Cars Don't Dream

Peter Keffer*, Fabio Klinge*, Lotta Piefke*
University of Osnabrück, 2022

Abstract

This paper aims to reimplement the DreamerV2 algorithm (Hafner et al. (2022)) and evaluate its performance on the "highway-env" (Bécsi et al. (2018)). Our implementation fails to train properly on the environment. Our analysis finds this is most likely due to an error in our code. Most likely the problem lies in the encoder-decoder architecture. We find DreamerV2 exhibits complications regarding hyperparameter-tuning and complexity.

We lay out our attempts to fix the implementation and show further approaches to find the root cause of the improper training behavior. We deduct that DreamerV2 should be able to train on the "highway-env" with nearly the same hyperparameters and model architecture we implemented. Even though the implementation does not work properly it can help as a guideline, being available under MIT-license, having much better readability than other implementations found online.

1 Introduction

"The great game of science is modeling the real world" Hestenes (1992).

A strategy humans use to grasp the complexity of reality in order to act in it, is to apply models on it. The goal is to reduce the multitude of available information to what is needed for a specific purpose. There exist models to explain complex physical events or how a free-market economy works. Despite these formulated models there is evidence for humans using mental models in order to learn, act and reason. Neuroscientists and other researchers in the field arguing for this are among others: Orban de Xivry and Ethier (2008), Gläscher et al. (2010), Lee et al. (2014) and Khemlani et al. (2014). This indicates decent support for the theory of mental models (Seel (2017)).

So it can be assumed that models are a vital part of human learning. Meanwhile the most popular and developed algorithms in deep reinforcement learning (DRL) are model-free, meaning the agent interacts directly with the environment by trial and error, using rewards to learn behaviour (OpenAI (2020)). This model-free approach should be questioned, as model-based learning seems to be a vital part of human learning (Seel (2017)).

One of the first conceptualisations on models in DRL, so called world models, was made by Wiering and Schmidhuber (1999). The classical notion of a model in model-based DRL is to learn a function predicting state-transitions and rewards (OpenAI (2020)). World models are a latent representation of the environment the actor can act upon to learn behaviour applicable to the actual environment. Ever since their first occurrence, there has been

*These authors contributed equally to this work

development and there are state-of-the-art algorithms using world-models (e.g. McGrath et al. (2021), Hafner et al. (2022) or Racanière et al. (2017)). This paper reproduces and evaluates one of these algorithms, namely DreamerV2, proposed by Hafner et al. (2022). The aim is to understand how world-models currently work and to see how applicable it is to another environment as the one used by Hafner et al. (2022) and to possibly find out why so few world-model approaches are used in current DRL.

One reason why model-based approaches in DRL are so few is the difficulty of learning the ground-truth of an environment (OpenAI (2020)). A model can be derived by rules from the environment like the one used in McGrath et al. (2021). The problem here is the need for an environment with rules defined explicitly, like the game Go. Most environments have their underlying rules not explicitly available which is why the majority of world model algorithms use the experience gained by the agent interacting with the environment in order to learn a model. DreamerV2 utilises the latter similar to Imagination-Augmented Agents like the I2A algorithm (Racanière et al. (2017)). Both learn models to predict future states and their rewards, but have different architectures. While I2A creates multiple models with different computational costs and accuracies, DreamerV2 creates a single world model. DreamerV2 proved to be more successful on the Atari Benchmark compared to other single GPU algorithms (Hafner et al. (2022), OpenAI (2020)).

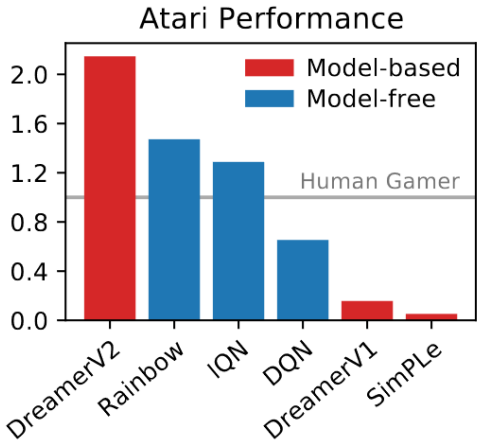


Figure 1: Comparison on Atari Performance between Model-Based and Model-Free Networks Hafner et al. (2022)

DreamerV2 implements model-based learning, where the agent can learn in a world model of the environment. Acting in this model instead of the actual environment is referred to as "dreaming". The models representation of environment states are small categorical representations. This way training happens more efficiently, as compact categorical model state representations are significantly smaller than actual environment states. As DreamerV2 also predicts transitions between these state representations, it allows the agent to plan its behaviour (Hafner et al. (2022)).

