Project AI: Representation learning for model-based reinforcement learning in Minecraft

Tom F. Lieberum
(Student ID: 13253042)

1 Introduction

Reinforcement learning (RL) agents typically require large amounts of training data to achieve good performance, even by standards of contemporary deep learning models, e.g Agent57 (Badia et al., 2020), which required on the order of 90 billion transitions to outperform the human baseline on all Atari games. One popular approach to reduce sample complexity in RL is so-called model-based reinforcement learning (MBRL) (Deisenroth et al., 2013). In MBRL, the goal is to learn a model of the environment, which is typically composed of a representation and a transition or dynamics model. This model can be trained efficiently, e.g. by pretraining it on the experience of other agents, thus reducing the amount of online data required for solving the RL problem. The dynamics model can then be used either *explicitly*, in the form of planning (Hafner et al., 2019; Ozair et al., 2021), or as an additional data source (Ha and Schmidhuber, 2018; Hafner et al., 2020, 2021) – or *implicitly*, using the internal state of the dynamics model in downstream tasks like Q-learning (Ha and Schmidhuber, 2018).

To facilitate the learning of the model, one can employ techniques from representation learning. In this work, we investigate several methods to learn such an internal representation in the 3D MineRLNavigateDenseVectorObf environment. We compare different visual representation models, namely Variational Autoencoders (VAE) (Kingma and Welling, 2014), Vector-Quantized VAEs (VQ-VAE) (van den Oord et al., 2018) and regular convolutional neural networks trained together with the RL model. For the dynamics model, we experiment with the MDN-RNN proposed by Ha and Schmidhuber (2018) and compare it to using no dynamics model at all. We restrict ourselves to an implicit approach, i.e. using the learned world model representations in downstream tasks, because we did not have the time or computational resources to also experiment with explicit approaches, like planning or data generation.

1.1 RL Setting

In this report we exclusively focus on the MineRL Navigate Dense vector-obfuscated environment.

We can model the environment as a partially-observable Markov Decision Process (POMDP) given by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, \Omega, \gamma)$, where we have defined

- $\mathcal{S}$: (latent) state space
- $\mathcal{A}$: action space
- $T : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$: transition function between states, conditioned on action
- $\mathcal{O}$: observation space
- $\Omega : \mathcal{S} \to \mathcal{O}$: observation function
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: reward function
- $\gamma \in [0, 1]$: discount factor

where we have used $\Delta\mathcal{S}$ to denote the set of all probability distributions over $\mathcal{S}$.
The agent receives two types of observations, an image of its first person point of view and a (64-dimensional) continuous vector:

$$\mathcal{O} = \mathcal{O}_{pov} \times \mathcal{O}_{vec}$$

The continuous vector represents a compass pointing towards the goal location that the agent tries to reach.
The goal of MBRL in the language of the above POMDP is then to learn a representational mapping from the observation space to some latent space $\mathcal{Z}$

$$f_{repr} : \mathcal{O} \to \mathcal{Z},$$

and a transition mapping

$$f_{trans} : \mathcal{Z}^t \times \mathcal{A}^t \to \Delta\mathcal{Z}$$

which maps a history of latent-state-action pairs to a distribution over next latent states. $\Delta\mathcal{Z}$ denotes the space of probability distributions over $\mathcal{Z}$. One could think of this as learning a mapping from the POMDP to a latent MDP as well as learning the latent MDP's dynamics. Note that it is necessary to condition the transition function on the whole history in order to retain the Markov property, due to the partial observability of the true POMDP. This approach was for example explicitly pursued by van der Pol et al. (2020), albeit in an MDP setting. In the present report however, we leave this correspondence to a latent MDP implicit. We note that while theoretically, f_{trans} could depend explicitly on the whole history, in practice it usually encodes the history in some form of memory state, e.g. the hidden state of a recurrent neural network, and only updates this state given the most recent state and action.

2 Methods

Due to the small dimensionality of $\mathcal{O}_{vec}$, we simply take $\mathcal{Z} \coloneqq \mathcal{Z}_{pov} \times \mathcal{O}_{vec}$. In Section 2.1, we will discuss methods to learn $f_{repr} : \mathcal{O}_{pov} \to \mathcal{Z}_{pov}$. Section 2.2 focuses on ways to learn f_{trans}. The RL algorithms used are explained in Section 2.3.

2.1 Representation Learning

2.1.1 Variational Autoencoder

Variational Autoencoders (VAE) (Kingma and Welling, 2014) are continuous latent variable models which learn an approximate posterior $q_\phi(z|o)$ over the latent variable z, given some observation o and the corresponding approximate likelihood $p_\theta(o|z)$. Their loss function is composed of a reconstruction loss and a regularization loss which pushes the approximate posterior to be close to a diagonal, zero-mean, unit-variance Gaussian:

$$\begin{aligned}\mathcal{L}_{\text{VAE}}(o;\phi,\theta) = &\mathbb{E}_{q_\phi(z|o)}\Big[-\log p_\theta(o|z)\Big] \\ &+ \beta \cdot D_{KL}(q_\phi(z|o)||p(z)),\end{aligned} \tag{1}$$

where $p(z) = \mathcal{N}(z|0,\mathbf{1})$. β is a hyperparameter which controls the regularization strength of the prior $p(z)$ (Higgins et al., 2017).

