Discrete Latent Space World Models for Reinforcement Learning

Jan Robine **Tobias Uelwer** **Stefan Harmeling**
Department of Computer Science
Heinrich Heine University Düsseldorf
{jan.robine,tobias.uelwer,harmeling}@hhu.de

Abstract

Sample efficiency remains a fundamental issue of reinforcement learning. Model-based algorithms try to make better use of data by simulating the environment with a model. We propose a new neural network architecture for world models based on a vector quantized-variational autoencoder (VQ-VAE) to encode observations and a convolutional LSTM to predict the next embedding indices. A model-free PPO agent is trained purely on simulated experience from the world model. We adopt the setup introduced by Kaiser et al. (2020), which only allows $100K$ interactions with the real environment, and show that we reach better performance than their SimPLe algorithm in five out of six randomly selected Atari environments, while our model is significantly smaller.

1 Introduction

Reinforcement learning is a generally applicable framework for finding actions in an environment which lead to the maximal sum of rewards. From a probabilistic perspective, the following probabilities lay the foundations of all reinforcement learning problems:

- The *dynamics* $p(r_t, s_{t+1} \mid s_t, a_t)$, i.e., the conditional joint probability of the next reward and state given the current state and action. This probability defines the environment and is usually unknown. In most real-world applications the underlying states s_t are not observable, but instead the environment produces observations x_t, which might not contain all state information. In this case we can only observe $p(r_t, x_{t+1} \mid x_t, a_t)$.
- The *policy* $p_\theta(a_t|x_t)$, i.e., the conditional probability of choosing an action given the current observation, where θ denotes the parameters of some model, indicating that this probability is not provided by the environment, but learned. Reinforcement learning methods provide means to optimize the policy in the sense that the actions which maximize the future sum of rewards have the highest probability.

Model-free algorithms try to optimize the policy $p_\theta(a_t|x_t)$ based on real experience from the environment, without any knowledge of the underlying dynamics. They have shown great success in a wide range of environments, but usually require a large amount of training data, i.e., many interactions with the environment. This low sample efficiency makes them inappropriate for real-world applications in which data collection is expensive.

Model-based algorithms approximate the dynamics $p_\phi(r_t, x_{t+1} \mid x_t, a_t) \approx p(r_t, x_{t+1} \mid x_t, a_t)$, where ϕ denotes some learned parameters, thus building a model of the environment, which we will call *world model* to differentiate it from other models. The process of improving the policy using the world model is called *planning*, but there are two types: first, we can generate training data by sampling from $p_\phi(r_t, x_{t+1} \mid x_t, a_t)$ and apply a model-free algorithm to this *simulated experience*.

Second, we can try to *look ahead* into the future using the world model, starting from the current observation, in order to dynamically improve the action selection during runtime.

In this work we follow a model-based approach and consider the first type of planning. The ability to generate new experience without acting in the real environment does effectively increase the sample efficiency. We try to find reasonable neural network architectures and training procedures.

2 Related Work

World Models (Ha and Schmidhuber, 2018). Modeling environments with complex visual observations is a hard task, but luckily predicting observations on pixel level is not necessary. The authors introduce latent variables that are produced by encoding the high-dimensional observations x_t into lower-dimensional, latent representations z_t. For this purpose they use a variational autoencoder (Kingma and Welling, 2014), which they call the "vision" component, that extracts information from the observation at the current time step. They use an LSTM combined with a mixture density network (Bishop, 1994) to predict the next latent variables stochastically. They call this component the "memory", that can accumulate information over multiple time steps. They also condition the policy on the latent variable, which enables them to stay in latent space, so that decoding back into pixels is not required (except for learning the representations). This makes simulation more efficient and can reduce the effect of accumulating errors. In Section 3.1 we describe in more detail how to integrate latent variables into the dynamics.

