Neural Processing Letters (2023) 55:11397–11427
https://doi.org/10.1007/s11063-023-11381-3

Smaller World Models for Reinforcement Learning

Jan Robine[1] · **Tobias Uelwer**[1] · **Stefan Harmeling**[1]

Accepted: 21 July 2023 / Published online: 10 August 2023

Abstract
Model-based reinforcement learning algorithms try to learn an agent by training a model that simulates the environment. However, the size of such models tends to be quite large which could be a burden as well. In this paper, we address the question, how we could design a model with *fewer* parameters than previous model-based approaches while achieving the same performance in the 100 K-interactions regime. For this purpose, we create a world model that combines a vector quantized-variational autoencoder to encode observations and a convolutional long short-term memory to model the dynamics. This is connected to a model-free proximal policy optimization agent to train purely on simulated experience from this world model. Detailed experiments on the Atari environments show that it is possible to reach comparable performance to the SimPLe method with a significantly smaller world model. A series of ablation studies justify our design choices and give additional insights.

Keywords Model-based reinforcement learning · World models · Discrete latent space · VQ-VAE · Atari

1 Introduction

Reinforcement learning environments can be formalized with Markov decision processes (MDPs). An MDP is defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, the starting distribution $p(s_1)$, and the dynamics distribution $p(s_{t+1}, r_t \mid s_t, a_t)$ for any time step t. The dynamics distribution describes the probability of the next state $s_{t+1} \in \mathcal{S}$ and reward $r_t \in \mathbb{R}$, given the current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$. This one-step dynamics distribution fully describes the environment's dynamics, since the environment is assumed to satisfy the Markov property, i.e., states retain all relevant information about the past. For practicality, we add another variable $d_t \in \{0, 1\}$ that indicates whether the state is non-terminal or terminal, respectively.

✉ Jan Robine
jan.robine@tu-dortmund.de

Tobias Uelwer
tobias.uelwer@tu-dortmund.de

Stefan Harmeling
stefan.harmeling@tu-dortmund.de

[1] Department of Computer Science, Technical University of Dortmund, 44227 Dortmund, Germany

The dynamics distribution then becomes $p(s_{t+1}, r_t, d_t \mid s_t, a_t)$. This still fits into the MDP definition, as d_t can be seen as part of the state.

An reinforcement learning agent interacts with an environment and tries to maximize the sum of rewards by taking the right actions. The behavior of an agent is defined by a policy distribution $\pi_\theta(a_t \mid s_t)$ that maps states onto actions, where θ denotes the parameters of some learned model. Reinforcement learning methods provide means to optimize the policy distribution in the sense that the actions that maximize the sum of rewards have high probability.

Model-free reinforcement learning algorithms try to optimize the policy $\pi_\theta(a_t \mid s_t)$ 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, yet they often require a large amount of training data, i.e., many interactions with the environment. This low sample efficiency still renders them inappropriate for real-world applications with expensive data collection. Many efforts have been made to alleviate this problem, e.g., experience replay, where data is stored and reused multiple times instead of being thrown away immediately.

Model-based algorithms approximate the true dynamics with a model of the environment $p_\phi(s_{t+1}, r_t, d_t \mid s_t, a_t) \approx p(s_{t+1}, r_t, d_t \mid s_t, a_t)$ with parameters ϕ. To differentiate it from other models, we will call $p_\phi(s_{t+1}, r_t, d_t \mid s_t, a_t)$ a world model. The process of improving a policy with a world model is called planning and can be carried out in two ways [23]: (i) Decision-time planning dynamically improves the action selection process during runtime by looking ahead into the future using the world model. This can be illustrated with the game of chess, where it is beneficial to think about the consequences of possible moves. This slows down the agent at decision-time, but it may be favorable for environments in which predictions about the future are crucial. (ii) Background planning generates novel training data by sampling from the world model. The agent can be trained with this simulated experience, possibly in combination with real experience. This increases training time, but is suited for environments in which fast decisions are required and collecting experience is expensive, e.g., self-driving cars or robots, because after training, the world model is not needed anymore.

Contributions

In this work we consider a model-based approach with background planning as described above. The ability to generate new experience without acting in the real environment increases the sample efficiency, because less real data is necessary. An overview of our approach can be seen in Fig. 1. Our main contributions and insights can be summarized as follows:

- We implement a world model based on a VQ-VAE which requires fewer parameters than existing approaches (see Table 4).
- Our world model uses a two-dimensional discrete latent representation combined with a dynamics network built from convolutional LSTMs (see Fig. 5) and we demonstrate that this choice is favorable for model-based reinforcement learning (see Table 6).
- We extensively evaluate our world model in the Arcade Learning Environment and restrict the training to 100K interactions instead of the usually used 10 M or 50 M interactions. We show that learning a latent space world model and training an agent is possible in this restricted regime (see Table 2).

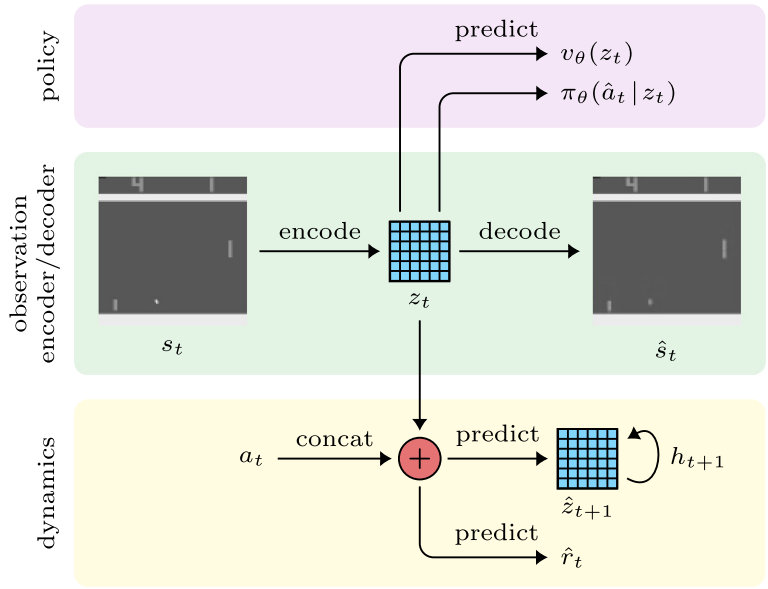


Fig. 1 Overview of our approach. A VQ-VAE encodes game frames into discrete latent representations, which are fed into the policy to determine the next actions. The dynamics model is conditioned on the latent representation and the action, and predicts the reward and the next latent representation using a convolutional LSTM with hidden states. The architecture of the individual components is described in detail in Sect. 3.2

2 Preliminaries

Many real world environments only expose observations o_t that might not contain all information of the underlying states s_t. The Markov property might not hold for these observations. For instance, the observation o_t could be a camera image while the state s_t could describe the actual position and velocity of objects. This partial observability can be seen as a special case of function approximation (e.g., the parameters can be chosen in such a way that the approximation does not depend on the unobservable parts of the state), and all algorithms that consider function approximation can be applied to partial observability [23]. Therefore, we will continue to denote the observations as s_t for simplicity.

2.1 Actor-Critic Algorithms

A central concept of reinforcement learning is the *value*, which is the expected discounted sum of rewards when starting at some position and following a specific policy. The state-value function $v^{\pi_\theta}(s_t)$ computes the value when starting at state s_t and following policy π_θ. The action-value function $q^{\pi_\theta}(s_t, a_t)$ computes the value when starting at state s_t and action a_t. Formally, they are defined as

$$v^{\pi_\theta}(s_t) := \mathbb{E}_{\pi_\theta}\left[\sum_{l=0}^{\infty} \gamma^l R_{t+l} \mid S_t = s_t\right] \tag{1}$$

$$q^{\pi_\theta}(s_t, a_t) := \mathbb{E}_{\pi_\theta}\left[\sum_{l=0}^{\infty} \gamma^l R_{t+l} \mid S_t = s_t, A_t = a_t\right] \tag{2}$$

where R_t, S_t, and A_t are random variables for the reward, state, and action at time step t, respectively, and $\gamma \in [0, 1)$ is the discount factor that determines the weighting of distant rewards. The exact definitions of the random variables and the expectations can be derived from the definition of the MDP and the dynamics distribution. Furthermore, the advantage of a particular action over the average action in terms of value can be computed with the advantage function $a^{\pi_\theta}(s_t, a_t) := q^{\pi_\theta}(s_t, a_t) - v^{\pi_\theta}(s_t)$ for a given policy π_θ.

Model-free algorithms can be divided into two classes: value-based methods, which estimate a value function and implicitly define the policy such that the value is maximized, and policy gradient methods, which directly learn a policy distribution. In this work we will focus on policy gradient methods. The policy gradient objective $\mathcal{L}^{\mathrm{PG}}(\theta)$ provides a learning signal to optimize the parameters θ of the policy distribution. There are several ways to formulate this objective, but we will use the following definition:

$$\mathcal{L}^{\mathrm{PG}}(\theta) := \mathbb{E}_{\pi_\theta}\left[\log \pi_\theta(A_t \mid S_t) a_\psi(S_t, A_t)\right], \tag{3}$$

where $a_\psi(s_t, a_t) \approx a^{\pi_\theta}(s_t, a_t)$ is an estimator of the true advantage function with parameters ψ [23]. If this estimator involves an estimation of a value function, it is called an actor-critic method. A common estimator of the advantage function is the expected multi-step temporal difference error

$$a_\psi^{\mathrm{TD}}(s_t, a_t) := \mathbb{E}_{\pi_\theta}\left[\sum_{l=0}^{T-1} \gamma^l R_{t+l} + \gamma^T v_\psi(S_{t+T}) - v_\psi(S_t) \mid S_t = s_t, A_t = a_t\right], \tag{4}$$

where $v_\psi(s_t) \approx v^{\pi_\theta}(s_t)$ is an estimator of the state-value function with parameters ψ. The horizon T determines the number of time steps before "bootstrapping" the remaining value with the learned state-value function and it controls the trade-off between bias and variance.

Typically, data is sampled from N environment instances for T time steps to obtain a minibatch of shape $N \times T$, in order to approximate the expectations in Eq. 3 and Eq. 4 and update the parameters θ. At the same time, the learned state-value function in Eq. 4 can be optimized by minimizing the following mean squared error

$$\mathcal{L}^{\mathrm{VF}}(\psi) := \mathbb{E}_{\pi_\theta}\left[\left(\underbrace{\sum_{l=0}^{T-1} \gamma^l R_{t+l} + \gamma^T \mathrm{sg}(v_\psi(S_{t+T}))}_{\text{target}} - \underbrace{v_\psi(S_t)}_{\text{prediction}}\right)^2\right], \tag{5}$$

where $\mathrm{sg}(\cdot)$ is the stop-gradient operator that prevents gradients from flowing through the target values when applying stochastic gradient descent.

