Unsuccessful Neural-Symbolic Descriptive Action Model from Images: The Search for STRIPS

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

Abstract

Recent work on Neural-Symbolic systems that learn the discrete planning model from images has opened a promising direction for expanding the scope of Automated Planning and Scheduling to the raw, noisy data. However, previous work only partially addressed this problem, utilizing the black-box neural model as the successor generator. In this work, we propose Double-Stage Action Model Acquisition (DSAMA), a system that obtains a descriptive PDDL action model with explicit preconditions and effects over the propositional variables unsupervised-learned from images. DSAMA trains a set of Random Forest rule-based classifiers and compiles them into logical formulae in PDDL. While we obtained a competitively accurate PDDL model compared to a black-box model, we observed that the resulting PDDL is too large and complex for the state-of-the-art standard planners such as Fast Downward primarily due to the PDDL-SAS+ translator bottleneck. From this negative result, we argue that this translator bottleneck cannot be addressed just by using a different, existing rule-based learning method, and we point to the potential future directions.

1 Introduction

Recently, Latplan system (Asai and Fukunaga 2018) successfully connected a subsymbolic neural network (NN) system and a symbolic Classical Planning system to solve various visually presented puzzle domains. The system consists of four parts: 1) The State AutoEncoder (SAE) neural network learns a bidirectional mapping between images and propositional states with unsupervised training. 2) Action Model Acquisition module generates an action model from the propositional state transitions encoded from the images. 3) Classical Planning module solves the problem in the propositional state space with the learned action model. 4) The decoding module maps the propositional plan back to an image sequence. The proposed framework opened a promising direction for applying a variety of symbolic methods to the real world — For example, the search space generated by Latplan was shown to be compatible with a symbolic Goal Recognition system (Amado et al. 2018a; 2018b). Several variations replacing the state encoding modules have also been proposed: Causal InfoGAN (Kurutach et al. 2018) uses a GAN-based framework, First-Order SAE (Asai 2019) learns the First Order Logic symbols (instead of the propositional ones), and Zero-Suppressed SAE (Asai and Kajino 2019, ZSAE) addresses the Symbol Stability issue of the regular SAE with ℓ_1 regularization.

```
(:action a0 :parameters () :precondition [D0]
 :effect (and (when [E00]             (z0))
              (when (not [E00]) (not (z0)))
              (when [E01]             (z1))
              (when (not [E01]) (not (z1))) ...))
```

Figure 1: An example DSAMA compilation result for the first action (i.e. `a0`) generated from the image planning domain. `[...]` consists of a negation normal form compiled from a Random Forest (Ho 1998, RF).

Despite these efforts, Latplan is missing a critical feature of the traditional Classical Planning systems: The use of State-of-the-Art heuristic functions. The main reason behind this limitation is the lack of *descriptive* action model consisting of logical formula for the preconditions and the effects, which allows the heuristics to exploit its causal structures. *Obtaining the descriptive action models from the raw observations with minimal human interference* is the next key milestone for expanding the Automated Planning applications to the raw unstructured inputs, as it fully unleashes the pruning power of state-of-the-art Classical Planning heuristic functions which allow the planner to scale up to much larger problems.

In this paper, we propose an approach called Dual-Stage Action Model Acquisition (DSAMA), a dual-stage process that first learns the set of action symbols and action effects via Action AutoEncoder neural network module in Latplan AMA_2 (Asai and Fukunaga 2018) and then trains a rule-based machine learning system which are then converted into propositional formula in a PDDL format. We tested DSAMA with Random Forest (RF) framework (Ho 1998) as the machine learning module due to its maturity and performance. As a result, we successfully generated a descriptive action model, as depicted in Fig. 1 for example, which is as accurate as the black-box neural counterpart.

Despite the success in terms of the model accuracy, the proposed approach turned out to be an impractical solution for descriptive action model acquisition and gave us an insight into the core problem of this approach. The generated logical formula and the resulting PDDL was too large and complex for the recipient classical planning system (Fast

Downward) to solve the given instance in a reasonable runtime and memory, and if we trade the accuracy with the model simplicity, the goal becomes unreachable. We provide an analysis on the reason and discuss possible future directions. The code reproducing the experiments will be published at `github.com/guicho271828/latplan/`.

2 Preliminaries

We denote a tensor (multi-dimensional array) in bold and denote its elements with a subscript, e.g. when $\boldsymbol{x} \in \mathbb{R}^{N \times M}$, the second row is $\boldsymbol{x}_2 \in \mathbb{R}^M$. We use dotted subscripts to denote a subarray, e.g. $\boldsymbol{x}_{2..5} = (\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4)$. For a vector or a set X, $|X|$ denotes the number of elements. $\mathbf{1}^D$ and $\mathbf{0}^D$ denote the constant matrix of shape D with all elements being 1/0, respectively. $\boldsymbol{a};\boldsymbol{b}$ denotes a concatenation of tensors $\boldsymbol{a}$ and $\boldsymbol{b}$ in the first axis where the rest of the dimensions are same between $\boldsymbol{a}$ and $\boldsymbol{b}$. For a dataset, we generally denote its i-th data point with a superscript ^{i} which we may sometimes omit for clarity.

Let $\mathcal{F}(V)$ be a propositional formula consisting of logical operations $\{\wedge, \vee, \neg\}$, constants $\{\top, \bot\}$, and a set of propositional variables V. We define a grounded (propositional) Classical Planning problem as a 4-tuple $\langle P, A, I, G\rangle$ where P is a set of propositions, A is a set of actions, $I \subset P$ is the initial state, and $G \subset P$ is a goal condition. Each action $a \in A$ is a 3-tuple $\langle \textsc{pre}(a), \textsc{eff}^+(a), \textsc{eff}^-(a)\rangle$ where $\textsc{pre}(a) \in \mathcal{F}(P)$ is a precondition and $\textsc{eff}^+(a)$, $\textsc{eff}^-(a)$ are the sets of effects called add-effects and delete-effects, respectively. Each effect is denoted as $c \triangleright e$ where $c \in \mathcal{F}(P)$ is an *effect condition* and $e \in P$. A state $s \subseteq P$ is a set of true propositions, an action a is *applicable* when $s \vDash \textsc{pre}(a)$ (s *satisfies* $\textsc{pre}(a)$), and applying an action a to s yields a new successor state $a(s)$ which is $a(s) = s \cup \{e \mid (c \triangleright e) \in \textsc{eff}^+(a), c \vDash s\} \setminus \{e \mid (c \triangleright e) \in \textsc{eff}^-(a), c \vDash s\}$.

