2018b), and DVAE# (Vahdat et al., 2018a). The difference between them is the training stability and accuracy. They may contribute to stable performance, but we leave the task of faster / easier training for future work.

A variant called Straight-Through Gumbel-Softmax (ST-GS) (Jang et al., 2017) combines a so-called Straight-Through estimator with Gumbel Softmax. ST-GS is outperformed by standard GS (Jang et al., 2017, Figure 3(b)), thus we do not use it in this paper. However, we explain it in an attempt to cover as many methods as possible, and also because this ST-estimator frequently appears in other discrete representation learning literature, including ST-estimator for Heaviside STEP function (Koul et al., 2019; Bengio et al., 2013) and VQVAE (van den Oord et al., 2017).

A Straight-Through estimator is implemented by a primitive operation called *stop gradient* SG, which is available in major deep learning / automatic differentiation frameworks. SG acts as an identity in the forward computation but acts as a zero during the weight update / backpropagation step of automatic differentiation in neural network training. To understand how it works, it is best

to see how ST-GS is implemented:

$$\text{ST-GS}(\mathbf{l}) = \text{sg}(\arg \max(\mathbf{l}) - \text{GS}_\tau(\mathbf{l})) + \text{GS}_\tau(\mathbf{l}) = \begin{cases} \arg \max(\mathbf{l}) & (\text{forward}) \\ 0 + \text{GS}_\tau(\mathbf{l}) & (\text{backward}) \end{cases} \quad (23)$$

Notice that this function acts exactly as $\arg \max(\mathbf{l})$ in the forward computation, but uses only the differentiable $\text{GS}_\tau(\mathbf{l})$ for backpropagation, thus eliminating the need for a gradient of a non-differentiable function $\arg \max$. Several applications of ST-estimator exist. ST estimator that combines Heaviside step function STEP and a linear function x was used in (Koul et al., 2019; Bengio et al., 2013) but was outperformed by Binary Concrete (Jang et al., 2017, Figure 3(a)). Similarly, an ST-estimator can combine STEP and Binary Concrete, but this was also outperformed by the standard Binary Concrete (Jang et al., 2017, Figure 3(a)).

Appendix D. ELBO Computation of Gumbel Softmax / Binary Concrete VAE

Having provided the overview, we explain the details of ELBO computation of Gumbel Softmax VAE and then Binary Concrete VAE, following (Jang et al., 2017; Maddison et al., 2017). Those who are not interested in these theoretical aspects can safely skip this section, but it becomes relevant in later sections where we analyze our contributions. This section also discusses several ad-hoc variations of the implementations shared by the original authors (and their issues) to sort out the information available in the field.

To compute ELBO, we need an analytical form of the reconstruction loss and the KL divergence. We focus on the latter because the choice of reconstruction loss $\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} | \mathbf{z})]$ is independent from the computation of the KL divergence $D_{\text{KL}}(q(\mathbf{z} | \mathbf{x}) || p(\mathbf{z}))$.

We denote a categorical distribution of C classes as $\text{Cat}(\mathbf{p})$ with parameters $\mathbf{p} \in \mathbb{B}^C$. Here, \mathbf{p} is a probability vector for C classes, thus it sums up to 1, i.e., $\sum_{k=1}^C \mathbf{p}_k = 1$. For example, when $\mathbf{p}_k = 1/6$ for all k and $C = 6$, it models a fair cube dice.

Gumbel-Softmax is a continuous relaxation of Gumbel-Max technique (Gumbel & Lieblein, 1954; Maddison et al., 2014), a method for drawing samples of categorical distribution $\text{Cat}(\mathbf{p})$ from a *log-unnormalized probability* or a *logit* \mathbf{l} . ‘‘Log-unnormalized’’ imply that $\mathbf{l} = \log \mathbf{p}'$, where \mathbf{p}' is an unnormalized probability. Since it is not normalized, \mathbf{p}' does not sum up to 1, but normalization is trivial ($\mathbf{p} = \mathbf{p}'_k / \sum_k \mathbf{p}'_k$). Combining these facts derives a following relation:

$$\mathbf{p}_k = \mathbf{p}'_k / \sum_k \mathbf{p}'_k = \exp \mathbf{l}_k / \sum_k \exp \mathbf{l}_k = \text{SOFTMAX}(\mathbf{l})_k. \quad (24)$$

Log-unnormalized probabilities \mathbf{l} are convenient for neural networks because it can take an arbitrary value in \mathbb{R}^C . Gumbel-Max draws samples from $\text{Cat}(\mathbf{p})$ using \mathbf{l} without computing \mathbf{p} explicitly:

$$\{0, 1\}^C \ni \text{GUMBELMAX}(\mathbf{l}) = \arg \max(\mathbf{l} + \text{GUMBEL}^C(0, 1)) \sim \text{Cat}(\mathbf{p}). \quad (25)$$

Again, note that we assume $\arg \max$ returns a one-hot representation rather than the index of the maximum value.

Unlike samples generated by Gumbel-Max technique, samples from Gumbel-Softmax function follows its own $\text{GS}(\mathbf{l}, \tau)$ distribution, not the original $\text{Cat}(\mathbf{p})$ distribution:

$$\mathbb{B}^C \ni \mathbf{z} = \text{GS}_\tau(\mathbf{l}) = \text{SOFTMAX}\left(\frac{\mathbf{l} + \text{GUMBEL}^C(0, 1)}{\tau}\right) \sim \text{GS}(\mathbf{l}, \tau). \quad (26)$$

An obscure closed-form probability density function (PDF) of this distribution is available (Jang et al., 2017; Maddison et al., 2017) as follows:

$$\mathbf{GS}(z | l, \tau) = (C - 1)! \tau^{C-1} \prod_{k=1}^C \frac{\exp l_k z_k^{-(\tau+1)}}{\sum_{i=1}^C \exp l_i z_i^{-\tau}}. \quad (27)$$

The factorial $(C - 1)!$ is sometimes denoted by a Gamma function $\Gamma(C)$ depending on the literature.

D.1 A Simple Add-Hoc Implementation

In practice, an actual VAE implementation may avoid using this complicated PDF of $\mathbf{GS}(l, \tau)$ by computing the KL divergence based on $\mathbf{Cat}(p)$ instead. While it could potentially violate the true lower bound (ELBO), in practice, it does not seem to cause a significant problem. This issue is discussed by Maddison et al. (2017, Eq.21,22). The following derivation is based on the actual implementation shared by an author on his website⁵, which corresponds to Eq.22 in Maddison et al. (2017). It made two modifications to the faithful formulation based on the complicated PDF of $\mathbf{GS}(l, \tau)$ in order to simplify the optimization objective. In this implementation, it computes the KL divergence as if the annealing is completed ($\tau = 0$), treating the variable z as a discrete random variable. The implementation also uses $p(z) = \mathbf{Cat}(1/C)$ (i.e., a uniform categorical distribution) as a prior distribution. The KL divergence in the VAE is thus as follows:

$$\begin{aligned} \int q(z | \mathbf{x}) \log \frac{q(z | \mathbf{x})}{p(z)} dz &\approx \sum_{k \in \{1..C\}} q(\mathbf{z}_k = 1 | \mathbf{x}) \log \frac{q(\mathbf{z}_k = 1 | \mathbf{x})}{p(\mathbf{z}_k = 1)} \quad \because z \text{ is treated as discrete.} \\ &= \sum_{k \in \{1..C\}} q(\mathbf{z}_k = 1 | \mathbf{x}) \log \frac{q(\mathbf{z}_k = 1 | \mathbf{x})}{\frac{1}{C}} \\ &= \sum_{k \in \{1..C\}} q(\mathbf{z}_k = 1 | \mathbf{x}) \log q(\mathbf{z}_k = 1 | \mathbf{x}) + \sum_{k \in \{1..C\}} q(\mathbf{z}_k = 1 | \mathbf{x}) (\log C) \\ &= \sum_{k \in \{1..C\}} q(\mathbf{z}_k = 1 | \mathbf{x}) \log q(\mathbf{z}_k = 1 | \mathbf{x}) + \log C \quad \because q \text{ sums up to 1.} \end{aligned} \quad (28)$$

Since it assumes that the annealing is completed, the distribution is also treated as if it is equivalent to $\mathbf{Cat}(q)$ where $q = \text{SOFTMAX}(l)$. Therefore, this formula can be computed using $q(\mathbf{z}_k = 1 | \mathbf{x}) = q_k = \text{SOFTMAX}(l)_k$.

Appendix E. Difference of the Non-Standard Bernoulli Prior (Section 5.4) from the Zero-Suppress SAE (Asai & Kajino, 2019)

Our conference paper (Asai & Kajino, 2019) proposed ZSAE (Zero-suppress SAE), a method which uses an additional ad-hoc loss term called Zero-suppress loss defined as follows:

$$-H(q) + \alpha \cdot \text{BC}_\tau(l) \quad (29)$$

5. <https://blog.evjang.com/2016/11/tutorial-categorical-variational.html>

where l is an output of the encoder network, which is a logit of q (i.e., $q = \text{SIGMOID}(l)$). Compare this with the KL divergence we defined in Equation 7 (repost):

$$q \log \frac{q}{\epsilon} + (1 - q) \log \frac{(1 - q)}{(1 - \epsilon)} = -H(q) + \alpha \cdot q - \log(1 - \epsilon) \quad (30)$$

Two losses are similar up to the constant difference $-\log(1 - \epsilon)$ because BC_τ is a modified form of SIGMOID (Equation 4), therefore $\text{BC}_\tau(l)$ in Equation 29 and q in Equation 7 behave similarly.

Although Zero-suppress loss was shown to work, it is an ad-hoc loss that is not theoretically justified as an ELBO. Moreover, ZSAE used the values *after* the Binary Concrete activation $\text{BC}_\tau(l)$: This introduces unnecessary noise in the loss due to the logistic noise in BC_τ , which makes the training slower and less stable than the theoretically justified formulation presented in this paper.

Appendix F. Loss derivation for AMA₃⁺

We define the statistical model of this network as a probability distribution $p(\mathbf{x}^0, \mathbf{x}^1)$ of observed random variables \mathbf{x}^0 and \mathbf{x}^1 , also known as a *generative model* of \mathbf{x}^0 and \mathbf{x}^1 . Specifically, we model the probability distribution $p(\mathbf{x}^0, \mathbf{x}^1)$ by introducing latent variables corresponding to an action label \mathbf{a} (one-hot vector) and the current and successor propositional states, \mathbf{z}^0 and \mathbf{z}^1 (binary vectors). The model is trained by maximizing the log-likelihood $\log p(\mathbf{x}^0, \mathbf{x}^1)$ observing a tuple $(\mathbf{x}^0, \mathbf{x}^1)$. Since it is difficult to compute the log-likelihood function of such a latent variable model, we resort to a variational method to approximately maximize the likelihood, which leads to the autoencoder architecture depicted in Figure 8.1.