To improve the exploration behaviour of the policy, [16] suggest an entropy regularization term that encourages the policy to stay slightly uncertain, $\mathcal{L}^{\mathrm{ENT}}(\theta) := \mathbb{E}_{\pi_\theta}\left[\mathrm{H}_{\pi_\theta}[A_t \mid S_t]\right]$ where $\mathrm{H}_{\pi_\theta}[A_t \mid S_t]$ denotes the conditional entropy of A_t given S_t for the distribution π_θ. We can define a combined maximization objective for both the policy and the value function

$$\mathcal{L}^{\mathrm{PG+VF+ENT}}(\theta, \psi) := \mathbb{E}_{\pi_\theta}\left[\mathcal{L}^{\mathrm{PG}}(\theta) + c_1 \mathcal{L}^{\mathrm{ENT}}(\theta) - c_2 \mathcal{L}^{\mathrm{VF}}(\psi)\right], \tag{6}$$

where $c_1, c_2 \geq 0$ are coefficients that control the contribution of the entropy regularization and the state-value function, respectively.

The "vanilla" policy gradient discussed so far can be further improved, for instance, via trust region methods, which bound large policy updates in terms of changes to the policy distribution. This can be achieved by imposing a constraint on the divergence between the old policy and the new policy after a parameter update and either requires expensive second-order optimization or KL divergence approximations. Proximal policy optimization (PPO,

[19]) instead proposes a heuristically determined first-order objective aimed at bounding large policy updates

$$\mathcal{L}^{\text{CLIP}}(\theta) := \mathbb{E}_{\pi_{\theta_{\text{old}}}}\Big[\min\Big\{r_\theta(A_t, S_t)a_\psi(S_t, A_t), \\ \text{clip}\left(r_\theta(A_t, S_t), 1-\epsilon, 1+\epsilon\right) a_\psi(S_t, A_t)\Big\}\Big], \tag{7}$$

where $\pi_{\theta_{\text{old}}}$ is the policy used to collect the data, $r_\theta(a_t, s_t) := \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$ is the likelihood ratio, $a_\psi(s_t, a_t) \approx a^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$ is an estimator of the advantage function of the old policy, and $\text{clip}(x, a, b) := \max\{a, \min\{b, x\}\}$. In order to reduce variance, the likelihood ratio is clipped between $1-\epsilon$ and $1+\epsilon$ for a small ϵ, thus keeping it close to 1. The bound is pessimistic, as the objective is only clipped if it would improve too heavily, while decreasing the objective is not prevented (because of the minimum). Clipping takes effect in two cases: (i) the advantage estimate $a_\psi(s_t, a_t)$ is positive and the policy gradient wants to increase the probabilities of the actions excessively, and (ii) the advantage estimate is negative and the policy wants to decrease the probabilities of the actions too heavily. The likelihood ratios allow for multiple parameter updates on a single minibatch, so more information can be squeezed out of the data before throwing it away. The PPO objective can be combined with the state-value function objective and the entropy regularization term, resulting in the final objective we will use in our work

$$\mathcal{L}^{\text{CLIP+VF+ENT}}(\theta, \psi) := \mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[\mathcal{L}^{\text{CLIP}}(\theta) + c_1\mathcal{L}^{\text{ENT}}(\theta) - c_2\mathcal{L}^{\text{VF}}(\psi)\right] \tag{8}$$

where the coefficients $c_1, c_2 \geq 0$ control the contribution of the objectives.

2.2 Arcade Learning Environment

The Arcade Learning Environment (ALE) by [3] is a commonly used framework to evaluate reinforcement learning algorithms. It provides access to many games of the Atari 2600 console.

The actions are discrete and correspond to the inputs that can be made using the Atari 2600 joystick, with a total number of 18. The reward is defined as the change in the score of a game. The observations are raw video frames, which are usually preprocessed before being passed to the agent. Typically, the RGB frames of shape $210 \times 160 \times 3$ are converted to grayscale and scaled down to a square image, in our case to 96×96. We follow the common preprocessing steps from [15]: (i) Convert frames to grayscale and scale them down. (ii) Frame skipping, i.e., only consider every nth frame and repeat the same action in-between, which reduces the effective frame rate and enables the agent to reach further advanced observations in less time steps. (iii) Frame stacking, i.e., concatenate the last k frames (after frame skipping), which allows the agent to observe short-time effects, e.g., the velocity of objects. (iv) Max pooling, i.e., take the component-wise maximum of the two most recent frames, since hardware limitations of the Atari 2600 can cause objects to be displayed only every other frame.

In the following, an *interaction* with an environment denotes a single action that is sent to the game. We will use the common frame stacking value of 4 (i.e., we only consider every fourth frame). Therefore, an interaction corresponds to four frames. There are common values for the number of frames that are collected during training of the reinforcement learning agent, among which are 10, 40, 100, and 200 M (or in terms of interactions 2.5, 10, 25, and 50 M, respectively). In terms of real time these are roughly 46, 185, 463, and 962 hours, respectively, which are orders of magnitude longer than the time a human would usually need

to be comparatively good at an Atari 2600 game. In this work we limit the agent to about $100K$ interactions, i.e., 400 K frames, as proposed by [11], which (approximately) equals 1 hour and 51 minutes of real time gameplay.

2.3 Variational Autoencoders

A variational autoencoder (VAE, [13]) is a powerful generative model that can produce new samples of a complex distribution $p(x)$ by decoding samples of a simpler distribution $p(z)$, e.g., samples from $p(x)$ could be images and samples from $p(z)$ low-dimensional real vectors. Conversely, a VAE can also map high-dimensional samples x onto low-dimensional representations z.

VAEs can be derived from latent variable models, which are generative models that sample a latent variable z and decode it into the actual data x via a learned decoder $p_\Phi(x \mid z)$ with parameters Φ, instead of directly sampling x. This has the advantage that high-level or global information can be decided upon before generating the high-dimensional output, e.g., the digit, its position, its shape, etc., before generating the pixels. However, these models suffer from the problem that maximizing the likelihood $p_\Phi(x) = \int p_\Phi(x \mid z) p_0(z) dz$, requires a lot of samples from the prior $p_0(z)$ in order to approximate the integral. Even if z is low-dimensional, the number of possible samples usually is really high, $p_\Phi(x \mid z)$ will be close to zero for most z, and capturing the entire latent space via random sampling is not tractable. VAEs approach this problem by additionally learning an encoder $p_\Phi(z \mid x)$ that can produce samples of z conditioned on training data x, so that only samples of z that are relevant for maximizing the likelihood are considered. Using variational inference, a lower bound on the log-likelihood can be derived [13]

$$\log p_\Phi(x) \geq \mathbb{E}_{p_\Phi(z \mid x)} \left[\log p_\Phi(x \mid Z) \mid X = x \right] - D_{\mathrm{KL}} \left(p_\Phi(z \mid x) \parallel p_0(z) \right). \tag{9}$$

The expected value in Eq. 9 can be interpreted as a "reconstruction" term and is often approximated using a single sample from $p_\Phi(z \mid x)$. The KL divergence acts as a regularizer on the encoder since it keeps its outputs closer to the prior distribution.

VAEs with a normally distributed latent space often have the issue that the outputs of the decoder are "blurry" (e.g., blurry images). One reason is that the training samples still only sparsely fill the latent space and sampling from a standard normal prior can generate latent variables with low probability in $p_\Phi(x \mid z)$. Vector quantized-variational autoencoders (VQ-VAEs, [25]) provide two improvements: (i) The latent space is discrete, which makes it more compact, and (ii) Samples from the prior are generated more carefully with an additionally trained autoregressive neural network.

A discrete latent space for image inputs is achieved as follows. An encoder convolutional neural network $f_\Phi(x)$ outputs tensors of shape $H \times W \times C$, with spatial dimensions H, W, and number of hidden channels C. This output is interpreted as a two-dimensional structure of C-dimensional vectors that get discretized via vector quantization (VQ). That is, a list of K embedding vectors is maintained and the output vectors are replaced by those embedding vectors that have the smallest Euclidean distance. The latent representation z then is a matrix of integers, where $z_{i,j}$ is the index of the closest embedding vector for the output vector $f_\Phi(x)_{i,j}$ at row i and column j. The embedding vectors are updated by either moving them slightly towards the output vectors in every parameter update or by keeping an exponential moving average of the output vectors. During backpropagation, the gradients are estimated by passing them straight through the vector quantization step, as if the quantization did not happen.

The probabilistic interpretation of the vector quantization is that the encoder $p_\Phi(z \mid x)$ outputs a two-dimensional grid of shape $H \times W$ of independent categorical distributions, where the number of categories corresponds to the number of embedding vectors. For each entry $z_{i,j}$ of the latent representation, the index of the embedding vector closest to the output vector has a probability of one; all other categories have zero probability. The prior $p_0(z)$ is a two-dimensional grid of shape $H \times W$ of uniform categorical distributions, i.e., each $p_0(z_{i,j}) = \text{Cat}\left(z_{i,j}; K, \left[\frac{1}{K} \cdots \frac{1}{K}\right]\right)$is a categorical distribution with equal probabilities. This choice of distributions leads to a constant KL divergence term in the lower bound from Eq. 9, which is another beneficial property of VQ-VAEs and makes them more robust across different training sets, since the magnitude of the KL divergence stays the same, independent of the training data x [25]. For the standard (Gaussian) VAEs, this is usually addressed by inserting a coefficient for the KL term in the lower bound in Eq. 9 [9], which needs to be manually fine-tuned for each dataset.

To generate latent variables autoregressively, a PixelCNN [24] is trained to predict the indices of the embedding vectors, after finishing the training of the encoder, decoder, and embedding vectors. The indices can be looked up in the list of embedding vectors, which then are decoded. This allows VQ-VAEs to generate stable, non-blurry samples.

3 Method

This section provides a summary of the methods that we use to build our world model, starting with a high-level view and gradually becoming more concrete. First, in Sect. 3.1 we introduce the concepts and our notation for latent space world models and policies. Second, in Sect. 3.2 we show the neural network architectures that we employ. Lastly, in Sect. 3.3 we explain our training procedure and how the models work together.

3.1 Latent Space World Models

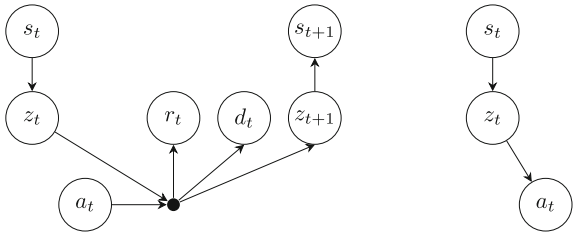
Simulating experience in latent space instead of the raw state or observation space can be beneficial. It is computationally more efficient and the potentially less complex latent space can alleviate the prediction tasks. Therefore, we estimate the dynamics $p(s_{t+1}, r_t, d_t \mid s_t, a_t)$ with latent variables. Note that the latent variables are no direct approximations of the true MDP states, as their sole purpose is to facilitate the generation of experience.