2 Methods

2.1 Model

DreamerV2 is an improved version of the Dreamer proposed by Hafner et al. (2020), which from here on will be referred to as DreamerV1.

The term "model-based" tends to create ambiguity towards the term "model". In the following, it denotes state-transition- and reward-functions for a given environment. It should not be confused with the classical notion of a model in Deep Learning which refers to the underlying architecture and logic of the Neural Network.

2.1.1 Architecture

The general architecture of DreamerV2 can be described as a loop of three steps:

1. Let the agent interact with the environment and add the collected experiences as sequences to the experience replay buffer
2. Train the world model on the collected data to be able to create compact state-representations
3. Train the agent with an Actor-Critic-Algorithm on the state-representations generated by the world model

In the following, we are going to explain each of the individual parts of DreamerV2 in order to better understand this training loop.

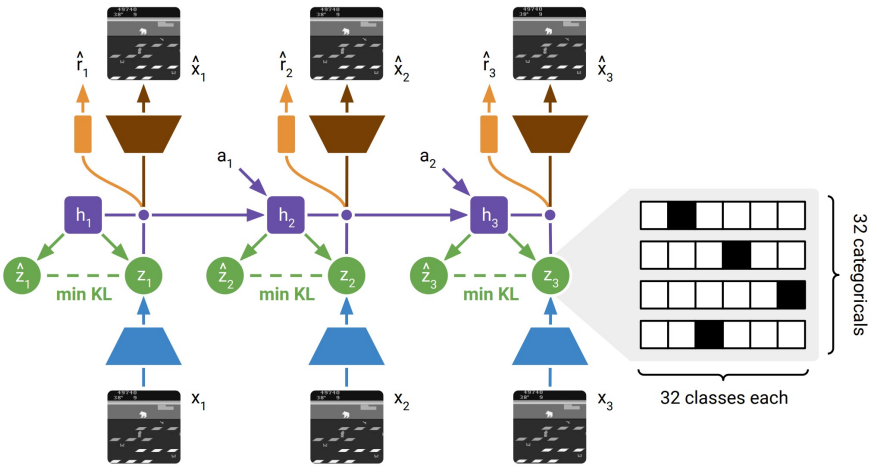


Figure 2: World Model Architecture (Hafner et al. (2022))

World model The world model (Figure 2) consists of a Recurrent State Space Model (RSSM) presented in Hafner et al. (2019). It is a recurrent neural network using a GRU cell to save long term information in the hidden state representing the latent space h_t in 2) shown as violet square. Furthermore it has two models, the representation- and the

transition model, represented by the green circles creating categorical representations z_t and $\hat{z}_t$.

They entail the relevant information of the image input, further referred to as state-representations. The representation model for categorical representations creates the posterior z_t from the image input x_t and the information stored in the hidden state h. As the goal is to model the environments behaviour, the transition predictor aims at creating a prior $\hat{z}_t$. It creates a compressed categorical representation for a state from the environment, by only using the information stored in the latent space h_t. Furthermore the world model includes an image encoder- and decoder network which are represented by the blue and the brown trapezoid. Similar to the typical autoencoder structure, the encoder first embeds the image input $\hat{z}_t$. The decoder is implemented to reconstruct the image, but different to a classical autoencoder, the decoder here recreates the image from our posterior state representation z_t.

Furthermore z_t and h_t are used as input for a reward predictor network (orange rectangle). Also there is a discount predictor network not displayed in Figure 2. Its main objective is to account for the occasion that an episode ends and to weigh the other losses in order to scale their impact down when an episode might have ended. Φ describes the combined parameter vector of the world model.

$$
\text{RSSM} \begin{cases} \text{Recurrent model:} & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Representation model:} & z_t \sim q_\phi(z_t \mid h_t, x_t) \\ \text{Transition predictor:} & \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t) \\ \text{Image predictor:} & \hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t) \\ \text{Reward predictor:} & \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t) \\ \text{Discount predictor:} & \hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t \mid h_t, z_t). \end{cases}
$$

Figure 3: Networks in World Model (Hafner et al. (2022))

The discount predictor is trained by taking the cross-entropy loss of its output and the terminal information for that state. The reward predictor is trained the same way but compares its output to the actual obtained reward. The representation model, image encoder and decoder networks are optimised by backpropagating the cross entropy loss between the image input x and the output of the decoder network.