Latplan is a framework for *domain-independent image-based classical planning* (Asai and Fukunaga 2018). It learns the state representation and the transition rules entirely from image-based observations of the environment with deep neural networks and solves the problem using a classical planner.

Latplan is trained on a *transition input* Tr: a set of pairs of raw data randomly sampled from the environment. The i-th pair in the dataset $\text{tr}^i = (\boldsymbol{x}^{i,0}, \boldsymbol{x}^{i,1}) \in \text{Tr}$ is a randomly sampled transition from a environment observation $\boldsymbol{x}^{i,0}$ to another observation $\boldsymbol{x}^{i,1}$ as the result of some unknown action. Once trained, Latplan can process the *planning input* $(\boldsymbol{x}^I, \boldsymbol{x}^G)$, a pair of raw data images corresponding to an initial and goal state of the environment. The output of Latplan is a data sequence representing the plan execution $(\boldsymbol{x}^I, \ldots \boldsymbol{x}^G)$ that reaches $\boldsymbol{x}^G$ from $\boldsymbol{x}^I$. While the original paper used an image-based implementation, conceptually any form of temporal data that can be auto-encoded to a learned representation is viable for this methodology.

Latplan works in 3 steps. In Step 1, a *State AutoEncoder* (SAE) (Fig. 2, left) neural network learns a bidirectional mapping between raw data $\boldsymbol{x}$ (e.g., images) and propositional states $\boldsymbol{z} \in \{0,1\}^F$, i.e., the F-dimensional bit vectors. The network consists of two functions $\textsc{Enc}$ and $\textsc{Dec}$, where $\textsc{Enc}$ encodes an image $\boldsymbol{x}$ to $\boldsymbol{z} = \textsc{Enc}(\boldsymbol{x})$, and $\textsc{Dec}$ decodes $\boldsymbol{z}$ back to an image $\tilde{\boldsymbol{x}} = \textsc{Dec}(\boldsymbol{z})$. The training is performed by minimizing the reconstruction loss $||\tilde{\boldsymbol{x}} - \boldsymbol{x}||$ under some norm (e.g., Mean Square Error for images).

In order to guarantee that $\boldsymbol{z}$ is a binary vector, the network must use a differentiable discrete activation function such as Heavyside Step Function with straight-through estimator (Koul, Fern, and Greydanus 2019; Bengio, Léonard, and Courville 2013) or Gumbel Softmax (GS) (Jang, Gu, and Poole 2017), which we use for its superior accuracy (Jang, Gu, and Poole 2017, Table 3). GS is an annealing-based continuous relaxation of $\arg\max$ (returns a 1-hot vector), defined as $\text{GS}(\boldsymbol{x}) = \textsc{Softmax}((\boldsymbol{x} - \log(-\log \boldsymbol{u})))/\tau)$, and its special case limited to 2 categories (Maddison, Mnih, and Teh 2017) is $\textsc{BinConcrete}(\boldsymbol{x}) = \textsc{Sigmoid}((\boldsymbol{x} + \log \boldsymbol{u} - \log(1 - \boldsymbol{u}))/\tau)$, where $\boldsymbol{u}$ is a vector sampled from $\textsc{Uniform}(0, 1)$ and τ is an annealing parameter. As $\tau \to 0$, $\text{GS}(\boldsymbol{x}) \to \arg\max(\boldsymbol{x})$ and $\textsc{BinConcrete}(\boldsymbol{x}) \to \textsc{Heavyside}(\boldsymbol{x})$.

After learning the mapping $\textsc{Enc}$ from $\{\ldots \boldsymbol{x}^{i,0}, \boldsymbol{x}^{i,1} \ldots\}$, SAE obtains the propositional transitions $\overline{\text{Tr}} = \{(\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1})\}$ with $\textsc{Enc}$. In Step 2, an Action Model Acquisition (AMA) method learns an action model from $\overline{\text{Tr}}$. In Step 3, a planning problem instance $(\boldsymbol{z}^I, \boldsymbol{z}^G)$ is generated from the planning input $(\boldsymbol{x}^I, \boldsymbol{x}^G)$ and the classical planner finds the path connecting them. In the final step, Latplan obtains a step-by-step, human-comprehensible visualization of the plan execution by $\textsc{Dec}$'oding the intermediate states of the plan into images. For evaluation, we use domain-specific validators for the visualized results because the representation learned by unsupervised learning is not directly verifiable.

Action Model Acquisition (AMA) The original Latplan paper proposed two approaches for AMA. AMA_1 is an oracular model that directly generates a PDDL without learning, and AMA_2 is a neural model that approximates AMA_1 by learning from examples.

AMA_1 is an oracular, idealistic AMA that does not incorporate machine learning, and instead generates the entire propositional state transitions from the entire image transitions in the search space. Each propositional transition is turned into a single, grounded action schema. For example, in a state space represented by 2 latent space propositions $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2)$, a transition from $\boldsymbol{z}^{i,0} = (0, 1)$ to $\boldsymbol{z}^{i,1} = (1, 0)$ is translated into an action with $\textsc{pre}(a) = \neg\boldsymbol{z}_1 \wedge \boldsymbol{z}_2, \textsc{eff}^+(a) = \{\top \triangleright \boldsymbol{z}_1\}, \textsc{eff}^-(a) = \{\top \triangleright \boldsymbol{z}_2\}$. It is impractical because it requires the entire image transitions, and also because the size of the PDDL is proportional to the number of transitions in the state space.

AMA_2 consists of two neural networks: Action AutoEncoder (AAE, Fig. 2 middle) and Action Discriminator (AD). AAE is an autoencoder that learns to cluster the state transitions into a (preset) finite number of action labels.