We will continue with the following notation for conditional expectations: $\mathbb{E}_{p(X \mid Y)}[f(X)]$. The sum rule of probability allows us to introduce latent variables z_t, z_{t+1} into the dynamics

$$\begin{aligned} &p(s_{t+1}, r_t, d_t \mid s_t, a_t) \\ &= \mathbb{E}_{p(Z_t \mid s_t, a_t)}\left[\mathbb{E}_{p(Z_{t+1} \mid s_t, a_t, Z_t)}\left[p(s_{t+1}, r_t, d_t \mid s_t, a_t, Z_t, Z_{t+1})\right]\right]. \end{aligned} \tag{10}$$

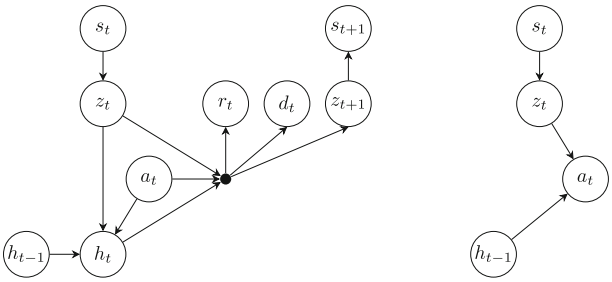
We assume that the graphical models in Fig. 2 describe the dependencies between the random variables. This allows us to estimate the true dynamics using the world model $p_\phi(s_{t+1}, r_t, d_t \mid s_t, a_t) \approx p(s_{t+1}, r_t, d_t \mid s_t, a_t)$ with parameters ϕ in a specific way, as described in the following. The conditional independencies induced by the graphical model in Fig. 2a lead to the following approximation of the dynamics

$$\begin{aligned} &p_\phi(s_{t+1}, r_t, d_t \mid s_t, a_t) \\ &:= \mathbb{E}_{p_\phi(Z_t \mid s_t)}\left[p_\phi(r_t \mid Z_t, a_t)\, p_\phi(d_t \mid Z_t, a_t) \mathbb{E}_{p_\phi(Z_{t+1} \mid Z_t, a_t)}\left[p_\phi(s_{t+1} \mid Z_{t+1})\right]\right]. \end{aligned} \tag{11}$$



(a) Non-recurrent world model (b) Policy

Fig. 2 Graphical models of the non-recurrent world model and the policy. From the perspective of the world model, the observation s_t and the action a_t are noise variables. From the perspective of the policy, s_t is a noise variable. The black dot is an intermediate variable, which reduces the number of arrows



(a) Recurrent world model (b) Policy

Fig. 3 Graphical models of the recurrent world model and the policy

In practice we approximate each expectation with a single Monte Carlo sample. This graphical model is just one of many possibilities to build a world model and now we will explain our motivation for choosing this one. We want an observation encoding model, $p_\phi(z_t \mid s_t)$, that does not depend on the action, so that we can apply a (vector quantized-)variational autoencoder to the observations analogous to the "vision" component of [6]. We also want to predict the next latent variable based on the previous latent variable and action, i.e., $p_\phi(z_{t+1} \mid z_t, a_t)$, independent of the observation s_t, so that no decoding into observations is necessary during the simulation of experience. Furthermore, the reward and next latent variable should not depend on each other, which allows us to use two prediction heads and to compute them in a single neural network pass.

In contrast to [11], our policy is conditioned on the latent variables, so that no decoding into high-dimensional observations is necessary when choosing actions,

$$\pi_\theta(a_t \mid s_t) = \mathbb{E}_{p_\phi(Z_t \mid s_t)}\left[\pi_\theta(a_t \mid Z_t)\right], \tag{12}$$

where θ are the parameters of the policy. We use the conditional independence $a_t \perp\!\!\!\perp s_t \mid z_t$ induced by the graphical model in Fig. 2b.

So far, the world model $p_\phi(s_{t+1}, r_t, d_t \mid s_t, a_t)$ has to predict s_{t+1}, r_t, and d_t based on the current observation s_t and action a_t. As discussed in Sect. 1, the observation might only contain partial information of the true state and the Markov property might not hold. Therefore, we introduce recurrent variables h_{t-1}, h_t that can capture information over multiple time steps, analogous to the "memory" component of [6], and we obtain a *recurrent* world model

Table 1 Summary of the models

	Observation encoder	$p_\phi(z_t \mid s_t)$
	Recurrent dynamics	$p_\phi(h_t \mid z_t, a_t, h_{t-1})$
World model	Latent predictor	$p_\phi(z_{t+1} \mid z_t, h_t, a_t)$
	Reward predictor	$p_\phi(r_t \mid z_t, h_t, a_t)$
	Terminal predictor	$p_\phi(d_t \mid z_t, h_t, a_t)$
	Observation decoder	$p_\phi(s_{t+1} \mid z_{t+1})$
Agent	Policy	$\pi_\theta(a_t \mid z_t)$ or $\pi_\theta(a_t \mid z_t, h_{t-1})$

$p_\phi(s_{t+1}, r_t, d_t, h_t \mid s_t, a_t, h_{t-1}) \approx p(s_{t+1}, r_t, d_t \mid s_t, a_t)$ that potentially is better at dealing with partial observability.

The graphical models in Fig. 3 describe the dependencies between the random variables for our recurrent world model. This allows us to derive the recurrent dynamics

$$\begin{aligned} & p_\phi(s_{t+1}, r_t, d_t, h_t \mid s_t, a_t, h_{t-1}) \\ & := \mathbb{E}_{p_\phi(Z_t \mid s_t)}\Big[p_\phi(h_t \mid Z_t, a_t, h_{t-1})\, p_\phi(r_t \mid Z_t, a_t, h_t)\, p_\phi(d_t \mid Z_t, a_t, h_t) \\ & \qquad \mathbb{E}_{p_\phi(Z_{t+1} \mid Z_t, a_t, h_t)}\left[p_\phi(s_{t+1} \mid Z_{t+1})\right]\Big]. \end{aligned} \tag{13}$$

Once more, we use the independencies induced by the graphical model. Following the graphical model in Fig. 3b, the policy can be conditioned on the recurrent variable, i.e., $\pi_\theta(a_t \mid s_t, h_{t-1}) = \mathbb{E}_{p_\phi(Z_t \mid s_t)}\left[\pi_\theta(a_t \mid Z_t, h_{t-1})\right]$.

We will implement a recurrent world model in this work. We believe that this type of world model is appropriate for a wide range of environments and can also be applied to domains other than Atari, since we made no domain-specific assumptions (except the independence between reward and next latent variable, which could be unfavorable for some environments). From Eq. 13 it becomes clear that several models have to be learned, which are summarized and labeled in Table 1, in addition to the policy. The parameters of the world model are denoted by ϕ and the parameters of the policy by θ. In the next section we describe our architectures that implement the models from Table 1.

3.2 Architecture

3.2.1 Observation Encoder and Decoder

We use a vector quantized-variational autoencoder [25] for the observation encoder and decoder, so that each latent variable z_t is a 6×6 matrix filled with discrete embedding indices. We set the size of the embedding vectors to 32 and there are 128 embedding vectors in total. We apply certain preprocessing steps to the raw frames of the Atari game (see Sect. 2.2), so that the observations s_t are a stack of four grayscale frames, which results in the shape $96 \times 96 \times 4$. Frame stacking allows the observation encoder to incorporate short-time information into the stationary latent representations.

The encoder consists of multiple convolutions each followed by batch normalization [10] and the decoder of multiple deconvolutions. See Fig. 4 for a visualization. After each batch normalization and deconvolution we add a leaky ReLU nonlinearity [14]. The encoder $p_\phi(z \mid s)$ and the prior $p_0(z)$ follow the default VQ-VAE distributions, i.e., one-hot categorical distributions and uniform categorical distributions, respectively. The last deconvolution com-

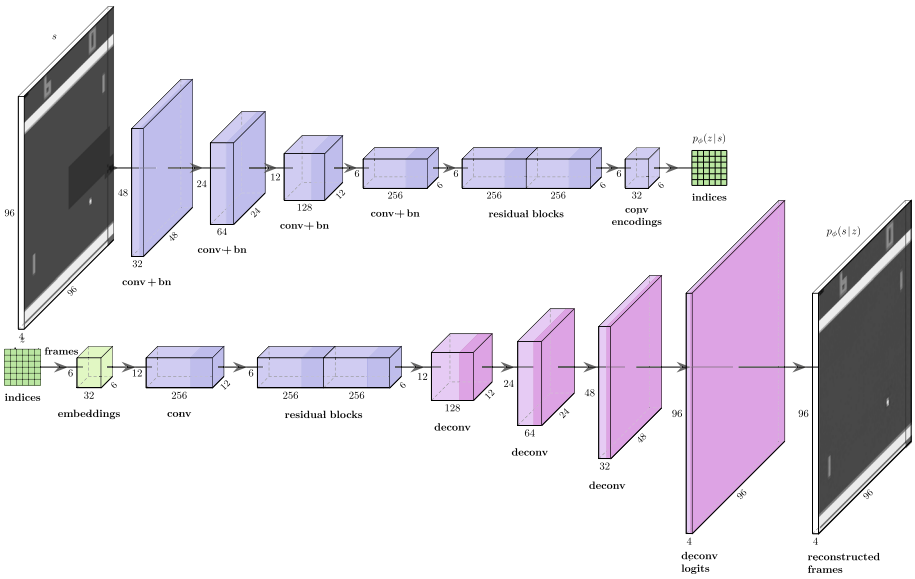


Fig. 4 Observation encoder network (top) and observation decoder network (bottom)

putes the logits of $96 \times 96 \times 4$ continuous Bernoulli distributions for the decoder $p_\phi(s \mid z)$, with independence among the stacked frames and pixels.

Our motivation for a two-dimensional latent representation is that it is able to maintain spatial correlations. In combination with convolutional operations, these correlations can be respected when predicting the next time step.

3.2.2 Dynamics

The recurrent dynamics model is implemented by a two-cell convolutional LSTM [22] with layer normalization [1] after each cell. The input consists of a $6 \times 6 \times 50$ tensor, where the first 32 channels are made up of the embedding vectors, which are looked up in the VQ-VAE using the indices of the latent representation; the last 18 channels contain one-hot encodings of the action, repeated along the spatial dimensions, in order to condition the dynamics on the action. In fact, for environments with less than 18 actions the number of channels is reduced accordingly. The action encodings are also concatenated to the output of each convolutional LSTM cell, because the action information might get lost during the forward pass. Since the output h_t of the last convolutional LSTM cell is deterministic, there is no stochasticity in $p_\phi(h_t \mid z_t, a_t, h_{t-1})$. In addition to a convolutional LSTM we have tested the follow-up architectures spatio-temporal LSTM [26] and causal LSTM [27], but they have increased the number of parameters and the training time significantly, while the performance has stayed the same.

After that there are two prediction heads: one for the latent predictor, consisting of one convolutional layer, and one for the reward predictor, consisting of a convolutional layer with layer normalization and two fully-connected layers, each followed by a leaky ReLU nonlinearity. The entire model is depicted in Fig. 5.