We start to model $p(\mathbf{x}^0, \mathbf{x}^1)$ by assuming a set of dependencies between variables and a set of distributional assumptions for the variables. This process is often called *statistical modeling*. *Following this process often helps to define a sound probabilistic model.*

Equations 31-33 below define dependencies between variables, where all summations are over the respective domains. In Equation 33, we assumed \mathbf{x}^0 and \mathbf{x}^1 depend only on \mathbf{z}^0 and \mathbf{z}^1 , respectively, because \mathbf{x}^0 and \mathbf{x}^1 are visualizations of \mathbf{z}^0 and \mathbf{z}^1 , respectively.

$$\begin{aligned} p(\mathbf{x}^0, \mathbf{x}^1) &= \sum_{\mathbf{z}^0, \mathbf{z}^1, \mathbf{a}} p(\mathbf{x}^0, \mathbf{x}^1, \mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) \\ &= \sum_{\mathbf{z}^0, \mathbf{z}^1, \mathbf{a}} p(\mathbf{x}^0, \mathbf{x}^1 \mid \mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) p(\mathbf{z}^0, \mathbf{z}^1, \mathbf{a}). \end{aligned} \quad (31)$$

$$p(\mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) = p(\mathbf{z}^1 \mid \mathbf{z}^0, \mathbf{a}) p(\mathbf{a} \mid \mathbf{z}^0) p(\mathbf{z}^0). \quad (32)$$

$$p(\mathbf{x}^0, \mathbf{x}^1 \mid \mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) = p(\mathbf{x}^0 \mid \mathbf{z}^0) p(\mathbf{x}^1 \mid \mathbf{z}^1). \quad (33)$$

The dependencies also define the I/O interfaces of the subnetworks (e.g., $p(\mathbf{x}^0 \mid \mathbf{z}^0)$ is modeled by a network that takes \mathbf{z}^0 and returns $\tilde{\mathbf{x}}^0$), and which ones are prior distributions that take no inputs (e.g., $p(\mathbf{z}^0)$). In general, when defining a generative model, it is better to avoid making unnecessary/unjustified independence assumptions. For example, although it is possible, we generally avoid assuming $p(\mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) = p(\mathbf{z}^0) p(\mathbf{z}^1) p(\mathbf{a})$, i.e., \mathbf{z}^0 , \mathbf{z}^1 and \mathbf{a} to be all independent, unless there is a domain knowledge that justifies the independence.

This process is also called *graphical modeling* (Russell et al., 1995, Chapter 14). It visualizes the statistical model using a directed acyclic graph (DAG) notation whose nodes are variables and

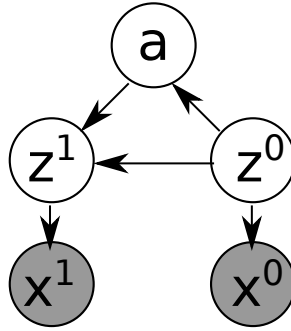


Figure F.1: DAG notation (graphical model) of our statistical model defined in Equations 31-33. The dark nodes are for observed variables, and the light nodes are for latent variables.

edges are conditional dependencies. For example, our model (Equations 31-33) can be visualized as Figure F.1.

After defining the dependencies, we assign an assumption to each distribution, as shown below. Thanks to the process of defining our generative model, we now realize that $p(\mathbf{a} \mid \mathbf{z}^0)$ requires an additional subnetwork `APPLICABLE(z)` that is missing in both the `AMA2` and the `AMA3` models. This network models $p(\mathbf{a} \mid \mathbf{z}^0)$ by taking a latent vector \mathbf{z} and returns a probability vector over actions. Note that some distributional assumptions apply to multiple vectors in Figure 8.1. For example, Equation 36 applies to both $\mathbf{z} = \mathbf{z}^1$ and $\mathbf{z} = \mathbf{z}^2$.

$$p(\mathbf{z}^0) = \text{Bernoulli}(\epsilon). \quad \text{Section 5.4.}$$

$$p(\mathbf{a} \mid \mathbf{z}^0 = \mathbf{z}) = \text{Cat}(\text{APPLICABLE}(\mathbf{z})) \quad (34)$$

$$p(\mathbf{x}^0 \mid \mathbf{z}^0 = \mathbf{z}) = \mathcal{N}(\tilde{\mathbf{x}}, \sigma), \text{ where } \tilde{\mathbf{x}} = \text{DECODE}(\mathbf{z}). \quad (35)$$

$$p(\mathbf{x}^1 \mid \mathbf{z}^1 = \mathbf{z}) = \mathcal{N}(\tilde{\mathbf{x}}, \sigma), \text{ where } \tilde{\mathbf{x}} = \text{DECODE}(\mathbf{z}). \quad (36)$$

$$p(\mathbf{z}^1 \mid \mathbf{z}^0 = \mathbf{z}, \mathbf{a} = \mathbf{a}) = \text{Bernoulli}(\mathbf{q}), \text{ where } \mathbf{q} = \text{SIGMOID}(\mathbf{l}), \mathbf{l} = \text{APPLY}(\mathbf{z}, \mathbf{a}). \quad (37)$$

Next, to compute and maximize $p(\mathbf{x}^0, \mathbf{x}^1)$, we should compute the integral/summation over latent variables $\mathbf{z}^0, \mathbf{z}^1, \mathbf{a}$ (Equation 31). Such an integration is known to be intractable; in fact, the complexity class of computing the likelihood of observations (e.g., $p(\mathbf{x}^0, \mathbf{x}^1)$) is shown to be #P-hard (Dagum & Chavez, 1993; Roth, 1996; Dagum & Luby, 1997) through reduction to #3SAT. Therefore, we have to resort to maximizing $p(\mathbf{x}^0, \mathbf{x}^1)$ approximately.

Variational modeling techniques, including VAEs, tackle this complexity by approximating the likelihood from below. This approximated objective is called ELBO (Evidence Lower Bound) and is defined by introducing a *variational model* q , an arbitrary distribution that a model designer can choose. To derive an autoencoding architecture, we model the variational distribution with encoders. Equations 38-40 define dependencies and assumptions in q , e.g., \mathbf{z}^1 depends only on \mathbf{x}^1 .

$$q(\mathbf{z}^0 \mid \mathbf{x}^0 = \mathbf{x}) = \text{Bernoulli}(\mathbf{q}), \text{ where } \mathbf{q} = \text{SIGMOID}(\mathbf{l}), \mathbf{l} = \text{ENCODE}(\mathbf{x}). \quad (38)$$

$$q(\mathbf{z}^1 \mid \mathbf{x}^1 = \mathbf{x}) = \text{Bernoulli}(\mathbf{q}), \text{ where } \mathbf{q} = \text{SIGMOID}(\mathbf{l}), \mathbf{l} = \text{ENCODE}(\mathbf{x}). \quad (39)$$

$$q(\mathbf{a} \mid \mathbf{x}^0 = \mathbf{x}, \mathbf{x}^1 = \mathbf{x}') = \text{Cat}(\mathbf{q}), \text{ where } \mathbf{q} = \text{SOFTMAX}(\mathbf{l}), \mathbf{l} = \text{ACTION}(\mathbf{x}, \mathbf{x}'). \quad (40)$$

Using this model, we derive a lower bound by introducing variables one by one. The first variable to introduce is \mathbf{z}^0 and the bound is derived using the same proof used for obtaining VAE's ELBO (Equation 2).

$$\begin{aligned}
 \log p(\mathbf{x}^0, \mathbf{x}^1) &= \log \left(\sum_{\mathbf{z}^0} p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0) p(\mathbf{z}^0) \right) \\
 &= \log \left(\sum_{\mathbf{z}^0} p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0) \frac{p(\mathbf{z}^0)}{q(\mathbf{z}^0 | \mathbf{x}^0)} q(\mathbf{z}^0 | \mathbf{x}^0) \right) \\
 &= \log \left(\mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0)} \left\langle p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0) \frac{p(\mathbf{z}^0)}{q(\mathbf{z}^0 | \mathbf{x}^0)} \right\rangle \right) \quad (\text{Definition of expectations}) \\
 &\geq \mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0)} \left\langle \log \left(p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0) \frac{p(\mathbf{z}^0)}{q(\mathbf{z}^0 | \mathbf{x}^0)} \right) \right\rangle \quad (\text{Jensen's inequality}) \\
 &= \mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0)} \langle \log p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0) \rangle \\
 &\quad - D_{\text{KL}}(q(\mathbf{z}^0 | \mathbf{x}^0) \| p(\mathbf{z}^0)). \quad (\text{Definition of KL divergence}) \tag{41}
 \end{aligned}$$

Next, we decompose $\log p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0)$ by introducing \mathbf{a} and deriving the lower bound. We are merely reapplying the same proof with a different set of variables and distributions:

$$\log p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0) \geq \mathbb{E}_{q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1)} \langle \log p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0, \mathbf{a}) \rangle - D_{\text{KL}}(q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) \| p(\mathbf{a} | \mathbf{z}^0)). \tag{42}$$

Since \mathbf{x}^0 does not depend on \mathbf{x}^1 and \mathbf{a} (Equation 33),

$$\begin{aligned}
 \log p(\mathbf{x}^0, \mathbf{x}^1 | \mathbf{z}^0, \mathbf{a}) &= \log p(\mathbf{x}^0 | \mathbf{z}^0) p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{a}) \\
 &= \log p(\mathbf{x}^0 | \mathbf{z}^0) + \log p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{a}). \tag{43}
 \end{aligned}$$

We further decompose $\log p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{a})$ by deriving its variational lower bound:

$$\begin{aligned}
 \log p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{a}) &= \log \sum_{\mathbf{z}^1} p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) \frac{p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})}{q(\mathbf{z}^1 | \mathbf{x}^1)} q(\mathbf{z}^1 | \mathbf{x}^1) \\
 &\geq \mathbb{E}_{q(\mathbf{z}^1 | \mathbf{x}^1)} \log p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) - D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})) \\
 &= \mathbb{E}_{q(\mathbf{z}^1 | \mathbf{x}^1)} \log p(\mathbf{x}^1 | \mathbf{z}^1) - D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})). \tag{44}
 \end{aligned}$$

However, we can also decompose it by applying Jensen's inequality directly. The reconstruction loss obtained from this formula corresponds to the reconstruction from the latent vector \mathbf{z}^2 (Figure 8.1) generated/sampled by the AAE, as the value is an expectation over $p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})$.

$$\begin{aligned}
 \log p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{a}) &= \log \sum_{\mathbf{z}^1} p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a}) \\
 &\geq \mathbb{E}_{p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})} \log p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{z}^1, \mathbf{a}) \\
 &= \mathbb{E}_{p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})} \log p(\mathbf{x}^1 | \mathbf{z}^1) \quad \text{Equation 33.} \tag{45}
 \end{aligned}$$

We wish to optimize both of these objectives (Equations 44-45). To understand the motivation, it is crucial to see the interpretation of these objectives. First, $D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a}))$

maintains the symbol stability we discussed in Section 5.2 by making two distributions ($q(\mathbf{z}^1 | \mathbf{x}^1)$ and $p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})$) identical. Without this KL term, latent vectors encoded by the encoder (i.e., $q(\mathbf{z}^1 | \mathbf{x}^1)$) and by the AAE (i.e., $p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})$) may diverge, despite them being the symbolic representation for the same raw observation \mathbf{x}^1 . Second, $\mathbb{E}_{p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})} \log p(\mathbf{x}^1 | \mathbf{z}^1)$ ensures that the latent vectors generated by the action model are reconstructable. If we optimize Equation 44 alone, the stability is maintained, but the resulting vector \mathbf{z}^1 is not guaranteed to be reconstructable. If we optimize Equation 45 alone, the stability is not maintained. Therefore, it is crucial to optimize both objectives at once.