They successfully evaluate their architectures on two environments, but it involves some manual fine-tuning of the policy. They use an evolution strategy to optimize the policy, which is not suitable for bigger networks. They also use a non-iterative training procedure, i.e., they randomly collect real experience only once and then train the world model and the policy. This implies that the improved policies cannot be used to obtain new experience, and a random policy has to ensure sufficient exploration, which makes the approach inappropriate for more complex environments.

Simulated Policy Learning (Kaiser et al., 2020). The authors introduce a new model-based algorithm (SimPLe) and successfully apply it to Atari games. They use an iterative training procedure, that alternates between collecting real experience, training the world model, and improving the policy using the world model. Another novelty is that they restrict the algorithm to about $100K$ interactions with the real environment, which is considerably less than the usual $10M$ to $50M$ interactions. They train the policy using the model-free PPO algorithm (Schulman et al., 2017) instead of an evolution strategy. They use a video prediction model similar to SV2P (Babaeizadeh et al., 2017) and incorporate the input action in the decoding process to predict the next frame. The latent variable is discretized into a bit vector, that is predicted autoregressively using an LSTM during inference time. The policy gets as input the predicted or real frames, which means that it requires decoding into pixel-level observations. They get excellent results on a lot of Atari environments, considering the low number of interactions.

DreamerV1 (Hafner et al., 2020a) and DreamerV2 (Hafner et al., 2020b). In both works a world model with meaningful latent representations is trained, such that the agent can operate directly on the latent variables. One of the main improvements of DreamerV2 over the model of DreamerV1 is the discretization of the latent space, as it uses categorical intead of Gaussian latent variables. They show that their agent beats model-free algorithms in many Atari games after $50M$ interactions.

3 Discrete Latent Space World Models

The goal of this work is to extend the idea of Ha and Schmidhuber (2018) by using more sophisticated neural network architectures and evaluating them on Atari environments with the model-free PPO algorithm instead of evolution strategies. In particular, we discretize the latent space, but have a fundamentally different architecture compared to DreamerV2 (Hafner et al., 2020b), since our latent space is two-dimensional. Moreover, we adopt the limitation to $100K$ interactions, the iterative training scheme and some other crucial ideas from Kaiser et al. (2020), which will be explained in later sections.

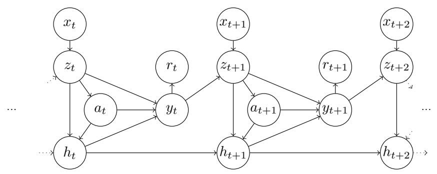

Figure 1: Graphical model of the world model and the policy, which arises from inserting the latent variables and from our independence assumptions.

3.1 Latent Variables

Similar to Ha and Schmidhuber (2018) we approximate the dynamics with the help of latent variables. The sum rule of probability allows us to introduce a latent variable z_t,

$$p(r_t, x_{t+1} \mid x_t, a_t) \approx \mathbb{E}_{z_t \sim p(z_t|x_t)} \left[\mathbb{E}_{z_{t+1} \sim p(z_{t+1}|z_t, a_t)} [p(r_t|z_t, a_t) \, p(x_{t+1}|z_{t+1})] \right], \tag{1}$$

where we have made multiple independence assumptions. Especially, we want an observation encoding model, $p_\phi(z_t|x_t)$, that does not depend on the action (analogous to the "vision" component of Ha and Schmidhuber (2018)). Second, we want to predict the next latent variable based on the previous latent variable and action, i.e., $p_\phi(z_{t+1}|z_t, a_t)$, independent of the observation x_t. Furthermore, the reward and next latent variable should not depend on each other, which allows to predict them using two heads and compute them in a single neural network pass.

