Planning From Pixels in Atari With Learned Symbolic Representations

Andrea Dittadi*
Technical University of Denmark
Copenhagen, Denmark
adit@dtu.dk

Frederik K. Drachmann*
Technical University of Denmark
Copenhagen, Denmark
fdrachmann@hotmail.dk

Thomas Bolander
Technical University of Denmark
Copenhagen, Denmark
tobo@dtu.dk

Abstract

Width-based planning methods have been shown to yield state-of-the-art performance in the Atari 2600 video game playing domain using pixel input. One approach consists in an episodic rollout version of the Iterated Width (IW) algorithm called RolloutIW, and uses the B-PROST boolean feature set to represent states. Another approach, π-IW, augments RolloutIW with a learned policy to improve how actions are picked in the rollouts. This policy is implemented as a neural network, and the feature set is derived from an intermediate representation learned by the policy network. Results suggest that learned features can be competitive with hand-crafted ones in the context of width-based search. This paper introduces a new approach, where we leverage variational autoencoders (VAEs) to learn features for the domains in a principled manner, directly from pixels, and without supervision. We use the inference network (or encoder) of the trained VAEs to extract boolean features from screen states, and use them for planning with RolloutIW. The trained model in combination with RolloutIW outperforms the original RolloutIW and human professional play on the Atari 2600 domain and reduces the size of the feature set from 20.5 million to 4,500.

Introduction

Width-based search algorithms have in the last few years become among the state-of-the-art approaches to automated planning, e.g. the original Iterated Width (IW) algorithm (Lipovetzky and Geffner 2012). As in propositional STRIPS planning, states are represented by a set of propositional literals, also called boolean *features*. The state space is searched with breadth-first search (BFS), but the state space explosion problem is handled by pruning states based on their *novelty*. First a parameter k is chosen, called the *width parameter* of the search. Searching with width parameter k essentially means that we only consider k literals/features at a time. A state s generated during the search is called *novel* if there exists a set of k literals/features not made true in any earlier generated state. Unless a state is novel, it is immediately pruned from the search. Clearly, we then reduce the size of the searched state space to be exponential in k. It has been shown that many classical planning problems, e.g. problems from the International Planning Competition (IPC) domains, can be solved efficiently using width-based search with very low values of k.

The essential benefit of using width-based algorithms is the ability to perform semi-structured (based on feature structures) exploration of the state space, and reach deep states that may be important for achieving the planning goals. In classical planning, width-based search has been integrated with heuristic search methods, leading to Best-First Width Search (Lipovetzky and Geffner 2017) that performed well at the 2017 International Planning Competition. Width-based search has also been adapted to reward-driven problems where the algorithm uses a simulator for interacting with the environment (Francès et al. 2017). This has enabled the use of width-based search in reinforcement learning environments such as the the Atari 2600 video game suite, through the Arcade Learning Environment (ALE) (Bellemare et al. 2013). There have been several implementations of width-based search directly using the RAM states of the Atari computer as features (Lipovetzky, Ramirez, and Geffner 2015; Shleyfman, Tuisov, and Domshlak 2016; Jinnai and Fukunaga 2017).

Motivated by the fact that humans do not have access to the internal RAM states of a computer when playing video games, methods for using the algorithms based on the raw screen pixels have been developed. Bandres, Bonet, and Geffner (2018) propose a modified version of IW, called RolloutIW, that uses pixel-based features and achieves results comparable to learning methods in almost real-time on the ALE. The combination of Reinforcement Learning (RL) and RolloutIW in the pixel domain is explored by Junyent, Jonsson, and Gómez (2019), who propose to train a policy to guide the action selection in rollouts.

Regardless of different efforts, width-based algorithms are highly exploratory and, not surprisingly, significantly dependent on the quality of the features defined. The original RolloutIW paper (Bandres, Bonet, and Geffner 2018) uses a set of features called B-PROST (Liang et al. 2015). These are features extracted from the screen pixels by splitting the screen into tiles and keeping track of which colors are present in which tiles. These features have been designed by humans to achieve good performance in ALE. Integrating learning of features efficient for width-based search into the algorithms themselves rather than using hand-crafted fea-

*Equal contribution.

ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning.

tures is an interesting challenge—and a challenge that will bring the algorithms more on a par with humans trying to learn to play those video games. In Junyent, Jonsson, and Gómez (2019), the features used were the output features of the last layer of the policy network, improving the performance of IW. With the attempt to learn efficient features for width-based search, and inspired by the results of Junyent, Jonsson, and Gómez (2019), this paper investigates the possibility of a more structured approach to generate features for width-based search using deep generative models.

More specifically, in this paper we use variational autoencoders (VAE)—latent variable models that have been widely used for representation learning (Kingma and Welling 2019) as they allow for approximate inference of the latent variables underlying the data—to learn propositional symbols directly from pixels and without any supervision. We then investigate whether the learned symbolic representation can be successfully used for planning in the Atari 2600 domain.

We compare the planning performance of RolloutIW using our learned representations with RolloutIW using the handcrafted B-PROST features, and report human scores as baseline. Our results show that our learned representations lead to better performance than B-PROST on RolloutIW (and often outperform human players). This is despite the fact that the B-PROST features also contain temporal information (how the screen image dynamically changes), whereas we only extract static features from individual frames. Apart from improving scores in the Atari 2600 domain, we also significantly reduce the number of features (by a factor of more than 10^3). We also investigate in more detail (with ablation studies) which factors seems to lead to good performance on games in the Atari domain.

The main contributions of our paper are hence:

- We train generative models to learn propositional symbols for planning, from pixels and without any supervision.
- We run a large-scale evaluation on Atari 2600 where we compare the performance of RolloutIW with our learned representations to RolloutIW with B-PROST features.
- We investigate with ablation studies the effect of various hyperparameters in both learning and planning.
- We show that our learned representations lead to higher scores than using B-PROST, even though our features don't have any temporal information, and our models are trained from data collected by RolloutIW with B-PROST, limiting the richness of the dataset.

Below, we provide the required background, present the original approach of this paper, discuss our experimental results, and conclude.

Background

In this section, we revise the relevant background on Iterated Width (IW), RolloutIW, planning with pixels, and variational autoencoders (VAEs).

Iterated Width

Iterated Width (IW) (Lipovetzky and Geffner 2012) is a blind-search planning algorithm in which the states are represented by sets of boolean features (sets of propositional atoms). The set of all features/atoms is the *feature set*, denoted F. IW is an algorithm parametrized by a *width parameter* k. We use IW(k) to denote IW with width parameter k. Given an initial state s_0, IW(k) is similar to a standard breadth-first search (BFS) from s_0 except that, when a new state s is generated, it is immediately pruned if it is not novel. A state s generated during search is defined to be *novel* if there is a k-tuple of features (atoms) $t = (f_1, \ldots, f_k) \in F^k$ such that s is the first generated state making all of the f_i true (s making f_i true of course simply means $f_i \in s$, and the intuition is that s then "has" the feature f_i). In particular, in IW(1), the only states that are not pruned are those that contain a feature $f \in F$ for the first time during the search. The maximum number of generated states is exponential in the width parameter (IW(k) generates $O(|F|^k)$ states), whereas in BFS it is exponential in the number of features/atoms.

Rollout IW

RolloutIW(k) (Bandres, Bonet, and Geffner 2018) is a variant of IW(k) that searches via rollouts instead of doing a breadth-first search, hence making it more akin to a depth-first search. A *rollout* is a state-action trajectory $(s_0, a_0, s_1, a_1, \ldots)$ from the initial state s_0 (a path in the state space), where actions a_i are picked at random. It continues until reaching a state that is not novel. A state s is *novel* if it satisfies one of the following: 1) it is distinct from all previously generated states and there exists a k-tuple of features $(f_1, \ldots, f_k)$ that are true in s, but not in any other state of the same depth of the search tree or lower; 2) it is an already earlier generated state and there exists a k-tuple of features $(f_1, \ldots, f_k)$ that are true in s, but not in any other state of lower depth in the search tree. The intuition behind case 1 is that in this case the new state s comes with a combination of k features occurring at a lower depth than earlier encountered, which makes it relevant to explore further. In case 2, s is an existing state already containing the lowest occurrence of one of the combinations of k features, again making it relevant to explore further. When a rollout reaches a state that is not novel, the rollout is terminated, and a new rollout is started from the initial state. The process continues until all possible states are explored, or a given time limit is reached. After the time limit has been reached, an action with maximal expected reward is chosen (the method was developed in the context of the Atari 2600 domain, making use of a simulator to compute action outcomes and rewards). The chosen action is executed, and the algorithm is repeated with the new state as initial state.