To optimize both objectives at once, we combine them into a single objective. There is some flexibility in how to achieve this: For example, one could consider alternating two loss functions in each epoch, or consider optimizing the maximum of two objectives. Optimizing their Pareto front may also be an option. However, since they both give a lower bound of $\log p(\mathbf{x}^1 | \mathbf{z}^0, \mathbf{a})$, a simple approach to optimizing them at once while maintaining the ELBO is to take a weighted sum between them, with equal weights 0.5.

Now that the log likelihood is decomposed into either reconstruction losses from a decoder or KL divergences, we can finally discuss the total optimization objective. Similar to β -VAE discussed in Section 2.3, we can apply coefficients $\beta_1, \beta_2, \beta_3 \geq 1$ to each of the KL terms. This does not overestimate the ELBO because KL divergence is always positive, and larger β results in a lower bound of the original ELBO. Combining Equations 41-45, we obtain the following total maximization objective:

$$\begin{aligned}
 \log p(\mathbf{x}^0, \mathbf{x}^1) &\geq -\beta_1 D_{\text{KL}}(q(\mathbf{z}^0 | \mathbf{x}^0) \| p(\mathbf{z}^0)) \\
 &\quad + \mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0)} \left[\begin{array}{l} -\beta_2 D_{\text{KL}}(q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) \| p(\mathbf{a} | \mathbf{z}^0)) \\ + \mathbb{E}_{q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1)} \left[\begin{array}{l} \log p(\mathbf{x}^0 | \mathbf{z}^0) \\ + \frac{1}{2} \mathbb{E}_{q(\mathbf{z}^1 | \mathbf{x}^1)} \log p(\mathbf{x}^1 | \mathbf{z}^1) \\ - \frac{1}{2} \beta_3 D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})) \\ + \frac{1}{2} \mathbb{E}_{p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})} \log p(\mathbf{x}^1 | \mathbf{z}^1) \end{array} \right] \end{array} \right] \\
 &= -\beta_1 D_{\text{KL}}(q(\mathbf{z}^0 | \mathbf{x}^0) \| p(\mathbf{z}^0)) \\
 &\quad + \mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0)} [-\beta_2 D_{\text{KL}}(q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) \| p(\mathbf{a} | \mathbf{z}^0))] \\
 &\quad + \mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0)} \log p(\mathbf{x}^0 | \mathbf{z}^0) \\
 &\quad + \frac{1}{2} \mathbb{E}_{q(\mathbf{z}^1 | \mathbf{x}^1)} \log p(\mathbf{x}^1 | \mathbf{z}^1) \\
 &\quad + \frac{1}{2} \mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0) q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1)} [-\beta_3 D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a}))] \\
 &\quad + \frac{1}{2} \mathbb{E}_{q(\mathbf{z}^0 | \mathbf{x}^0) q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})} \log p(\mathbf{x}^1 | \mathbf{z}^1).
 \end{aligned} \tag{46}$$

Compared to Equation 9 we showed in the main text, this formula appears much more complex. In particular, it contains a large number of complex expectations ($\mathbb{E} \dots$). We now explain what these expectations imply.

Practical VAE implementations typically compute the expectations of quantities that depend on stochastic variables (e.g., reconstruction loss $\mathbb{E}_{q(\mathbf{z}^1 | \mathbf{x}^1)} \log p(\mathbf{x}^1 | \mathbf{z}^1)$ in Equation 44, which depends

on stochastic \mathbf{z}^1 sampled from $q(\mathbf{z}^1 | \mathbf{x}^1)$) by merely taking a *single* sample of the variables rather than taking multiple samples and computing the mean of the quantities. Recall that $\mathbf{z}^1 = \text{BC}_\tau(\mathbf{l}^1)$ is a noisy function, thus computing the value of \mathbf{z}^1 from \mathbf{l}^1 is equivalent to sampling the value once.

The reason VAE implementations, including ours, follow this approach is that it reduces the runtime, simplifies the code, and empirically works well. If we want to compute an expectation $\mathbb{E}_{q(\mathbf{z}^1 | \mathbf{x}^1)} \log p(\mathbf{x}^1 | \mathbf{z}^1)$ from multiple samples instead, we could do as follows: First, we compute $\mathbf{l}^1 = \text{ENCODE}(\mathbf{x}^1)$, which is deterministic. Next, we compute a noisy function $\text{BC}_\tau(\mathbf{l}^1)$ for K times, i.e., take K samples $\mathbf{z}^{1,1}, \dots, \mathbf{z}^{1,K}$ from a single \mathbf{l}^1 . We then run the decoder K times to obtain K samples of $\tilde{\mathbf{x}}^{1,k} = \text{DECODE}(\mathbf{z}^{1,k})$, compute K reconstruction losses $\log p(\mathbf{x}^{1,k} | \mathbf{z}^{1,k})$ (e.g., scaled squared error in Equation 22), then obtain the mean $\frac{1}{K} \sum_{k=1}^K \log p(\mathbf{x}^{1,k} | \mathbf{z}^{1,k})$. Limiting $K = 1$ avoids these complications.

As a result, practically, Equation 46 for each data sample $\mathbf{x}^0, \mathbf{x}^1$ is implemented as follows. Reordering the formula for readability,

$$\begin{aligned}
 & \log p(\mathbf{x}^0 | \mathbf{z}^0) + \frac{1}{2} \log p(\mathbf{x}^1 | \mathbf{z}^1) + \frac{1}{2} \log p(\mathbf{x}^1 | \mathbf{z}^2) && \text{(Reconstruction losses)} \\
 & - \beta_1 D_{\text{KL}}(q(\mathbf{z}^0 | \mathbf{x}^0) \| p(\mathbf{z}^0)) && \text{(Prior for } \mathbf{z}^0) \\
 & - \beta_2 D_{\text{KL}}(q(\mathbf{a} | \mathbf{x}^0, \mathbf{x}^1) \| p(\mathbf{a} | \mathbf{z}^0)) && \text{(Prior for } \mathbf{a}) \\
 & - \frac{1}{2} \beta_3 D_{\text{KL}}(q(\mathbf{z}^1 = \mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^1 = \mathbf{z}^2 | \mathbf{z}^0, \mathbf{a})). && (47)
 \end{aligned}$$

Reconstruction losses are square errors due to Gaussian assumptions. For KL terms, for example, $D_{\text{KL}}(q(\mathbf{z}^1 | \mathbf{x}^1) \| p(\mathbf{z}^2 | \mathbf{z}^0, \mathbf{a}))$ is obtained by converting the logits $\mathbf{l}^1, \mathbf{l}^2 \in \mathbb{R}$ to probabilities $\mathbf{q}^1 = \text{SIGMOID}(\mathbf{l}^1)$, $\mathbf{q}^2 = \text{SIGMOID}(\mathbf{l}^2)$, then computing the KL divergence as follows.

$$D_{\text{KL}}(\mathbf{q}^1 \| \mathbf{q}^2) = \mathbf{q}^1 \frac{\log \mathbf{q}^1}{\log \mathbf{q}^2} + (1 - \mathbf{q}^1) \frac{\log(1 - \mathbf{q}^1)}{\log(1 - \mathbf{q}^2)}.$$

Comparison to AMA₃: AMA₃ contains a number of ad-hoc differences from AMA₃⁺. For example, AMA₃ uses an *absolute error loss* between two latent vectors \mathbf{z}^1 and \mathbf{z}^2 , instead of using the KL divergence $D_{\text{KL}}(\mathbf{q}^1 \| \mathbf{q}^2)$. This is not theoretically justified, and is inefficient because losses using the sampled values \mathbf{z}^1 introduce noise in the loss function.

Appendix G. Bayesian Interpretation of Back-to-Logit

We can view BTL as a mechanism that injects a new assumption into $p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a})$ by introducing a random variable \mathbf{e} and removing the dependency from \mathbf{e} to \mathbf{z}^0 as follows:

$$\begin{aligned}
 p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a}) &= \sum_{\mathbf{e}} p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{a}, \mathbf{e}) p(\mathbf{e} | \mathbf{z}^0, \mathbf{a}) \\
 &= \sum_{\mathbf{e}} p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{e}) p(\mathbf{e} | \mathbf{a})
 \end{aligned} \tag{48}$$

where $p(\mathbf{e} | \mathbf{a})$ is modeled by EFFECT, and $p(\mathbf{z}^1 | \mathbf{z}^0, \mathbf{e})$ is modeled by m and an addition. This approach is supported by the Bayesian theoretical analysis of batch normalization (Teye et al., 2018).

Obtaining a value of (i.e., sampling the value of) e from \mathbf{a} with $p(e|\mathbf{a})$ is a stochastic process because it is affected by other data points in the same batch, which are randomly sampled in each training iteration. This model can be visualized as a graphical model in Figure G.1.

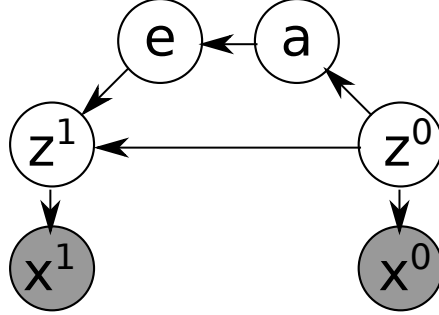


Figure G.1: DAG notation (graphical model) of CSAE.

Appendix H. Proof of Monotonicity

Proof of Theorem Theorem 1:

Proof. For readability, we omit j and assume a 1-dimensional case. Let $e = \text{EFFECT}(\mathbf{a}) \in \mathbb{R}$, which is a constant for a fixed action \mathbf{a} . In the test time, Binary Concrete is replaced by a step function (Section 5.3). The BTL is then simplified to

$$z^{i,1} = \text{STEP}(m(z^{i,0}) + e). \quad (49)$$

The possible values of a pair $(z^{i,0}, z^{i,1})$ is $(0, 0), (0, 1), (1, 0), (1, 1)$. Since both STEP and $m = \text{BN}$ are deterministic at the testing time (Ioffe & Szegedy, 2015), we consider only the deterministic mapping from $z^{i,0}$ to $z^{i,1}$. There are only 4 deterministic mappings from $\{0, 1\}$ to $\{0, 1\}$: $\{(0, 1), (1, 1)\}, \{(1, 0), (0, 0)\}, \{(0, 0), (1, 1)\}, \{(0, 1), (1, 0)\}$. Thus our goal is to show that the last mapping $\{(0, 1), (1, 0)\}$ is impossible in the latent space produced by an ideally trained BTL.