AAE's encoder takes a propositional state pair $(\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1})$ as the input. The last layer of the encoder is activated by a discrete activation function (such as Gumbel Softmax) to become a one-hot vector of A categories, $\boldsymbol{a}^i \in \{0,1\}^A$

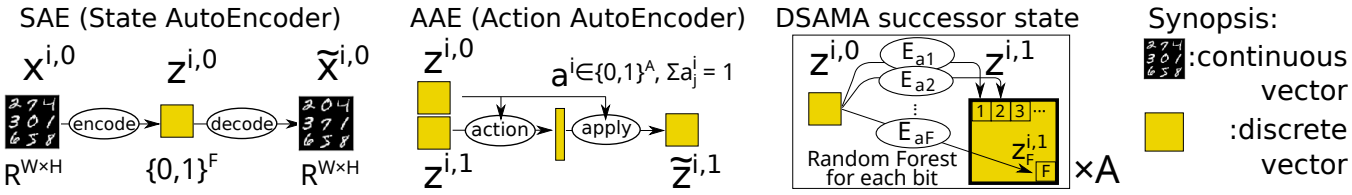

Figure 2: The illustration of State AutoEncoder, Action AutoEncoder, and Double Stage AMA for effect prediction.

($\sum_{1 \le j \le A} \boldsymbol{a}^i_j = 1$), where A is a hyperparameter for the maximum number of action labels and $\boldsymbol{a}^i$ represents an action label. For clarity, we use the one-hot vector $\boldsymbol{a}^i$ and the index $a^i = \arg\max \boldsymbol{a}^i$ interchangeably. AAE's decoder takes the current state $\boldsymbol{z}^{i,0}$ and $\boldsymbol{a}^i$ as the input and output $\tilde{\boldsymbol{z}}^{i,1}$, which is a reconstruction of $\boldsymbol{z}^{i,1}$. The encoder $\text{ACTION}(\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) = \boldsymbol{a}^i$ acts as a function that tells "what action has happened" and the decoder can be seen as a progression function $\text{APPLY}(\boldsymbol{a}^i, \boldsymbol{z}^{i,0}) = \tilde{\boldsymbol{z}}^{i,1}$. The action assigned by the network does not necessarily correspond to any meaningful human-understandable action due to its unsupervised nature.

AD is a binary classifier that models the preconditions of the actions. AD learns the condition from the observed propositional state transitions $\overline{\text{Tr}}$ and a "fake" state transitions. Let $P = \overline{\text{Tr}}$ and U be the fake transitions. U could be generated by applying a random action $1 \le a \le A$ to the states in $\overline{\text{Tr}}$.

This learning task is a Positive-Unlabeled learning task (Elkan and Noto 2008, PU-learning). While all examples in P are guaranteed to be the positive (valid) examples obtained from the observations, the examples in U are *unlabeled*, i.e., we cannot guarantee that the examples in U are always negative (invalid). Unlike the standard binary classification task, which takes the purely positive and the purely negative dataset, PU-learning takes such a positive and an unlabeled dataset and returns a positive-negative classifier. Under the assumption that the positive examples are i.i.d.-sampled from the entire distribution of positive examples, one can obtain a positive-negative classifier $D(\boldsymbol{s})$ for the input $\boldsymbol{s}$ by correcting the confidence value of a labeled-unlabeled classifier $D_{PU}(\boldsymbol{s})$ by an equation $D(\boldsymbol{s}) = D_{PU}(\boldsymbol{s})/c(V_P)$, where V_P is a positive validation set and $c(V_P) = \mathbb{E}_{\boldsymbol{s} \in V_P}[D_{PU}(\boldsymbol{s})]$ is a constant computed after the training of $D_{PU}(\boldsymbol{s})$ (Elkan and Noto 2008). In AD, $\boldsymbol{s}$ is $(\boldsymbol{z}^{i,0}; \boldsymbol{z}^{i,1})$, i.e., the concatenation of the propositional current state and the successor state, unlike the standard STRIPS setting where the precondition only sees the current state.

Combining AAE and AD yields a successor function that can be used for graph search algorithms: It first enumerates the potential successor states from the current state $\boldsymbol{z}$ by iterating $\boldsymbol{z}^a = \mathit{apply}(\boldsymbol{z}, \boldsymbol{a})$ over $1 \le a \le A$, then prunes the generated successor states using AD, i.e., whether $D(\boldsymbol{z}; \boldsymbol{z}^a) > 0.5$. The major drawback of this approach is that both AAE and AD are black-box neural networks, and thus are incompatible with the standard PDDL-based planners and heuristics, and requires a custom heuristic graph search solver.

3 Double-Stage Learning

To overcome the lack of PDDL compatibility of the black-box NNs in AMA_2, we propose Double-Stage Action Model Acquisition method (DSAMA) which consists of 3 steps: (1) It trains the same AAE networks to identify actions and perform the clustering, (2) transfers the knowledge to a set of Random Forest binary classifiers (Fig. 2, right), then finally (3) converts the classifiers into logical preconditions / effects in PDDL. Let M be a process that returns a Random Forest binary classifier and let $\text{TOPDDL}(b)$ be a function that converts a classifier b into a logical formula $\bar{b}$. The overall DSAMA process is shown in Algorithm 1.

Algorithm 1 *Double-Stage Action Model Acquisition* for $\overline{\text{Tr}}$ with a Random Forest classifier M.