RolloutIW is an anytime algorithm that will return an action independently of the time budget. In principle, IW could also be used as an anytime algorithm in the context of the Atari 2600 domain, but it will be severely limited by the breadth-first search strategy that will in practice prevent it from reaching nodes of sufficient depth in the state space, and hence prevent it from discovering rewards that only occur later in the game (Bandres, Bonet, and Geffner 2018).

Planning With Pixels

The Arcade Learning Environment (ALE) (Bellemare et al. 2013) provides an interface to Atari 2600 video games, and has been widely used in recent years as benchmark for reinforcement learning and planning algorithms. In the visual setting, the sensory input consists of a pixel array of size 210×160, where each pixel can have 128 distinct colors. Although in principle the set of $210 \times 160 \times 128$ booleans could be used as features, we will follow Bandres, Bonet, and Geffner (2018) and focus on features that capture meaningful structures from the image.

An example of a visual feature set that has proven successful is *B-PROST* (Liang et al. 2015), which consists of *Basic*, *B-PROS*, and *B-PROT* features. The screen is split into 14×16 disjoint tiles of size 15×10. For each tile (i, j) and color c, the basic feature $f_{i,j,c}$ is 1 iff c is present in (i, j). A B-PROS feature $f_{i,j,c,c'}$ is 1 iff color c is present in tile t, color c' in tile t', and the relative offsets between the tiles are i and j. Similarly, a B-PROT feature $f_{i,j,c,c'}$ is 1 iff color c is present in tile t *in the previous decision point*, color c' is present in tile t' in the current one, and the relative offsets between the tiles are i and j. The number of features in B-PROST is the sum of the number of features in these 3 sets, in total 20,598,848.

Variational Autoencoders

Latent variable models. *Latent variable models* (LVMs) are a type of probabilistic models in which some variables are unobserved. For one datapoint, the marginal distribution over the observed variables $\mathbf{x}$ is:

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \tag{1}$$

where $\mathbf{z}$ are the unobserved *latent variables* and θ denotes the model parameters. This quantity is typically referred to as *marginal likelihood* or *model evidence*. A simple and rather common structure for LVMs is:

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x} \,|\, \mathbf{z}) p_\theta(\mathbf{z}) \tag{2}$$

where both $p_\theta(\mathbf{x} \,|\, \mathbf{z})$ and $p_\theta(\mathbf{z})$ are specified.

If the model's distributions are parameterized by neural networks, the marginal likelihood is typically intractable for lack of an analytical solution or a practical estimator. Since $p_\theta(\mathbf{z} \,|\, \mathbf{x}) = p_\theta(\mathbf{x}, \mathbf{z}) / p_\theta(\mathbf{x})$ and $p_\theta(\mathbf{x}, \mathbf{z})$ is tractable to compute, it follows that the intractability of $p_\theta(\mathbf{x})$ derives in fact from that of the posterior $p_\theta(\mathbf{z} \,|\, \mathbf{x})$.

Amortized variational inference. *Variational inference* (VI) is a common approach to approximating this intractable posterior, where we define a distribution $q_\phi(\mathbf{z} \,|\, \mathbf{x})$ with *variational parameters* ϕ, and optimize it to be "close" to the exact posterior. In the LVM above, for any choice of q_ϕ:

$$\log p_\theta(\mathbf{x}) = \log \mathbb{E}_{q_\phi(\mathbf{z} \,|\, \mathbf{x})} \left[\frac{p_\theta(\mathbf{x} \,|\, \mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z} \,|\, \mathbf{x})} \right] \tag{3}$$

$$\geq \mathbb{E}_{q_\phi(\mathbf{z} \,|\, \mathbf{x})} \left[\log \frac{p_\theta(\mathbf{x} \,|\, \mathbf{z}) p_\theta(\mathbf{z})}{q_\phi(\mathbf{z} \,|\, \mathbf{x})} \right] \tag{4}$$

$$= \mathcal{L}_{\theta,\phi}(\mathbf{x}) \tag{5}$$

where $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ is a lower bound on the marginal log likelihood also known as *Evidence Lower BOund* (ELBO).

In contrast to traditional VI methods, where per-datapoint variational parameters are optimized separately, *amortized variational inference* utilizes function approximators like neural networks to share variational parameters across datapoints and improve learning efficiency. In this setting, $q_\phi(\mathbf{z} \,|\, \mathbf{x})$ is typically called *inference model* or *encoder*.

Variational Autoencoders (VAEs) (Kingma and Welling 2013; Rezende, Mohamed, and Wierstra 2014) are a framework for amortized VI, in which the ELBO is maximized by jointly optimizing the inference model and the LVM (i.e., ϕ and θ, respectively) with stochastic gradient ascent.

VAE optimization. The ELBO, the objective function to be maximized, can be decomposed as follows:

$$\begin{aligned} \mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi} \left[\log p_\theta(\mathbf{x} \,|\, \mathbf{z}) \right] - \mathbb{E}_{q_\phi} \left[\log \frac{q_\phi(\mathbf{z} \,|\, \mathbf{x})}{p_\theta(\mathbf{z})} \right] \\ &= \mathbb{E}_{q_\phi} \left[\log p_\theta(\mathbf{x} \,|\, \mathbf{z}) \right] - D_{\mathrm{KL}}(q_\phi(\mathbf{z} \,|\, \mathbf{x}) \,||\, p_\theta(\mathbf{z})) \end{aligned} \tag{6}$$

where the first term can be interpreted as negative expected *reconstruction error*, and the second term is the KL divergence from the prior $p_\theta(\mathbf{z})$ to the approximate posterior. Minimizing the reconstruction error (i.e., maximizing the first term) pushes 1) the decoder to accurately reconstruct the input (in expectation over the encodings of $\mathbf{x}$), and 2) the inference model to encode $\mathbf{x}$ in such a way that the decoder can do so more effectively.

One of the major challenges in the optimization of VAEs is that the gradients of some terms of the objective function cannot be backpropagated through the sampling step. However, for a rather wide class of probability distributions, a random variable following such distribution can be expressed as a differentiable, deterministic transformation of an auxiliary variable with independent marginal distribution. For example, if z is a sample from a Gaussian random variable with mean $\mu_\phi(x)$ and standard deviation $\sigma_\phi(x)$, then $z = \sigma_\phi(x)\,\epsilon + \mu_\phi(x)$, where $\epsilon \sim \mathcal{N}(0, 1)$. Thanks to this *reparameterization*, z can be differentiated with respect to ϕ by standard backpropagation. This approach, called *pathwise gradient estimator*, typically exhibits lower variance than the alternatives, and is widely used in practice.

From an information theory perspective, optimizing the variational lower bound (6) involves a tradeoff between rate and distortion (Alemi et al. 2017) or, equivalently, between how much the data is compressed (more compression corresponds to a lower KL divergence) and how much information we retain (more information corresponds to a lower reconstruction loss). A straightforward way to control the rate–distortion tradeoff is to use the β-VAE framework (Higgins et al. 2017), in which the training objective (6) is modified by scaling the KL term:

$$\begin{aligned} \mathcal{L}_{\theta,\phi,\beta}(\mathbf{x}) = \mathbb{E}_{q_\phi} &\left[\log p_\theta(\mathbf{x} \,|\, \mathbf{z}) \right] \\ &- \beta D_{\mathrm{KL}}(q_\phi(\mathbf{z} \,|\, \mathbf{x}) \,||\, p_\theta(\mathbf{z})) \end{aligned} \tag{7}$$

VAEs with discrete variables. Since the categorical distribution is not reparameterizable, training VAEs with categorical latent variables is generally impractical. A solution

to this problem is to replace samples from a categorical distribution with samples from a Gumbel-Softmax distribution (Jang, Gu, and Poole 2016; Maddison, Mnih, and Teh 2016), which can be smoothly annealed into a categorical distribution by making the temperature parameter τ tend to 0. Because Gumbel-Softmax is reparameterizable, the pathwise gradient estimator can be used to get a low-variance—although in this case biased—estimate of the gradient. In this work, we use Bernoulli latent variables (categorical with 2 classes) and their Gumbel-Softmax relaxations.

VAE-IW

Although width-based planning has been shown to be generally very effective for classical planning domains (Lipovetzky and Geffner 2012; 2017; Francès et al. 2017), its performance in practice significantly depends on the quality of the features used. Features that better capture meaningful structure in the environment typically translate to better planning results (Junyent, Jonsson, and Gómez 2019). Here we propose VAE-IW, in which representations extracted by a VAE are used as features for RolloutIW. The main advantages are the following:

- The number of features can be orders of magnitude smaller than B-PROST, leading to faster planning and a smaller memory footprint.
- Autoencoders, in particular VAEs, are a natural approach for learning meaningful, compact representations from data (Bengio, Courville, and Vincent 2013; Tschannen, Bachem, and Lucic 2018; Kingma and Welling 2019).
- No additional preprocessing is needed, such as background masking (Junyent, Jonsson, and Gómez 2019).