In contrast to Kaiser et al. (2020), the policy is conditioned on the latent variables, so that no decoding into high-dimensional observations is necessary,

$$p(a_t|x_t) \approx \mathbb{E}_{z_t \sim p(z_t|x_t)} [p(a_t|z_t)]. \tag{2}$$

3.2 Recurrent Dynamics

Predicting the next latent variable and reward can be improved by introducing a recurrent variable h_t to the dynamics model, similar to Ha and Schmidhuber (2018),

$$\begin{aligned}
&p(x_{t+1}, r_t, h_t|x_t, a_t, h_{t-1}) \\
&\approx \mathbb{E}_{z_t \sim p(z_t|x_t)} \left[p(h_t|z_t, a_t, h_{t-1}) \, \mathbb{E}_{z_{t+1} \sim p(z_{t+1}|z_t, a_t, h_t)} [p(r_t|z_t, a_t, h_t) \, p(x_{t+1}|z_{t+1})] \right] &(3) \\
&= \mathbb{E}_{z_t \sim p(z_t|x_t)} \left[p(h_t|z_t, a_t, h_{t-1}) \, \mathbb{E}_{z_{t+1} \sim p(z_{t+1}|y_t)} [p(r_t|y_t) \, p(x_{t+1}|z_{t+1})] \right]. &(4)
\end{aligned}$$

We have made independence assumptions analogous to Eq. (1). In particular, the observation encoder and decoder do not depend on the recurrent variable, but the latent and reward dynamics do. For notation purposes, we introduced an intermediate representation $y_t = f(z_t, a_t, h_t)$ in Eq. (4), which is a deterministic function of z_t, a_t, and h_t. The resulting graphical model can be seen in Fig. 1.

We can also condition the policy on the recurrent variable, which adds a dependency from a_t to h_{t-1},

$$p(a_t|x_t, h_{t-1}) \approx \mathbb{E}_{z_t \sim p(z_t|x_t)} [p(a_t|z_t, h_{t-1})]. \tag{5}$$

From Eq. (4), Eq. (2), and Eq. (5) it becomes clear that several models have to be learned. We denote the parameters of the world model by ϕ, and the parameters of the policy by θ. Table 1 provides an overview of all models that need to be learned.

Table 1: Summary of the learned models. Note that y_t is a deterministic function of z_t, a_t, and h_t.

World model	Observation encoder	$p_\phi(z_t\|x_t)$
	Recurrent dynamics	$p_\phi(h_t\|z_t, a_t, h_{t-1})$
	Reward dynamics	$p_\phi(r_t\|y_t)$
	Latent dynamics	$p_\phi(z_{t+1}\|y_t)$
	Observation decoder	$p_\phi(\hat{x}_{t+1}\|z_{t+1})$
Agent	Policy	$p_\theta(a_t\|z_t)$ or $p_\theta(a_t\|z_t, h_{t-1})$