To prove this, first, assume that there is $(z^{i,0}, z^{i,1}) = (0, 1)$ for some index i . Then

$$1 = \text{STEP}(m(0) + e) \Rightarrow m(0) + e > 0 \Rightarrow m(1) + e > 0 \Rightarrow \forall i; m(z^{i,0}) + e > 0. \quad (50)$$

The second step is due to the monotonicity $m(0) < m(1)$. This shows $z^{i,1}$ is constantly 1 regardless of $z^{i,0}$, therefore it proves that $(z^{i,0}, z^{i,1}) = (1, 0)$ cannot happen in any i .

Likewise, if $(z^{i,0}, z^{i,1}) = (1, 0)$ for some index i ,

$$0 = \text{STEP}(m(1) + e) \Rightarrow m(1) + e < 0 \Rightarrow m(0) + e < 0 \Rightarrow \forall i; m(z^{i,0}) + e < 0. \quad (51)$$

Therefore, $z^{i,1} = 0$ regardless of $z^{i,0}$, and thus $(z^{i,0}, z^{i,1}) = (0, 1)$ cannot happen in any i .

Finally, if the data points do not contain $(0, 1)$ or $(1, 0)$, then by assumption they do not coexist. Therefore, the embedding learned by BTL cannot contain $(0, 1)$ and $(1, 0)$ at the same time. \square

Appendix I. Training Curve

To facilitate reproducibility, we show the training and validation loss curves and what to expect. Figure I.1 shows the results of training AMA_4^+ networks on 8-Puzzle. Similar behaviors were observed in other domains and AMA_3^+ networks. We see multiple curves in each subfigure due to multiple hyperparameters. The figure includes three metrics as well as the annealing parameter τ . We could make several observations from these plots.

ELBO: In the successful training curves with lower final ELBO, we do not observe overfitting behavior. ELBO curves show that annealing τ and training the network with higher τ is essential, as the networks stop improving the ELBO after the annealing phase is finished at epoch 1000. Indeed, we performed the same experiments with a lower initial value $\tau_{\min} = 1.0$, and they exhibited significantly less accuracy. This is because lower τ in BC_τ makes the latent SIGMOID function closer to a step function (steeper around 0, flatter away from 0) and produce smaller gradients, as discussed by Jang et al. (2017).

KL divergence against the prior distribution: The group with the lower initial,

$$D_{\text{KL}}(q(\mathbf{z}^{i,0}|\mathbf{x}^0)||p(\mathbf{z}^{i,0})),$$

consists of those with hyperparameter $F = 50$. This is because the KL divergence is the sum across latent dimensions and the KL divergence in each dimension tends to have a similar value initially.

KL divergence for successor prediction: While the overall ELBO loss monotonically decreases, the $D_{\text{KL}}(q(\mathbf{z}^{i,1}|\mathbf{x}^{i,1})||p(\mathbf{z}^{i,2}|\mathbf{z}^{i,0}, \mathbf{a}^i))$ loss representing successor prediction accuracy initially goes up, then goes down. This indicates that the network initially focuses on learning the reconstructions, then moves on to adjust the state representation and the action model in the later stage of the training. The KL divergence continues to improve after the annealing is finished at epoch 1000. The curves seem to indicate that we could train the network even longer in order to achieve better successor prediction loss.

Appendix J. Domain-wise colorization of heuristics figure

An alternative version of Figure 11.1, colored by the domains, is shown in Figure J.1.

Appendix K. Domain-wise colorization of solvability experiments

An alternative version of Figure 11.3, colored by the domains, is shown in Figure K.1.

Appendix L. Plan Validators

We developed a plan validator for each visual planning domain to evaluate the accuracy of the learned representation and the performance of the planner. This section describes the details of these validators.

All validators consist of two stages: State-wise validation (`validate_states`) and transition validation (`validate_transitions`). State-wise validation typically checks for inaccurate reconstructions and violation of domain-specific constraints in the image. Transition validation checks for the correctness of the transitions.

The source code of these validators are available in the source code of Latplan repository ⁶.

6. <https://github.com/guicho271828/latplan/>

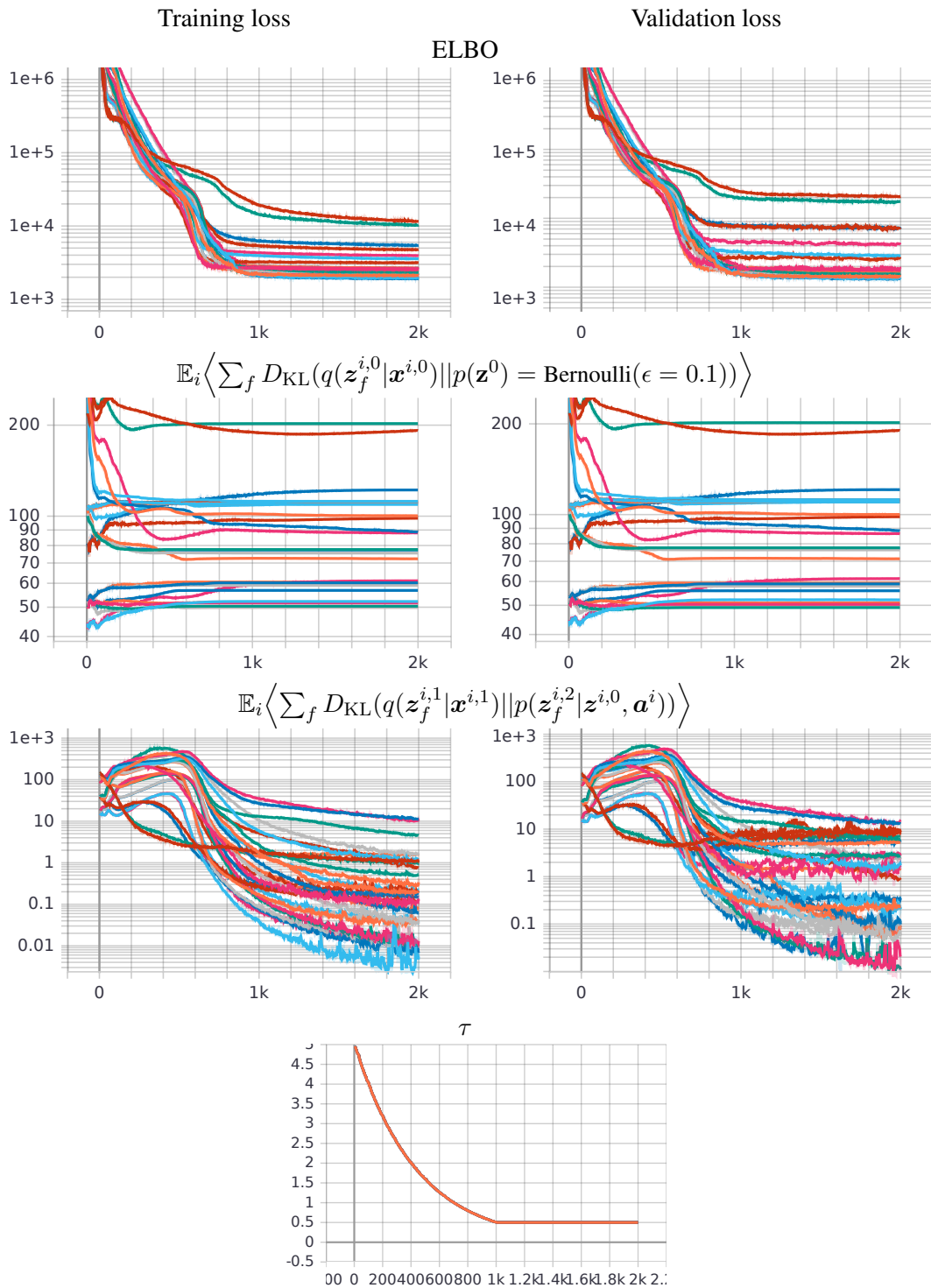


Figure I.1: Training and validation curves. The x -axes show the training epochs. The y -axes are in a logarithmic scale except for τ .

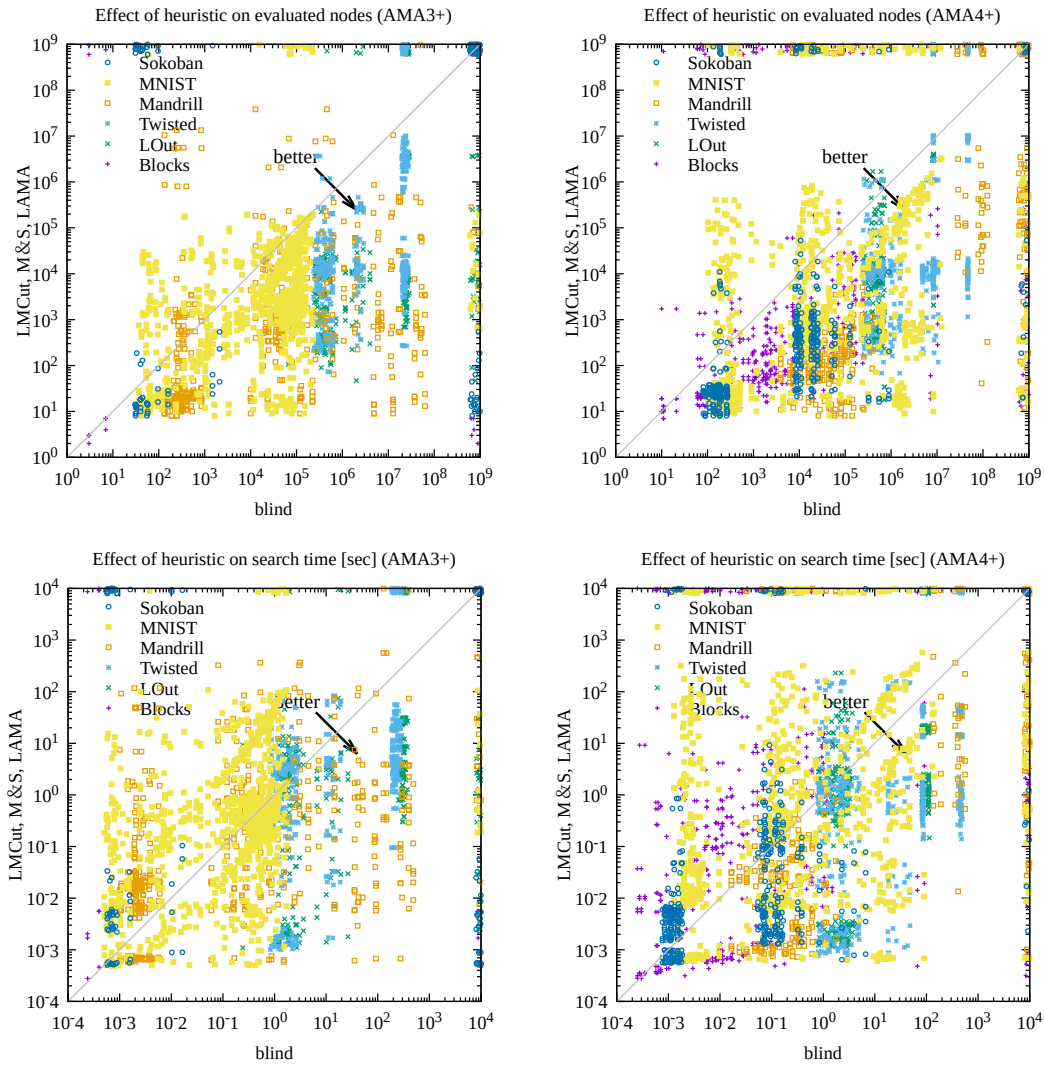


Figure J.1: Domain-wise colorization of Figure 11.1.