The latent predictor head computes the unnormalized scores $U \in \mathbb{R}^{6\times6\times128}$ for the embedding indices. These scores get normalized via the softmax function, in order to obtain an independent categorical distribution for each component of z_{t+1}, i.e.,

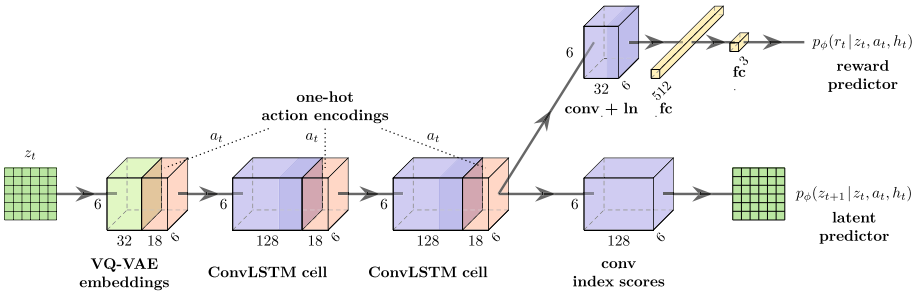


Fig. 5 Dynamics network. After the second convolutional LSTM cell, the network splits into the reward predictor head at the top, and the latent predictor head at the bottom. The recurrent states h_t, h_{t-1} are omitted for clarity

$$p_\phi(z_{t+1} \mid z_t, a_t, h_t) := \prod_{j=1}^{6} \prod_{k=1}^{6} p_\phi\left((z_{t+1})_{j,k} \mid z_t, a_t, h_t\right) \tag{14}$$

$$= \prod_{j=1}^{6} \prod_{k=1}^{6} \text{Cat}\left((z_{t+1})_{j,k}; 128, \text{Softmax}(U_{j,k})\right), \tag{15}$$

where $(z_{t+1})_{j,k}$ is the latent matrix entry at row j and column k, and $U_{j,k}$ is the output vector at row j and column k. We suppose that the discretization of the latent space stabilizes the latent predictor, since it has to predict scores for a predefined set of categories instead of real values. This is especially important when we take into account that the targets are moving, i.e., that the latent representations, which are produced by the VQ-VAE, change during training.

The rewards are discretized into three categories $\{-1, 0, 1\}$ by clipping them into the interval $[-1, 1]$ and rounding to the nearest integer. The reward predictor head computes the unnormalized scores $V \in \mathbb{R}^3$ for the corresponding reward categories. These scores are normalized via the softmax function to obtain a categorical distribution, i.e.,

$$p_\phi(r_t \mid z_t, a_t, h_t) := \text{Cat}\left(r_t; 3, \text{Softmax}(V)\right). \tag{16}$$

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.

Since Atari games are episodic, the world model has to terminate episodes by predicting the binary terminal variable d_t. This prediction has to be reliable, as an incorrect prediction of $d_t = 1$ can have a severe impact on the simulated experience, and thus on the policy. We follow [11] and employ a naive but effective solution, which is to end all episodes after a fixed number of time steps T_{sim}, i.e., we assign the following fixed distribution,

$$p_\phi(d_t = 1 \mid z_t, a_t, h_t) := \begin{cases} 1, & \text{if } t \geq T_{\text{sim}} \\ 0, & \text{otherwise,} \end{cases} \tag{17}$$

where $T_{\text{sim}} := 50$ for all of our experiments. On the downside, this prevents the policy to learn from effects that are longer than T_{sim} [11].

3.2.3 Policy

Our policy $\pi_\theta(a_t \mid z_t)$ is only conditioned on the latent representation, and not on the recurrent variable h_{t-1}, as presented in Eq. 12. The policy network is visualized in Fig. 6.

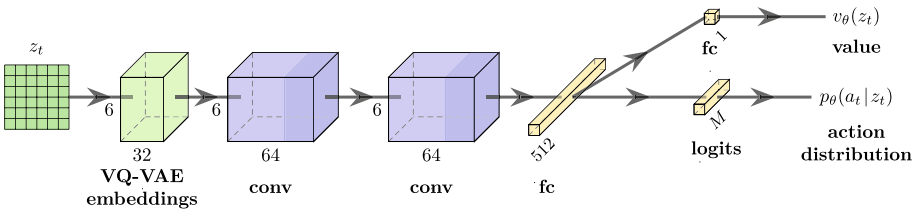


Fig. 6 Policy network. After the first fully-connected layer that computes a "hidden" vector, the network splits up into the value prediction head at the top, and the action distribution head at the bottom

The embedding vectors of the VQ-VAE are sent through two convolutional layers with layer normalization and a fully connected layer; all layers are followed by leaky ReLU nonlinearities. This "hidden" output is passed to two separate fully connected layers: the estimated state-value function $v_\theta(z_t)$, and the action logits $W \in \mathbb{R}^M$, which are the unnormalized scores for the M possible actions. The action distribution is a categorical distribution, which we get by normalizing the scores with the softmax function,

$$\pi_\theta(a_t \mid z_t) := \text{Cat}\left(a_t; M, \text{Softmax}(W)\right). \tag{18}$$

3.3 Training Procedure

We follow the Simulated Policy Learning (SimPLe) training procedure that was presented by [11]. The idea is to iteratively repeat three steps, namely, (i) Collecting real experience and storing it in a data buffer, (ii) Training the world model with experience from the data buffer, and (iii) Training the agent by simulating experience with the world model. This loop can be illustrated with the following pseudocode:

```
for i = 1 to num_iterations:
  batch = collect_experience(env, agent)
  data_buffer.append(batch)
  train_world_model(world_model, data_buffer)
  train_agent(agent, world_model)
```

In the following we provide more detailed descriptions of the individual steps.

3.3.1 Collecting Experience

In the first step of each iteration experience is collected from the real environment. In the first iteration, we apply a random policy that samples actions uniformly. In subsequent iterations, we use the trained policy. The observation encoder processes the raw observations to produce latent representations, since the policy is conditioned on them. This process is visualized in Fig. 7a. There is no recurrency involved at this point.

We adopt the evaluation method from [11] and limit the total number of interactions (i.e., the number of actions taken) throughout the entire training process to roughly $100K$. We follow [11] and perform 15 iterations with 6400 interactions per iteration. They perform additional 6400 interactions prior to the first iteration, therefore, we perform 12800 interactions in the first iteration, resulting in the same number of total interactions, i.e., 102400. We sample from a single environment instance and do not reset the environment between iterations.

As [11] have mentioned, collecting useful experience is crucial for the performance of the world model, especially in the low data regime. To improve exploration, we increase the randomness of the policy by inserting a temperature parameter $\tau > 0$ into the Softmax calculation of the policy, i.e.,

$$\pi_\theta(a_t = a_i \mid z_t) := \frac{e^{f_{\theta(z_t)_i}/\tau}}{\sum_{j=1}^{M} e^{f_{\theta(z_t)_j}/\tau}}, \tag{19}$$

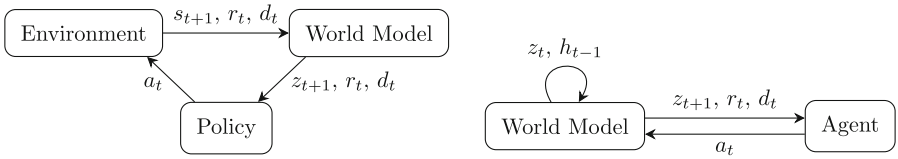
where $f_\theta(z_t) \in \mathbb{R}^M$ are the unnormalized scores computed by the policy network, and a_i is the ith action for $i \in \{1, \ldots, M\}$. Increasing the temperature τ causes the softmax normalization to be "softer", i.e., it makes the distribution more uniform, while decreasing τ moves the distribution closer to a maximum.

3.3.2 Training the World Model

In the second step of each iteration the world model is trained in a supervised manner using the collected real experience. We sample minibatches of N_{obs} observations from the data buffer, $s^{(i)} \sim p_{\text{buffer}}(s)$ for $i \in \{1, \ldots, N_{\text{obs}}\}$, where $p_{\text{buffer}}(s)$ is a uniform distribution over all collected real observations in the data buffer. With these minibatches we minimize the VQ-VAE loss using the Adam optimizer [12], in order to optimize the observation encoder and decoder models.

To perform a parameter update of the dynamics models, we first sample a minibatch of N_{seq} sequences of length T_{seq} of observations, actions, and rewards from the data buffer. Then, we initialize the recurrent states with all zeros and iterate over the sequences step by step. At each step the observation encoder converts the observations to latent variables. The dynamics models take the latent variables, the actions, and the last recurrent states to predict the next latent variables and rewards, while also computing the next recurrent states. After all T_{seq} time steps are processed, the sum over the sequence of the negative log-likelihood of the predicted latent and reward distributions is minimized with the Adam optimizer. To make training more robust, the "true" encoded latent variables are used as input to the dynamics models in the first T_{context} time steps (also called the context size), and after that the latent variables from the latent predictor are used. This can mitigate the issue that bad predictions at the beginning of a sequence degenerate the remaining ones. Gradients coming from the reward predictor head (see Fig. 5) can have a relatively high magnitude, which can have a degrading effect on the performance of the other models. Therefore, we scale down the negative log-likelihood of the rewards, but increase the learning rate of the reward predictor head to compensate for the scaled down gradients.

The VQ-VAE and the dynamics models are trained in an alternating fashion, i.e., we perform a single parameter update of the VQ-VAE and then of the dynamics models. When the entire data buffer was used for training the dynamics models, one epoch is finished. Multiple epochs are executed per iteration, and the number of epochs decreases in every iteration since the data buffer grows; the exact numbers can be found in Table 5. After collecting the first batch of data we train the VQ-VAE separately for 50 epochs with a higher learning rate (in the "warm-up" phase). We want to give the dynamics models a better starting point by feeding it with latent representations that already contain useful information. This cannot be done in later training stages, however, as the representations would change and the dynamics models would not be able to keep up easily. Indeed, in the entire course of training, the rate at which the observation encoder model changes proved to be important. We have to adjust the training of the VQ-VAE in such a way, that it can learn from newly obtained



(a) Pipeline when collecting experience and at inference time. The policy selects an action which is sent to the environment. The world model encodes the experience of the environment.

(b) Pipeline at training time. The world model simulates experience and maintains a recurrent state, and the agents selects actions based on the experience.

Fig. 7 The interactions between world model, agent, and environment, at inference and at training time. In both cases the agent receives the experience in simulation space and does not notice a difference

observations, without changing the latent representations too heavily. After this initial phase we even slow down the training of the observation encoder and decoder models by performing only every second or third parameter update, and effectively fixing the parameters in between (see *update intervals* in Table 5).

As a side note, [11] state that the scale of the KL divergence loss of a variational autoencoder is game dependent, which makes VAEs impractical to apply to all Atari games without fine-tuning. The VQ-VAE does not suffer from this problem, because the KL term is constant and only depends on the dimensions of the latent space and the number of embedding vectors, as explained in Sect. 2.3.