Typically, the approximate posterior $q_\phi(z|o)$ is modeled by a deep convolutional network parameterizing a diagonal Gaussian distribution. Similarly, the approximate likelihood $p_\theta(o|z)$ is modeled by a deep convolutional network. Those networks are also referred to as the encoder and decoder networks, respectively. The functional form of a Gaussian for $q_\phi(z|o)$ enables the exact computation of the KL divergence term in the loss.

In our particular POMDP setting, the VAE encoder maps an observed POV image o to a distribution over latent vectors $q_\phi(z|o)$. Then, we draw a single sample z and let the decoder reconstruct it to approximate the expected value term in Equation (1):

1. $f_{enc}(\phi) : \mathcal{O}_{pov} \to \Delta\mathcal{Z}_{vae} : o \mapsto q_\phi(z|o)$
2. $z \sim q_\phi(z|o)$
3. $f_{dec}(\theta) : \mathcal{Z}_{vae} \to \mathcal{O}_{pov} : z \mapsto \hat{o} \equiv p_\theta(\hat{o}|z)$

where $\Delta\mathcal{Z}_{vae}$ denotes the space of all probability distributions over $\mathcal{Z}_{vae}$. The reparameterization trick enables us to backpropagate through the sampling process in 2.

The loss is then computed as

$$\mathcal{L}_{\text{VAE}}(o;\theta,\phi) = \text{MSE}(o,\hat{o}) + \beta \cdot D_{KL}(q_\phi(z|o)||p(z))$$

where MSE is the mean squared error in pixel space. Note that while the choice of MSE is commonly used by practitioners, it does implicitly assume a Gaussian with unit variance and mean $\hat{o}$ as the output distribution $p_\theta(o|\hat{z})$, which is usually not theoretically justified – it does however yield good results in practice.

2.1.2 Vector-Quantized VAE

Recently, discrete latent variable models like for example Vector-Quantized VAEs (VQ-VAE) (van den Oord et al., 2018), have been applied with significant success in model-based reinforcement learning in Atari settings and even in 3D environments (Hafner et al., 2021; Ozair et al., 2021).

The core components of a VQ-VAE are the encoder function f_{enc}, N codebooks $\{c_i^n\}_{i=1}^K$ of K D-dimensional embedding vectors c_i^n, representing N different categorical variables of cardinality K, and the decoder function f_{dec}. The encoding function maps an input image to categorical distributions, one for each variable:

$$f_{enc} : \mathcal{O}_{pov} \to \bigtimes_{n=1}^{N} \Delta\mathcal{Z}_n$$
$$: o \mapsto \prod_{n=1}^{N} q_\phi(z_n|o),$$

with $\mathcal{Z}_n = \{c_i^n\}_{i=1}^K$.
From this latent distribution, we sample a codebook vector for each variable via the Gumbel-Softmax trick (Jang et al., 2017). During (pre-) training, this vector is a convex combination of the codebook vectors according to the probabilities given by f_{enc}, but during inference a hard sample is drawn. The sampled codebook vectors are decoded by f_{dec} into $\hat{o}$ which is scored via the MSE against the input image o. In full generality, the loss function is the same as Equation (1). Assuming a uniform prior $p(z)$ over codebook vectors, we obtain

$$\mathcal{L}_{\text{VQ}}(o;\phi,\theta) = \text{MSE}(o,\hat{o}) + \beta\left(\frac{1}{N}\sum_{n=1}^{N} -\mathcal{H}(q_\phi(z_n|o)) + \log K\right),$$

where $\mathcal{H}(q_\phi(z_n|o))$ is the entropy of $q_\phi(z_n|o)$.

2.2 Learning Dynamics

2.2.1 MDN-RNN

As dynamics model we investigate the mixture-density network with recurrent neural network (MDN-RNN) introduced by Ha and Schmidhuber (2018). The MDN-RNN we use is comprised of a single-layer gated recurrent unit (GRU) (Cho et al., 2014) and a mixture-density network (MDN) which we will explain now in that order.

GRU: The GRU computes its hidden states h_t at time t:

$$h_{t+1} = f_{\text{GRU}}(h_t, z_t, a_t),$$

where $z_t \in \mathcal{Z}$ is the latent output of the visual representation model at time t concatenated with the vector observation at time t, and a_t denotes the action at time t. For the exact definition of the update function f_{GRU} we refer to Cho et al. (2014).

MDN: The MDN is a single linear layer which computes the parameters ψ_k of K mixture component distributions, as well as the corresponding mixture coefficients π_k, resulting in a mixture of distributions over the next latent state:

$$\hat{p}(z_{t+1}|h_{t+1},\{\psi_k\},\{\pi_k\}) = \sum_{k=1}^{K} \pi_k p(z_{t+1}|h_{t+1},\psi_k)$$

In the case of a VAE, the individual components are diagonal Gaussians parameterized by the ψ_k, whereas for VQ-VAEs, they are categorical distributions with their class probabilities given by the ψ_k.
In addition to the ψ_k and π_k, the MDN also computes a direct prediction of the next state's $\mathcal{O}_{vec}$ part and is scored according to the MSE between this prediction and the ground truth.