On the planning side, we follow Junyent, Jonsson, and Gómez (2019) and use RolloutIW(k) with width $k = 1$ to keep planning efficient even with a small time budget. For example, with a budget of 0.5s per planning phase RolloutIW(2) generates only a few nodes per step, whereas RolloutIW(1) generates hundreds. Width 1 is sufficient to get good results, and this choice makes it easier to compare our experimental results with the aforementioned paper. In the remainder of this section, we detail the two main components of the proposed method: feature learning and planning.

Unsupervised Feature Learning

For feature learning we use a variational autoencoder with a discrete latent space:

$$p_\theta(\mathbf{x}) = \sum_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p_\theta(\mathbf{x} \,|\, \mathbf{z}) p(\mathbf{z}) \tag{8}$$

where the prior is a product of independent Bernoulli distributions with a fixed parameter:

$$p(\mathbf{z}) = \prod_{i=1}^{K} p(z_i) = \prod_{i=1}^{K} \text{Bernoulli}(\mu) \,. \tag{9}$$

Given an image $\mathbf{x}$, we would like to retrieve the binary latent factors $\mathbf{z}$ that generated it, and use them as propositional representation for planning. Since the likelihood function $p_\theta(\mathbf{x} \,|\, \mathbf{z})$ is parameterized by a neural network, and therefore highly nonlinear with respect to the latent variables, inference of the latent variables is intractable (it would require the computation of the sum in (8), which has a number of terms exponential in the number of latent variables).

We define an inference model:

$$q_\phi(\mathbf{z} \,|\, \mathbf{x}) = \prod_{i=1}^{K} q_\phi(z_i \,|\, \mathbf{x}) = \prod_{i=1}^{K} \text{Bernoulli}((\mu_\phi(\mathbf{x}))_i)$$

to approximate the true posterior $p_\theta(\mathbf{z} \,|\, \mathbf{x})$. The *encoder* μ_ϕ is a deep neural network with parameters ϕ that outputs the approximate posterior probabilities of all latent variables given an image $\mathbf{x}$. Using stochastic amortized variational inference, we train the inference and generative models end-to-end by maximizing the ELBO with respect to the parameters of both models. We approximate the discrete Bernoulli distribution of $\mathbf{z}$ with the Gumbel-Softmax relaxation (Jang, Gu, and Poole 2016; Maddison, Mnih, and Teh 2016), which is reparameterizable. This allows us to estimate the gradients of the inference network with the pathwise gradient estimator. Alternatively, the approximate posterior could be optimized directly using a score-function estimator (Williams 1992) with appropriate variance reduction techniques or alternative variational objectives (Bornschein and Bengio 2014; Mnih and Rezende 2016; Le et al. 2018; Masrani, Le, and Wood 2019; Liévin et al. 2020).

Note that we are not necessarily interested in the best generative model—e.g. in terms of marginal log likelihood of the data or visual quality of the generated samples—as much as in representations that are useful for planning. In order to control the tradeoff between the competing terms in (6), we use the β-VAE framework, and following Burgess et al. (2018) we decrease β until good quality reconstructions are obtained.

Planning With Learned Features

After training, the inference model $q_\phi(\mathbf{z} \,|\, \mathbf{x})$ yields approximate posterior distributions of the latent variables. The binary features for downstream planning can be obtained by sampling from the approximate posteriors or by directly thresholding their probabilities:

$$f_i = \begin{cases} 1 & \text{if } \mu_\phi(\mathbf{x})_i \geq \lambda \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where $\lambda \in (0, 1)$ is the threshold. In this work we choose to deterministically threshold the probabilities, as it empirically yields more stable features for planning. Overall, this approach provides a way of efficiently computing a compact, binary representation of an image, which can in turn be interpreted as a set of propositional features to be used in symbolic reasoning systems like IW or RolloutIW.

The planning phase in VAE-IW is based on RolloutIW, but uses the features extracted by the inference network instead of the B-PROST features. Note that while the B-PROST features include temporal information (specifically in the B-PROT subset), the VAE features are computed independently for each frame. This should, everything else being equal, give an advantage to planners that rely on B-PROST features.

Experiments

We evaluate the proposed approach on a suite of 47 Atari 2600 games. We separately trained a VAE for each domain by maximizing (7) using stochastic gradient ascent. As reconstruction loss we used the binary cross-entropy. We used the Adam optimizer (Kingma and Ba 2014) with default parameters and learning rate 10^{-4}. To train the VAEs, we collected 15,000 frames by running RolloutIW(1) with B-PROST features, and split them into a training and validation set of 14,250 and 750 images, respectively. The RGB images of size 210×160 are grayscaled and downsampled to 128×128. VAE-IW experiments were performed with 4GB of RAM on one core of a Xeon Gold 6126 CPU, and a Tesla V100 GPU. The GPU was used both to train the models and to evaluate the encoder in the planning phase.

The encoder consists of 3 convolutional layers interleaved with residual blocks. The single convolutional layers downsample the feature map by a factor of 2, and each residual block includes 2 convolutional layers, resulting in 7 convolutional layers in total. The decoder mirrors such architecture, and performs upsampling with transposed convolutional layers. Further architectural details are provided in the Appendix. The inference network outputs a feature map of shape $H \times W \times C$ which represents the approximate posterior probabilities of the latent variables. Thus, unlike in traditional VAEs where latent variables do not carry any spatial meaning, in our case they are spatially arranged (Vahdat and Kautz 2020). As outlined in the previous section, the Bernoulli probabilities computed by the encoder are thresholded, and the resulting binary features are used as propositional representations for planning in RolloutIW(1), instead of the B-PROST features. For the planning phase of VAE-IW, we use RolloutIW(1) with partial caching and risk aversion (RA) as described by Bandres, Bonet, and Geffner (2018). With partial caching, after each iteration of RolloutIW the branch of the chosen action is kept in memory. Risk aversion is attained by multiplying all negative rewards by a factor $\alpha \gg 1$ when they are propagated up the tree.

Because of their diversity, Atari games vary widely in complexity. We empirically chose a set of hyperparameters that performed reasonably well on a small subset of the games, assuming this would generalize well enough to other games. Following previous work (Lipovetzky and Geffner 2012; Bandres, Bonet, and Geffner 2018), we used a frame skip of 15 and a planning budget of 0.5s per time step. We set an additional limit of 15,000 executed actions for each run, to prevent runs from lasting too long. Note that this constraint is only applied to our method. Table 1 summarizes the main hyperparameters used in our experiments. We expect that better results can be obtained by performing an extensive hyperparameter search on the whole suite of Atari 2600 games.

Parameter	Value
Batch size	64
Learning rate	10^{-4}
Latent space size	$15 \times 15 \times 20 = 4500$
τ	0.5
β	10^{-4}
μ	0.5
α	50000
γ	0.99
λ	0.9
Frame skip	15
Planning budget	0.5s

Table 1: Hyperparameters in VAE-IW experiments unless specified otherwise.

Main Results

In Table 2, we compare the planning performance of our RA VAE-IW(1) to IW(1) with B-PROST features (Lipovetzky and Geffner 2012) and RA RolloutIW(1) (Bandres, Bonet, and Geffner 2018) with B-PROST features. In Figure 1, we normalize the scores of width-based planning methods using scores from random and human play, as in Bandres, Bonet, and Geffner (2018). Note, however, that human play is only meant as a baseline: since humans do not have access to a simulator, a direct comparison cannot be made. Following the literature, we report the average score over 5 runs, with each run ending when the game is over. These results show that learning binary features purely from images, without any supervision, can be beneficial for width-based planning. In particular, using the learned representations in RolloutIW leads to generally higher scores in the Atari 2600 domain.

Note that the performance of VAE-IW depends on the quality and expressiveness of the features extracted by the VAE, which in turn depend on the images the VAE is trained on. Crucially, since we collect data by running RolloutIW(1) with B-PROST features, the performance of VAE-IW is constrained by the effectiveness of the data collection algorithm. The VAE features will not necessarily be meaningful on parts of the game that are significantly different from those in the training set. Surprisingly, however, our method significantly outperforms the baseline that was used for data collection, providing even stronger evidence that the compact set of features learned by a VAE can be successfully utilized for width-based planning algorithms. This form of bootstrapping can be iterated: images collected by VAE-IW could be used for re-training or fine-tuning VAEs, potentially leading to further performance improvements. Although in principle the first iteration of VAEs could be trained from images collected via random play, this is not a viable solution in hard-exploration problems such as many Atari games (Ecoffet et al. 2019).

In addition, there appears to be a significant negative correlation between the average number of true features and the average number of expanded nodes per planning phase (Spearman rank correlation of -0.51, p-value < 0.001). In other words, in domains where the VAE extracts on average more true features, the planning algorithm tends to expand fewer nodes. Thus, it could be potentially fruitful to further investigate the interplay between the average number of true features, the meaningfulness and usefulness of the features, and the efficiency of width-based planning algorithm.