The transition predictor is optimised by minimising the Kullback–Leibler divergence between z_t and $\hat{z}_t$. KL-divergence is not a metric, but a statistical distance, meaning *KL(a,b)* is different from *KL(b,a)*. *KL(a,b)* says how much distribution *b* differs from *a*. The loss created would thereby move distribution *b* closer to *a*. DreamerV2 makes use of this by introducing *KL balancing*. Here, the KL-loss is distributed from z_t and $\hat{z}_t$ perspectives with learning rate $\alpha = 0.8$ for KL(z_t, $\hat{z}_t$) and 1 - α for KL($\hat{z}_t$, z_t). This way, not only does the prior learn to better approximate, but also the representations of z_t are regularised towards the prior. This supports exploration.

The goal of DreamerV2 is to create small, meaningful categorical representations $\hat{z}$ for image inputs to be able to train multiple agents at once on a single GPU. While DreamerV1 uses samples from a learned Gaussian as embedding, Hafner et al. (2022) proposes that in order to properly entail all the information included in one frame, multiple Gaussian's are needed. This can be seen in Figure 4. In order to tackle this problem, the chosen representations are matrices representing a 32x32 categorical distribution. Each of the 32

categories is represented by a specific class as shown in Figure 2. To backpropagate through samples of categorical distributions, DreamerV2 uses straight-through gradients proposed by Bengio et al. (2013). This way it is possible to propagate the sample in the forward-step but update the probabilities of the distribution in the backward-step, neglecting the sample as it has no gradients.

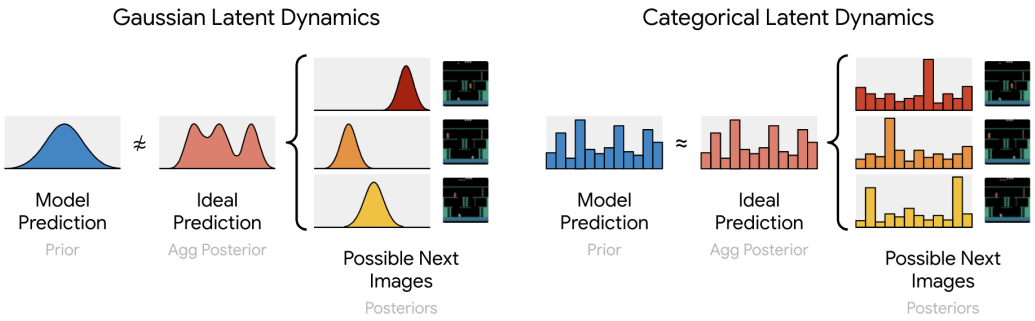


Figure 4: Comparison of Gaussian and Categorical Variables for a Latent Embedding of Picture Inputs s: https://www.googblogs.com/mastering-atari-with-discrete-world-models/

Actor Critic DreamerV2 uses the Advantage Actor Critic (A2C) algorithm. The stochastic actor gets a state as input, and outputs a categorical distribution with probabilities over the action-space. The deterministic critic gets a state and an action as input and outputs a state-action-value. During their training, the parameters of the world model are not changed.

Both actor and critic are solely trained on dreamed sequences of priors, i.e. categorical representations $\hat{z}$. These dreamed sequences have a length of horizon h. The prior for the next step is created by the world model, which uses a sampled action from the stochastic actor network and the current prior. The first step of each sequence is the posterior $z_{t=1}$. From there on, the actor input is $\hat{}z_{t=2,3,...}$, as the agent is supposed to learn its actions on the predictions made by the world model which are dreamed sequences.

For each state, represented by categorical representations, the actor outputs a categorical distribution over the action-space. Normally, a state-action pair would be the input for the critic. In this case, information on the action is represented in our latent space h_t. Therefore, the inputs for the critic, reward- and the discount predictor are a combination of the latent space h_t and the categorical representations. The actor is optimised by making actions more probable that maximize the output of the critic. It uses a combination of classical Reinforce gradients (unbiased, high-variance) and straight-through gradients (biased, low-variance) together with an entropy regularizer to control exploration. Reinforce computes gradients by multiplying probabilities of sampled actions with the values given for them by the critic. To reduce the high variance Reinforce has, the state-value is deducted as a baseline. The critic tries to correctly predict the sum of rewards obtained by the reward predictor for the given latent space h_t and categorical representation pair. The predicted sum of rewards get weighted by the mean of the discount network for each sequence $t=1:H$. The cumulated predictions of the discount network are used to weight both actor and critic loss to softly account for the occurrence of a terminal state.