2.3 Reinforcement Learning

Since we investigate methods for reducing the amount of online samples required, we use an RL method which makes efficient use of the provided off-policy samples, namely Deep Q-learning from Demonstrations (DQfD) (Hester et al., 2017). This report focuses on the effect of different models for MBRL and as such we are not investigating/comparing multiple RL schemes.
DQfD combines Deep Double Q-Network learning (Mnih et al., 2013; van Hasselt et al., 2015) with prioritized replay (Schaul et al., 2016) and additionally makes use of provided expert demonstrations. In DQfD, a deep Q network is trained via the loss function

$$J(Q) = J_{\text{TD}(1)}(Q) + J_{\text{TD}(n)}(Q) + J_E(Q)$$

$J_{\text{TD}(1)}(Q)$ and $J_{\text{TD}(n)}(Q)$ are the 1-step and n-step squared TD error respectively:

$$\text{TD}(k) = \sum_{i=0}^{k-1} \gamma^i r_i + \gamma^k \max_{a'} Q(s_k, a') - Q(s_0, a)$$

To reduce overestimation of the learned Q values, the final term in the TD error is evaluated using a target network that is a copy of the original Q network and is only updated every N_T steps.
$J_E(Q)$ is the large margin classification loss introduced by Hester et al. (2017), and ensures that actions that are not taken by the expert will receive a lower Q value than those actions which are present in the expert demonstrations:

$$J_E(Q) = \max_{a\in\mathcal{A}} [Q(s,a) + \ell(a_E, a)] - Q(s, a_E),$$

where a_E is the expert action and $\ell(a_E, a)$ is 0 when $a = a_E$ and equal to a hyperparameter $\alpha > 0$ otherwise. This is necessary because otherwise actions which are not present in the off-policy dataset could receive unreasonably large values, since they would not be constrained by the real reward structure of the environment.
Training in DQfD is split in two phases. First, in the pretraining phase, the network is pretrained for k steps on the expert demonstrations alone.
Then, the network is trained as described in Hester et al. (2017):

1. Sample a trajectory using ϵ-greedy policy induced by Q.
   (a) Add trajectory to agent replay buffer.
   (b) Overwrite old transitions if the agent replay buffer is full.
2. Perform M minibatch updates against loss function $J(Q)$, by sampling from the combined expert-agent replay buffer via prioritized experience replay (Schaul et al., 2016).
3. Update the target network.
4. Repeat until N_o online transitions have been sampled.

The replay buffer contains the expert demonstrations and the transitions collected by the agent. The expert demonstrations always remain within the buffer. Once the buffer limit is reached, new incoming agent-collected transitions will override the oldest agent-collected transitions. This ensures that the agent will always re-encounter expert transitions.
In contrast to (Hester et al., 2017), we sample a complete trajectory and then perform M updates instead of updating once after every online transition, this was necessary for adequate sampling speed.

3 Experiments and Results

Due to limited computational resources, we perform all runs that are part of the pretraining only on a single seed. The reasoning is that the variance during pretraining is relatively small compared to the variance present in interactions with the procedurally generated 3D environment, so that the uncertainty of the results is most likely dominated by the online part of DQfD. We train the pretrained models in the online setting for three different seeds.

3.1 Environment and Demonstration Data

The MineRLNavigateDenseVectorObf environment is a 3D environment in which an agent tries to reach a certain goal coordinate. The agent receives two types of observations, a 64×64 RGB image of its current point-of-view, and a 64-dim. vector with its entries between -1 and 1, which is an obfuscated representation of a compass pointing towards the goal destination. It receives a small positive reward whenever it moves towards the goal and a small negative reward when it goes further away from it. Once it reaches the goal point, it receives 100 reward and the episode ends. The episode also ends if the agent dies (e.g. by drowning) in which case the agent does not receive any reward upon termination.
The actions in this environment are 64-dim. continuous vectors with values between -1 and 1 in each dimension. Internally, each vector is mapped onto discrete actions by the environment handler. This provision ensures that no hard coding or action shaping is undertaken. These are the 'research track' restrictions of the MineRL Diamond competition (Guss et al., 2021), which inspired this report. We follow the baseline of the aforementioned competition and use the k-means algorithm to cluster the action vectors into 1000 discrete actions, to enable deep Q learning[1]. The clustering is performed on the expert demonstration data and is fixed going forward from there.
We chose this particular environment for the present report, because its dense reward structure allows for faster iteration and evaluation of different methods, while still providing the challenge of learning to navigate a 3D environment from first-person-view pixel inputs.
The environment comes with a dataset of 194 episodes of human gameplay, containing roughly between 500 and 5000 transitions per episode. This demonstration data is used to pretrain the world model components as well as providing an initial pretraining for the Q-network.

3.2 Representation Learning

3.2.1 Hyperparameters for VQ-VAE

The exact architecture is described in Appendix A.1 and largely follows Ha and Schmidhuber (2018). Important hyperparameters of the VQ-VAE architecture are the number of variables, the size of the

[1] https://github.com/minerllabs/baselines: We chose 1000 instead of 150 actions to make sure to have less action overlap.

codebook for each variable (i.e. its cardinality) and the dimension of the codebook vectors. We did not have the computational resource for an exhaustive sweep. We were however able to test different hyperparameter settings within a reasonable interval. The exact search space and results are shown in Appendix B.1. However, the variance of the loss dominated the difference between parameter settings. Thus, we stuck with a setting of 32 variables with 32 32-dimensional codebook vectors, which seemed to provide a good trade-off between model size and performance.