Perform AAE training on $\overline{\text{Tr}}$ and obtain ACTION function.
for all $1 \le a \le A$ **do**
  $Z^0_a \leftarrow \left\{\boldsymbol{z}^{i,0} | (\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) \in \overline{\text{Tr}}, \text{ACTION}(\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) = a\right\}$
  $Z^1_a \leftarrow \left\{\boldsymbol{z}^{i,1} | (\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) \in \overline{\text{Tr}}, \text{ACTION}(\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) = a\right\}$
  $Z_a \leftarrow \left\{\boldsymbol{z}^{i,0}; \boldsymbol{z}^{i,1} | (\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) \in \overline{\text{Tr}}, \text{ACTION}(\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) = a\right\}$
  $U_a \leftarrow \left\{\boldsymbol{z}^{i,0}; \boldsymbol{z}^{i,1} | (\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) \in \overline{\text{Tr}}, \text{ACTION}(\boldsymbol{z}^{i,0}, \boldsymbol{z}^{i,1}) \ne a\right\}$
  for all $1 \le f \le F$ **do**
    $Z^1_{af} \leftarrow \left\{\boldsymbol{z}_f | \boldsymbol{z} \in Z^1_a\right\}$
    $E_{af} \leftarrow M(Z^0_a, Z^1_{af})$
    $\overline{E}_{af} \leftarrow \text{TOPDDL}(E_{af})$ (Effect conditions for $\boldsymbol{z}_f$)
  $D_a \leftarrow M((Z_a; U_a), (\mathbf{1}^{|Z_a|}; \mathbf{0}^{|U_a|}))$ ($(X;Y)$ = concat of X and Y)
  $\overline{D}_a \leftarrow \text{TOPDDL}(D_a)$ (precondition)
  collect action $\left\langle \overline{D}_a, \bigcup_f \left\{\overline{E}_{af} \triangleright \boldsymbol{z}_f\right\}, \bigcup_f \left\{\neg\overline{E}_{af} \triangleright \neg\boldsymbol{z}_f\right\} \right\rangle$.

In order to learn the action preconditions, DSAMA performs a PU-learning following Action Discriminator (Sec. 2). Similar to AD, it takes both the current and successor states as the input – in the later experiments, we show that the accuracy drops when we limit the input to the current state. Unlike AD in AMA_2, DSAMA trains a specific classifier for each action. For the action effects, DSAMA learns the effect condition c of the conditional effect $c \triangleright e$ (Sec. 2). DSAMA iterates over every action a and every bit f and trains a binary classifier E_{af} that translates to c.

Random Forest (RF) While the binary classifier in DSAMA could be any binary classifier that could be converted to a logical formula, we chose Random Forest (Ho 1998), a machine learning method based on decision trees. It constructs an ensemble of decision trees $\boldsymbol{t} = (\boldsymbol{t}_1 \dots \boldsymbol{t}_T)$ and

averages the predictions returned by each tree $\boldsymbol{t}_i$. We do not describe the details of its training, which is beyond the scope of this paper. It is one of the most widely used rule-based learning algorithms whose implementations are available in various machine learning packages (we used (Imai 2017)). To address the potential data imbalance, we used a Balanced Random Forest algorithm (Chen, Liaw, and Breiman 2004).

A decision tree for classification consists of decision nodes and leaf nodes. A decision node is a 4-tuple $(i, \theta, \mathit{left}, \mathit{right})$, where each element is the feature index, a threshold, and the left / right child nodes. A leaf node contains a class probability vector $\boldsymbol{p} \in [0,1]^C$, where C is a number of classes to be classified, which is 2 in our binary case. To classify a feature vector $\boldsymbol{x} \in \mathbb{R}^F$, where F is the number of features, it tests $\boldsymbol{x}_i \leq \theta$ at each decision node and recurses into the left/right children depending on success. When a single tree is used for classification, it takes the $\arg\max$ over the probability vector at the leaf node and returns the result as the prediction. For an ensemble of trees, prediction is performed either by taking the $\arg\max$ of the average of $\boldsymbol{p}$ returned by the trees, or by taking the majority vote of each prediction.

Since STRIPS/PDDL cannot directly represent the numeric averaging operation, we simulate the voting version in the PDDL framework by compiling the ensemble into a *majority gate* boolean circuit. First, the decision nodes and the leaf nodes can be recursively converted into a Negation Normal Form in the TOPDDL(tree t) method (Algorithm 2, Fig. 3). Next, we express the majority voting mechanism in a *majority circuit* (Lee and Jen 1992, Fig. 4), a boolean circuit of T fan-ins and a single fan-out which returns 1 when more than $\lfloor T/2 \rfloor$ inputs are 1. One practical approach for implementing such a function is based on *bitonic sorting network* (Knuth 1997; Batcher 1968). Our bitonic sorting (Algorithm 2) for $\boldsymbol{t}_i$ takes $\vee$ and $\wedge$ of the elements being swapped instead of taking the $\max$ and $\min$, assuming $0 = \bot, 1 = \top$.

Since our preconditions D_a takes the current and the successor states as the input, we need to take care of the decision nodes that points to the successor state. When the binary latent space has F dimensions, the input vector to the random forest has $2F$ dimensions where the first and the second half of $2F$ dimensions is for the current and the successor state. In TOPDDL, when the index i of the decision node satisfies $F < i$, we insert $\overline{E}_{a(i-F)}$ instead of $\boldsymbol{z}_i$ because our DSAMA formulation guarantees that $\boldsymbol{z}_i$ is true in the successor bit when the effect condition $\overline{E}_{a(i-F)}$ is satisfied.

Algorithm 2 TOPDDL method for Random Forest (Ho 1998) based on bitonic sorting network (Batcher 1968)