To leverage the possibility of being able to generate on-policy trajectories using the

world model, the critic is trained using temporal-difference learning with a lambda-target. That means that not only the reward but also future critic outputs in the horizon h are accounted for computing the value. The lambda-target parameter $\lambda = 0.95$ is chosen to increase focus on future rewards to make use of predictions created by the world model. To stabilise training, the actor is trained on a target-network (Mnih et al. (2015)), a copy of the critic network. The critic network is updated every step while the target network copies the critic network every n steps (Hafner et al. (2022)).

2.2 Environment and Data Preprocessing

The environment that we chose is called "highway-env". It uses the OpenAI Gym interface. It is a collection of simple 2D environments, where a self-driving car has to solve different tasks. We decided to use the "highway-v0", a highway setting with three lanes, where the agent is rewarded for driving fast and avoiding crashing with other cars. To achieve faster training, we used a modification called "highway-fast-v0", which has improved speed, but less simulation accuracy.

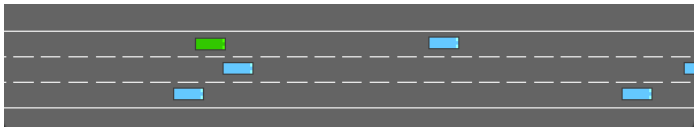
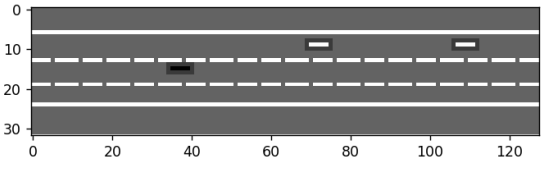


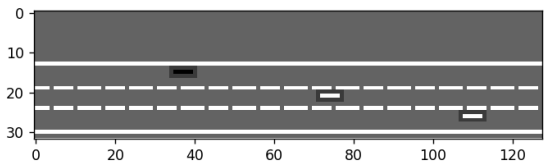
Figure 5: The Rendered "highway-fast-env"

We chose a configuration of our environment, where the observations are a greyscale image of the size $width \cdot height$. We chose an discrete action space, where five actions are possible: Changing to left lane, changing to right lane, accelerating, slowing down, and doing nothing (Bécsi et al. (2018)). The reward R can be calculated as $R(s,a) = a \cdot \frac{v - v_{min}}{v_{max} - v_{min}} - b \cdot collision$. In the equation, v is the velocity of the agent, and v_{max}, v_{min} are the maximum and minimum speed of the car, determined by the environment. The reward is maximized when the car is driving at full velocity and does not crash. The reward is scaled by the two coefficients a, b and then normalized to be in the range of $[0, 1]$ (Leurent (2018)).

We configure our environment to a scaling of 1.5 and an observation size of 128 x 32 pixels. We chose this observation size, because it is as small as possible, while still displaying all lanes and enough cars to the front and rear. Our observation size equals 4096 pixels. This is the same amount as the preprocessed Atari games observations used in the original DreamerV2 paper that have a resolution of 64 x 64 pixels. The greyscale values of the observation were normalised to be in the interval $[-1, 1]$ (Hafner et al. (2022)).



(a) An Exemplary Greyscale Observation.



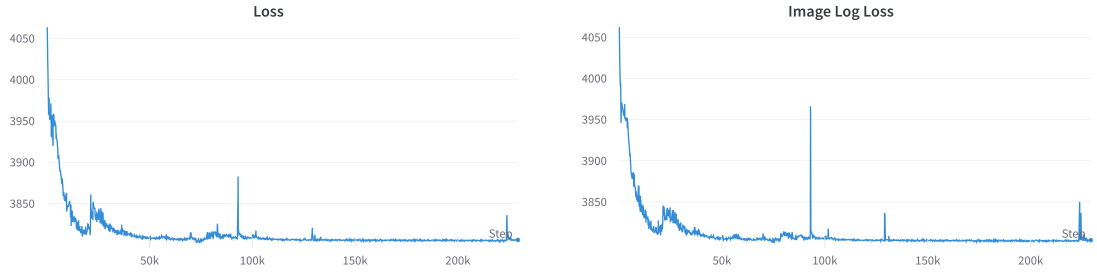
(b) A Second Exemplary Observation.