Algorithm	Human	IW	RA Rollout IW	RA VAE-IW
Features		B-PROST	B-PROST	VAE $15 \times 15 \times 20$
Planning budget		0.5s	0.5s	0.5s
Alien	6,875.0	1,316.0	7,170.0	**8,144.0**
Amidar	1,676.0	48.0	1,049.2	**1,551.0**
Assault	1,496.0	268.8	336.0	**1,250.0**
Asterix	8,503.0	1,350.0	46,100.0	**999,500.0***
Asteroids	13,157.0	840.0	4,698.0	**14,816.0**
Atlantis	29,028.0	33,160.0	122,220.0	**1,955,280.0***
Battle zone	37,800.0	6,800.0	74,600.0	**191,000.0**
Beam rider	5,775.0	715.2	2,552.8	**3,432.0**
Berzerk		280.0	**1,208.0**	828.0
Bowling	154.0	30.6	44.2	**63.0**
Breakout	31.8	1.6	**86.2**	50.8
Centipede	11,963.0	88,890.0	56,328.0	**113,369.2***
Crazy climber	35,411.0	16,780.0	40,440.0	**941,660.0***
Demon attack	3,401.0	106.0	6,958.0	**276,586.0***
Double dunk	-15.5	-22.0	3.2	**10.4**
Elevator action		1,080.0	0.0	**38,940.0***
Fishing derby	5.5	-83.8	-77.0	**-19.6**
Freeway	29.6	0.6	2.0	**4.8**
Frostbite	4,335.0	106.0	146.0	**254.0**
Gopher	2,321.0	1,036.0	**8,388.0**	7,968.0
Gravitar	2,672.0	380.0	1,660.0	**2,360.0**
Ice hockey	0.9	-13.6	-12.4	**37.6**
James bond 007	406.7	40.0	**10,760.0**	5,280.0
Krull	2,395.0	3,206.8	2,091.8	**3,455.2**
Kung-fu master	22,736.0	440.0	2,620.0	**5,300.0**
Ms. Pac-man	15,693.0	2,578.0	15,115.0	**16,200.6**
Name this game	4,076.0	7,070.0	6,558.0	**18,526.0**
Phoenix		1,266.0	**6,790.0**	6,160.0
Pitfall!		-8.6	-302.8	**-5.8**
Pong	9.3	-20.8	-4.2	**10.4**
Private eye	69,571.0	**2,690.8**	-480.0	60.0
Q*bert	13,455.0	515.0	**15,970.0**	2,070.0
River raid	13,513.0	664.0	6,288.0	**6,818.0**
Road Runner	7,845.0	200.0	**31,140.0**	2,740.0
Robotank	11.9	3.2	**31.2**	30.8
Seaquest	20,182.0	168.0	**2,312.0**	560.0
Skiing		-16,511.0	-16,006.8	**-10,443.8**
Space invaders	1,652.0	280.0	1,149.0	**2,943.0**
Stargunner	10,250.0	840.0	**14,900.0**	1,040.0
Tennis	-8.9	-23.4	-5.4	**-0.6**
Time pilot	5,925.0	2,360.0	3,540.0	**32,440.0**
Tutankham	167.7	71.2	135.6	**180.6**
Up'n down	9,082.0	928.0	34,668.0	**764,264.0***
Video pinball	17,298.0	28,706.4	**216,468.6**	149,284.6
Wizard of wor	4,757.0	5,660.0	43,860.0	**199,900.0**
Yars' revenge		6,352.6	7,848.8	**105,637.0**
Zaxxon	9,173.0	0.0	**15,500.0**	12,120.0
# > Human	n/a	4	18	24
# > 75% Human	n/a	4	21	27
# best	n/a	1	12	34

Table 2: Score comparison of width-based methods in 47 Atari games in the pixel setting. Scores in bold indicate the best overall width-based method, and scores with a star indicate that the algorithm reached the limit on executed actions at least once. In the bottom rows we also report the number of domains in which an algorithm's score was greater than the human score, or greater than 75% of the human score. Results for IW and Rollout IW B-PROST are from Bandres, Bonet, and Geffner (2018); human scores are from Liang et al. (2015).

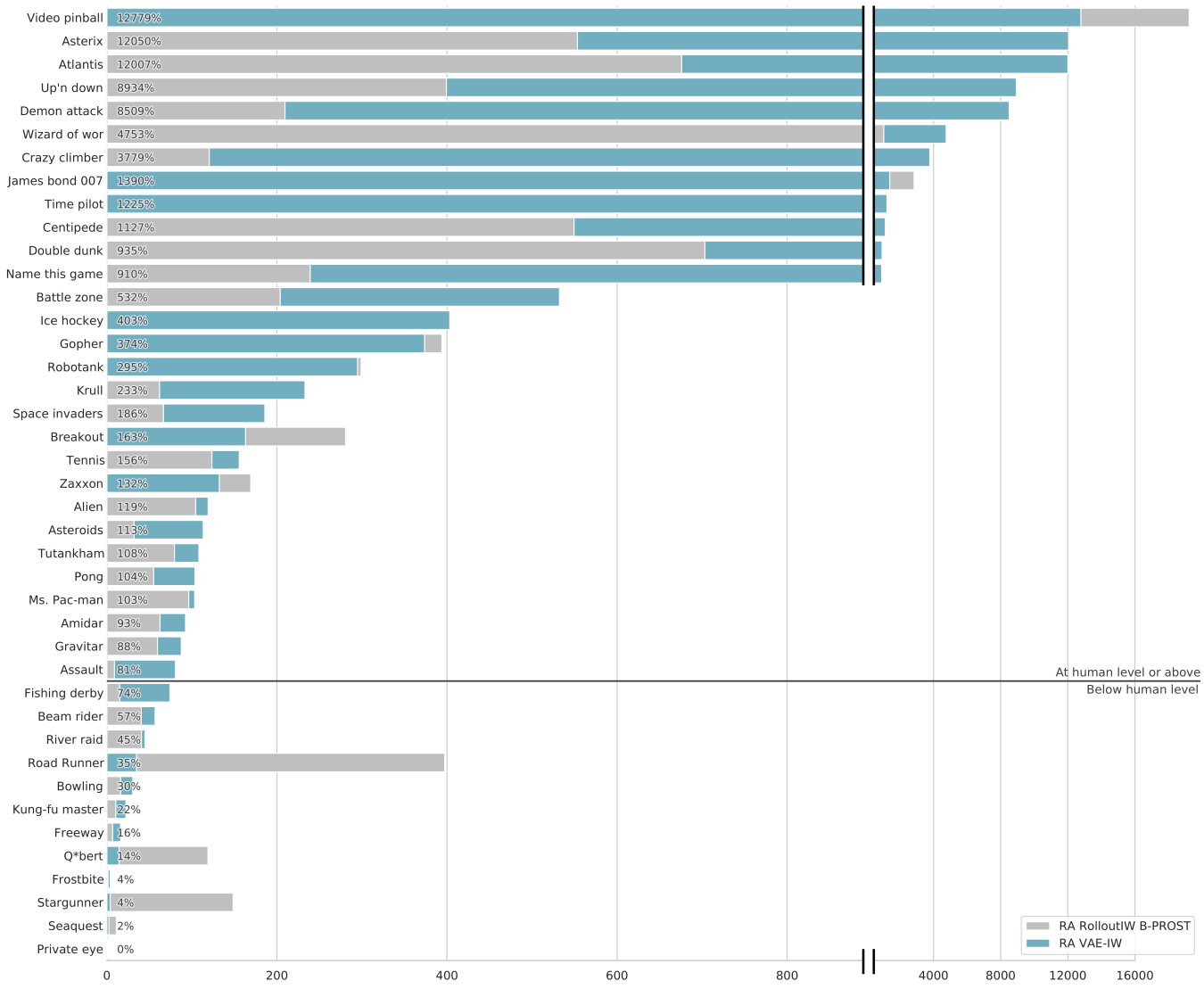

Figure 1: Comparison of the risk-averse variants of VAE-IW (ours, using VAE features) and RolloutIW B-PROST (using B-PROST features). Following Mnih et al. (2015), the performance of both methods is normalized with respect to a professional human game tester (100% level) and random play (0%) as: $100 \times (\text{VAE} - \text{random play})/(\text{human score} - \text{random play})$. RA VAE-IW obtains the highest score among width-based approaches in most games, and it performs at a level that is superior to or comparable with professional human play. The reported percentages are for VAE features.

Ablation Studies

The performance of VAE-IW depends on several hyperparameters related to the planning algorithm and to the probabilistic model. Here we attempt to investigate the effect of some of these parameters.