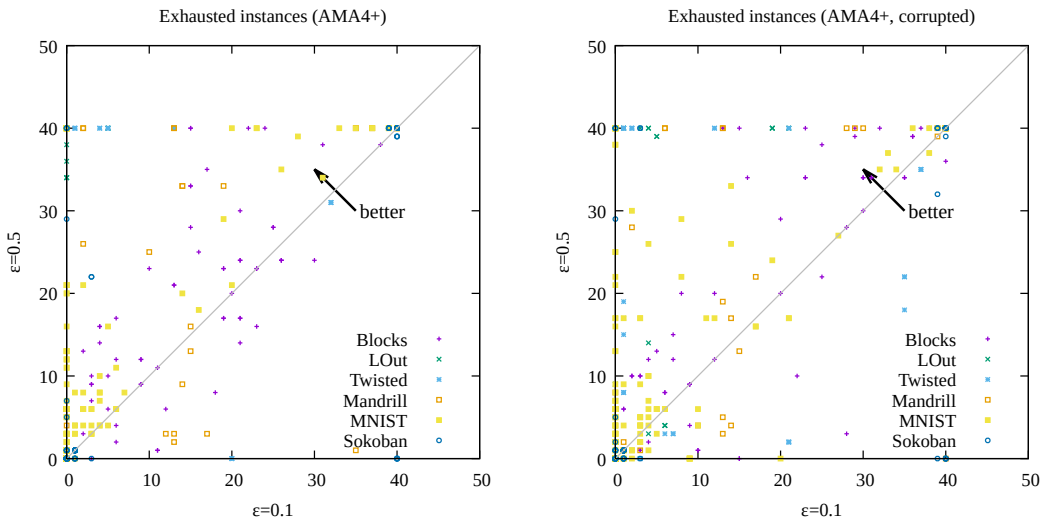


Figure K.1: Domain-wise colorization of Figure 11.3.

L.1 Sliding Tile Puzzle Validators (MNIST, Mandrill)

Since we know the number of rectangular grids in each domain, we first segment each image in the plan into tile patches. For each patch, we compare the mean absolute errors against a fixed set of ground-truth tile patterns which are used to generate the dataset.

After computing the distance, we do not simply pick the nearest tile — it would result in selecting the same ground-truth tile pattern for several patches. Instead, the algorithm searches for the best threshold value θ for the absolute error by a binary search under $0.0 \leq \theta \leq 0.5$. A patch is considered a “match” when the absolute error is below θ . The reason we search for the threshold is because MNIST and Mandrill images have different pixel value ranges. Manually searching for the threshold is not only cumbersome but would also result in an arbitrary decision (for example, we could set a very forgiving / strict value and claim that it works well / does not work well). Another reason is that Sliding Tile Puzzle assumes that there are no duplicate tiles, which can be used to detect invalid states.

The threshold θ is increased/decreased so that it balances the number n_1 of ambiguous patches that matches more than one ground-truth tiles and the number n_2 of patches that fail to match any ground truth. θ is increased when $n_1 < n_2$ and decreased when $n_1 > n_2$. The search is stopped when $|n_1 - n_2| \leq 1$. We consider an image is invalid when $n_1 \neq 0$ and $n_2 \neq 0$.

To validate a transition, we map an image into a compact state configuration representing each tile as an integer id using the threshold value discovered in the previous step. We then check if only two tiles are changed, if the tiles are adjacent, and if one of them is the empty tile (tile id 0).

L.2 LightsOut and Twisted LightsOut Validator

We reuse some of the code from Sliding Tile Puzzle validators with two ground-truth tiles being the “On” tile (indicated by a + sign) and the “Off” tile (a blank black tile).

Unlike MNIST and Mandrill, LightsOut images consist of complete black-and-white pixels, and also multiple patches can match the same ground-truth tile (either “on” or “off”). We therefore set

the threshold to 0.01, a value small enough that any noticeable + marks will be detected as an “On” state of each button. We map each state into a 0/1 configuration representing a puzzle state and validate the transitions.

To handle Twisted LightsOut images, we undo this swirl effect and apply the same validator. The threshold is increased to 0.04 to account for the numerical errors (e.g., anti-aliasing effects) caused by the swirl effect.

L.3 Blocksworld Validator

We use the ground-truth knowledge of the image generator that the image consists of 4 blocks with four different colors (red, green, blue, black).

The first step of the validator parses the image into a compact representation. To do so, it quantifies the image into 3-bit colors, then counts the number of pixels for each color to detect the objects. The most frequent color (gray) is ignored as the background. Also, colors which occur less than 0.1% of the entire image are ignored as a noise.

For each color, we compute the centroid and the width/height to obtain the estimate of the object location. While doing so, we ignore the outliers by first computing the 25% and 75% quantiles, and ignoring the coordinates outside 1.5 times the width of the quantiles. Assume \mathbf{x} is a vector of x -axis or y -axis coordinates for each pixel in the image with the target color of interest. Formally, we perform the following operation for each coordinate axis and each color:

$$\begin{aligned}
 Q_1 &\leftarrow \text{Quantile}(\mathbf{x}, 1/4) \\
 Q_3 &\leftarrow \text{Quantile}(\mathbf{x}, 3/4) \\
 w &\leftarrow Q_3 - Q_1 \\
 l &\leftarrow Q_1 - 1.5 \cdot w \\
 u &\leftarrow Q_3 + 1.5 \cdot w \\
 \text{Return: } &\{x \in \mathbf{x} \mid l < x < u\}.
 \end{aligned}$$

After parsing the image into a set of objects, we check the validity of the state as follows. For each object o , it first searches for objects below it using the half-width $o.w$ (half the width of the bounding box, i.e., dimension from the center), the half-height $o.h$ and the centroid coordinates $o.x, o.y$. An object o_1 is below another object o_2 when $o_1.y > o_2.y$ and in the same tower, i.e., $|o_1.x - o_2.x| < \frac{o_1.w + o_2.w}{2}$. (Note that the pixel coordinates are measured with the top-left being the origin.)

If an object o is a “bottom” object (nothing is directly below it), it compares the bottom edge of the bounding box $o.y + o.h$ with other bottom objects and check if they have the similar y -coordinates, ensuring that none of them are in the mid air. The y -coordinates are considered “similar” when the differences are within half the average height of two objects, i.e., $|o_1.y - o_2.y| < \frac{o_1.h + o_2.h}{2}$.

Otherwise, there are other objects below the object o_1 . It collects all objects below it, and extracts the object o_2 with the smallest y -coordinate. It checks if o_1 and o_2 are in direct contact by checking if y -coordinate difference is around the sum of both heights, i.e., $(o_1.h + o_2.h) \cdot 0.5 < |o_1.y - o_2.y| < (o_1.h + o_2.h) \cdot 1.5$.

To check the validity of the transitions, it first looks for unaffected objects, and check if exactly one object is moved. If this is satisfied, it further checks if the moved object is a “top” object both before and after the transition.

L.4 Sokoban Validator

The sokoban validator partially shares code with LightsOut, therefore it does not use the binary search to find the threshold. Moreover, it does not use the threshold and it assigns each patch to the closest ground-truth panel (wall, stone, player, goal, clear, stone-at-goal) available in PDDLgym.

To validate the state, we use the fact that the environment was generated from a single PDDL instance (p006-microban-sequential). We enumerate the entire state space with Dijkstra search and store the ground-truth configurations of the states into an archive. A state is valid when the same configuration is found in an archive.

To validate the transition, we can’t use the same approach used for states because the archive contains all states but not all transitions — Since the enumeration uses a Dijkstra search to enumerate states, there is only one transition for each state as a destination. We reimplemented the precondition and the effects of each PDDL action (move, push-to-goal, push-to-nongoal) for our representation.

Appendix M. Additional Experiments for Towers of Hanoi Domain (Negative Results)

As a failure case of Latplan, we describe **Towers of Hanoi** (ToH) domain and the experiments we performed. A (d, t) -ToH domain with t towers and d disks has t^d states and $t^{d+1} - 2$ transitions in total. The optimal solution for moving a complete tower takes $2^d - 1$ steps if $t = 3$, but we parameterized our image generator so that it can generate images with more than 3 towers. Each input image has a dimension of $(H, W, C) = (hd, wt, 3)$ (height, width, color channel), where each disk is presented as a $h \times w$ pixel rectangle ($h = 1, w = 4$) with distinct colors. The validator of ToH images is identical to that of the Sliding Tile Puzzles except for the fixed patterns that are matched against each image patch.

We generated several datasets with increasing generator parameters, namely $(d, t) = (4, 4), (3, 9), (4, 9), (5, 9)$. The number of states in each configurations is 256, 729, 6561, 59049. We generated 20 problem instances for each configuration, where the number of optimal solution length is 5 for $(3, 9)$ because it lacks 7-step optimal plans⁷, and is 7 for $(4, 4), (4, 9),$ and $(5, 9)$ because they lack 14-step optimal plans. While several 3-disk instances were solved successfully, no valid plans were produced with larger number of disks, as seen in Table M.1. Visualizations of successful and failure cases are shown in Figure N.9.

The apparent failure of Latplan in ToH is surprising because its state space tends to be smaller and thus is “simpler” than the other domains we tested. It could be because the size of the dataset is too small for deep-learning based approaches, or that the images are horizontally long and the 5x5 convolutions are not adequate. It could also be due to the lack of visual cues – however, alternative image renderers did not change the results (e.g., black/white tiles with varying disk size, thicker disks, disks painted with hand-crafted patterns.) Lastly, there may be imbalance in the visual features because the smallest disk rarely appears in the top region of the image. For d -disk,

7. The diameter of the state space graph is less than 7.

t -tower ToH domain, the smallest disk can appear only t times in the top row among t^d states, which is exponentially rare in a uniformly randomly sampled dataset. As a result, pixel data in the top rows are biased toward being gray (the background color). We suspect deep-learning may have an assumption that all of the factors (i.e., fluents that distinguish the states) are seen often enough, which is not satisfied in this dataset.

domain	Blind			LAMA			LMCut			M&S					
	kltune			default			kltune			default					
	f	v	o	f	v	o	f	v	o	f	v	o	f	v	o
AMA ₃ ⁺															
$(d, t) = (4, 4)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(d, t) = (3, 9)$	3	2	0	0	0	0	0	0	0	3	2	0	0	0	0
$(d, t) = (4, 9)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(d, t) = (5, 9)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
AMA ₄ ⁺															
$(d, t) = (4, 4)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(d, t) = (3, 9)$	20	12	8	13	5	0	17	1	0	20	12	7	13	5	0
$(d, t) = (4, 9)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(d, t) = (5, 9)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table M.1: The number of solutions found (f), valid solutions (v), and optimal solutions (o) for (d, t) -Towers of Hanoi domains using AMA₃⁺ and AMA₄⁺.