3.3.3 Simulating Experience

The third step in each iteration is to improve the policy by simulating experience from the world model. To simulate a minibatch of N_{sim} episodes, we initialize the latent variables by encoding N_{sim} randomly selected observations from the data buffer. This enables the policy to learn from experience from any stage of the environment, although the number of time steps is limited to T_{sim} (see Sect. 3.2.2). The rest of the experience is generated iteratively. At any time step t, the policy selects actions based on the current latent states and the world model computes the next latent state, reward, and hidden state based on the current latent states, actions, and recurrent states. See Fig. 7b for a visualization of this process.

3.3.4 Training the Policy

The last part of the third step is to train the policy in a model-free manner. We apply proximal policy optimization [19] to the simulated experience with the action distribution and state-value being computed by the policy network (see Fig. 6). We set the temperature parameter from Eq. 19 back to $\tau := 1$, since we do not want the extra randomness in the context of simulation. We approximate the PPO objective from Eq. 8 using minibatches of simulated experience with batch size N_{sim} and horizon T_{horizon}. We align the horizon with the maximum number of simulated time steps, i.e., we set $T_{\text{horizon}} := T_{\text{sim}}$. We estimate the advantage function using generalized advantage estimation (GAE, [20]). In most iterations 1000 minibatches are simulated and trained upon, but in some iterations we gnerate 2000 or 3000 minibatches. These numbers were determined empirically by [11] and can be found in Table 5.

4 Evaluation

In this section we evaluate our world model on Atari games, provide analyses on the results, and perform further ablation studies on a smaller scale.

4.1 Performance Comparison

We restrict our agent to 100 K interactions per game. We evaluate two approaches: using a single frame and using four stacked frames. Both approaches have slight advantages in some of the games. Further discussion on this can be found in Sect. 4.3.

We average the results over five training runs with different seeds. For every run we evaluate the latest policy in each iteration by rolling out 32 episodes in the real environment and computing the mean of the (cumulative) episode rewards. In Table 2 we report the mean final episode rewards (i.e., the mean episode reward after the final iteration; averaged over five runs) for 36 Atari games. In Table 5 we provide a detailed list of all hyperparameters. In Fig. 9 we show the course of the mean episode reward over the 15 iterations for all games.

We compare our method with the default variant of SimPLe (stochastic discrete, 50 steps, $\gamma = 0.99$), 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 [11] that we show are not necessarily the best results across all of their variants, but the best for a fairer comparison. Our method achieves a higher value than this SimPLe variant in 20 out of 36 tested games.

4.1.1 Comparison with Human Performance and 10*M*

In Table 3 we compare our method (the best score of the default frame stacking and no frame stacking) with a random agent, a human baseline [15], and PPO trained on 10 M interactions [19]. The human score is the episode reward averaged over 20 episodes, after around two hours of practice in each game, meaning that it is comparable to the 100 K interactions. We also state the human normalized score [15] of our method, which is calculated as $(\textit{our score} - \textit{random score})/(\textit{human score} - \textit{random score})$, such that 0% corresponds to the random score and 100% corresponds to the human score. Values lower than 0% mean that the performance is worse than random and values higher than 100% indicate "superhuman" performance. We achieve superhuman performance in three games. In Fig. 8 we compare our human normalized scores with SimPLe.

4.1.2 Model Size

In Table 4 we show the number of parameters of our world model compared with the SimPLe world model by [11], which uses about seven times as many parameters. It also shows the number of parameters of all of our models in detail. At training time all models are used, but at inference time only the observation encoder and the policy network are necessary.

4.1.3 Training Times

A single run takes approximately 12 hours on an Nvidia A100 GPU. Due to the small model size and small batch sizes, the memory usage is quite low. On an Nvidia P100 GPU, our

Table 2 Mean final episode reward of our method (with and without frame stacking) in comparison with SimPLe [11] and model-free PPO [19] trained with 100 K interactions. The numbers are taken from [11]

Game	Ours (4 frames)		Ours (1 frame)		SimPLe (SD, $\gamma = 0.99$)		PPO 100 K	
	Mean	Std. dev	Mean	Std. dev	Mean	Std. dev	Mean	Std. dev
Alien	409.9	(± 73.0)	**423.3**	(± 48.1)	405.2	(± 130.8)	291.0	(± 40.3)
Amidar	37.6	(± 13.6)	30.5	(± 10.1)	**88.0**	(± 23.8)	56.5	(± 20.8)
Assault	375.4	(± 111.9)	408.3	(± 27.8)	369.3	(± 107.8)	**424.2**	(± 55.8)
Asterix	504.4	(± 53.3)	456.5	(± 146.4)	**1089.5**	(± 335.3)	385.0	(± 104.4)
Asteroids	862.9	(± 85.4)	989.9	(± 88.7)	731.0	(± 165.3)	**1134.0**	(± 326.9)
Atlantis	9413.1	(± 3349.8)	15463.7	(± 5478.7)	14481.6	(± 2436.9)	**34316.7**	(± 5703.8)
BankHeist	101.2	(± 17.4)	**249.3**	(± 49.8)	8.2	(± 4.4)	16.0	(± 12.4)
BattleZone	**5631.2**	(± 1179.1)	5531.3	(± 2515.4)	5184.4	(± 1347.5)	5300.0	(± 3655.1)
BeamRider	410.4	(± 55.4)	527.6	(± 61.8)	422.7	(± 103.6)	**563.6**	(± 189.4)
Bowling	27.9	(± 4.8)	24.5	(± 5.1)	**34.4**	(± 16.3)	17.7	(± 11.2)
Boxing	−2.8	(± 5.7)	−9.3	(± 12.6)	**9.1**	(± 8.8)	−3.9	(± 6.4)
Breakout	8.8	(± 1.5)	8.4	(± 1.5)	**12.7**	(± 3.8)	5.9	(± 3.3)
ChopperCommand	766.2	(± 195.3)	590.6	(± 335.0)	**1246.9**	(± 392.0)	730.0	(± 199.0)
CrazyClimber	**47536.9**	(± 6114.9)	36923.8	(± 2780.6)	39827.8	(± 22582.6)	18400.0	(± 5275.1)
DemonAttack	195.0	(± 76.4)	**211.3**	(± 86.2)	169.5	(± 41.8)	192.5	(± 83.1)
FishingDerby	−89.6	(± 4.5)	**−87.9**	(± 4.1)	−91.5	(± 2.8)	−95.6	(± 4.3)
Freeway	**24.6**	(± 3.4)	11.3	(± 9.6)	20.3	(± 18.5)	8.0	(± 9.8)
Frostbite	214.4	(± 10.2)	219.1	(± 45.6)	**254.7**	(± 4.9)	174.0	(± 40.7)
Gopher	687.2	(± 91.1)	**1398.4**	(± 166.5)	771.0	(± 160.2)	246.0	(± 103.3)
Gravitar	87.2	(± 60.9)	82.2	(± 64.5)	198.3	(± 39.9)	**235.0**	(± 197.2)

Table 2 continued

Game	Ours (4 frames)		Ours (1 frame)		SimPLe (SD, $\gamma = 0.99$)		PPO 100 K	
	Mean	Std. dev	Mean	Std. dev	Mean	Std. dev	Mean	Std. dev
Hero	3453.6	(± 594.7)	**3911.6**	(± 1259.3)	1295.1	(± 1600.1)	569.0	(± 1100.9)
IceHockey	−13.6	(± 2.5)	−12.0	(± 2.1)	−10.5	(± 2.2)	**−10.0**	(± 2.1)
Jamesbond	66.6	(± 7.8)	46.6	(± 24.2)	**125.3**	(± 112.5)	65.0	(± 46.4)
Kangaroo	245.0	(± 99.6)	276.3	(± 136.7)	**323.1**	(± 359.8)	140.0	(± 102.0)
Krull	3520.2	(± 211.4)	3241.0	(± 448.4)	**4539.9**	(± 2470.4)	3750.4	(± 3071.9)
KungFuMaster	11903.1	(± 4399.5)	8521.2	(± 1330.9)	**17257.2**	(± 5502.6)	4820.0	(± 983.2)
MsPacman	652.2	(± 92.6)	668.3	(± 86.4)	**762.8**	(± 331.5)	496.0	(± 379.8)
NameThisGame	**2448.4**	(± 179.5)	2119.4	(± 217.9)	1990.4	(± 284.7)	2225.0	(± 423.7)
Pong	**11.8**	(± 6.9)	−3.9	(± 7.6)	5.2	(± 9.7)	−20.5	(± 0.6)
PrivateEye	**99.4**	(± 1.2)	96.9	(± 5.4)	58.3	(± 45.4)	10.0	(± 20.0)
Qbert	480.9	(± 143.5)	**617.5**	(± 149.5)	559.8	(± 183.8)	362.5	(± 117.8)
Riverraid	2100.2	(± 50.4)	**2273.3**	(± 188.5)	1587.0	(± 818.0)	1398.0	(± 513.8)
RoadRunner	1562.5	(± 440.2)	1723.8	(± 688.2)	**5169.4**	(± 3939.0)	1430.0	(± 760.0)
Seaquest	458.1	(± 155.7)	**531.5**	(± 105.1)	370.9	(± 128.2)	370.0	(± 103.3)
UpNDown	1128.2	(± 247.6)	1354.6	(± 741.4)	2152.6	(± 1192.4)	**2874.0**	(± 1105.8)
YarsRevenge	4096.0	(± 520.9)	4360.3	(± 1156.9)	2980.2	(± 778.6)	**5182.0**	(± 1209.3)

Table 3 Mean final episode reward and human normalized score of our method compared with a random agent, the human baseline, and PPO at 10M

Game	Random	PPO 10M	Human	Ours (best)	HNS
Alien	227.8	1850.3	6875.4	423.3	2.9%
Amidar	5.8	674.6	1675.8	37.6	1.9%
Assault	222.4	4971.9	1496.4	408.3	14.6%
Asterix	210.0	4532.5	8503.3	504.4	3.5%
Asteroids	719.1	2097.5	13156.7	989.9	2.2%
Atlantis	12850.0	2311815.0	29028.1	15463.7	16.2%
BankHeist	14.2	1280.6	734.4	249.3	32.6%
BattleZone	2360.0	17366.7	37800.0	5631.2	9.2%
BeamRider	363.9	1590.0	5774.7	527.6	3.0%
Bowling	23.1	40.1	154.8	27.9	3.6%
Boxing	0.1	94.6	4.3	−2.8	−69.0%
Breakout	1.7	274.8	31.8	8.8	23.6%
ChopperCommand	811.0	3516.3	9881.8	766.2	−0.5%
CrazyClimber	10780.5	110202.0	35410.5	47536.9	**149.2**%
DemonAttack	152.1	11378.4	3401.3	211.3	1.8%
FishingDerby	−91.7	17.8	5.5	−87.9	3.9%
Freeway	0.0	32.5	29.6	24.6	83.1%
Frostbite	65.2	314.2	4334.7	219.1	3.6%
Gopher	257.6	2932.9	2321.0	1398.4	55.3%
Gravitar	173.0	737.2	2672.0	87.2	−3.4%
Hero	1027.0	N/A	25762.5	3911.6	11.7%
IceHockey	−11.2	−4.2	0.9	−12	−6.6%
Jamesbond	29.0	560.7	406.7	66.6	10.0%
Kangaroo	52.0	9928.7	3035.0	276.3	7.5%
Krull	1598.0	7942.3	2394.6	3520.2	**241.3**%
KungFuMaster	258.5	23310.3	22736.2	11903.1	51.8%
MsPacman	307.3	2096.5	15693.4	668.3	2.3%
NameThisGame	2292.3	6254.9	4076.2	2448.4	8.8%
Pong	−20.7	20.7	9.3	11.8	**108.3**%
PrivateEye	24.9	69.5	69571.3	99.4	0.1%
Qbert	163.9	14293.3	13455.0	617.5	3.4%
Riverraid	1338.5	8393.6	13513.3	2273.3	7.7%
RoadRunner	11.5	25076.0	7845.0	2337.5	21.9%
Seaquest	68.4	1204.5	20181.8	531.5	2.3%
UpNDown	533.4	95445.0	9082.0	1354.6	9.6%
YarsRevenge	3092.9	N/A	54576.9	4360.3	2.4%