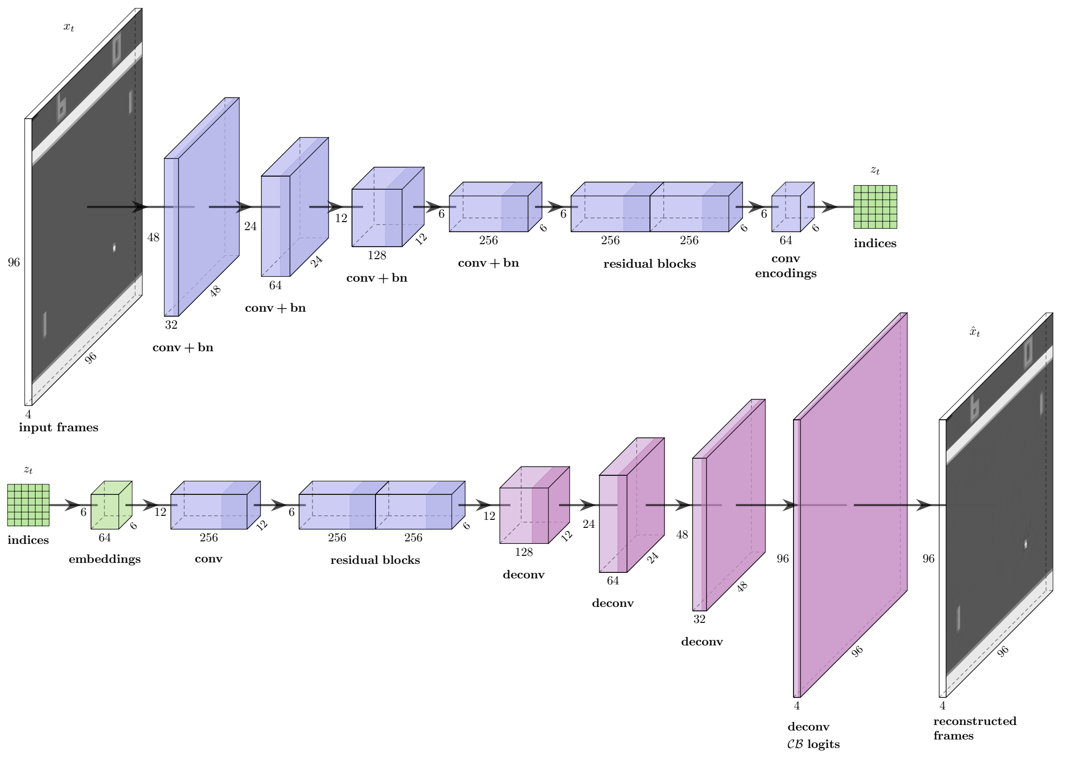

Figure 2: Visualization of the observation encoder architecture (top) and decoder architecture (bottom). The encoder uses batch normalization but the decoder does not. All convolutions and deconvolutions are followed by ReLU nonlinearities (after the batch norm). The last deconvolution outputs the logits of $96 \times 96 \times 4$ continuous Bernoulli distributions (Loaiza-Ganem and Cunningham, 2019) for all stacked frames.

3.3 Architecture

Latent Representations. We use a vector quantized-variational autoencoder (van den Oord et al., 2017) for the observation encoder and decoder, so each latent variable z_t is a matrix filled with discrete embedding indices. We suppose that a two-dimensional representation which can keep local spatial correlations is more suitable for predicting the representation at the next time step, especially when combined with convolutional operations.

The observations x_t are the last four frames of the Atari game, stacked and converted from RGB to grayscale. We model the outputs of the decoder with continuous Bernoulli distributions (Loaiza-Ganem and Cunningham, 2019) with independence among the stacked frames and pixels. Frame stacking allows the observation encoder to incorporate short-time information, e.g., the velocity of objects, into the otherwise stationary latent representations z_t. See Fig. 2 for a detailed description of the model.

Dynamics. For the recurrent dynamics we use a two-cell convolutional LSTM (Shi et al., 2015) with layer normalization, conditioned on the action. The intermediate representation y_t from Fig. 1 is realized by concatenating the output of the convolutional LSTMs and the actions, which means that we actually drop its direct dependence on z_t. Then, there are two prediction heads, one for the next latent variable, $f_\phi(y_t)$, and one for the reward, $g_\phi(y_t)$, as seen in Fig. 3. We also tried follow-up architectures like the spatio-temporal LSTM (Wang et al., 2017) or causal LSTM (Wang et al., 2018), but for our problem the performance gain was not significant enough, compared with the imposed additional training time and number of parameters.

The output of $f_\phi(y_t)$ is a $H \times W \times K$ tensor ($6 \times 6 \times 128$) containing the unnormalized scores for the embedding indices, which get normalized via the softmax function,

$$p_\phi\Big(z_{t+1}^{(j,k)} \,\Big|\, y_t\Big) = \mathrm{Cat}\Big(K,\, \mathrm{Softmax}\Big(f_\phi^{(j,k)}(y_t)\Big)\Big). \tag{6}$$