Modeling choices. One of the major modeling choices is the dimensionality of the latent space, and the spatial structure $H \times W \times C$ of the latent variables. Both of these factors are tightly coupled with the neural architecture underlying the inference and generative networks. As there is no clear heuristic, we explored different neural architectures and latent space sizes. Based on the performance on a few selected domains, we chose two different settings, with latent space size $15 \times 15 \times 20$ and $4 \times 4 \times 200$ (see the Appendix for further details). In Table 12 we compare the performance of RA VAE-IW on these two configurations, keeping the rest of the hyperparameters fixed as the ones used in Table 2. While overall the $15 \times 15 \times 20$ configuration leads to a higher score in most domains, the effect of this modeling choice seems to significantly depend on the domain.

As previously mentioned, we consider the framework of β-VAEs in which β controls the trade-off between reconstruction accuracy and amount of information encoded in

the latent space. For our purposes, β has to be small enough that the model can capture all relevant details in an image. In practice, we decreased β until the model generated sufficiently accurate reconstructions on a selection of Atari games (Burgess et al. 2018). Table 13 reports the performance of VAE-IW when varying $\beta \in \{10^{-4}, 10^{-3}\}$, and shows that the effect of varying β depends on the domain. Intuitively, while a stronger regularization (i.e. higher β) can be detrimental for the reconstructions and thus also for the informativeness of the learned features, it may lead to better representations in less visually challenging domains. In practice, one could for example train VAEs with different regularization strengths, and do unsupervised model selection separately for each domain by using the "elbow method" (Ketchen and Shook 1996) on the reconstruction loss or other relevant metrics.

Planning parameters. Regardless of the features used for planning, the performance of VAE-IW depends on the variant of RolloutIW being used, and on its parameters. In Table 8 we compare the average score of VAE-IW with and without risk aversion, and observe that the risk-averse variant achieves an equal or higher average score in 33 of the 47 domains.

Table 9 shows the results of VAE-IW and RolloutIW with B-PROST features, similarly to Table 2, except that both methods are run *without* risk aversion. With this modification, our method still obtains a higher average score in the majority of domains (32/47).

Another crucial planning parameter is the time budget for planning at each time step. While the main results are based on a 0.5s budget, we also consider a 32s budget, following Bandres, Bonet, and Geffner (2018). In Table 10 we observe that, not surprisingly, the high time budget outperforms the low budget in most domains (34/47). However, in some of them the shorter planning budget yields a significantly higher score (e.g. in Asterix, CrazyClimber, and ElevatorAction). Interestingly, increasing the planning budget seems to leave the average rollout depth unaffected, while the average number of expanded nodes in each planning phase grows significantly. This behaviour is consistently observed in all tested domains (see Figures 2 and 3) and points to the fact that increasing the planning budget improves results mostly by allowing more rollouts.

In Table 11 we compare the average scores obtained by VAE-IW and RolloutIW with B-PROST features, using a 32s planning budget for both methods. Once again, using the compact features learned by a VAE seems to be beneficial, as VAE-IW obtains the highest average score in 29 of the 47 domains.

Related Work

Variational Autoencoders (VAEs) have been extensively used for representation learning (Kingma and Welling 2019) as their amortized inference network lends itself naturally to this task. In the context of automated planning, Asai and Fukunaga (2017) proposed the State Autoencoder (SAE) for propositional symbolic grounding. An SAE is in fact a VAE with Bernoulli latent variables. It is trained by maximizing a modified ELBO that includes an additional entropy regularizer, defined as twice the negative KL divergence. Thus, the objective function being maximized is the ELBO (6) with the sign of the KL flipped. Although unintentional (Asai and Kajino 2019), this proved to be fundamental for the mitigation of the issue of representation instability. A variation of SAE, the zero-suppressed state autoencoder (Asai and Kajino 2019), adds a further regularization term to the propositional representation (features), leading to more stable representations (Asai and Kajino 2019).

Zhang et al. (2018) take a supervised approach to representation learning for planning, and learn a transition graph for planning in the representation space with Dijkstra's algorithm. Konidaris, Kaelbling, and Lozano-Perez (2018) specify a set of skills for a task, and then automatically extract state representations from raw observations. Kurutach et al. (2018) use generative adversarial networks to learn structured representations of images and a deterministic dynamics model, and plan with graph-search methods.

Junyent, Jonsson, and Gómez (2019) proposed π-IW, a variant of RolloutIW(1) where a neural network guides the action selection process in the rollouts, which would otherwise be random. This is reminiscent of AlphaZero (Silver et al. 2018), where a policy network guides the rollouts of Monte Carlo Tree Search (MCTS). Moreover, π-IW plans using features obtained from the last hidden layer of the policy network, instead of B-PROST.

Conclusion

We have introduced a novel combination of width-based planning with learning techniques. The learning employs a VAE to learn relevant features in video games from the Atari 2600 suite, given raw images of screen states as training data. The planning is done with RolloutIW(1) using the features learned by the VAE. Our approach reduces the size of the feature set from the 20.5 million B-PROST features used in previous work in connection with RolloutIW, to only 4,500. Our algorithm, VAE-IW, outperforms the previous methods, proving that VAEs can learn meaningful representations that can be effectively used for width-based planning.

In VAE-IW, the symbolic representations are learned from data collected by RolloutIW using B-PROST features. Increasing the diversity and quality of the training data could potentially lead to better representations, and thus better planning results. One possible way to achieve this could be to iteratively retrain or fine-tune the VAEs on data collected by the current iteration of VAE-IW: The planner would produce new images to retrain the VAE, which could again be used in combination with RolloutIW, resulting in a new generation of VAE-IW. The quality of the representations could also be improved by using more expressive discrete models, for example with a hierarchy of discrete latent variables (Van Den Oord, Vinyals, and Kavukcuoglu 2017; Razavi, van den Oord, and Vinyals 2019). Finally, we can expect further improvements orthogonal to this work, by learning a rollout policy for more effective action selection, as investigated by Junyent, Jonsson, and Gómez (2019).

References

Alemi, A. A.; Poole, B.; Fischer, I.; Dillon, J. V.; Saurous, R. A.; and Murphy, K. 2017. Fixing a broken elbo. *arXiv preprint arXiv:1711.00464*.

Asai, M., and Fukunaga, A. 2017. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154*.

Asai, M., and Kajino, H. 2019. Towards stable symbol grounding with zero-suppressed state autoencoder. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 592–600.

Bandres, W.; Bonet, B.; and Geffner, H. 2018. Planning with pixels in (almost) real time. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8):1798–1828.

Bornschein, J., and Bengio, Y. 2014. Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*.

Burgess, C. P.; Higgins, I.; Pal, A.; Matthey, L.; Watters, N.; Desjardins, G.; and Lerchner, A. 2018. Understanding disentangling in $beta$-vae. *arXiv preprint arXiv:1804.03599*.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Francès, G.; Ramírez Jávega, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely declarative action descriptions are overrated: Classical planning with simulators. In *IJCAI 2017. Twenty-Sixth International Joint Conference on Artificial Intelligence*.

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. In *ICLR 2017*.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jinnai, Y., and Fukunaga, A. 2017. Learning to prune dominated action sequences in online black-box planning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Junyent, M.; Jonsson, A.; and Gómez, V. 2019. Deep policies for width-based planning in pixel domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 646–654.

Ketchen, D. J., and Shook, C. L. 1996. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal* 17(6):441–458.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P., and Welling, M. 2019. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research* 61:215–289.

Kurutach, T.; Tamar, A.; Yang, G.; Russell, S. J.; and Abbeel, P. 2018. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, 8733–8744.

Le, T. A.; Kosiorek, A. R.; Siddharth, N.; Teh, Y. W.; and Wood, F. 2018. Revisiting reweighted wake-sleep. *arXiv preprint arXiv:1805.10469*.

Liang, Y.; Machado, M. C.; Talvitie, E.; and Bowling, M. 2015. State of the art control of atari games using shallow reinforcement learning. *arXiv preprint arXiv:1512.01563*.

Liévin, V.; Dittadi, A.; Christensen, A.; and Winther, O. 2020. Optimal variance control of the score function gradient estimator for importance weighted bounds. *arXiv preprint arXiv:2008.01998*.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*.

Lipovetzky, N., and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *AAAI*, 3590–3596.

Lipovetzky, N.; Ramirez, M.; and Geffner, H. 2015. Classical planning with simulators: Results on the atari video games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Masrani, V.; Le, T. A.; and Wood, F. 2019. The thermodynamic variational objective. In *Advances in Neural Information Processing Systems*, 11521–11530.

Mnih, A., and Rezende, D. J. 2016. Variational inference for monte carlo objectives. *arXiv preprint arXiv:1602.06725*.

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.

Razavi, A.; van den Oord, A.; and Vinyals, O. 2019. Generating diverse high-fidelity images with VQ-VAE-2. In *Advances in Neural Information Processing Systems*, 14866–14876.