function TOPDDL(forest $\boldsymbol{t} = (\boldsymbol{t}_1 \dots \boldsymbol{t}_T)$)
  SORT($\top, \boldsymbol{t}$) ▷ Bitonic sorting
  return $\boldsymbol{t}_{\lfloor T/2 \rfloor}$
function TOPDDL(tree t)
  if t is a decision node $(i, \theta, \mathit{left}, \mathit{right})$ **then**
    if $0 < \theta < 1$ **then**
      return $(\boldsymbol{x}_i \wedge$ TOPDDL(left)$) \vee (\neg \boldsymbol{x}_i \wedge$ TOPDDL(right))
    if $\theta < 0$ **return** TOPDDL(right), **else return** TOPDDL(left)
  else if t is a leaf node (p_0, p_1) **then**
    if $p_0 < p_1$ **return** $\top$, **else return** $\bot$
function SORT(bool u, vector $\boldsymbol{x}$)
  if $|\boldsymbol{x}| \leq 1$ **then return** $\boldsymbol{x}$
  else ▷ Note: $(\boldsymbol{a}; \boldsymbol{b})$ is a concatenation of vector $\boldsymbol{a}$ and $\boldsymbol{b}$ (Sec. 2).
    $d \leftarrow \lfloor |\boldsymbol{x}|/2 \rfloor, \quad \boldsymbol{a} \leftarrow \boldsymbol{x}_{0..d}, \quad \boldsymbol{b} \leftarrow \boldsymbol{x}_{d..|\boldsymbol{x}|}$
    return MERGE$(u, ($SORT$(\top, \boldsymbol{a})$; SORT$(\bot, \boldsymbol{b})))$
function MERGE(bool u, vector $\boldsymbol{x}$)
  if $|\boldsymbol{x}| \leq 1$ **then return** $\boldsymbol{x}$
  else
    COMPAREANDSWAP$(u, \boldsymbol{x})$
    $d \leftarrow \lfloor |\boldsymbol{x}|/2 \rfloor, \quad \boldsymbol{a} \leftarrow \boldsymbol{x}_{0..d}, \quad \boldsymbol{b} \leftarrow \boldsymbol{x}_{d..|\boldsymbol{x}|}$
    return (MERGE$(u, \boldsymbol{a})$; MERGE$(u, \boldsymbol{b})$)
function COMPAREANDSWAP(bool u, vector $\boldsymbol{x}$)
  $d \leftarrow \lfloor |\boldsymbol{x}|/2 \rfloor$
  for all $i \in \{0..d\}$ **do**
    if u **then** ▷ Compare in increasing order?
      $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i \vee \boldsymbol{x}_{i+d}$ ▷ originally $\max(\boldsymbol{x}_i, \boldsymbol{x}_{i+d})$
      $\boldsymbol{x}_{i+d} \leftarrow \boldsymbol{x}_i \wedge \boldsymbol{x}_{i+d}$ ▷ $\min(\boldsymbol{x}_i, \boldsymbol{x}_{i+d})$
    else
      $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i \wedge \boldsymbol{x}_{i+d}$ ▷ $\min(\boldsymbol{x}_i, \boldsymbol{x}_{i+d})$
      $\boldsymbol{x}_{i+d} \leftarrow \boldsymbol{x}_i \vee \boldsymbol{x}_{i+d}$ ▷ $\max(\boldsymbol{x}_i, \boldsymbol{x}_{i+d})$

4 Evaluation

We evaluated our approach in the dataset used by Asai and Fukunaga, which consists of 5 image-based domains. **MNIST 8-puzzle** is an image-based version of the 8-puzzle, where tiles contain hand-written digits (0-9) from the MNIST database (LeCun et al. 1998). Valid moves in this domain swap the "0" tile with a neighboring tile, i.e., the "0" serves as the "blank" tile in the classic 8-puzzle. The **Scrambled Photograph 8-puzzle (Mandrill, Spider)** cuts and scrambles real photographs, similar to the puzzles sold in stores). These differ from the MNIST 8-puzzle in that "tiles" are *not* cleanly separated by black regions (we re-emphasize that Latplan has no built-in notion of square or movable region). **LightsOut** is a video game where a 4x4 grid of lights is in some on/off configuration, and pressing a light toggles its state as well as the states of its neighbors. The goal is all lights Off. **Twisted LightsOut** distorts the original LightsOut game image by a swirl effect, showing that Latplan is not limited to handling rectangular "objects"/regions. In all domains, we used 9000 transitions for training and 1000 transitions for testing. Note that 8-puzzle contains 362880 states and 967680 transitions, and LightsOut contains 65536 states and 1048576 transitions.

We used the SAE with $F = 100$, i.e., it produces 100 latent propositions. Following the work of (Asai and Kajino

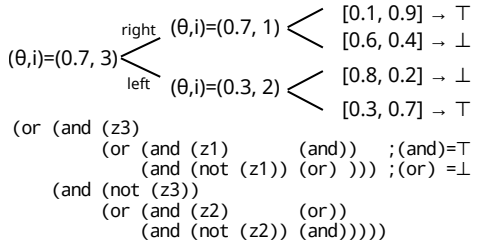

Figure 3: Compilation of a decision tree into a logical formula in PDDL (as a precondition or an effect condition).

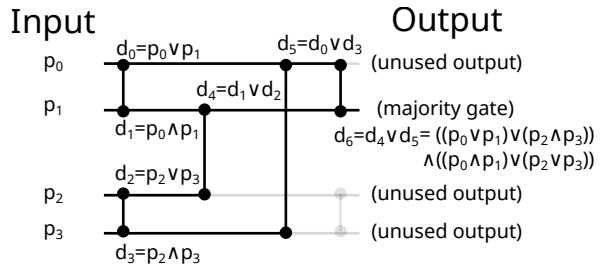

Figure 4: Majority-gate implementation for 4 inputs $p_0 \dots p_3$, using 7 comparators $d_0 \dots d_6$ generated as part of bitonic sorting network.

2019), we used the standard version of SAE and a regularized version of SAE (ZSAE) with a regularization constant $\alpha = 0.5$. For the AAE, we tuned the upper-bound A of the number of actions in AAE by iteratively increasing A from $A = 8$ to $A = 128$ by $\Delta A = 8$ until the mean absolute error of AAE ($|\boldsymbol{z}^{i,1} - \tilde{\boldsymbol{z}}^{i,1}|$) goes below 0.01, i.e., below 1 bit on average. This is because a large A reduces the number of transitions that fall into a single action label and makes the random forest training harder due to the lack of examples.

4.1 Accuracy

We compared the accuracy of DSAMA and AMA_2. DSAMA has two primary controlling hyperparameters for RF — the maximum depth D of the tree and the number of trees T. Other hyperparameters of Random Forest follows the standard parameters for classification tasks.

We first compared the successor generation accuracy between AAE (black-box NN model) and DSAMA. The dataset $\overline{\text{Tr}}$ is divided into the training set and the test set by 9:1. AAE uses the same hyperparameters used in Latplan (Asai and Fukunaga 2018). DSAMA uses a RF with $T = 80$ and $D = 100$, the largest number we used in this paper. Table 1 shows the average reconstruction accuracy for the successor states over F bits, over all transitions in the test dataset. The results indicate that RF is competitive against the black box NN.