Appendix N. Examples of Planning Results

Finally, Figures N.1-N.9 show several examples of successful plans (optimal and suboptimal) as well as invalid plans in order to illustrate how Latplan could behave or fail.

We did not see any qualitative differences between the visualized results obtained by various configurations, conditioned by the same validity status of the plan (invalid/suboptimal/optimal). For example, we did not find any particular differences between the sets of invalid results regardless of the choice of AMA₃⁺, AMA₄⁺, $\epsilon = 0.1$ (kltune), $\epsilon = 0.5$ (default), or other hyperparameter differences. All invalid results look similarly invalid (albeit individual variations), except the statistical differences observed in the overall number of invalid plans as discussed in Section 11. Similarly, suboptimal results look similarly suboptimal and no configuration particularly / visually / qualitatively stands out. Optimal results also looks similarly optimal. One exception is the case of using LAMA, in which case plans tend to be much longer than in the other configurations.

Due to this and the obvious space reasons, we do not attempt to exhaustively enumerate the results of all configurations. We instead hand-picked several examples without much consideration on the configuration. **Thus, although we describe each configuration in the description, it is not our intention to compare these results and make any claims based on them.**

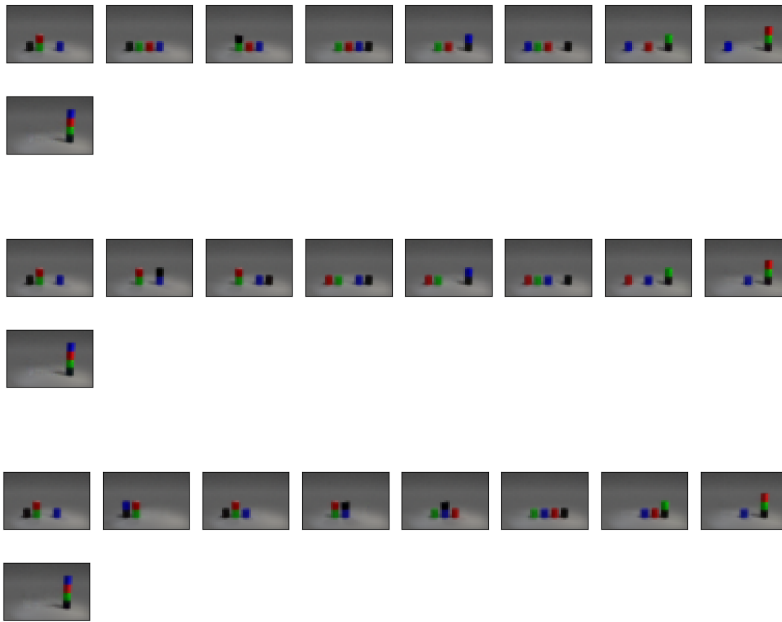


Figure N.1: Examples of valid suboptimal plans in Blocksworld obtained from AMA_4^+ with default priors ($\epsilon = 0.5$). The first and the second plans were generated by the same network, but with LMcute and M&S heuristics. The third plan was generated by M&S with another training result with different hyperparameters. They all resulted in different plans because different heuristics and state encoding result in a different node expansion order. See also: Section 11.4 discusses why A^* + admissible heuristics can generate suboptimal visual plans.

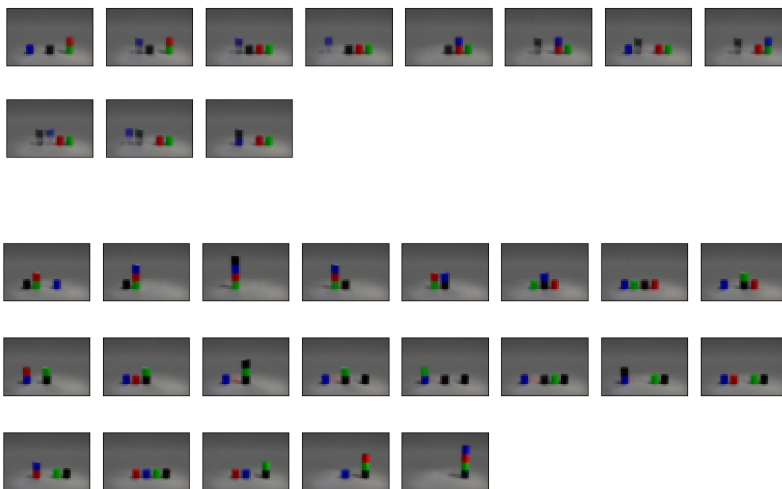


Figure N.2: Examples of invalid plans in Blocksworld for a 7-steps problem and a 14-steps problem. In the first instance, floating objects can be observed. In the second instance, the colors of some blocks change and some objects are duplicated. (The first instance was generated by AMA_4^+ , default prior, LAMA. The second instance was produced by AMA_4^+ , $\epsilon = 0.1$ prior, M&S.)

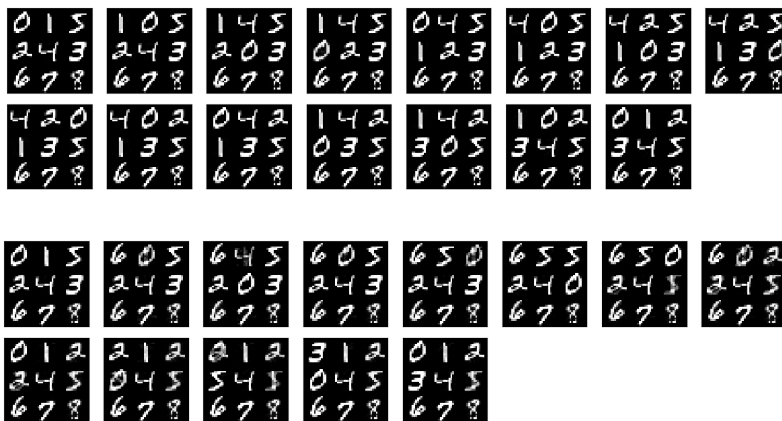


Figure N.3: Examples from MNIST 8-puzzle. The first plan is optimal (14 steps). The second plan is invalid due to the duplicated “6” tiles in the second step. (The first instance was generated by AMA_3^+ , default prior, blind. The second instance was produced by AMA_4^+ , $\epsilon = 0.1$ prior, blind.)



Figure N.4: Examples from Mandrill 15-puzzle. The first plan is optimal (7 steps). The second plan is invalid because it is hard to distinguish different tiles. (Both instances were generated by AMA_3^+ , $\epsilon = 0.1$ prior, blind but with different hyperparameters (F, β_1, β_3) .)

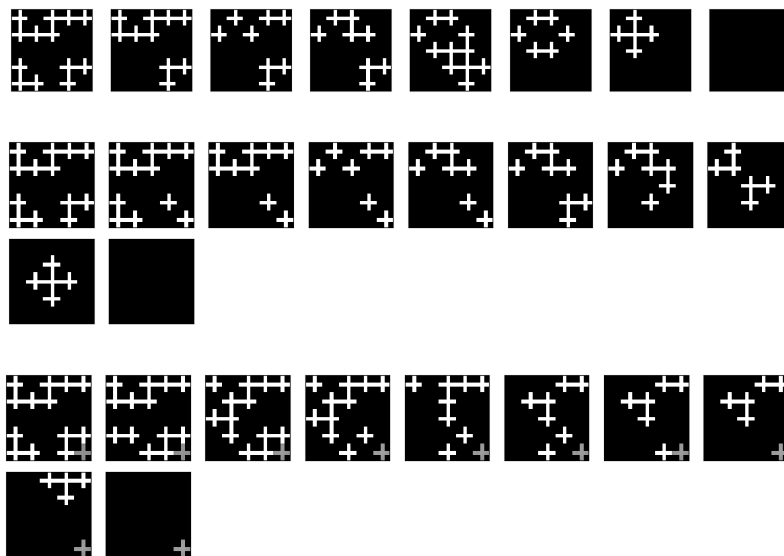


Figure N.5: Examples from LightsOut. The first plan is optimal (7 steps). The second plan is sub-optimal (9 steps) because it hits the bottom-right button twice (step 1 and 5), which is unnecessary (it reverses the effect). The third plan is invalid because of the bottom-right tile being above the threshold and the init/goal states do not match. (The first instance was generated by AMA_3^+ , $\epsilon = 0.1$ prior, blind. The second instance was generated by the same network combined with LAMA. The third instance was produced by AMA_3^+ , $\epsilon = 0.1$ prior, blind with a different hyperparameter.)



Figure N.6: An optimal plan generated for Twisted LightsOut (7 steps). (Generated by AMA_3^+ , $\epsilon = 0.1$ prior, blind.)

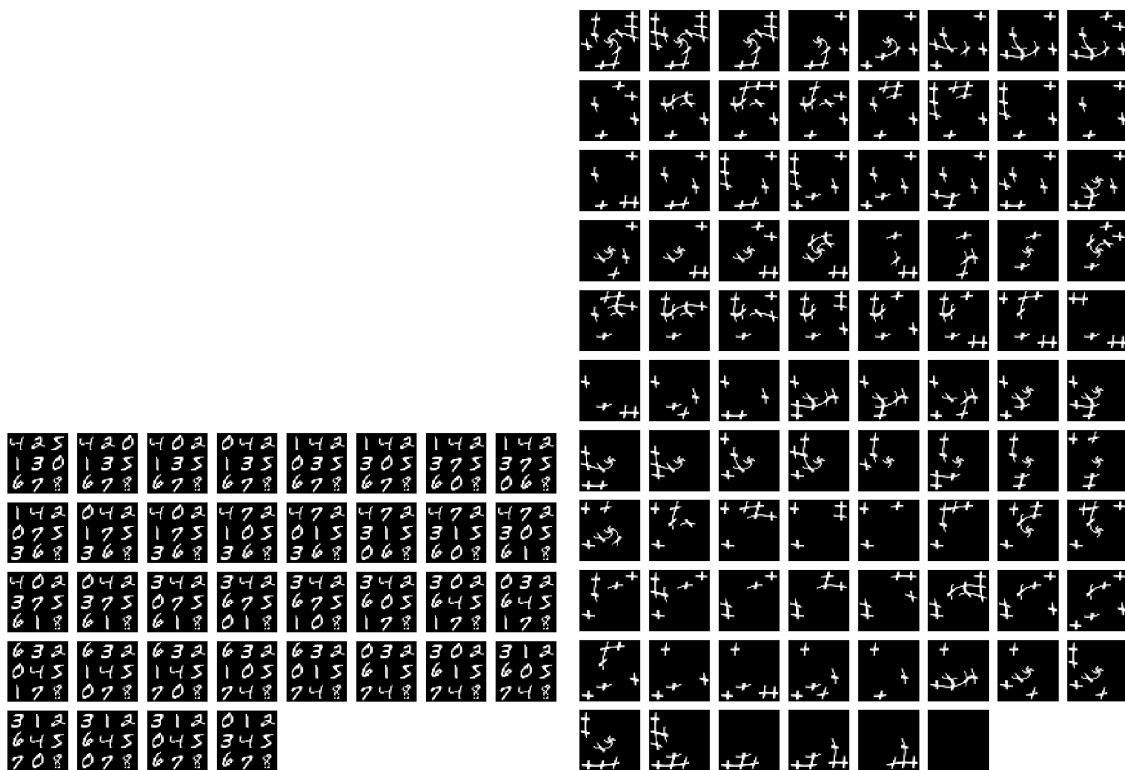


Figure N.7: Suboptimal plans generated for 7-steps instances of MNIST 8-Puzzle and Twisted LightsOut by LAMA. Plans generated by LAMA tends to be hugely suboptimal compared to the suboptimal visual plans generated by A^* and admissible heuristics (e.g., blind, LMcut, M&S). See also: Section 11.4 discusses why A^* + admissible heuristics can generate suboptimal visual plans. (The first instance was generated by AMA_4^+ , default prior, LAMA. The second instance was generated by AMA_3^+ , $\epsilon = 0.1$ prior, LAMA.)