Rezende, D. J.; Mohamed, S.; and Wierstra, D. 2014. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Shleyfman, A.; Tuisov, A.; and Domshlak, C. 2016. Blind

search for atari-like online planning revisited. In *IJCAI*, 3251–3257.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* 362(6419):1140–1144.

Tschannen, M.; Bachem, O.; and Lucic, M. 2018. Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069*.

Vahdat, A., and Kautz, J. 2020. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*.

Van Den Oord, A.; Vinyals, O.; and Kavukcuoglu, K. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, 6306–6315.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Zhang, A.; Sukhbaatar, S.; Lerer, A.; Szlam, A.; and Fergus, R. 2018. Composable planning with attributes. In *International Conference on Machine Learning*, 5842–5851.

Appendix

BatchNorm
LeakyReLU(0.01)
Conv 3×3, padding=1
Dropout(0.2)
BatchNorm
LeakyReLU(0.01)
Conv 3×3, padding=1
Dropout(0.2)
Residual connection
LeakyReLU(0.01)

Table 3: Residual block. The number of channels is always 64. The residual connection consists of summing the activation to the block's input.

Conv 3×3, stride=2
Residual block
Conv 3×3, stride=2
Residual block
Conv 3×3, stride=2
Residual block
Conv 3×3, stride=2
Residual block
Conv 3×3, stride=2, padding=1

Table 4: Encoder with output of spatial size 4×4. The number of channels is always 64 except for the last convolution, in which the number of output channels controls the latent space size.

ConvT 3×3, stride=2
Residual block
ConvT 4×4, stride=2
Residual block
ConvT 4×4, stride=2
Crop 128×128
Sigmoid activation

Table 5: Decoder with input of spatial size 15×15. The number of channels is always 64 except for the last convolution, in which the number of output channels is 1. ConvT denotes transposed convolutional layers.

Conv 4×4, stride=2
Residual block
Conv 4×4, stride=2
Residual block
Conv 3×3, stride=2, padding=1

Table 6: Encoder with output of spatial size 15×15. The number of channels is always 64 except for the last convolution, in which the number of output channels controls the latent space size.

ConvT 3×3, stride=2
Residual block
ConvT 3×3, stride=2
Residual block
ConvT 3×3, stride=2
Residual block
ConvT 3×3, stride=2
Residual block
ConvT 3×3, stride=2
Crop 128×128
Sigmoid activation

Table 7: Decoder with input of spatial size 4×4. The number of channels is always 64 except for the last convolution, in which the number of output channels is 1. ConvT denotes transposed convolutional layers.

Algorithm	VAE-IW	RA VAE-IW
β	10^{-4}	10^{-4}
Planning horizon	0.5s	0.5s
Features	VAE $15 \times 15 \times 20$	VAE $15 \times 15 \times 20$
Alien	5,576.0	**8,144.0**
Amidar	1,093.2	**1,551.0**
Assault	826.4	**1,250.0**
Asterix	**999,500.0***	**999,500.0***
Asteroids	772.0	**14,816.0**
Atlantis	40,620.0	**1,955,280.0***
Battle zone	91,200.0	**191,000.0**
Beam rider	3,179.6	**3,432.0**
Berzerk	566.0	**828.0**
Bowling	**64.6**	63.0
Breakout	13.0	**50.8**
Centipede	22,020.2	**113,369.2**
Crazy climber	661,460.0	**941,660.0***
Demon attack	9,656.0	**276,586.0***
Double dunk	7.2	**10.4**
Elevator action	**76,160.0***	38,940.0*
Fishing derby	-20.2	**-19.6**
Freeway	**6.2**	4.8
Frostbite	248.0	**254.0**
Gopher	6,084.0	**7,968.0**
Gravitar	**2,770.0**	2,360.0
Ice hockey	36.0	**37.6**
James bond 007	640.0	**5,280.0**
Krull	**3,543.2**	3,455.2
Kung-fu master	5,160.0	**5,300.0**
Ms. Pac-man	**20,161.0**	16,200.6
Name this game	13,332.0	**18,526.0**
Phoenix	5,328.0	**6,160.0**
Pitfall!	**0.0**	-5.8
Pong	9.2	**10.4**
Private eye	**139.0**	60.0
Q*bert	1,890.0	**2,070.0**
River raid	4,884.0	**6,818.0**
Road Runner	**4,720.0**	2,740.0
Robotank	21.4	**30.8**
Seaquest	**596.0**	560.0
Skiing	**-9,664.8**	-10,443.8
Space invaders	1,962.0	**2,943.0**
Stargunner	**1,120.0**	1,040.0
Tennis	**5.2**	-0.6
Time pilot	24,840.0	**32,440.0**
Tutankham	167.4	**180.6**
Up'n down	35,964.0	**764,264.0***
Video pinball	**462,619.4**	149,284.6
Wizard of wor	89,380.0	**199,900.0***
Yars' revenge	85,800.6	**105,637.0**
Zaxxon	7,320.0	**12,120.0**
# best	15	33

Table 8: Score comparison of VAE-IW with and without risk aversion. Scores in bold indicate the best method, and scores with a star indicate that the algorithm reached the limit on executed actions at least once. Ties are counted as won for both methods.

Algorithm	Rollout IW	VAE-IW
β		10^{-4}
Planning horizon	0.5s	0.5s
Features	B-PROST	VAE $15 \times 15 \times 20$
Alien	4,238.0	**5,576.0**
Amidar	659.8	**1,093.2**
Assault	285.0	**826.4**
Asterix	45,780.0	**999,500.0**
Asteroids	**4,344.0**	772.0
Atlantis	**64,200.0**	40,620.0
Battle zone	39,600.0	**91,200.0**
Beam rider	2,188.0	**3,179.6**
Berzerk	**644.0**	566.0
Bowling	47.6	**64.6**
Breakout	**82.4**	13.0
Centipede	**36,980.2**	22,020.2
Crazy climber	39,220.0	**661,460.0***
Demon attack	2,780.0	**9,656.0**
Double dunk	3.6	**7.2**
Elevator action	0.0	**76,160.0***
Fishing derby	-68.0	**-20.2**
Freeway	2.8	**6.2**
Frostbite	220.0	**248.0**
Gopher	**7,216.0**	6,084.0
Gravitar	1,630.0	**2,770.0**
Ice hockey	-6.0	**36.0**
James bond 007	450.0	**640.0**
Krull	1,892.8	**3,543.2**
Kung-fu master	2,080.0	**5,160.0**
Ms. Pac-man	9,178.4	**20,161.0**
Name this game	6,226.0	**13,332.0**
Phoenix	**5,750.0**	5,328.0
Pitfall!	-81.4	**0.0**
Pong	-7.4	**9.2**
Private eye	-322.0	**139.0**
Q*bert	**3,375.0**	1,890.0
River raid	**6,088.0**	4,884.0
Road Runner	2,360.0	**4,720.0**
Robotank	**31.0**	21.4
Seaquest	**980.0**	596.0
Skiing	-15,738.8	**-9,664.8**
Space invaders	**2,628.0**	1,962.0
Stargunner	**13,360.0**	1,120.0
Tennis	-18.6	**5.2**
Time pilot	7,640.0	**24,840.0**
Tutankham	128.4	**167.4**
Up'n down	**36,236.0**	35,964.0
Video pinball	203,765.4	**462,619.4**
Wizard of wor	37,220.0	**89,380.0**
Yars' revenge	5,225.4	**85,800.6**
Zaxxon	**9,280.0**	7,320.0
# best	15	32

Table 9: Score comparison between RolloutIW B-PROST and VAE-IW. Scores in bold indicate the best method, and scores with a star indicate that the algorithm reached the limit on executed actions at least once. Ties are counted as won for both methods.