Next, we compared the F-measure based on the true positive rate (=*recall*) and the true negative rate (=*specificity*) of the black-box precondition model (Action Discriminator) and the DSAMA precondition model using Random Forest. Similar results are obtained in Table 2: Rule-based method (Random-Forest) is competitive against the black box method when a sufficiently large capacity is provided to the model ($T = 80, D = 100$). To address the concern on using the successor states as part of the precondition, we also tested the variants which learns only from the current state. We observed a significant drop in the accuracy both in the black-box NN (AD) and the DSAMA D_a.

In order to see the effect of the random forest hyperparameters on the learned results, we performed an exhaustive experiment on $(T, D) \in \{1, 2, 5, 10, 20, 40, 80\} \times \{4, 7, 12, 25, 50, 100\}$ and compared the precondition accuracy, the effect accuracy and the size of the PDDL files. Note that $T = 1$ is a degenerative case for a single decision tree without ensembles. Due to the space constraint, we only show the results for Mandrill 8-Puzzle with ZSAE ($\alpha = 0.5$), but the overall characteristics were the same across domains and the choice of SAE / ZSAE.

	SAE		AMA_2 AAE	DSAMA E_{af}
Domain	α	MSE	A	Successor bit accuracy
LOut	0.0	2.25E-12	37	99.1% / 98.5%
LOut	0.5	8.18E-11	43	99.4% / 99.2%
Twisted	0.0	1.03E-02	48	99.3% / 98.8%
Twisted	0.5	1.08E-02	39	98.8% / 98.4%
Mandrill	0.0	7.14E-03	14	99.9% / 98.9%
Mandrill	0.5	7.09E-03	9	99.4% / 98.9%
Mnist	0.0	3.34E-04	67	98.8% / 94.0%
Mnist	0.5	3.44E-03	3	99.8% / 99.5%
Spider	0.0	9.51E-03	9	99.2% / 98.5%
Spider	0.5	8.88E-03	8	99.5% / 98.4%

Table 1: Accuracy comparison between AAE and DSAMA on the successor state prediction task, where the discrete transitions are generated by the SAE or ZSAE ($\alpha = 0.5$) for 5 domains. The accuracy is measured by the number of bits correctly predicted by the model in the test dataset. DSAMA uses a Random Forest with the number of trees $T = 80$ and the maximum depth of the tree $D = 100$. We observed that the Random Forest is competitive for this task against a black box NN and sometimes even outperformed it. As a reference, we also showed: (1) A, the number of action labels generated by the AAE, and (2) the image reconstruction error of the SAE, where the values are MSE for the pixel values in $[0, 1] \subset \mathbb{R}$.

We observed that the effect of larger T and D saturates quickly, while small numbers negatively affect the performance. Larger T and D also implies larger file sizes. (Note: When generating the PDDL file, we apply De-Morgan's law to simplify obvious invariants when one is encountered, e.g., $v \wedge \top = v$, $v \vee \bot = v$, $v \wedge \neg v = \bot$ and $v \vee \neg v = \top$.)

4.2 Evaluation in the Latent Space

To measure the effectiveness of our approach for planning, we ran Fast Downward on the domain PDDL files generated by DSAMA system. We gave 1 hour time limit and a maximum of 256 GB memory to the planner.

We tested several configurations, including Blind, max (Haslum and Geffner 2000) and FF (Hoffmann and Nebel 2001) heuristics (with A^* and GBFS), on the PDDL domain files generated by different T and D. As stated in the introduction, despite our RF models achieving high accuracy in terms of prediction, we did not manage to find a plan at all. The failure modes are threefold: The planner failed to find the goal after exhaustively searching the state space, the initial heuristic value being infinity in the reachability analysis (in h^{FF} and $h^{\max}$), or the problem transformation to SAS+ (Bäckström and Nebel 1995) does not finish within the resource limit. From the results in the previous tables, the reason of the failure is obvious: There is a trade off between the accuracy and the PDDL file size. When the PDDL model is inaccurate, the search graph becomes disconnected and the search fails. If we increase the accuracy of the PDDL

Input dataset →		AMA$_2$ AD		DSAMA D_a	
		$\overline{\text{Tr}}$	$\{(\boldsymbol{z}^{i,0}; \boldsymbol{a}^i)\}$	Z_a	Z_a^0
LOut	α =0.0	**89.7%**	74.2%	81.5%	75.0%
LOut	0.5	**89.1%**	68.7%	85.5%	71.7%
Twisted	0.0	**84.3%**	75.5%	80.3%	71.5%
Twisted	0.5	**84.5%**	72.6%	78.8%	69.6%
Mandrill	0.0	81.6%	76.2%	**82.4%**	76.5%
Mandrill	0.5	66.0%	29.1%	**77.2%**	59.6%
Mnist	0.0	**87.5%**	81.6%	84.4%	78.2%
Mnist	0.5	**63.8%**	57.0%	61.6%	57.5%
Spider	0.0	**82.5%**	64.3%	80.7%	69.2%
Spider	0.5	60.1%	15.9%	**77.9%**	68.3%

Table 2: F-measure of true positive rate (*recall*) and true negative rate (*specificity*), comparing AD and DSAMA on the action applicability task. The discrete transitions are generated by the SAE or ZSAE ($\alpha = 0.5$) for 5 domains. Given the prediction X and the ground truth G, recall $= P(X = 1 \mid G = 1)$, specificity $= P(X = 0 \mid G = 0)$, and F $= \frac{2 \cdot \text{recall} \cdot \text{specificity}}{\text{recall} + \text{specificity}}$. The numbers are for a test dataset where each transition is assigned a boolean ground-truth value by a validator that works on the reconstructed image pairs. DSAMA uses a Random Forest with the number of trees $T = 80$ and the maximum depth of the tree $D = 100$. We show the results with the current state as the input, and those with the concatenation of the current and the successor states as the input. The accuracy drops when the input dataset is limited to the current state only.

model, the file size increases and Fast Downward fails even to start the search. Moreover, we observed that the translation fails even with a PDDL domain file with the moderate file size (e.g. 11MB). In contrast, PDDL file sizes of IPC Cybersec domain are typically 35MB and Fast Downward finds no problem translating the file.