One challenge of the highway-environment, compared to the Atari Games, is that the

camera is moving with the agent. So the position of the lanes and other cars change relative to the agents car. This can be seen in Figure 6a and Figure 6b. The agent is on the same pixel position, but the road marks are not.

3 Results

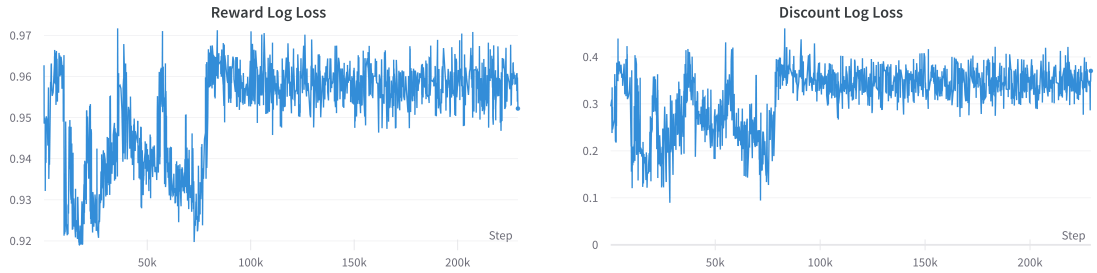
To assess the performance of our DreamerV2 algorithm, we have a look at our losses and at the score our agent achieves. Our overall loss is calculated by summing over the individual losses. Those are the image loss, reward and discount prediction loss, actor and critic loss, and KL balance loss. The overall loss (Figure 7a) is highly dependent on the image loss (Figure 7b). Both seem to stagnate around a value of 3800. This behavior could be observed over all training runs we conducted.



(a) Overall Loss (b) Image Loss

Figure 7: Overall and Image Loss

The reward and the discount predictor loss (Figure 8) are oscillating downwards in the beginning, but start to stagnate together with actor and critic loss.



(a) Reward Prediction Loss (b) Discount Prediction Loss

Figure 8: Reward and Discount Prediction Loss

The KL loss (Figure 9) seems to decrease until it suddenly gets near zero and stagnates there. We could observe this behavior in multiple training runs, but in slightly different manifestations. Sometimes it spikes again, after a long time of being close to zero.

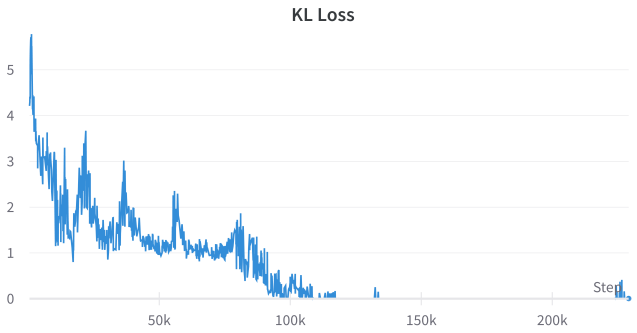
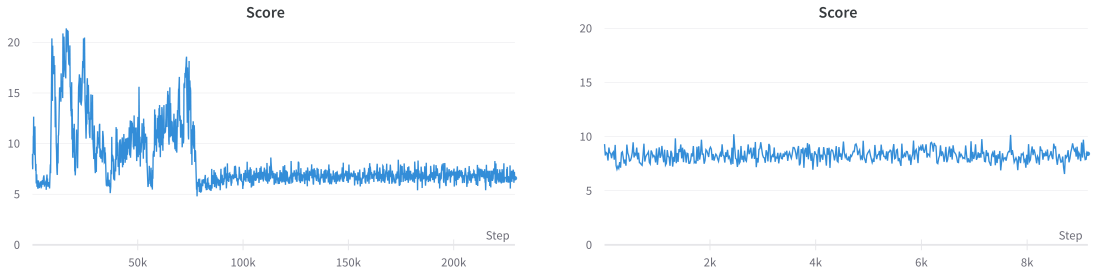


Figure 9: KL Balance Loss

The reward score (Figure 10a) is the score that our agent achieves when acting in the highway environment. The score of the random baseline (Figure 10b) is determined by having an agent in the environment that performs random actions. The score of our actor is on average slightly higher than the random baseline until around 70k steps. From then on, the reward score behaves like a random agent.