3.2.2 VAE vs. VQ-VAE

The latent space of the VQ-VAE consists of 32 discrete variables which can each take on 32 different 32-dim. values, i.e. $N = K = D = 32$. The latent space of the VAE is modeled by a 512-dim. diagonal Gaussian whose parameters are given by the encoder network.

We aimed at giving them roughly the same number of parameters for their respective encoding models (1.7M each) and similar architecture, although a small degree of difference remains. Namely, the latent dimension of the VAE is 512, whereas for the VQ-VAE it is $32 * 32 = 1024$. The VAE does model a continuous latent space which might compensate for the lower latent dimension. We also trained the VAE with a latent dimension of 1024 but could not detect a meaningful difference, so we opted to keep the parameter count as close as possible instead. Both models were trained via the Adam optimizer, with a cosine-annealed latent loss parameter β. Each model was trained for thirty epochs on the same $\sim$ 230K images from the MineRLNavigateDenseVectorObf-v0 expert data. More hyperparameters can be found in Appendices A.1 and A.2.

The latent losses of the two models are fundamentally not comparable, since the latent distributions are continuous Gaussian and a discrete distribution respectively. We can, however, at least compare the reconstruction loss, which is shown in Figure 1. Example reconstructions are shown in Figure 2. We see that the VQ-VAE learns slightly better reconstructions, which is corroborated by the example reconstructions in Figure 2, in which we see slightly more details in the VQ-VAE reconstruction. In Section 3.4, we investigate how both models perform in downstream RL related tasks.

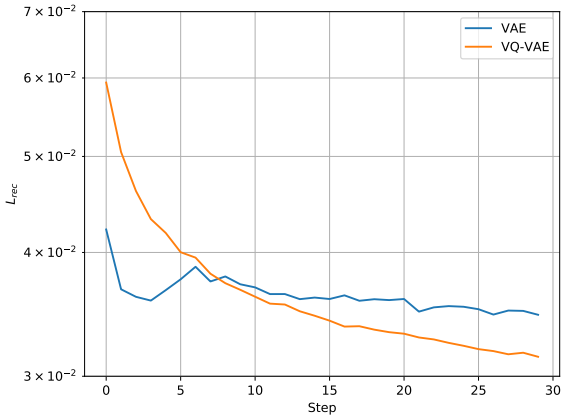


Figure 1: Validation reconstruction loss $\text{MSE}(o, \hat{o})$ over the course of thirty epochs.

3.3 Dynamics Model

3.3.1 MDN-RNN

We train the MDN-RNN to predict the parameters of the next latent state's distribution. The MDN-RNN is trained for 100 epochs on the expert trajectories, since we found that this was necessary for satisfactory predictions. It is scored via the KL divergence between its predicted distribution $\hat{p}(z_{t+1}|h_{t+1}, \{\psi_k\}, \{\pi_k\})$ and the actual next latent state distribution $q_\phi(z_{t+1}|o_{t+1})$ of the pretrained visual model. We note that the losses are not directly comparable between the VAE and VQ-VAE setting since the latent distributions are fundamentally different. However we can compare the MSE loss of the $\mathcal{O}_{vec}$ part of the observation. From the description in prior work (Hafner et al., 2021; Ozair et al., 2021), it was unclear whether the used dynamics models were using the learned codebook vectors of the VQ-VAE or a multi-hot representation to indicate the active variables instead. We trained the model in either setting and could detect a slight difference in predictive quality of the multi-hot setting over the learned-feature setting, shown in Figure 3. Thus, going forward, we will be using the multi-hot setting.

In Figure 4, we display the one-step prediction error of the vector observation for the MDN-RNN with a VAE vs. VQ-VAE visual model.

To better judge quality of the models we show example predictions in Figure 5. We observe that while the one-step predictions are of decent quality, the n-step extrapolations quickly deteriorate over time. This is expected, since small errors in the one-step predictions compound when extrapolating. This could be ameliorated by training the network

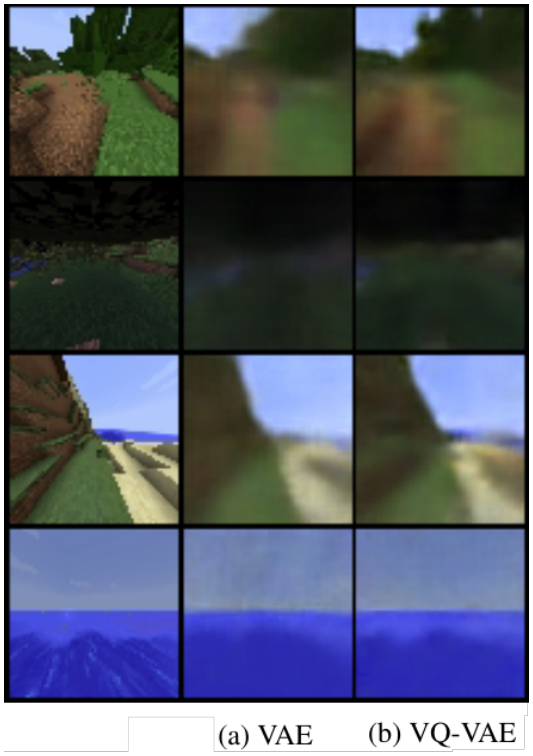


Figure 2: Some final reconstructions of VAE (a) and VQ-VAE (b). Original image is shown in the left column. The VQ-VAE has slightly higher resolution, e.g. the cliffside in row 3, and captures distinct features better, e.g. the light green spot in row 2.