Our experiments showed that Random-Forest based DSAMA approach does not work even if it achieves the competitive accuracy as the black-box model. Based on this observation, one question arises: *Can the translator bottleneck be addressed just by using a different rule-based learning method, such as MAX-SAT based approaches (Yang, Wu, and Jiang 2007) or planning based approaches (Aineto, Jiménez, and Onaindia 2018)?* We argue that this would not be the case because (1) our Random Forest based DSAMA approach can be considered as the upper bound of existing Action Model Acquisition method in terms of *accuracy* and (2) should the same accuracy be achieved by other approaches, *the resulting PDDL must have the same complexity*. We explain the reasoning below.

First, we note that the translation failure is due to the heavy use of disjunctions in our PDDL files originating from Random Forest. In Fast Downward, disjunctions are eliminated (Helmert 2009) by making the separate actions for each branch of the disjunction. This causes an exponential blowup when a huge number of disjunctions are presented to the translator.

Next, in order to avoid this blowup, the rules must be disjunction-free, but the effect is negated by the exponential number of rules. For example, one trivial approach to learn disjunction-free rules in DSAMA is to use Decision Lists (Cohen 1995), the degenerate case of decision trees where the children (*left*, *right*) of every decision node can contain at most one decision node. This is ineffective because a decision tree can be decomposed into an exponential number of decision lists that express the same classification function. Eliminating the disjunctions during the translation and finding a set of disjunction-free rules both end up in an exponentially large list of disjunction-free actions. Note that existing approaches (Yang, Wu, and Jiang 2007; Aineto, Jiménez, and Onaindia 2018) learn the disjunction-free action models.

Finally, given that our Random Forest based DSAMA achieved almost-perfect accuracy in the successor generation task (effect condition), we could argue that the rules generated by our approach are quite close to the ground truth rules and, therefore, *the ground truth rules are at least as complex as the rules found by DSAMA*. If the existing or future approaches achieved the same accuracy on the same task, their resulting disjunction-free set of conditions would be as large and complex as our exponentially large SAS+ compilation.

5 Related Work

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. Many systems require a structured input representation (i.e., First Order Logic) that are partially hand-crafted and exploits the symmetry and the structures provided by the structured representation, although the requirements of the systems may vary (Yang, Wu, and Jiang 2007; Cresswell, McCluskey, and West 2013; Aineto, Jiménez, and Onaindia 2018).

There are several lines of work that extracts a PDDL action model from a natural language corpus. Framer (Lindsay et al. 2017) uses a CoreNLP language model while EASDRL (Feng, Zhuo, and Kambhampati 2018) uses Deep Reinforcement Learning (Mnih et al. 2015). The difference from our approach is that they are reusing the symbols found in the corpus while we generate the discrete propositional symbols from the visual perception which completely lacks such a predefined set of discrete symbols.

While there are recent efforts in handling the complex state space without having the action description (Frances et al. 2017), action models could be used for other purposes, including Goal Recognition (Ramírez and Geffner 2009), macro-action generation (Botea and Braghin 2015; Chrpa, Vallati, and McCluskey 2015), or plan optimization (Chrpa and Siddiqui 2015).

There are three lines of work that learn the binary representation of the raw environment. Latplan SAE (Asai and Fukunaga 2018) uses the Gumbel-Softmax VAE (Maddison, Mnih, and Teh 2017; Jang, Gu, and Poole 2017) which was modified from the original to maximize the KL divergence term for the Bernoulli distribution (Asai and Kajino 2019). Causal InfoGAN (Kurutach et al. 2018) uses GAN(Goodfellow et al. 2014)-based approach combined with Gumbel

		Action applicability (precondition) F-measure						Successor state reconstruction (effect) accuracy						PDDL domain file size					
		D						D						D					
		4	7	12	25	50	100	4	7	12	25	50	100	4	7	12	25	50	100
	1	36.5%	56.1%	65.5%	49.2%	49.8%	49.5%	87.7%	90.6%	91.7%	91.8%	91.8%	91.8%	964K	1.5M	2.0M	3.0M	2.9M	2.9M
	2	40.5%	59.1%	67.0%	63.8%	63.8%	64.0%	91.5%	93.0%	91.8%	91.9%	91.9%	91.9%	2.2M	3.2M	4.1M	6.0M	6.0M	6.0M
	5	37.3%	56.8%	70.1%	69.0%	69.5%	68.8%	94.4%	95.8%	96.3%	96.3%	96.3%	96.3%	7.8M	11M	13M	18M	18M	18M
T	10	35.7%	56.2%	71.1%	72.7%	72.9%	72.5%	95.5%	97.3%	97.5%	97.5%	97.5%	97.5%	24M	29M	34M	43M	44M	43M
	20	35.9%	56.8%	72.4%	75.0%	74.8%	74.6%	96.4%	98.0%	98.2%	98.2%	98.2%	98.2%	71M	81M	91M	109M	109M	109M
	40	32.8%	55.6%	73.1%	76.6%	76.2%	76.4%	96.7%	98.2%	98.8%	98.8%	98.8%	98.8%	204M	224M	244M	281M	281M	281M
	80	33.4%	55.6%	73.1%	77.1%	77.1%	77.2%	96.9%	98.5%	98.9%	98.8%	98.9%	98.9%	557M	597M	636M	710M	710M	709M

Table 3: Accuracy for the precondition and the effects by DSAMA using various Random Forest hyperparameters, as well as the PDDL domain file sizes (in bytes) resulted from their compilation. T is the number of trees in each Random Forest ensemble, and D is the maximum depth of the tree. The table is showing the results for the test dataset of Mandrill 8-Puzzle, where the discrete state vectors are generated by ZSAE ($\alpha = 0.5$).

Softmax prior and Mutual Information prior. Latplan ZSAE (Asai and Kajino 2019) additionally penalizes the "true" category in the binary categorical distribution to suppress the chance of random flips in the latent vector caused by the input noise. Quantized Bottleneck Network (Koul, Fern, and Greydanus 2019) uses quantized activations (i.e., step functions) in the latent space with Straight-Through gradient estimator (Bengio, Léonard, and Courville 2013), which enables the backpropagation through the step function. There are more complex alternatives such as VQVAE (van den Oord, Vinyals, and others 2017), DVAE++(Vahdat et al. 2018), DVAE# (Vahdat, Andriyash, and Macready 2018).