Figure N.8: Examples from Sokoban. The first plan is optimal (7 steps for a 7-steps instance). The second plan is suboptimal (16 steps for a 14-steps instance). The third plan is invalid because it duplicates the player. (The first instance was generated by AMA_3^+ , $\epsilon = 0.1$ prior, blind. The second instance was generated by AMA_4^+ , $\epsilon = 0.1$ prior, blind. The third instance was produced by AMA_4^+ , default prior, M&S.)

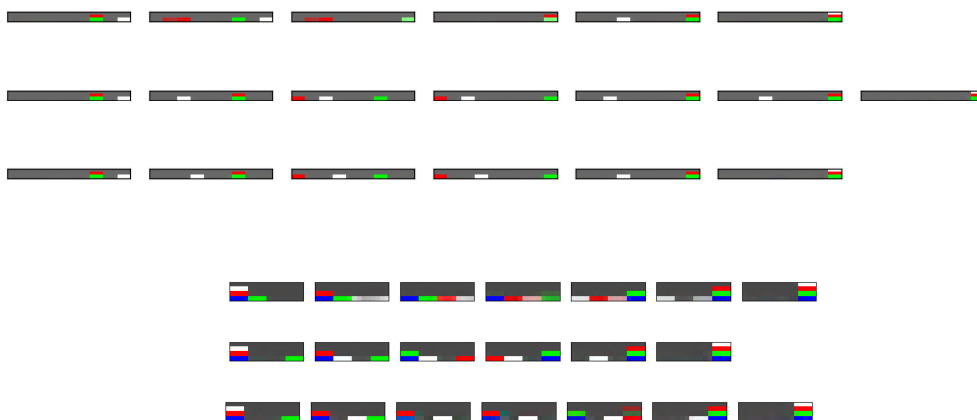


Figure N.9: An invalid plan, a valid suboptimal plan and an optimal plan generated in (3, 9)-ToH, and 3 invalid plans for (4, 4)-ToH. (Each example was generated by AMA_4^+ , $\epsilon = 0.1$ prior, blind with different hyperparameters (F, β_1, β_3) .)

References

- Ahmetoglu, A., Seker, M. Y., Sayin, A., Bugur, S., Piater, J. H., Öztop, E., & Ugur, E. (2020). Deep-Sym: Deep Symbol Generation and Rule Learning from Unsupervised Continuous Robot Interaction for Planning. *CoRR*, *abs/2012.02532*.
- Aineto, D., Jiménez, S., & Onaindia, E. (2018). Learning STRIPS Action Models with Classical Planning. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.
- Alcázar, V., Borrajo, D., Fernández, S., & Fuentetaja, R. (2013). Revisiting Regression in Planning. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.
- Amado, L., Pereira, R. F., Aires, J. P., Magnaguagno, M. C., Granada, R., & Meneguzzi, F. (2018). Goal Recognition in Latent Space. In *Proc. of International Joint Conference on Neural Networks (IJCNN)*.
- Andersen, G., & Konidaris, G. (2017). Active Exploration for Learning Symbolic Representations. In *Advances in Neural Information Processing Systems*, pp. 5009–5019.
- Anderson, M., & Feil, T. (1998). Turning lights out with linear algebra. *Mathematics Magazine*, *71*(4), 300–303.
- Arfae, S. J., Zilles, S., & Holte, R. C. (2010). Bootstrap learning of heuristic functions. In Felner, A., & Sturtevant, N. R. (Eds.), *Proc. of Annual Symposium on Combinatorial Search*. AAAI Press.
- Arfae, S. J., Zilles, S., & Holte, R. C. (2011). Learning Heuristic Functions for Large State Spaces. *Artificial Intelligence*, *175*(16-17), 2075–2098.
- Arora, A., Fiorino, H., Pellier, D., Etivier, M., & Pesty, S. (2018). A Review of Learning Planning Action Models. *Knowledge Engineering Review*, *33*.
- Asai, M. (2018). Photo-realistic blocksworld dataset. *CoRR*, *abs/1812.01818*.
- Asai, M. (2020). Unsuccessful Neural-Symbolic Descriptive Action Model from Images: The Search for STRIPS. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling(KEPS)*.
- Asai, M., & Fukunaga, A. (2018). Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In McIlraith, S. A., & Weinberger, K. Q. (Eds.), *Proc. of AAAI Conference on Artificial Intelligence*, pp. 6094–6101. AAAI Press.
- Asai, M., & Kajino, H. (2019). Towards Stable Symbol Grounding with Zero-Suppressed State AutoEncoder. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.
- Asai, M., & Muise, C. (2020). Learning Neural-Symbolic Descriptive Planning Models via Cube-Space Priors: The Voyage Home (to STRIPS). In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.
- Bäckström, C., & Nebel, B. (1995). Complexity Results for SAS+ Planning. *Computational Intelligence*, *11*(4), 625–655.

- Barbu, A., Narayanaswamy, S., & Siskind, J. M. (2010). Learning Physically-Instantiated Game Play through Visual Observation. In *Proc. of IEEE International Conference on Robotics and Automaton (ICRA)*, pp. 1879–1886.
- Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *J. Artif. Intell. Res. (JAIR)*, 47, 253–279.
- Bengio, Y., Léonard, N., & Courville, A. C. (2013). Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *CoRR*, abs/1308.3432.
- Bieszczad, A., & Kuchar, S. (2015). Neurosolver Learning to Solve Towers of Hanoi Puzzles. In *Proc. of International Joint Conference on Computational Intelligence (IJCCI)*, Vol. 3, pp. 28–38. IEEE.
- Bonet, B., & Geffner, H. (2001). Planning as Heuristic Search. *Artificial Intelligence*, 129(1), 5–33.
- Bonet, B., & Geffner, H. (2020). Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *Proc. of European Conference on Artificial Intelligence*.
- Botea, A., & Braghin, S. (2015). Contingent versus Deterministic Plans in Multi-Modal Journey Planning. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 268–272.
- Bowman, S., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., & Bengio, S. (2016). Generating Sentences from a Continuous Space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 10–21.
- Chen, X., Duan, Y., Houthoofd, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pp. 2172–2180.
- Chollet, F., et al. (2015). Keras. <https://keras.io>.
- Chrupa, L., & Siddiqui, F. H. (2015). Exploiting Block Deordering for Improving Planners Efficiency. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.
- Chrupa, L., Vallati, M., & McCluskey, T. L. (2015). On the Online Generation of Effective Macro-Operators. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.
- Cresswell, S., & Gregory, P. J. (2011). Generalised Domain Model Acquisition from Action Traces. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*.
- Cresswell, S., McCluskey, T. L., & West, M. (2009). Acquisition of object-centred domain models from planning examples. In *Nineteenth International Conference on Automated Planning and Scheduling*.
- Cresswell, S., McCluskey, T. L., & West, M. M. (2013). Acquiring planning domain models using LOCM. *Knowledge Engineering Review*, 28(2), 195–213.
- Culberson, J. (1998). Sokoban is PSPACE-complete. In *Proceedings in Informatics 4, International Conference on Fun with Algorithms*, pp. 65–76. Carleton Scientific.
- Dagum, P., & Chavez, R. M. (1993). Approximating Probabilistic Inference in Bayesian Belief Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3), 246–255.

- Dagum, P., & Luby, M. (1997). An Optimal Approximation Algorithm for Bayesian Inference. *Artificial Intelligence*, 93(1), 1–28.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. IEEE.
- Deng, L., Hinton, G. E., & Kingsbury, B. (2013). New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview. In *Proc. of IEEE Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603. IEEE.
- Deng, L., Seltzer, M. L., Yu, D., Acero, A., Mohamed, A.-r., & Hinton, G. E. (2010). Binary Coding of Speech Spectrograms using a Deep Auto-Encoder. In *Interspeech*, pp. 1692–1695. Citeseer.
- Dinh, L., & Dumoulin, V. (2014). Training Neural Bayesian Nets..
- Edelkamp, S. (2012). PDB or not PDB? - that's the Question. A Tribute to Blind Search Planning. In *Festivus in the 22nd International Conference on Automated Planning and Scheduling(ICAPS)*.
- Ferber, P., Helmert, M., & Hoffmann, J. (2020). Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In *Proc. of European Conference on Artificial Intelligence*, pp. 2346–2353.
- Fikes, R. E., & Nilsson, N. J. (1972). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3), 189–208.
- Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.(JAIR)*, 20, 61–124.
- Frances, G., Ramírez, M., Lipovetzky, N., & Geffner, H. (2017). Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4294–4301.
- Fukushima, K. (1980). Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 36(4).
- Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., & Schneider, M. (2011). Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, 24(2), 107–124.
- Glorot, X., & Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proc. of International Conference on Artificial Intelligence and Statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings.
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., & Bengio, Y. (2014). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680.
- 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.
- Gregory, P. J., & Cresswell, S. (2015). Domain Model Acquisition in the Presence of Static Relations in the LOP System. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*, pp. 97–105.

- Gumbel, E. J., & Lieblein, J. (1954). *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*, Vol. 33. US Government Printing Office.
- Ha, D., & Schmidhuber, J. (2018). World Models. *CoRR*, *abs/1803.10122*.
- Harman, H., & Simoons, P. (2020). Learning Symbolic Action Definitions from Unlabelled Image Pairs. In *2020 The 4th International Conference on Advances in Artificial Intelligence*, pp. 72–78.
- Harnad, S. (1990). The Symbol Grounding Problem. *Physica D: Nonlinear Phenomena*, *42*(1-3), 335–346.
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification. In *Proc. of the IEEE International Conference on Computer Vision*, pp. 1026–1034.
- Helmert, M. (2006). The Fast Downward Planning System. *J. Artif. Intell. Res.(JAIR)*, *26*, 191–246.
- Helmert, M., & Domshlak, C. (2009). Landmarks, Critical Paths and Abstractions: What’s the Difference Anyway?. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.
- Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of ACM*, *61*(3), 16:1–16:63.
- Higgins, I., Matthey, L., Pal, A., et al. (2017). β -VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *Proc. of the International Conference on Learning Representations*.
- Hinton, G. E., Nitish, S., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving Neural Networks by Preventing Co-adaptation of Feature Detectors. *CoRR*, *abs/1207.0580*.
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507.
- Ho, T. K. (1998). The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(8).
- Hoffmann, J., & Nebel, B. (2001). The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)*, *14*, 253–302.
- Hopfield, J. J., & Tank, D. W. (1985). ”Neural” Computation of Decisions in Optimization Problems. *Biological Cybernetics*, *52*(3), 141–152.
- Howey, R., & Long, D. (2003). VAL’s Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *Proceedings of ICAPS Workshop on the IPC*, pp. 28–37. Citeseer.
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. R., & Blei, D. M. (Eds.), *Proc. of the International Conference on Machine Learning*, Vol. 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org.
- Jang, E., Gu, S., & Poole, B. (2017). Categorical Reparameterization with Gumbel-Softmax. In *Proc. of the International Conference on Learning Representations*.