on n-step predictions. However, we did not have sufficient computational resources to perform this experiment.

3.4 Effect on Q-learning

To determine the effect of using a pretrained feature extractor, we train a shallow Q-network on top of the extracted features, where those features are either computed from a freshly initialized CNN which is trained together with the Q-network, the pretrained VQ-VAE or the pretrained VAE. If a dynamics model is used, then the hidden state h_{t-1} is appended to the features z_t computed by the visual model. This way, the Q-network can use the predictive information implicitly encoded in h_{t-1}. All Q-networks are first pretrained on 100 epochs of the available expert data for MineRLNavigateDenseVectorObf-v0. Then, they are trained in the manner outlined in Section 2.3. We choose the following parameters:

- $\epsilon = 0.05$
- Number of updates $M = 200$
- Batch size $= 100$

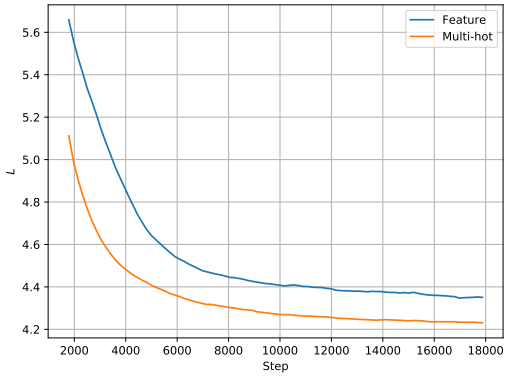


Figure 3: Total validation loss (KL divergence + $\mathcal{O}_{vec}$ reconstruction loss) for MDN-RNN with VQ-VAE model, using either multi-hot latents or the learned feature vectors, computed as moving average over 10 steps. Each step represents one expert episode.

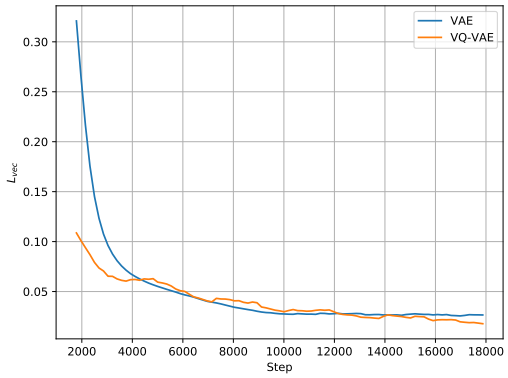
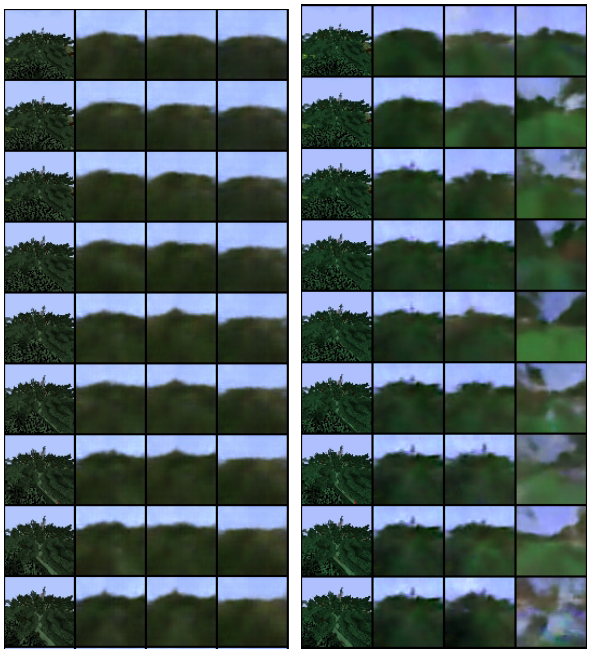


Figure 4: Validation reconstruction loss for $\mathcal{O}_{vec}$ for MDN-RNN with either VAE or VQ-VAE visual model, computed as moving average over 10 steps. Each step represents one expert episode.

- Max. online transitions $N_o = 10^5$

Online episodes are terminated prematurely if they exceed 3000 steps.
All hyperparameters of the DQfD algorithm can also be found in Appendix A.2. The prioritized replay parameters were set to the values used by Hester et al. (2017), as was the margin α. The other parameters were set to reasonable values, but we did not perform a sweep to systematically test different settings.
The total loss $J(Q)$ during online training is shown in Figure 6. A breakdown of the loss components can be found in Appendix B.2. We observe that at least for the loss, using a VQVAE over VAE or CNN and using the MDN-RNN over not using it both confer a significant advantage. The VAE alone does not seem to outperform the CNN. From the loss breakdown it is clear that the advantage



(a) VAE (b) VQ-VAE

Figure 5: From left to right: the original sequence, the direct reconstruction by the visual model, the reconstructed one-step predictions and the reconstructed n-step extrapolations of the dynamics model. Temporal order is from top to bottom.

is driven by the increased n-step prediction performance and the improved classification loss.