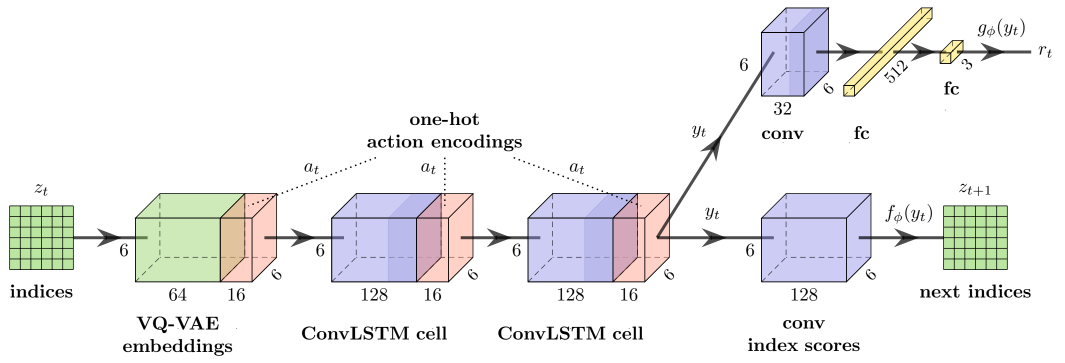

Figure 3: A visualization of the architecture of the dynamics network. The embeddings are obtained by looking up the embedding vectors for the given indices. In order to condition on the action, it is one-hot encoded, repeated along the spatial dimensions, and concatenated with the input. After the second convolutional LSTM cell the network splits into the reward prediction head $g_\phi(y_t)$ at the top, consisting of one convolutional and two fully connected layers, and the next latent prediction head $f_\phi(y_t)$ at the bottom, which predicts the next indices with one convolutional layer. The convolutions are followed by layer normalization and leaky ReLU nonlinearities. We repeat the action input at the second cell and at both prediction heads, since the action information might get lost during the forward pass. The recurrent states h_t, h_{t-1} of the LSTM are not visualized for clarity.

We suppose that the discretization of the latent space stabilizes the dynamics model, since it has to predict scores for a predefined set of categories instead of real values, especially considering that the target is moving, i.e., the latent representations change during training.

The rewards are discretized into three categories $\{-1, 0, 1\}$ by clipping them into the interval $[-1, 1]$ and rounding them to the nearest integer. The output of $g_\phi(y_t)$ is a 3-dimensional vector containing the scores for each reward category, which are also normalized via the softmax function,

$$p_\phi(r_t|y_t) = \mathrm{Cat}(3, \mathrm{Softmax}(g_\phi(y_t))). \tag{7}$$

The support of this distribution is $r \in \{1, 2, 3\}$, so we have to map the rewards accordingly ($r = r_{\text{orig}} + 2$) when we compute the likelihood.

Policy. The input of the policy network are the embedding vectors and it processes them using two convolutional layers with layer normalization, followed by a fully connected layer. Unlike Eq. (5) the policy does not depend on the recurrent variable h_{t-1} from the world model in our experiments, but this could be useful for more complex environments. The output of the network $f_\theta(z_t)$ is again a vector of unnormalized scores of a categorical distribution for the M possible actions,

$$p_\theta(a_t|z_t) = \mathrm{Cat}(M, \mathrm{Softmax}(f_\theta(z_t))). \tag{8}$$

3.4 Training

The distributions in Eq. (6) and Eq. (7) are trained using maximum likelihood. The policy is optimized on simulated experience using proximal policy optimization (Schulman et al., 2017), a model-free reinforcement learning algorithm. We approximate the expectations in Eq. (1) and Eq. (2) with single Monte Carlo samples. While we simulate experience, we use the same sample $z_t \sim p(z_t|x_t)$ for both the dynamics, Eq. (4), and the policy, Eq. (2). At inference time, when the policy is applied to a real environment, Eq. (2) still needs access to the observation encoder $p_\phi(z_t|x_t)$ in order to sample the latent variables.