Algorithm	VAE-IW	VAE-IW
β	10^{-4}	10^{-4}
Planning horizon	0.5s	32s
Features	VAE $15 \times 15 \times 20$	VAE $15 \times 15 \times 20$
Alien	5,576.0	**8,536.0**
Amidar	1,093.2	**1,955.8**
Assault	826.4	**1,338.4**
Asterix	**999,500.0**	417,100.0
Asteroids	772.0	**1,158.0**
Atlantis	40,620.0	**49,500.0**
BattleZone	91,200.0	**234,200.0**
BeamRider	3,179.6	**5,580.0**
Berzerk	**566.0**	554.0
Bowling	**64.6**	46.8
Breakout	13.0	**72.4**
Centipede	22,020.2	**166,244.8**
CrazyClimber	**661,460.0**	129,840.0
DemonAttack	**9,656.0**	5,397.0
DoubleDunk	**7.2**	5.2
ElevatorAction	**76,160.0**	0.0
FishingDerby	-20.2	**20.6**
Freeway	6.2	**29.4**
Frostbite	248.0	**280.0**
Gopher	6,084.0	**17,604.0**
Gravitar	**2,770.0**	2,640.0
IceHockey	36.0	**44.2**
Jamesbond	640.0	**650.0**
Krull	3,543.2	**6,664.0**
KungFuMaster	5,160.0	**20,960.0**
MsPacman	20,161.0	**25,759.0**
NameThisGame	13,332.0	**15,276.0**
Phoenix	5,328.0	**5,960.0**
Pitfall	**0.0**	-0.4
Pong	9.2	**12.0**
PrivateEye	139.0	**157.8**
Qbert	1,890.0	**4,760.0**
Riverraid	4,884.0	**5,372.0**
RoadRunner	4,720.0	**8,540.0**
Robotank	21.4	**24.2**
Seaquest	**596.0**	324.0
Skiing	**-9,664.8**	-9,705.0
SpaceInvaders	1,962.0	**2,972.0**
StarGunner	1,120.0	**1,180.0**
Tennis	5.2	**12.6**
TimePilot	**24,840.0**	24,220.0
Tutankham	167.4	**197.2**
UpNDown	35,964.0	**91,592.0**
VideoPinball	462,619.4	**833,518.4**
WizardOfWor	**89,380.0**	76,460.0
YarsRevenge	85,800.6	**188,551.2**
Zaxxon	7,320.0	**30,200.0**
# best	13	34

Table 10: Score comparison of VAE-IW with planning budget of 0.5 or 32 seconds. Scores in bold indicate the best method, and scores with a star indicate that the algorithm reached the limit on executed actions at least once. Ties are counted as won for both methods.

Algorithm	RolloutIW	VAE-IW
β		10^{-4}
Planning horizon	32s	32s
Features	B-PROST	VAE $15 \times 15 \times 20$
Alien	6,896.0	**8,536.0**
Amidar	1,698.6	**1,955.8**
Assault	319.2	**1,338.4**
Asterix	66,100.0	**417,100.0**
Asteroids	**7,258.0**	1,158.0
Atlantis	**151,120.0**	49,500.0
BattleZone	**414,000.0**	234,200.0
BeamRider	2,464.8	**5,580.0**
Berzerk	**862.0**	554.0
Bowling	45.8	**46.8**
Breakout	36.0	**72.4**
Centipede	65,162.6	**166,244.8**
CrazyClimber	43,960.0	**129,840.0**
DemonAttack	**9,996.0**	5,397.0
DoubleDunk	**20.0**	5.2
ElevatorAction	**0.0**	**0.0**
FishingDerby	-16.2	**20.6**
Freeway	12.6	**29.4**
Frostbite	**5,484.0**	280.0
Gopher	13,176.0	**17,604.0**
Gravitar	**3,700.0**	2,640.0
IceHockey	6.6	**44.2**
Jamesbond	**22,250.0**	650.0
Krull	1,151.2	**6,664.0**
KungFuMaster	14,920.0	**20,960.0**
MsPacman	19,667.0	**25,759.0**
NameThisGame	5,980.0	**15,276.0**
Phoenix	**7,636.0**	5,960.0
Pitfall	-130.8	**-0.4**
Pong	**17.6**	12.0
PrivateEye	**3,157.2**	157.8
Qbert	**8,390.0**	4,760.0
Riverraid	**8,156.0**	5,372.0
RoadRunner	**37,080.0**	8,540.0
Robotank	**52.6**	24.2
Seaquest	**10,932.0**	324.0
Skiing	-16,477.0	**-9,705.0**
SpaceInvaders	1,980.0	**2,972.0**
StarGunner	**15,640.0**	1,180.0
Tennis	-2.2	**12.6**
TimePilot	8,140.0	**24,220.0**
Tutankham	184.0	**197.2**
UpNDown	44,306.0	**91,592.0**
VideoPinball	382,294.8	**833,518.4**
WizardOfWor	73,820.0	**76,460.0**
YarsRevenge	9,866.4	**188,551.2**
Zaxxon	22,880.0	**30,200.0**
# best	19	29

Table 11: Score comparison between RolloutIW B-PROST and VAE-IW with planning budget of 32 seconds. Scores in bold indicate the best method, and scores with a star indicate that the algorithm reached the limit on executed actions at least once. Ties are counted as won for both methods.

Algorithm	RA VAE-IW	RA VAE-IW
β	10^{-4}	10^{-4}
Planning horizon	0.5s	0.5s
Features	VAE $15 \times 15 \times 20$	VAE $4 \times 4 \times 200$
Alien	**8,144.0**	7,592.0
Amidar	**1,551.0**	1,526.6
Assault	1,250.0	**1,308.6**
Asterix	**999,500.0***	**999,500.0***
Asteroids	**14,816.0**	8,852.0
Atlantis	**1,955,280.0***	1,912,700.0*
Battle zone	191,000.0	**228,000.0**
Beam rider	3,432.0	**3,450.0**
Berzerk	**828.0**	752.0
Bowling	**63.0**	47.6
Breakout	**50.8**	28.4
Centipede	113,369.2	**253,823.6**
Crazy climber	**941,660.0***	930,420.0*
Demon attack	276,586.0*	**292,099.0***
Double dunk	**10.4**	6.0
Elevator action	38,940.0*	**109,680.0***
Fishing derby	**-19.6**	-31.2
Freeway	**4.8**	2.0
Frostbite	**254.0**	244.0
Gopher	7,968.0	**8,504.0**
Gravitar	**2,360.0**	1,560.0
Ice hockey	**37.6**	37.2
James bond 007	5,280.0	**11,000.0**
Krull	**3,455.2**	3,219.0
Kung-fu master	**5,300.0**	3,600.0
Ms. Pac-man	**16,200.6**	15,066.8
Name this game	**18,526.0**	13,670.0
Phoenix	6,160.0	**9,210.0**
Pitfall!	**-5.8**	-9.6
Pong	**10.4**	3.6
Private eye	60.0	**115.4**
Q*bert	2,070.0	**4,935.0**
River raid	**6,818.0**	6,790.0
Road Runner	**2,740.0**	2,320.0
Robotank	30.8	**45.2**
Seaquest	560.0	**1,084.0**
Skiing	**-10,443.8**	-11,906.4
Space invaders	**2,943.0**	2,753.0
Stargunner	1,040.0	**1,200.0**
Tennis	**-0.6**	-6.6
Time pilot	**32,440.0**	23,460.0
Tutankham	**180.6**	158.4
Up'n down	**764,264.0***	627,706.0*
Video pinball	149,284.6	**248,101.2**
Wizard of wor	**199,900.0***	111,580.0
Yars' revenge	**105,637.0**	97,004.6
Zaxxon	12,120.0	**17,360.0**
# best	31	17

Table 12: Score comparison of VAE-IW with latent space size $15 \times 15 \times 20$ and $4 \times 4 \times 200$. Scores in bold indicate the best method, and scores with a star indicate that the algorithm reached the limit on executed actions at least once. Ties are counted as won for both methods.

Algorithm	RA VAE-IW	RA VAE-IW
β	10^{-3}	10^{-4}
Planning horizon	0.5s	0.5s
Features	VAE $15 \times 15 \times 20$	VAE $15 \times 15 \times 20$
Alien	4,902.0	**8,144.0**
Amidar	1,186.2	**1,551.0**
Assault	**1,468.2**	1,250.0
Asterix	**999,500.0***	**999,500.0***
Asteroids	7,204.0	**14,816.0**
Atlantis	**1,978,540.0***	1,955,280.0*
Battle zone	**406,600.0**	191,000.0
Beam rider	**4,377.6**	3,432.0
Berzerk	774.0	**828.0**
Bowling	35.0	**63.0**
Breakout	**53.4**	50.8
Centipede	**296,791.4**	113,369.2
Crazy climber	**976,580.0***	941,660.0*
Demon attack	**301,886.0**	276,586.0
Double dunk	7.4	**10.4**
Elevator action	**68,920.0**	38,940.0
Fishing derby	**-15.6**	-19.6
Freeway	4.0	**4.8**
Frostbite	**262.0**	254.0
Gopher	5,420.0	**7,968.0**
Gravitar	2,300.0	**2,360.0**
Ice hockey	34.0	**37.6**
James bond 007	630.0	**5,280.0**
Krull	**3,486.4**	3,455.2
Kung-fu master	4,960.0	**5,300.0**
Ms. Pac-man	**17,483.0**	16,200.6
Name this game	15,120.0	**18,526.0**
Phoenix	5,524.0	**6,160.0**
Pitfall!	-6.8	**-5.8**
Pong	-2.2	**10.4**
Private eye	40.0	**60.0**
Q*bert	**8,040.0**	2,070.0
River raid	6,078.0	**6,818.0**
Road Runner	2,080.0	**2,740.0**
Robotank	**44.6**	30.8
Seaquest	316.0	**560.0**
Skiing	-11,027.6	**-10,443.8**
Space invaders	2,721.0	**2,943.0**
Stargunner	**1,100.0**	1,040.0
Tennis	-16.6	**-0.6**
Time pilot	30,920.0	**32,440.0**
Tutankham	165.8	**180.6**
Up'n down	682,080.0*	**764,264.0***
Video pinball	**445,085.8**	149,284.6
Wizard of wor	184,260.0	**199,900.0***
Yars' revenge	77,950.8	**105,637.0**
Zaxxon	10,520.0	**12,120.0**
# best	18	30

Table 13: Score comparison of RA VAE-IW with different values of β. Scores in bold indicate the best method, and scores with a star indicate that the algorithm reached the limit on executed actions at least once. Ties are counted as won for both methods.