To gain a better measure of the models' actual performances we evaluate every model (i.e. three seeds per architecture) on 40 episodes. Episodes are stopped prematurely after 6000 steps, if the model has not reached the goal by then. We average the results and report the mean episode reward together with the estimated standard deviation of the sample mean in Table 1. We choose to report the standard deviation of the sample mean instead of the standard deviation of the reward directly, since that way we can better detect differences between the mean performance of the algorithms in the high-noise environment. In terms of the mean accumulated reward, there is significant difference between using a CNN and using a VAE. Using an MDN-RNNs predictive state seems to make the performance of the VAE worse. The VQ-VAE models are performing the worst, with a slight but not significant improvement when using an MDN-RNN together with a VQ-VAE.

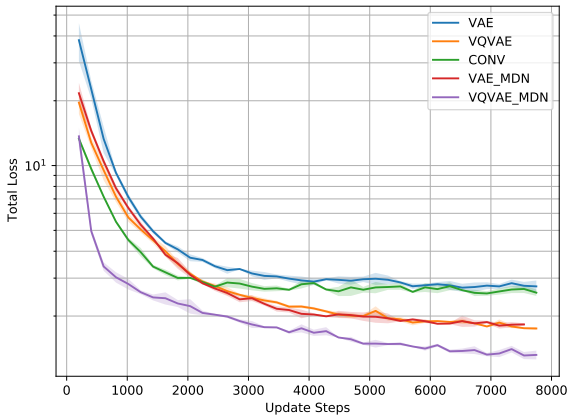


Figure 6: Moving average over 50 steps of the total loss $J(Q)$, averaged over three seeds. Shaded area represents $\pm 1\sigma$.

	CNN	VAE	VQ-VAE
no dyn.	56.0 ± 3.7	59.8 ± 3.4	37.0 ± 3.3
MDN-RNN	–	46.5 ± 4.2	39.7 ± 3.2

Table 1: Average episode reward after completed training of the Q-network on the extracted features z_t (and hidden states h_{t-1} in the MDN-RNN condition). Shown are sample mean and its estimated standard deviation

4 Discussion

4.1 VQ-VAE Hyperparameters

We could not detect a signifcant advantage in terms of reconstruction quality for different settings of the codebook size, number of variables and embedding dimension. Possibly, significant gains could be made for even larger settings then studied in this report, but we could not observe such a trend in our results.

4.2 VAE vs. VQ-VAE

In terms of reconstruction quality, the VQ-VAE seems to outperform the VAE both qualitatively and quantitatively, which is in line with previous work (van den Oord et al., 2018).

4.3 MDN-RNN

Interestingly, we found that using a multi-hot representation of the VQ-VAE output was beneficial to predicting the next latent state, compared to using the learned embeddings.
We could not detect a significant difference in the ability of the MDN-RNN to predict the next vector observation when using either visual model architecture. However there was a notable difference

when extrapolating beyond one-step predictions. While the VAE's predictions was roughly stable and mainly failed to adapt to new changes, the VQ-VAE's prediction quickly deteriorated from the ground truth. This could be related to the different inductive biases of their respective latent spaces, since the VQ-VAE only has a fixed set of variable values to choose from, so choosing a 'wrong' value can result in drastically different reconstruction, whereas the VAE's reconstructions are likely more robust to small errors in the latent state.

4.4 Effects on Q-Learning

As we can see from Figure 6, at least in terms of the total loss $J(Q)$ we were able to confirm the advantage of using a VQ-VAE over using a VAE and of using an MDN-RNN (even if it is only implicitly) over not using a dynamics model. The loss breakdown also suggests that the VQ-VAE and MDN-RNN based models should be better at predicting the n-return which should be most indicative of being able to choose the correct action. This indicates that the learned representations by the world model components are indeed useful for predicting Q-values.

However, the aforementioned trends seem to be reversed when it comes to the actually accumulated mean reward of the respective models which is quite surprising.

We were not able to find a satisfying explanation for this perplexing behavior, especially since the networks are doing well in predicting long-term rewards, indicated by the $J_{\text{TD}(n)}(Q)$ in Figure 9. Speculative possibilities include particularities of the high variance and complex environment combined with overfitting to the expert demonstrations, or wrong settings of the DQfD hyperparameters. We leave it up to future work to investigate this issue more closely.

5 Conclusion

We investigated different representation models together with a dynamics model to facilitate more sample efficient reinforcement learning in the MineRLNavigateDenseVectorObf environment. For Q-learning, We could show the advantage of using vector-quantized representations over continuous ones and the benefits of using a dynamics model. The accumulated reward showed the opposite behavior, favoring continuous representation models or even simple convolutional neural networks, and favoring not using a dynamics model.