Episodic Environments. Atari games are episodic, i.e., the environment resets after a game is over. Therefore, the world model needs to predict terminal states, for instance by predicting another binary variable that indicates the end of the episode. This prediction has to be reliable, otherwise a false prediction of “true” can have a severe impact on the simulated experience and the policy’s learning performance. As a first simple solution, we follow Kaiser et al. (2020) and end all episodes after a fixed number of steps (e.g., 50), so the world model does not have to decide on termination.

Table 2: Number of parameters of our world model compared with the model of Kaiser et al. (2020) (their number is approximate). Their model requires about seven times as many parameters.

Model	# parameters
Ours	10,655,057
SimPLe	74,000,000

Table 3: Number of parameters of our models in detail. At training time all models are used, but at test time only the encoder and the policy network are required. The encoder and decoder both need access to the embedding vectors, therefore their individual number of parameters is slightly higher than the total VQ-VAE.

Model	# parameters
World model	10,655,057
VQ-VAE	4,102,510
Encoder	2,027,658
Decoder	2,087,269
Dynamics network	6,552,547
Policy network	1,303,571
World model + policy (training)	11,958,628
Encoder + policy (inference)	3,331,229

We also adopt the idea of randomly selecting the initial observation of a simulated episode from the collected real data. This enables the policy to learn from experience from any stage of the environment, although the number of time steps is limited. On the downside, this prevents the policy to learn from effects that are longer than the fixed number of steps (Kaiser et al., 2020).

Iterative Training. We adopt the iterative training procedure from Kaiser et al. (2020) and alternate between interacting with the real environment, training the world model, and training the policy. We also adopt the number of interactions per iteration, 6400, and the number of iterations, 15. The authors state that they perform another 6400 interactions before the first iteration. We perform $12{,}800$ in the first iteration, resulting in the same number of total interactions, $12{,}800 + 6400 \times 14 = 102{,}400$. In the first iteration we use a random uniform policy instead of the randomly initialized network.

Warming Up Latent Representations. After collecting the first batch of data, we train the VQ-VAE separately for 50 epochs with a higher learning rate. We want to give the dynamics model a better starting point, with representations that already contain useful information. Note that this cannot be done in later training stages, as then the representations would move too far away and the dynamics model would not be able to keep up. After the initial phase, we even slow down the training, as stated in the next paragraph.

Fixed Representation Learning. We fix the parameters of the VQ-VAE for two training steps and update them in every third step, so the targets of the dynamics model are moving more slowly.

Reward Loss. If the gradients coming from the reward prediction head have a high magnitude, they have a degrading effect on the performance of the next latent prediction head. We solve this issue by scaling down the cross-entropy loss of the rewards to reduce its influence on the entire model, but use a higher learning rate for the reward prediction head, to compensate for the smaller gradients.

Constant KL. Kaiser et al. (2020) state that the weight of the KL divergence loss of a variational autoencoder is game dependent, which makes VAEs impractical to apply on all Atari games without fine-tuning. The VQ-VAE does not suffer from this problem, since the KL term is constant and only depends on the number of embedding vectors.