6 Conclusion

In this paper, we negatively answered a question of *whether simply replacing a neural, black-box Action Model Acquisition model with a rule-based machine learning model would generate a useful descriptive action model from the raw, unstructured input.* Our approach hybrids a neural unsupervised learning approach to the action label generation and the precondition/effect-condition learning using State-of-the-Art rule-based machine learning. While the proposed method was able to generate accurate PDDL models, the models are too complex for the standard planner to preprocess in a reasonable runtime and memory.

It is worth noting that the straightforward model based on binary classification is causing such a problem. We speculate that the planning domains written by humans, e.g. IPC domains, tend to have a specific human-originated property that suppresses this type of phenomenon, and that our current method lacks this property because the state encoding and the action labels are generated by the State/Action AutoEncoder neural networks. The lack of such a human-like regularization made an otherwise trivial task of PDDL-SAS+ translation intractable in a modern planner.

Future directions are twofold. The first one is to find the right regularization / architecture for the neural networks that reflects such a property. This is in line with the approach pursued by (Asai and Kajino 2019) which tries to suppress the number of propositions in the latent space. Machine Learning community is also increasingly focusing on the disentangled representation learning (Higgins et al. 2017) that tries to separate the meaning of each feature in the latent space. Finding the right structural bias for neural networks has a long history, notably the convolutional neural networks (Fukushima 1980) for images, or LSTMs (Hochreiter and Schmidhuber 1997) and transformers (Vaswani et al. 2017) for sequence modeling.

The second approach is to develop a planner that can directly handle the disjunctions in an efficient manner. Fast downward removes the disjunctions during the PDDL-SAS+ translation, assuming that the number of disjunctions are relatively small. While this may hold for most hand-crafted domains (such as IPC domains), it may not be a viable approach when the symbolic input is generated by neural networks.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS Action Models with Classical Planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Amado, L.; Pereira, R. F.; Aires, J.; Magnaguagno, M.; Granada, R.; and Meneguzzi, F. 2018a. Goal Recognition in Latent Space. In *Proc. of International Joint Conference on Neural Networks (IJCNN)*.

Amado, L.; Pereira, R. F.; Aires, J.; Magnaguagno, M.; Granada, R.; and Meneguzzi, F. 2018b. LSTM-based Goal Recognition in Latent Space. *arXiv preprint arXiv:1808.05249*.

Asai, M., and Fukunaga, A. 2018. Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. In *Proc. of AAAI Conference on Artificial Intelligence*.

Asai, M., and 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. 2019. Unsupervised Grounding of Plannable First-Order Logic Representation from Images. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

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

Batcher, K. E. 1968. Sorting Networks and Their Applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, 307–314. ACM.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or Propagating Gradients through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*.

Botea, A., and Braghin, S. 2015. Contingent versus Deterministic Plans in Multi-Modal Journey Planning. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*, 268–272.

Chen, C.; Liaw, A.; and Breiman, L. 2004. Using Random Forest to Learn Imbalanced Data. Technical Report Technical Report 666, Department of Statistics, UC Berkeley.

Chrpa, L., and Siddiqui, F. H. 2015. Exploiting Block Deordering for Improving Planners Efficiency. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2015. On the Online Generation of Effective Macro-Operators. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Cohen, W. W. 1995. Fast Effective Rule Induction. In *Proc. of the International Conference on Machine Learning*, 115–123.

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

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

Feng, W.; Zhuo, H. H.; and Kambhampati, S. 2018. Extracting Action Sequences from Texts Based on Deep Reinforcement Learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 4064–4070. Proc. of International Joint Conference on Artificial Intelligence (IJCAI).

Frances, G.; Ramırez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, 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):193–202.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, 2672–2680.

Haslum, P., and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proc. of the International Conference on Artificial Intelligence Planning and Scheduling*.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence* 173(5-6):503–535.

Higgins, I.; Matthey, L.; Pal, A.; Burgess, C.; Glorot, X.; Botvinick, M.; Mohamed, S.; and Lerchner, A. 2017. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. volume 2, 6.

Ho, T. K. 1998. The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(8):832–844.

Hochreiter, S., and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation* 9(8):1735–1780.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)* 14:253–302.

Imai, S. 2017. `cl-random-forest`. `https://github.com/masatoi/cl-random-forest`.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. In *Proc. of the International Conference on Learning Representations*.

Knuth, D. E. 1997. *The Art of Computer Programming*, volume 3. Pearson Education.

Koul, A.; Fern, A.; and 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.; and Abbeel, P. 2018. Learning Plannable Representations with Causal InfoGAN. In *In Proceedings of ICML / IJCAI / AAMAS 2018 Workshop on Planning and Learning (PAL-18)*.

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

Lee, C. L., and Jen, C.-W. 1992. Bit-Sliced Median Filter Design based on Majority Gate. *IEE Proceedings G (Circuits, Devices and Systems)* 139(1):63–71.

Lindsay, A.; Read, J.; Ferreira, J. F.; Hayton, T.; Porteous, J.; and Gregory, P. J. 2017. Framer: Planning Models from Natural Language Action Descriptions. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Proc. of the International Conference on Learning Representations*.

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.

Ramírez, M., and Geffner, H. 2009. Plan Recognition as Planning. In *Proc. of AAAI Conference on Artificial Intelligence*.

Vahdat, A.; Andriyash, E.; and Macready, W. 2018. DVAE#: Discrete variational autoencoders with relaxed Boltzmann priors. In *Advances in Neural Information Processing Systems*, 1864–1874.

Vahdat, A.; Macready, W. G.; Bian, Z.; Khoshaman, A.; and Andriyash, E. 2018. DVAE++: Discrete variational autoencoders with overlapping transformations. *arXiv preprint arXiv:1802.04920*.

van den Oord, A.; Vinyals, O.; et al. 2017. Neural Discrete Representation Learning. In *Advances in Neural Information Processing Systems*, 6306–6315.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

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