6 Acknowledgements

I would like to thank Herke van Hoof and David Kuric for their supervision and guidance. Likewise, I am thankful to Microsoft for supporting me with a Microsoft Azure compute grant for the MineRL Diamond competition and to the University of Amsterdam for lending me access to the Lisa compute cluster for this project.

References

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. 2020. Agent57: Outperforming the atari human benchmark.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. 2013.

William H. Guss, Mario Ynocente Castro, Sam Devlin, Brandon Houghton, Noboru Sean Kuno, Crissman Loomis, Stephanie Milani, Sharada P. Mohanty, Keisuke Nakata, Ruslan Salakhutdinov, John Schulman, Shinya Shiroshita, Nicholay Topin, Avinash Ummadisingu, and Oriol Vinyals. 2021. The minerl 2020 competition on sample efficient reinforcement learning using human priors. *CoRR*, abs/2101.11071.

David Ha and Jürgen Schmidhuber. 2018. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc. `https://worldmodels.github.io`.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. 2020. Dream to control: Learning behaviors by latent imagination.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Learning latent dynamics for planning from pixels.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2021. Mastering atari with discrete world models.

Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep reinforcement learning with double q-learning.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. 2017. Deep q-learning from demonstrations.

Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. 2017. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2018. Neural discrete representation learning.

Sherjil Ozair, Yazhe Li, Ali Razavi, Ioannis Antonoglou, Aäron van den Oord, and Oriol Vinyals. 2021. Vector quantized models for planning.

Elise van der Pol, Thomas N. Kipf, Frans A. Oliehoek, and Max Welling. 2020. Plannable approximations to MDP homomorphisms: Equivariance under actions. *CoRR*, abs/2002.11963.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay.

A Experimental Parameters

A.1 Network Architectures

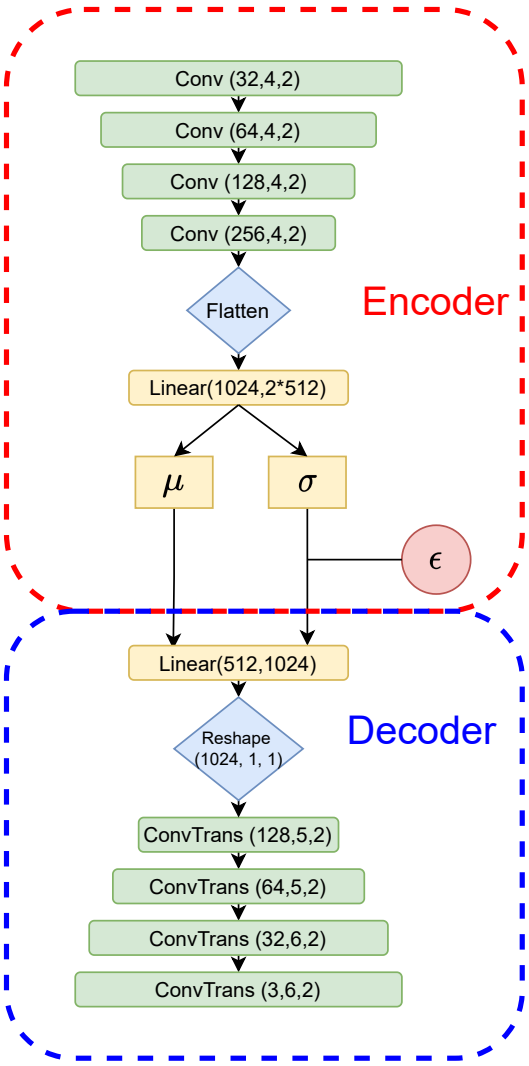


Figure 7: VAE architecture. All Convolutions, except the very last transposed convolution are followed by a ReLU activation function. The linear layer in the decoder is also followed by a ReLU, whereas the linear layer in the encoder does not use an activation function directly. Conv(C, K, S) refers to a convolution with C output channels, kernel size K and stride S.