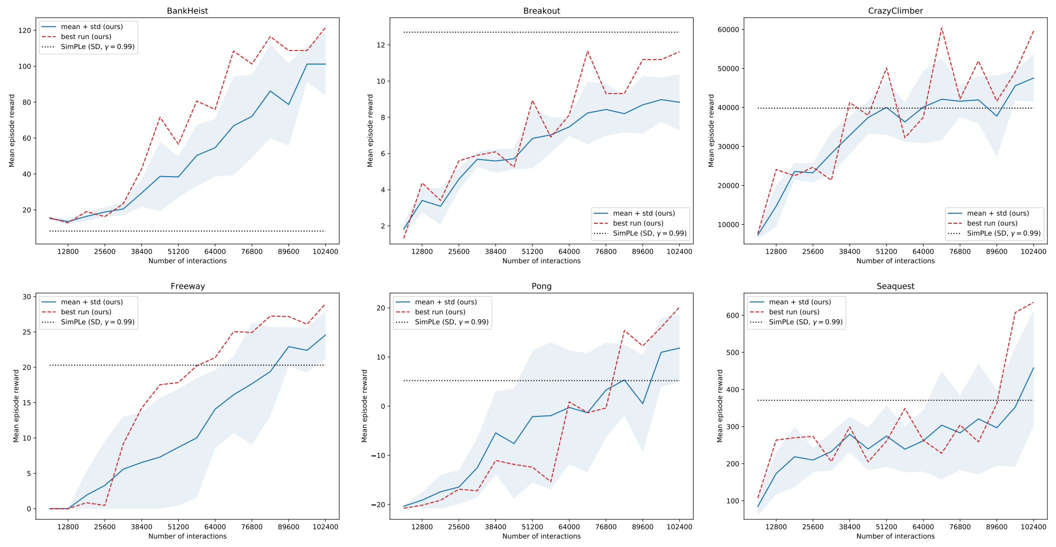

Figure 4: Total episode reward, averaged over five runs $\pm$ standard deviation. The x-axis shows the number of interactions with the real environment. The red plots show the best of the five runs and the horizontal bar the mean final performance of Kaiser et al. (2020).

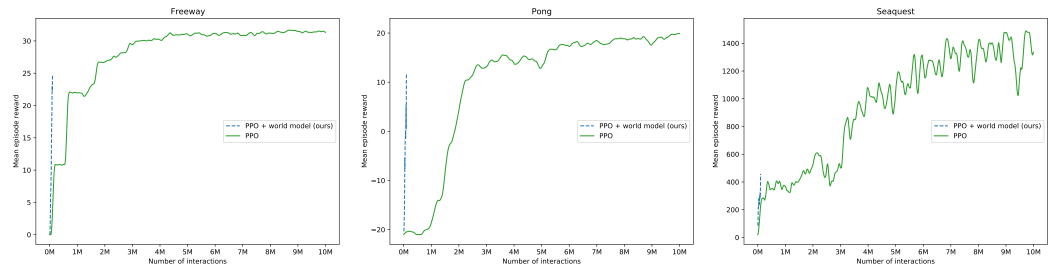

Figure 5: Comparison of our model-based approach with model-free PPO (Schulman et al., 2017), where we plot the mean of our five runs. The sample efficiency in the first $100K$ interactions is significantly higher in the model-based setting.

4 Evaluation

We compare our method with the default variant of Kaiser et al. (2020) (stochastic discrete, 50 steps, $\gamma = 0.99$, normal model training), which comes closest to our model in terms of hyperparameters (discount rate, batch size etc.) and number of parameter updates. This implies that the results of Kaiser et al. (2020) that we show are not necessarily the best results across all of their variants, but the best for a fairer comparison.

Due to limited resources, finding the right architectures and hyperparameters and evaluating on many environments over multiple runs takes a long time (currently a single run takes about 18 hours). At the moment, we have the results for six environments and cannot thoroughly compare our method to Kaiser et al. (2020), but we get similar results compared with the default variant of their model, although our model has much fewer parameters, as can be seen in Table 2. We do not know the performance difference to Kaiser et al. (2020) in terms of training time and test time, but considering the model size our method should be a lot faster.

Exploration. In our experiments we observe the same phenomenon as Kaiser et al. (2020), namely that the results can vary drastically for different runs with the same hyperparameters but different random seeds. The main reason might be that the world model cannot infer dynamics of regions of the environment's state space that it has never seen before, and the algorithm is very sensitive to the exploration-exploitation trade-off, as the number of interactions is very low.

Table 4: Comparison of our method with SimPLe (Kaiser et al., 2020) and model-free PPO (Schulman et al., 2017) trained with $100K$ steps. Mean and standard deviation of the final total reward over five training runs. The numbers for PPO are taken from Kaiser et al. (2020).