- Jiménez, S., de la Rosa, T., Fernández, S., Fernández, F., & Borrajo, D. (2012). A Review of Machine Learning for Automated Planning. *Knowledge Engineering Review*, 27(4), 433.
- Junghanns, A., & Schaeffer, J. (2001). Sokoban: Enhancing General Single-Agent Search Methods using Domain Knowledge. *Artificial Intelligence*, 129(1), 219–251.
- Kaiser, L. (2012). Learning Games from Videos Guided by Descriptive Complexity. In *Proc. of AAAI Conference on Artificial Intelligence*.
- Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., & Michalewski, H. (2020). Model Based Reinforcement Learning for Atari. In *Proc. of the International Conference on Learning Representations*.
- Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. In *Proc. of the International Conference on Learning Representations*.
- Konidaris, G., Kaelbling, L. P., & Lozano-Pérez, T. (2014). Constructing Symbolic Representations for High-Level Planning. In *Proc. of AAAI Conference on Artificial Intelligence*, pp. 1932–1938.
- Konidaris, G., Kaelbling, L. P., & Lozano-Pérez, T. (2015). Symbol Acquisition for Probabilistic High-Level Planning. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3619–3627.
- Konidaris, G., Kaelbling, L. P., & Lozano-Pérez, T. (2018). From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *J. Artif. Intell. Res.(JAIR)*, 61, 215–289.
- Korf, R. E. (1985). Macro-Operators: A Weak Method for Learning. *J. Artif. Intell. Res.(JAIR)*, 26(1), 35–77.
- Koul, A., Fern, A., & Greydanus, S. (2019). Learning Finite State Representations of Recurrent Policy Networks. In *Proc. of the International Conference on Learning Representations*.
- Kurutach, T., Tamar, A., Yang, G., Russell, S. J., & Abbeel, P. (2018). Learning Plannable Representations with Causal InfoGAN. In *Advances in Neural Information Processing Systems*.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4).
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proc. of the IEEE*, 86(11), 2278–2324.
- Li, J., Luong, M.-T., & Jurafsky, D. (2015). A Hierarchical Neural Autoencoder for Paragraphs and Documents. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.
- Lindsay, A., Read, J., Ferreira, J. F., Hayton, T., Porteous, J., & Gregory, P. J. (2017). Framer: Planning Models from Natural Language Action Descriptions. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.
- Lipovetzky, N. (2017). Best-First Width Search: Exploration and Exploitation in Classical Planning . In *Proc. of AAAI Conference on Artificial Intelligence*.

- Liu, L., et al. (2019). On the Variance of the Adaptive Learning Rate and Beyond. *CoRR*, *abs/1908.03265*.
- Maddison, C. J., Mnih, A., & Teh, Y. W. (2017). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Proc. of the International Conference on Learning Representations*.
- Maddison, C. J., Tarlow, D., & Minka, T. (2014). A* sampling. In *Advances in Neural Information Processing Systems*, pp. 3086–3094.
- McDermott, D. V. (2000). The 1998 AI Planning Systems Competition. *AI Magazine*, *21*(2), 35–55.
- Miglani, S., & Yorke-Smith, N. (2020). NLtoPDDL: One-Shot Learning of PDDL Models from Natural Language Process Manuals. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling(KEPS)*. ICAPS.
- 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., & Steedman, M. (2012). Learning STRIPS Operators from Noisy and Incomplete Observations. In *Proc. of the International Conference on Uncertainty in Artificial Intelligence*, pp. 614–623.
- Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT press.
- Newell, A. (1980). Physical Symbol Systems. *Cognitive science*, *4*(2), 135–183.
- Newell, A., & Simon, H. A. (1976). Computer Science as Empirical Inquiry: Symbols and Search. *Commun. ACM*, *19*(3), 113–126.
- Nilsson, N. J. (2007). The Physical Symbol System Hypothesis: Status and Prospects. In *50 Years of Artificial Intelligence*, pp. 9–17. Springer.
- Nix, D. A., & Weigend, A. S. (1994). Estimating the Mean and Variance of the Target Probability Distribution. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'94)*, Vol. 1, pp. 55–60. IEEE.
- Payan, C. (1992). On the Chromatic Number of Cube-Like Graphs. *Discrete mathematics*, *103*(3).
- Ramírez, M., & Geffner, H. (2009). Plan Recognition as Planning. In *Proc. of AAAI Conference on Artificial Intelligence*.
- Reinefeld, A. (1993). Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 248–253.
- Reiter, R. (1981). On Closed World Data Bases. In *Readings in Artificial Intelligence*, pp. 119–140. Elsevier.
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, pp. 91–99.
- Richter, S., & Westphal, M. (2010). The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.(JAIR)*, *39*(1), 127–177.

- Rodriguez, I. D., Bonet, B., Romero, J., & Geffner, H. (2021). Learning First-Order Representations for Planning from Black-Box States: New Results. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*.
- Roth, D. (1996). On the Hardness of Approximate Reasoning. *Artificial Intelligence*, 82(1-2), 273–302.
- Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (1995). *Artificial Intelligence: A Modern Approach*, Vol. 2. Prentice hall Englewood Cliffs.
- Satzger, B., & Kramer, O. (2013). Goal Distance Estimation for Automated Planning using Neural Networks and Support Vector Machines. *Natural Computing*, 12(1), 87–100.
- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., & Silver, D. (2019). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *CoRR*, [abs/1911.08265](https://arxiv.org/abs/1911.08265).
- Settles, B. (2012). *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.
- Shen, W., Trevizan, F., & Thiébaux, S. (2020). Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*, Vol. 30, pp. 574–584.
- 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.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K., & Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR*, [abs/1712.01815](https://arxiv.org/abs/1712.01815).
- Silver, T., & Chitnis, R. (2020). PDDL Gym: Gym Environments from PDDL Problems..
- Steels, L. (2008). 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.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.
- Taddeo, M., & Floridi, L. (2005). Solving the Symbol Grounding Problem: A Critical Review of Fifteen Years of Research. *Journal of Experimental & Theoretical Artificial Intelligence*, 17(4), 419–445.
- Teye, M., Azizpour, H., & Smith, K. (2018). Bayesian Uncertainty Estimation for Batch Normalized Deep Networks. In *Proc. of the International Conference on Machine Learning*, pp. 4907–4916. PMLR.
- Thayer, J. T., Dionne, A., & Ruml, W. (2011). Learning Inadmissible Heuristics during Search. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*, Vol. 21.
- Toyer, S., Trevizan, F., Thiébaux, S., & Xie, L. (2018). Action Schema Networks: Generalised Policies with Deep Learning. In *Proc. of AAAI Conference on Artificial Intelligence*, Vol. 32.

- Ugur, E., & Piater, J. H. (2015). Bottom-up Learning of Object Categories, Action Effects and Logical Rules: From Continuous Manipulative Exploration to Symbolic Planning. In *Proc. of IEEE International Conference on Robotics and Automaton (ICRA)*, pp. 2627–2633. IEEE.
- Vahdat, A., Andriyash, E., & Macready, W. G. (2018a). DVAE#: Discrete variational autoencoders with relaxed Boltzmann priors. In *Advances in Neural Information Processing Systems*.
- Vahdat, A., Macready, W. G., Bian, Z., & Khoshaman, A. (2018b). DVAE++: Discrete Variational Autoencoders with Overlapping Transformations. *CoRR*, *abs/1802.04920*.
- van den Oord, A., Vinyals, O., et al. (2017). Neural Discrete Representation Learning. In *Advances in Neural Information Processing Systems*.
- Vazirani, V. V. (2013). *Approximation Algorithms*. Springer Science & Business Media.
- Vera, A. H., & Simon, H. A. (1993). Situated Action: A Symbolic Interpretation. *Cognitive science*, *17*(1), 7–48.
- Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders. In *Proc. of the International Conference on Machine Learning*, pp. 1096–1103. ACM.
- Wang, J. X., King, M., Porcel, N. P. M., Kurth-Nelson, Z., Zhu, T., Deck, C., Choy, P., Cassin, M., Reynolds, M., Song, H. F., Buttimore, G., Reichert, D. P., Rabinowitz, N. C., Matthey, L., Hassabis, D., Lerchner, A., & Botvinick, M. (2021). Alchemy: A Benchmark and Analysis Toolkit for Meta-Reinforcement Learning Agents. In *NIPS, Datasets and Benchmarks Track (Round 2)*.
- Wang, X. (1994). Learning planning operators by observation and practice. In Hammond, K. J. (Ed.), *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pp. 335–340. AAAI.
- Wang, X. (1995). Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. In *Proc. of the International Conference on Machine Learning*, pp. 549–557. Elsevier.
- Weber, A. G. (1997). The USC-SIPI image database version 5. *USC-SIPI Report*, *315*(1).
- Yang, Q., Wu, K., & Jiang, Y. (2007). Learning Action Models from Plan Examples using Weighted MAX-SAT. *Artificial Intelligence*, *171*(2-3), 107–143.
- Yoon, S. W., Fern, A., & Givan, R. (2006). Learning Heuristic Functions from Relaxed Plans.. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*, Vol. 2, p. 3.
- Yoon, S. W., Fern, A., & Givan, R. (2008). Learning Control Knowledge for Forward Search Planning.. *Journal of Machine Learning Research*, *9*(4).
- Zhang, A., McAllister, R. T., Calandra, R., Gal, Y., & Levine, S. (2021). Learning Invariant Representations for Reinforcement Learning without Reconstruction. In *Proc. of the International Conference on Learning Representations*.
- Zhuo, H. H. (2015). Crowdsourced Action-Model Acquisition for Planning. In *Proc. of AAAI Conference on Artificial Intelligence*, Vol. 29.

- Zhuo, H. H., & Kambhampati, S. (2013). Action-Model Acquisition from Noisy Plan Traces. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.
- Zhuo, H. H., Peng, J., & Kambhampati, S. (2019). Learning Action Models from Disordered and Noisy Plan Traces. *CoRR*, *abs/1908.09800*.
- Zhuo, H. H., & Yang, Q. (2014). Action-Model Acquisition for Planning via Transfer Learning. *Artificial Intelligence*, *212*, 80–103.