A.2 Training parameters

- Visual model

  learning rate: 3e-4

  β: cosine annealed from 0 to 5e-4 over 15000 steps

- MDN-RNN

  number of mixture components: 5

  learning rate: 3e-4

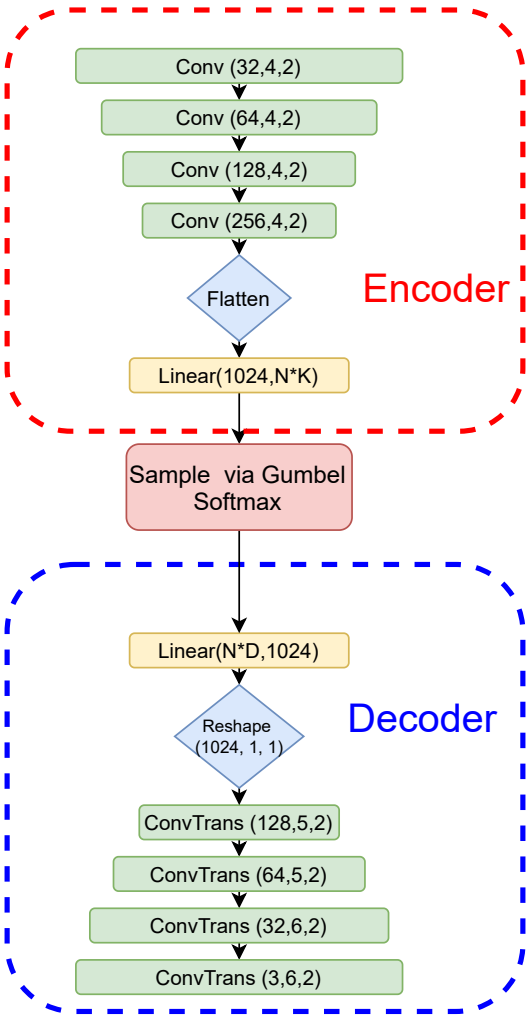


Figure 8: VQVAE architecture. All Convolutions, except the very last transposed convolution are followed by a ReLU activation function. The linear layer in the decoder is also followed by a ReLU, whereas the linear layer in the encoder does not use an activation function directly. Conv(C, K, S) refers to a convolution with C output channels, kernel size K and stride S.

- DQfD

  general parameters

  horizon (n-step): 50

  margin α: 0.8

  γ: 0.99

  target update rate: 100

  learning rate: 3e-4

  online-exclusive parameters:

  max episode length: 3000

  max environment interactions: 10^5

  batch size: 100

  update steps per iteration: 200

agent memory capacity: 50000

ϵ (for exploration): 0.05

prioritized replay parameters:

α_{PER}: 0.4

β_0: 0.6

ϵ_{agent}: 0.001

ϵ_{expert}: 1

B Additional Results

B.1 VQ-VAE Hyperparameter Sweep

We try out all possible combinations for the following parameters:
Embedding dimension $D \in \{32, 64\}$
Number of variables $N \in \{16, 32, 64, 128\}$
Codebook size $K \in \{16, 32, 64, 128\}$

Final results are shown in Tables 2 and 3. The noise within each run was on the order of 0.01, thus dominating the differences seen in the tables below.

N \ K	16	32	64	128
16	0.037	0.032	0.044	0.042
32	0.043	0.034	0.033	0.032
64	0.034	0.029	0.037	0.029
128	0.036	0.034	0.033	0.033

Table 2: VQ-VAE reconstruction loss after 1 epoch of training. N = number of variables, K = number of codebook vectors per variable, dimension of codebook vectors = 32

N \ K	16	32	64	128
16	0.047	0.032	0.036	0.031
32	0.035	0.032	0.036	0.032
64	0.029	0.028	0.023	0.032
128	0.033	0.032	0.038	0.032

Table 3: VQ-VAE reconstruction loss after 1 epoch of training. N = number of variables, K = number of codebook vectors per variable, dimension of codebook vectors = 64

B.2 Loss breakdown for DQfD

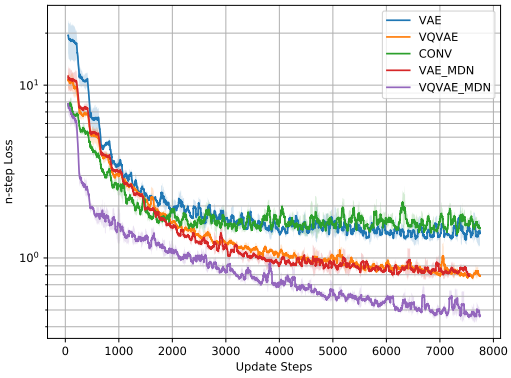


Figure 9: Moving average over 50 steps of $J_{\text{TD}(n)}(Q)$, averaged over 3 seeds. Shaded area represents $\pm 1\sigma$

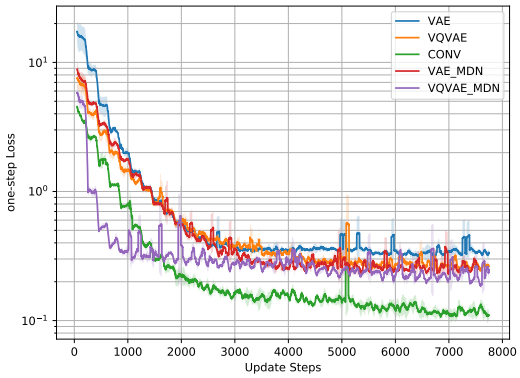


Figure 10: Moving average over 50 steps of $J_{\text{TD}(1)}(Q)$, averaged over 3 seeds. Shaded area represents $\pm 1\sigma$

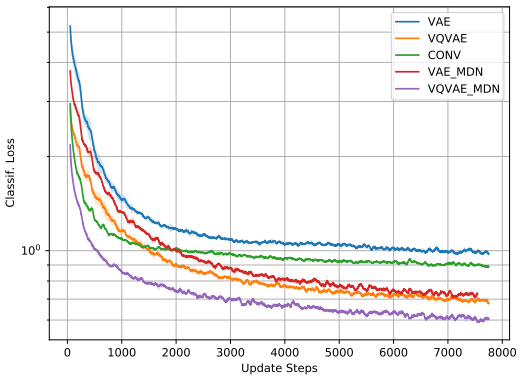


Figure 11: Moving average over 50 steps of $J_E(Q)$, averaged over 3 seeds. Shaded area represents $\pm 1\sigma$