	Ours		SimPLe (SD, $\gamma = 0.99$)		PPO $100K$	
Game	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
BankHeist	$\mathbf{101.2}$	(± 17.4)	8.2	(± 4.4)	16.0	(± 12.4)
Breakout	8.8	(± 1.5)	$\mathbf{12.7}$	(± 3.8)	5.9	(± 3.3)
CrazyClimber	$\mathbf{47536.9}$	(± 6114.9)	39827.8	(± 22582.6)	18400.0	(± 5275.1)
Freeway	$\mathbf{24.6}$	(± 3.4)	20.3	(± 18.5)	8.0	(± 9.8)
Pong	$\mathbf{11.8}$	(± 6.9)	5.2	(± 9.7)	-20.5	(± 0.6)
Seaquest	$\mathbf{458.1}$	(± 155.7)	370.9	(± 128.2)	370.0	(± 103.3)

Table 5: Comparison of our method with SimPLe (Kaiser et al., 2020). Median and best of the final total reward over five training runs.

	Median		Best	
Game	Ours	SimPLe	Ours	SimPLe
BankHeist	$\mathbf{101.2}$	8.9	$\mathbf{121.6}$	13.9
Breakout	8.4	$\mathbf{11.0}$	11.6	$\mathbf{19.5}$
CrazyClimber	$\mathbf{45256.2}$	41396.9	59609.4	$\mathbf{67250.0}$
Freeway	26.2	$\mathbf{33.5}$	29.0	$\mathbf{34.0}$
Pong	$\mathbf{8.2}$	1.4	20.2	$\mathbf{21.0}$
Seaquest	$\mathbf{472.5}$	386.9	$\mathbf{635.0}$	497.2

5 Conclusion and Future Work

We believe that in theory and practice our method has advantages over Kaiser et al. (2020). Our experiments show that acting entirely in latent space is possible, which speeds up training since no decoding into high-dimensional frames is required. We chose a discrete and two-dimensional latent space, so that the representations are more stable and expressive at the same time. Moreover, the VQ-VAE is a powerful model that can be extended to really complex visual observations. We reach better performance in five out of six environments compared with Kaiser et al. (2020), with a seven times smaller model. In this work we presented preliminary results that are already promising, and it seems that further tuning of hyperparameters could lead to even better results, which we could not realize yet due to resource constraints.

The next improvement could be to predict the latent variables autoregressively, e.g., with a PixelCNN as in the original VQ-VAE architecture. This would make it possible to model dependencies between the components of the latent variables and could potentially solve prediction errors like duplication or incoherent movement of objects.

As Kaiser et al. (2020) have already suggested, another important step is to improve exploration in order to enable even higher sample efficiency. Furthermore, we would like the world model to predict terminal states and move away from terminating episodes after a fixed number of steps, since this introduces a new hyperparameter that needs to be tuned, and prevents the agent from learning beyond this time horizon.

References

Babaeizadeh, M., Finn, C., Erhan, D., Campbell, R. H., and Levine, S. (2017). Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*.

Bishop, C. M. (1994). Mixture density networks. Technical report, Aston University, Birmingham. Available at `http://publications.aston.ac.uk/373/`.

Ha, D. and Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 2450–2462. Curran Associates, Inc.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2020a). Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020b). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

Kaiser, L., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. (2020). Model based reinforcement learning for atari. In *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Loaiza-Ganem, G. and Cunningham, J. P. (2019). The continuous bernoulli: fixing a pervasive error in variational autoencoders. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 13287–13297. Curran Associates, Inc.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., and WOO, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 802–810. Curran Associates, Inc.

van den Oord, A., Vinyals, O., and kavukcuoglu, k. (2017). Neural discrete representation learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 6306–6315. Curran Associates, Inc.

Wang, Y., Gao, Z., Long, M., Wang, J., and Yu, P. S. (2018). PredRNN++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. volume 80 of *Proceedings of Machine Learning Research*, pages 5123–5132, Stockholmsmässan, Stockholm Sweden. PMLR.

Wang, Y., Long, M., Wang, J., Gao, Z., and Yu, P. S. (2017). Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 879–888. Curran Associates, Inc.