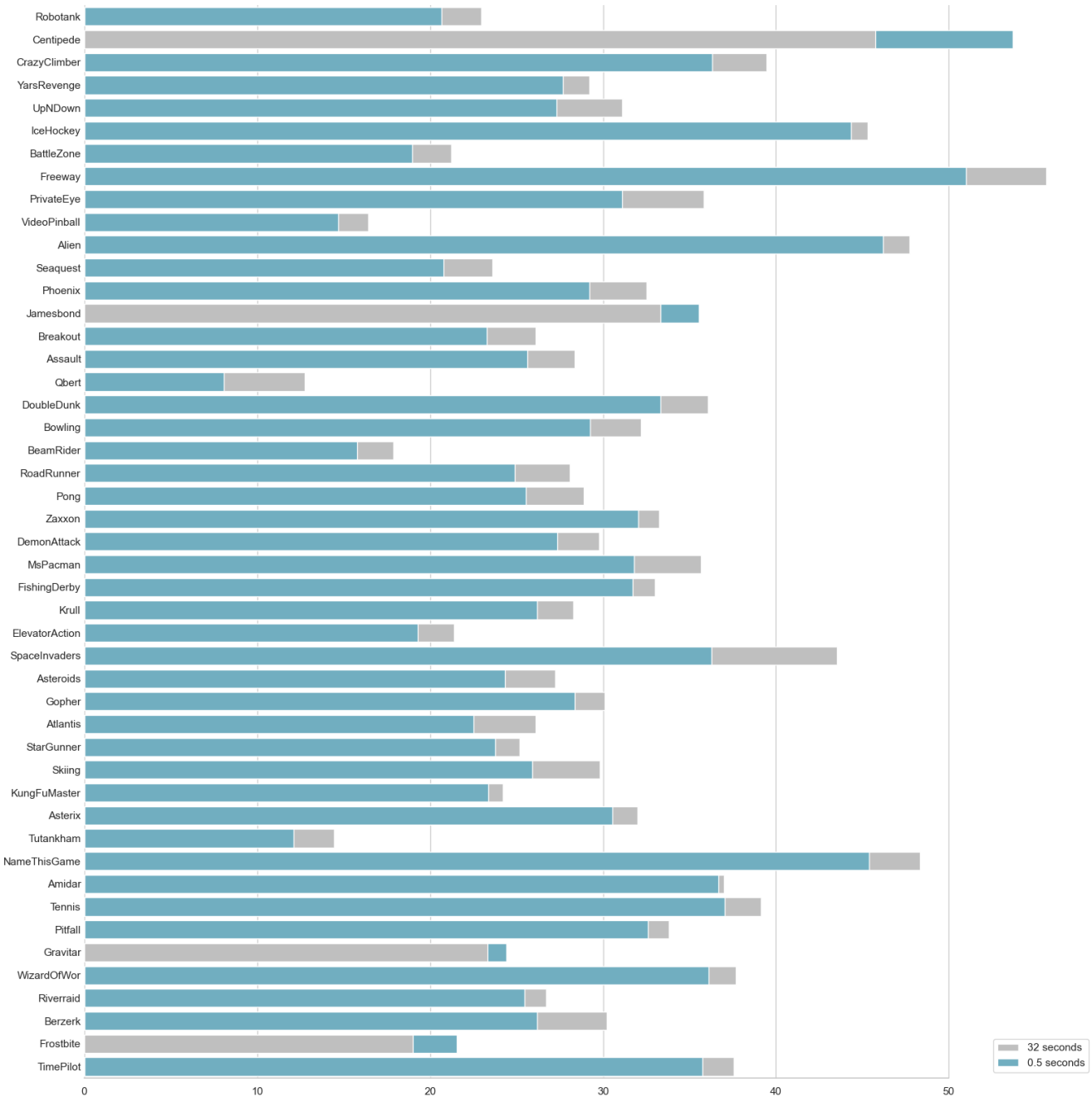

Figure 2: We calculate the mean number of expanded nodes of after each planning phase for each domain. The data is collected with one run for each domain. The comparison is between the 0.5 second RA VAE-IW and 32 second RA VAE-IW.

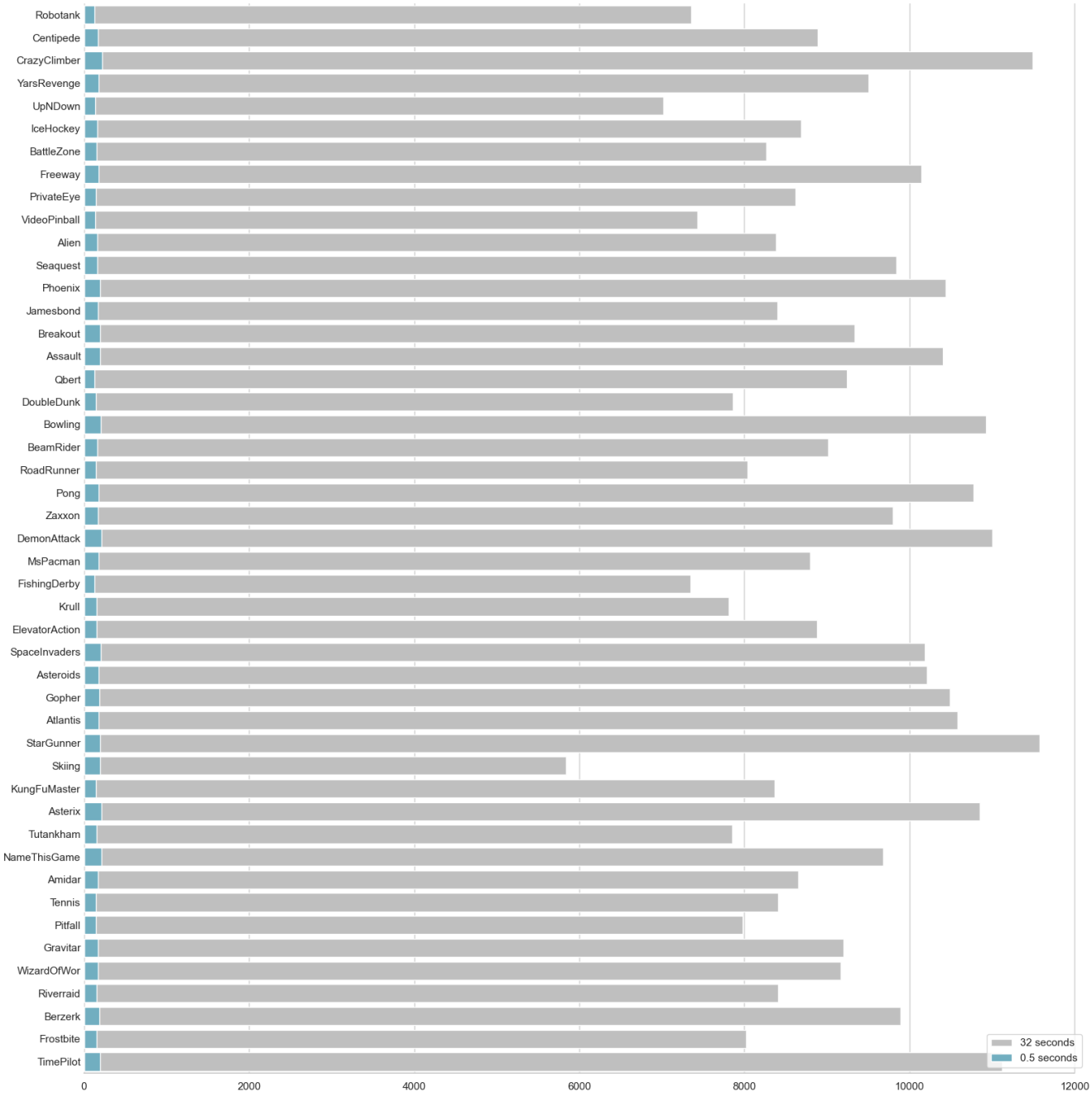

Figure 3: We calculate the mean of the max depth nodes in the planning tree after each planning phase for each domain. The data is collected with one run for each domain. The comparison is between the 0.5 second RA VAE-IW and 32 second RA VAE-IW.