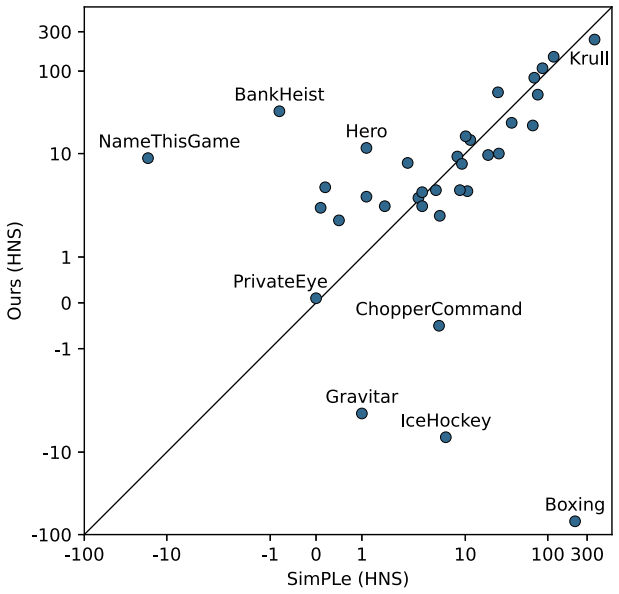


Fig. 8 Human normalized score (HNS) of our models and SimPLe [11] on a symmetrical logarithmic scale

Table 4 Number of parameters of our world model, the SimPLe world model [11], and the DreamerV2 world model [8]. The numbers are approximate

	# Parameters
VQ-VAE encoder	2.0 M
VQ-VAE decoder	2.1 M
Dynamics network	6.2 M
Our world model	10.3 M
SimPLe world model	74.0 M
DreamerV2 world model	20.0 M

method takes roughly 25 hours, while SimPLe [11] requires 500 hours on the same hardware [21], which is 20 times slower.

4.2 Analysis

In our experiments we observe the same phenomenon as [11], namely, that the results can vary drastically for different runs with the same hyperparameters but with different random seeds. A possible explanation for this is that the world model cannot infer dynamics of regions of the environment's state space that it has never seen before and that the algorithm is sensitive to the exploration-exploitation trade-off, as the number of interactions is relatively low.

4.2.1 Latent Representations

In Fig. 10 we show a visualization of the latent variables and their reconstructions. We assign random colors to the embedding indices and draw a colored square for each entry of the latent variable matrix. Note that a single square does not correspond to the same area in the

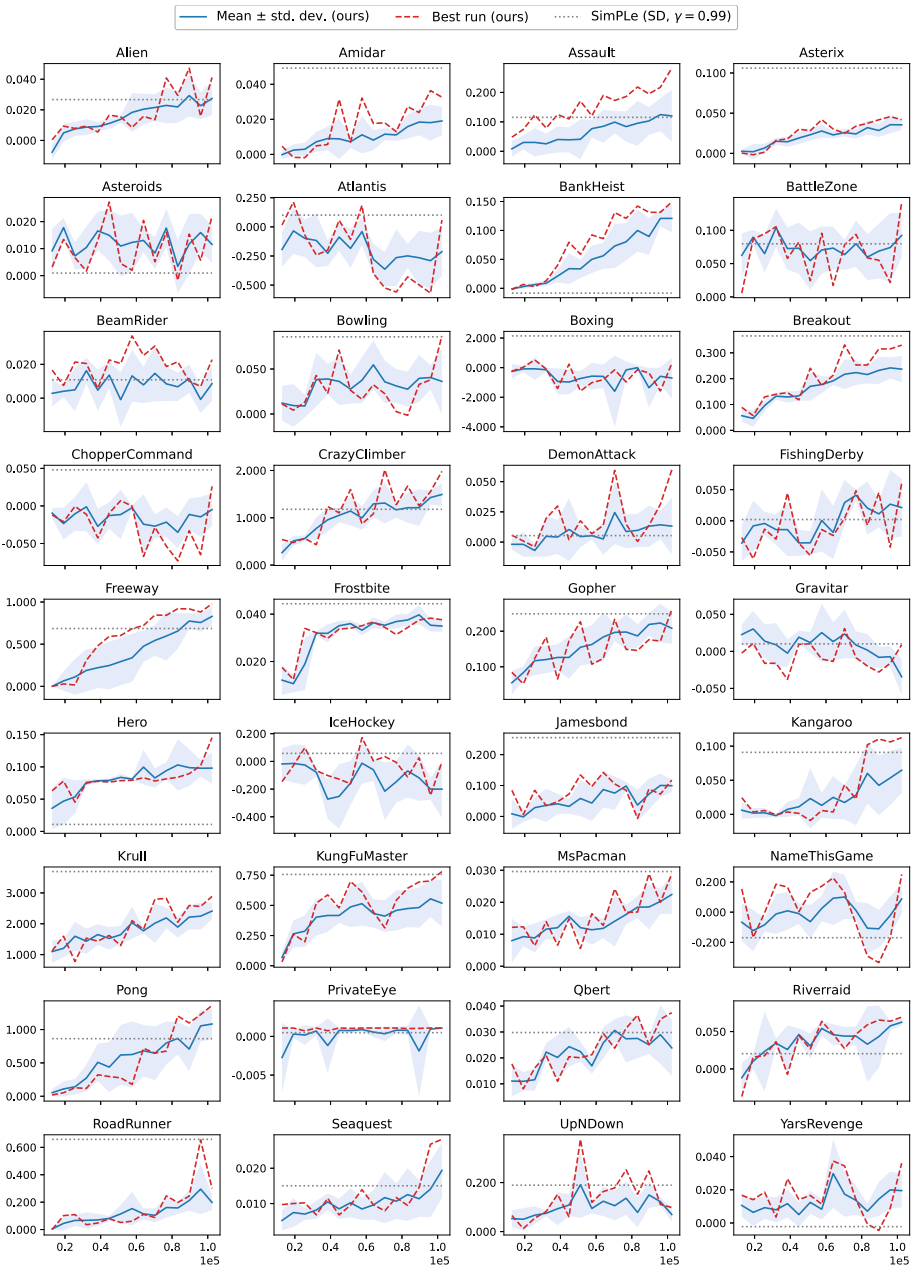
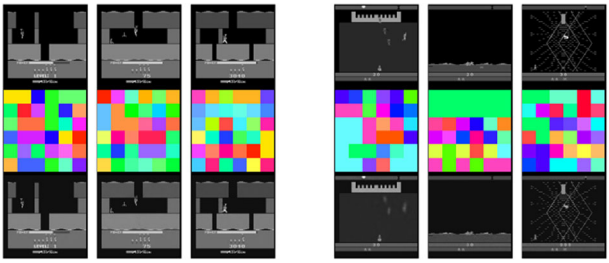
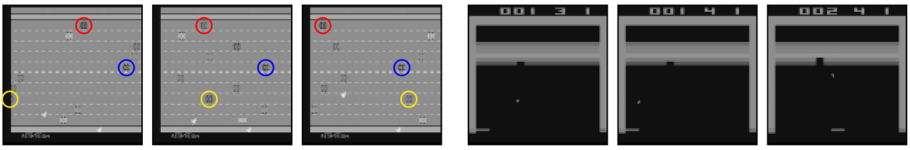


Fig. 9 Human normalized scores of our method (4 frames), averaged over five runs ± standard deviation. The x-axis shows the number of interactions with the real environment, but does not reflect the number of parameter updates that were performed in between. The dashed lines show the best of the five runs and the horizontal lines show the final human normalized score of SimPLe [11], since we only know their final scores



(a) Hero (b) Krull

Fig. 10 Original game frames (top row), the encoded 6×6 discrete latent variables (middle), and the reconstructions from the VQ-VAE (bottom)



(a) Freeway (b) Breakout

Fig. 11 Snapshots of good sequences produced by our dynamics models

frame, as they have larger receptive fields (also see Sect. 4.3). The VQ-VAE is able to encode most information into the latent representation. We have picked these two games to show that changes in the scene (Fig. 10a) or even switching scenes (Fig. 10b) can be represented, although this can cause some loss of details (e.g., see the left reconstruction in Fig. 10b).

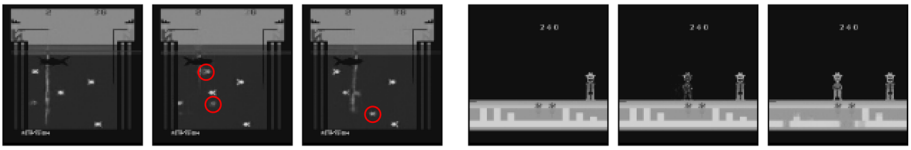
4.2.2 Generated Sequences

A straightforward method to evaluate the dynamics models is to observe the generated sequences. A video of some generated sequences can be found at https://github.com/jrobine/smaller-world-models. In this section we analyze the sequences in more detail.

First, Fig. 11a shows frames of Freeway at a 40 frame interval. The world model has learned to predict the car movement in the correct direction (to the left at the top, and to the right at the bottom) and with varying speeds (but constant speeds per car; the red [top] and yellow [bottom] highlighted cars move faster than the blue [middle] one). Second, Fig. 11b shows that the world model is able to predict the ball movement in Breakout. In the first frame the ball moves down to the left; in the second frame the ball has correctly bounced off the paddle and the wall, and moves up to the right; in the third frame the ball has removed a block and has bounced off, and the score has been increased. However, the life counter has been increased incorrectly in the second frame.

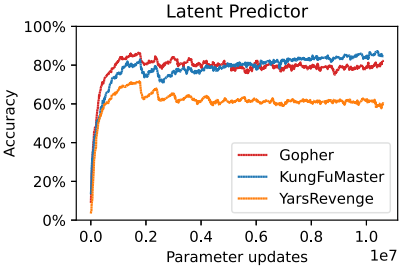
In Fig. 12a we see a consecutive sequence of frames of FishingDerby in which some fish appear and disappear (highlighted in red). We observe this phenomenon in several games that are complex and involve multiple objects. Moreover, Fig. 12b shows consecutive frames of Gopher in which a second farmer appears out of nowhere, which should be impossible, and the tunnels at the bottom change arbitrarily in the third frame.

These examples illustrate that the world model tries to predict everything that is visible, e.g., even scores or other elements of the user-interface, no matter how helpful the information is for the agent. Furthermore, although the world model is not capable of modeling the

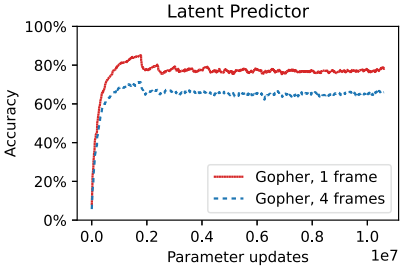


(a) FishingDerby (b) Gopher

Fig. 12 Snapshots of bad sequences produced by our dynamics models



(a) Accuracy for three Atari games. Each curve corresponds to a single run, so that the drops in performance are more noticeable.



(b) Accuracy for Gopher with frame stacking and without frame stacking, averaged over five runs.

Fig. 13 Accuracy of the latent predictor

dynamics of all games, we have observed that the predictions are relatively stable in the sense that the frames rarely degenerate completely (for example, fixed parts of the frames like the background are relatively stable), even way beyond the number of time steps that are used to train the dynamics models.

4.2.3 Latent Prediction Accuracy

On the one hand, acting on latent representations is computationally more efficient, since simulation can be performed purely in latent space without reconstructing observations. On the other hand, the representations can be less stable since they change during training and do not need to converge, and after each world model training iteration the agent gets confronted with new representations. In Fig. 13a we show the accuracy of the latent predictor $p_\phi(z_{t+1} \mid z_t, a_t, h_t)$ over the course of training for three Atari games. There is a drop in accuracy every time a new iteration starts, since new unseen data is fed into the network, especially in early iterations. As already mentioned, it has proved to be better to train the observation encoder and decoder and the dynamics models simultaneously, because otherwise we would see more drops similar to Fig. 13a. Notice that the accuracy stays approximately constant, so we might ask whether the system is learning anything. The reason is that Fig. 13a reports the accuracy on the data seen so far, so the accuracy on the right has a different meaning than the accuracy on the left.

4.2.4 Mean Episode Reward

We consider three learning curves from Fig. 9 that are typical for our world model. First, in Freeway the agent's performance successfully increases over the course of training. Second, in

Gravitar the agent's performance decreases over the course of training. This can have various reasons, e.g., when the agent reaches a new area of the state space and the environment's dynamics change drastically. Finally, in Hero the agent has comparably high performance from the beginning, which is most likely due to the inductive bias of the model.

4.3 Ablation Studies

To better understand which components of our model contribute to the overall performance, we conducted a series of ablation studies to answer the following questions (short answers in parentheses). We describe the results of the ablation studies in detail in the next sections.

(1) Can frame stacking improve the performance? (In a lot of environments feeding a single frame into the VQ-VAE model is sufficient.)
(2) Which role plays the discretization of the latent space and how does it interact with the dimensionality of the latent space? (A two-dimensional discrete latent space yields the best results.)
(3) Is conditioning the policy on reconstructed frames more suitable than acting in the latent space? (The latent representation is sufficient in most environments.)
(4) Can the performance be improved by autoregressively sampling the latent variable? (No, and it furthermore slows down the inference time.)
(5) Does a smaller receptive field improve the performance, since it simplifies the task of the dynamics model? (No, but it allows for more interpretable visualizations.)

Q1: Can Frame Stacking Improve the Performance?

As described earlier, our default VQ-VAE works with four stacked frames as input and as output. On the downside, this introduces complexity for the dynamics models, since the same frame can get a different representation, depending on the three other frames in the stack. In this experiment we set the frame stacking value to one, i.e., we disable frame stacking. The results indicate that this can lead to significant improvements, which is why we have expanded this experiment to all environments and have included it in the main results. In Fig. 13b we show the increase in accuracy of the latent predictor in Gopher when using no frame stacking. However, this increase is not necessarily present in all environments, even if the overall performance is improved, since there are other important factors, e.g., how useful the latent representations are for the policy. The training times stay roughly the same, since the only change is that the depth of the filters in the first convolutional layer increases.

Q2: Which Role Plays the Discretization of the Latent Space and How Does it Interact with the Dimensionality of the Latent Space?

In this experiment we have compared different models to evaluate a discrete vs. a continuous latent space and a two-dimensional vs. a one-dimensional latent space. This leads to four model configurations:

(1) *Discrete, two-dimensional latent space* This is the default configuration, so no changes have been made.
(2) *Discrete, one-dimensional latent space* We have changed the size of the latent representation from 6×6 to 36×1 by stacking the rows of the output of the observation encoder. This way we can still use the convolutional LSTM that takes embedding vectors as input. The architecture of the policy network only needs to be adjusted to the new input size.

Table 5 Summary of the hyperparameters

Hyperparameter	Value
Number of iterations	15
Number of world model epochs (per iteration)	140, 32, 24, 20, 16, 14, 12, 11, 10, 9, 8, 8, 7, 7, 6
Collecting experience	
Number of environments	1
Number of interactions (per iteration)	6400
Temperature parameter τ	1.5
Simulating experience	
Number of environments/batch size N_{sim}	16
Maximum number of simulation steps T_{sim}	50
Number of minibatches (per iteration)	1000
Temperature parameter τ	1.0
VQ-VAE	
Input and output channels (frame stacking)	1 or 4
Shape of latent representations	6×6
Number of embeddings	128
Size of embedding vectors	32
Batch size	40
Learning rate	0.0001
Adam β_1 parameter	0.9
Adam β_2 parameter	0.999
"Warm-up" learning rate	0.001
Number of "warm-up" epochs	50
Update intervals (per iteration)	3
Dynamics network	
Batch size N_{seq}	4
Sequence length T_{seq}	11
Context size T_{context}	4
Learning rate	0.0005
Adam β_1 parameter	0.9
Adam β_2 parameter	0.999
Reward head learning rate	0.001
Reward loss weight	0.000001
Policy network and PPO	
Learning rate	0.00025
Adam β_1 parameter	0.9
Adam β_2 parameter	0.999
Horizon T_{horizon}	50
Discount factor γ	0.99
Generalized advantage estimation factor λ	0.95
Entropy regularization coefficient c_1	0.01
Value function loss coefficient c_2	0.5

Table 5 continued

Hyperparameter	Value
Number of PPO optimization steps	4
PPO optimization minibatch size	256
PPO clipping value ϵ	0.1

Table 6 Mean final episode reward of the model configurations

Game	Discrete 2d	Discrete 1d	Continuous 2d	Continuous 1d
BankHeist	**249.3**	150.4	107.4	106.2
Gopher	**1398.8**	784.4	538.6	357.3
Hero	**3911.6**	3162.1	2868.8	2187.9
Seaquest	**531.5**	302.9	354.4	245.9
Riverraid	2273.3	2256.1	2253.8	**2472.8**

(3) *Continuous, two-dimensional latent space* We have changed the posterior distribution of the observation encoder to a product of independent normal distributions (instead of categorical distributions), and the latent space prior to a standard multivariate normal distribution. This corresponds to the usual setup for VAEs, but with a two-dimensional latent space. The output of the encoder is a $6 \times 6 \times 2$ tensor with means in the first channel and standard deviations (actually, the logarithm of the variance) in the second channel. The latent representations are sampled from the posterior and no vector quantization is applied. Most parts of the dynamics model stays the same, but the input of the convolutional LSTM is a $6 \times 6 \times 1$ tensor instead of the 6×6 tensor of embedding vectors, and the next latent head computes the parameters of independent normal distributions, analogous to the observation encoder. The policy network is adjusted to the one channel input.

(4) *Continuous, one-dimensional latent space* This is a combination of the previous two configurations, i.e., the observation encoder computes two vectors of length 36, one for the mean and one for the standard deviations of 36 independent normal distributions. The latent space prior is a standard multivariate normal distribution. In the dynamics model we have used an LSTM instead of a convolutional LSTM, since the latent representations are real-valued vectors. For the same reason, the policy network uses linear layers instead of convolutional layers.

For the evaluation we have selected five environments in which our method performed best and no frame stacking has been used. The results are shown in Table 6. The main observations are that a discrete latent space tends to lead to a better performance (compare the first two columns with the last two columns), as well as a two-dimensional one (compare column one with column three, and column two with column four).

Q3: Is Conditioning the Policy on Reconstructed Frames More Suitable Than Acting in the Latent Space?

In this experiment we condition the policy on the reconstructed frames similar to the SimPLe world model by [11], i.e., the outputs of the VQ-VAE decoder. To improve the quality of

Table 7 Mean final episode reward of our models when the policy is conditioned on the reconstructed frames (without frame stacking) compared with the default models, averaged over five runs

Game	Reconstructed frames (1 frame)	Latent representations (1 frame)	Latent representations (4 frames)
BankHeist	241.3	**249.3**	101.2
Breakout	**10.9**	8.4	8.8
CrazyClimber	**53652.5**	36923.8	47536.9
Freeway	14.8	11.3	**24.6**
KungFuMaster	9848.8	8521.2	**11903.1**
MsPacman	**742.3**	668.3	652.2
Pong	−2.2	−3.9	**11.8**
Seaquest	482.9	**531.5**	458.1

the reconstructions, we apply no frame stacking. We adjust the policy network to match the $96 \times 96 \times 1$ input tensors, and add downsampling convolutional layers. We evaluate this setup in a randomly selected subset of the environments and achieve better results in three out of eight environments compared with the policy conditioned on latent representations, while the results in the remaining five environments are relatively good as well, as can be seen in Table 7.

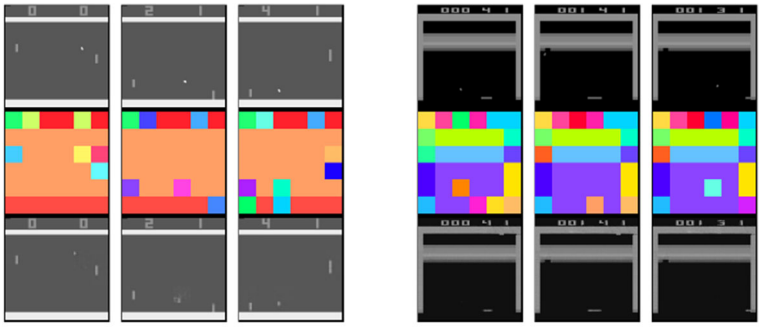
This means that in some environments the frames are a more suitable representation for the agent than the latent variables. The training time is increased quite significantly, due to the additional costs of the observation decoder and the policy network, from roughly 12 hours to 21 hours per run on an Nvidia A100 GPU.

Q4: Can the Performance be Improved by Autoregressively Sampling the Latent Variable?

Currently, the latent predictor samples the indices of the next latent variable independently. This might be disadvantageous because conditional dependencies between the indices, which correspond to certain areas of the video frames, are ignored. In this experiment the world model predicts the next latent variable autoregressively. For this purpose, we change the architecture of the dynamics network to use a conditional PixelCNN with three gated convolutional layers [24] that is conditioned on the output of the convolutional LSTM. We reduce the convolutional LSTM to one cell, and also apply no frame stacking. We only have preliminary results for this experiment, shown in Table 8, since the training time of a single run increases from 12 hours to 70 hours on an Nvidia A100 GPU. Most of the time is spent on simulating experience, because this is where the (sequential) autoregressive sampling is required. We did not adjust the hyperparameters, so they are not fine-tuned for this architecture. We see a minor improvement in two of the four environments, however the results are not convincing considering the additional training time.

Table 8 Mean final episode reward of our autoregressive world model (without frame stacking), averaged over three training runs, compared with the default models which independently samples the indices of the next latent variable

Game	Auto-regressive	Independent (1 frame)	Independent (4 frames)
BankHeist	**259.7**	249.3	101.2
KungFuMaster	7259.4	8521.2	**11903.1**
MsPacman	**696.3**	668.3	652.2
Seaquest	479.4	**531.5**	458.1



(a) Pong (b) Breakout

Fig. 14 Original game frames (top row), the encoded 6×6 discrete latent variables (middle), and the reconstructions from the VQ-VAE (bottom)

Q5: Does a Smaller Receptive Field Improve the Performance, Since it Simplifies the Task of the Dynamics Model?

In this experiment we narrow the receptive field of the latent representations by changing the kernel sizes, strides, and paddings of the convolutional layers in the observation encoder and decoder. As a consequence, each entry $z_{i,j}$ of the latent representation matrix corresponds to a 16×16 image patch in the 96×96 frame without overlap. The results that we get on a small scale indicate that this does not improve the performance, but it allows for a more interpretable visualization of the latent space. In Fig. 14 we can observe that the individual objects like the balls or the paddles are assigned a specific index in the latent representation. However, depending on its location inside of an image patch, the same object may get assigned varying indices. When an object is located between two patches, both corresponding indices are changed.

Summary

Our ablation study shows that the two-dimensional discrete latent representation is essential for the good performance of our approach. We have also seen that the latent representation is more efficient than reconstructed frames.

5 Related Work

5.1 World Models [6]

Modeling environments with complex visual observations is a hard task. The complexity can be alleviated by encoding observations into low-dimensional representations and acting

completely in this latent space. [6] train a variational autoencoder, which they call the "vision" component. It extracts information of the observation at the current time step. An LSTM combined with a mixture density network [4] predicts the latent variable at the next time step stochastically. They call it the "memory" component, which can accumulate information over multiple time steps. The policy is conditioned on latent variables instead of observations, which enables them to simulate experience in latent space, without decoding back into pixel space. This is more efficient and can reduce the effect of accumulating errors [6]. In Sect. 3.1 we describe in more detail how to integrate latent variables and recurrent states into the world model.

They successfully evaluate their architectures on two environments, but it involves some manual fine-tuning of the policy. The policy is optimized with an evolution strategy, which is not suitable for bigger networks. Additionally, their training procedure is non-iterative, i.e., they randomly collect real experience only once and then train the world model and the policy. This implies that no new experience can be obtained with the improved policies and that a random policy has to ensure sufficient exploration, which makes the approach inappropriate for more complex environments.

5.2 Simulated Policy Learning [11]

[11] succesfully train a world model on a subset of games of the Arcade Learning Environment, while restricting each game to 400 K frames. First, they introduce an iterative training procedure (SimPLe) that alternates between collecting real experience, training the world model, and improving the policy with simulated experience from the world model. Second, a video prediction model similar to SV2P [2] predicts the next frame conditioned on the current frame, while the input action is incorporated in the decoding process. The latent variables are discretized into a bit vector and an LSTM predicts it autoregressively during inference time (similar to the PixelCNN of a VQ-VAE). Although a latent space is used, simulating new experience is relatively expensive, as decoding into pixel space is necessary at each time step. They train the policy with the model-free PPO algorithm [19], conditioned on the real or predicted frames, and get excellent results in a lot of Atari games considering the low number of interactions.

5.3 DreamerV1 [7] and DreamerV2 [8]

[7, 8] train a world model with meaningful latent representations, so that the agent can operate directly on the latent variables. Similar to [6], a variational autoencoder encodes observations into latent space and a recurrent neural network predicts the next time step. The authors cleverly split the state into a stochastic and a deterministic part. One of the main improvements of DreamerV2 over the world model of DreamerV1 is the discretization of the latent space, as it uses a vector of categorical variables instead of Gaussian variables. The probabilities of the categorical distributions are computed using the Softmax function.

In contrast to this, we base our world model on a VQ-VAE [25] and thus discretize the latent space using vector quantization. This means that we use a dictionary of embedding vectors to discretize the feature representations via a nearest neighbor look-up. These embedding vectors are also learned during training and can contain richer information than class scores. Furthermore, with our approach we can easily increase the number of discrete categories without increasing the complexity of the encoder and the decoder, since it only affects the number of embedding vectors. Another difference of our approach to DreamerV2 is that the

latent space is a 2-dimensional matrix instead of a 1-dimensional vector, since we do not flatten the output of the encoder. Therefore, we employ a convolutional LSTM instead of a gated recurrent unit (GRU; 5) to compute the recurrent states.

They show that their agent beats model-free algorithms in many Atari games after 50 M interactions. We attempt to learn as much as possible from only 100 K interactions. Unfortunately, DreamerV2 was developed concurrently with our work.

5.4 MuZero [18]

One of the first successful applications of planning at decision-time to visually complex environments like Atari 2600 games has been accomplished by the MuZero algorithm. Without prior knowledge of the environment's dynamics, a trained world model looks ahead via Monte-Carlo tree search. Their results are remarkable, but at the cost of long training times and large models.

6 Conclusion

We have successfully shown that it is possible to build a world model with much fewer parameters than previous model-based reinforcement learning approaches, while performing comparably. To achieve this, we employ a VQ-VAE with a two-dimensional discrete latent space, a convolutional LSTM as the dynamics network, and a convolutional neural network for parameterizing the policy. This setup is able to effectively encode the complex visual observations of the Atari environments into representations, on which a policy is learned. Furthermore, acting entirely in latent space speeds up training, since we avoid the computational costs of decoding high-dimensional frames. Detailed experiments including several ablation studies confirm our design choices.

In the future, we would like to predict the end of episodes, instead of terminating episodes after a fixed number of time steps (since the latter prevents the agent from learning beyond this time horizon). Similar to DreamerV2 [8], one could learn a discount factor that down-weights the rewards instead of actually ending the episodes abruptly. Moreover, we train the encoder and the dynamics model separately, since this leads to more control over the individual components and makes hyperparameter tuning easier. However, the prediction of rewards could potentially be improved by incorporating it into the training of the encoder.

Another important question is how to improve exploration in order to get higher sample efficiency. Currently, we increase the randomness of our policy in Eq. 19 to introduce exploration. Other ideas could incorporate knowledge of the world model, e.g., by measuring the novelty of observations. [17] train a dynamics model and use the prediction error as an intrinsic reward. Such an approach would be a good match to our world model since it is inherently able to compute the likelihood of the next observations. However, we will leave that to future work.

Author Contributions Conceptualization, JR and SH; Methodology, JR; Software, JR; Validation, JR, TU, and SH; Investigation, JR; Writing—Original Draft, JR; Writing—Review & Editing, JR, TU, and SH; Visualization, JR and TU; Supervision, SH.

Funding Open Access funding enabled and organized by Projekt DEAL. Not applicable.

Data Availability Not applicable.

Code Availability Code is publicly available at https://github.com/jrobine/smaller-world-models

Declarations

Conflicts of interest The authors declare that they have no conflicts of interest.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

1. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization
2. Babaeizadeh M, Finn C, Erhan D, Campbell RH, Levine S (2017) Stochastic variational video prediction, Dumitru Erhan
3. Bellemare MG, Naddaf Y, Veness J, Bowling M (2013) The arcade learning environment: an evaluation platform for general agents. J Artif Intell Res 47:253–279. https://doi.org/10.1613/jair.3912
4. Bishop CM (1994) Mixture density networks. Aston University, Birmingham
5. Cho K, Van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y.(2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation
6. Ha D, Schmidhuber J (2018) Recurrent world models facilitate policy evolution. Advances in neural information processing systems, 31
7. Hafner D, Lillicrap T, Ba J, Norouzi M (2020) Learning behaviors by latent imagination, Dream to control
8. Hafner D, Lillicrap T, Norouzi M, Ba J (2020b) Mastering atari with discrete world models
9. Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S, Lerchner A (2017) beta-vae: Learning basic visual concepts with a constrained variational framework. In: International conference on learning representations
10. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift
11. Kaiser L, Babaeizadeh M, Milos P, Osinski B, Campbell RH, Czechowski K, Erhan D, Finn C, Kozakowski P, Levine S, Mohiuddin A (2020) Model based reinforcement learning for atari. In: International conference on learning representations
12. Kingma DP, Ba J (2017) Adam: A method for stochastic optimization
13. Kingma DP, Welling M (2014) Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, Editors, 2nd International conference on learning representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings
14. Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. In: ICML workshop on deep learning for audio, Speech and Language Processing
15. Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Rusu Andrei A, Veness Joel, Bellemare Marc G, Graves Alex, Riedmiller Martin, Fidjeland Andreas K, Ostrovski Georg, Petersen Stig, Beattie Charles, Sadik Amir, Antonoglou Ioannis, King Helen, Kumaran Dharshan, Wierstra Daan, Legg Shane, Hassabis Demis (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/10.1038/nature14236
16. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016). Asynchronous methods for deep reinforcement learning, Lillicrap, Tim Harley

17. Pathak D, Agrawal P, Efros AA, Darrell T (2017) Curiosity-driven exploration by self-supervised prediction
18. Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, Guez A, Lockhart E, Hassabis D, Graepel T, Lillicrap T (2020) Mastering atari, go, chess and shogi by planning with a learned model
19. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms, Prafulla Dhariwal
20. Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2018) High-dimensional continuous control using generalized advantage estimation
21. Schwarzer M, Anand A, Goel R, Hjelm RD, Courville A, Bachman P (2021) Data-efficient reinforcement learning with self-predictive representations. In: 9th international conference on learning representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021
22. Shi X, Chen Z, Wang H, Yeung DY, Wong WK, Woo WC (2015) Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 802–810. Curran Associates, Inc
23. Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction, 2nd edn. The MIT Press, Cambridge, MA, USA
24. Van Den Oord A, Kalchbrenner N, Kavukcuoglu K (2016) Pixel recurrent neural networks
25. Van Den Oord A, Vinyals O (2017) Neural discrete representation learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 6306–6315. Curran Associates, Inc
26. Wang Y, Long M, Wang J, Gao Z, Yu PS (2017) Predrnn: Recurrent neural networks for predictive learning using spatiotemporal LSTMS. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 879–888. Curran Associates, Inc
27. Wang Y, Gao Z, Long M, Wang J, Philip SY (2018) PredRNN++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. In : International conference on machine learning, PMLR, pp 5123–5132