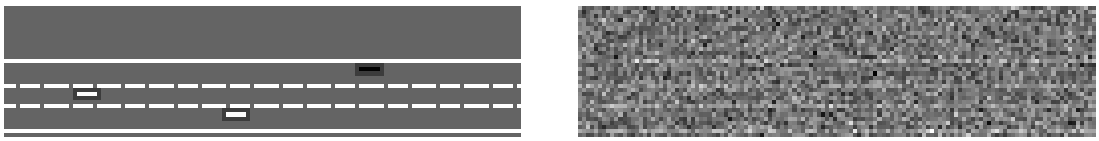


(a) Our DreamerV2

(b) Random Baseline

Figure 10: Comparison of Reward Score between our DreamerV2 Implementation and a Random Baseline

The generated images of the decoder (Figure 11b) are barely close to the original input image (Figure 11a). But still, one can clearly see that the decoder was able reconstruct some of the road lines. On the other hand, cars are not visible. Those road lines are already slightly visible after only a few thousands steps. From there on, they improve only slightly over the rest of the training run.



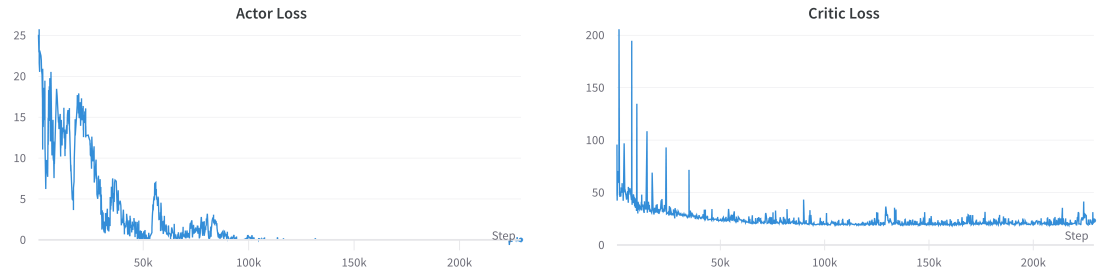
(a) Real State

(b) Predicted State

Figure 11: Comparison Between Predicted and Real state

The actor and the critic loss (Figure 12) improve a lot in the beginning, compared

to their initial loss. Eventually they stagnate, while the actor loss gets close to zero. Regardless, the reward stops correlating with the actor and critic loss very early in the training run.



(a) Actor Loss

(b) Critic Loss

Figure 12: Actor-Critic Loss

4 Discussion

Looking at our results, it clearly looks like our DreamerV2 implementation is not properly training. But it does seem to be an implementation error, instead of being by-design unable to train on the "highway-env" environment. The world model should be able to reconstruct the input image, but it barely does it. This shows us that some part from the encoder to the decoder is not working properly. The stagnating loss indicates vanishing gradients, but this is just a symptom and we haven't found the root cause. Whether our A2C implementation works, can't be evaluated at the current state of our implementation, because it relies on the outputs of the world model. As long as those are insufficient, A2C cannot learn to perform well on the environment. We still see it as a success that the predicted states of the decoder show correct road lines. This shows that it is working to some extent and we are likely very close to a working implementation. We decided to make our implementation publicly accessible, because we believe it to have good readability and explanations. It can be found here.

We experimented with changing the hyperparameters to improve the training of specifically the world model. We tried adjusting the hyperparameters of the batch size, sequence length, horizon, buffer size, encoding size, learning rate, gradient clipping and a few more less influential ones. We also used different network sizes and scaled network layer amounts and sizes up and down. Still, they showed nearly the same behavior, i.e. stagnation or oscillating loss signal when reaching an image loss of around 3800.

One approach that improved our performance slightly, was to add a missing straight-through gradient computation for KL balancing. Here, a self-written one-hot categorical distribution is needed. It inherits the OneHotCategorical distribution from the TensorFlow framework to be able to get a gradient when sampling from it. This enables the usage of the TensorFlow KL loss function while still having straight-through gradients.

While our environment is different from the Atari environments, we still think that DreamerV2's original design and parameters should be able to learn at least a decent performing policy. The autoencoder-like world model should be capable to output images that are close

to the input images.

Our next steps in debugging would include an analysis of the gradients as well as the model weights. There could be some problems with that, because there are multiple occasions were we used the stop_gradient method of TensorFlow. These complex interlacing could lead to undetected bugs. Furthermore, the extensive use of distributions and straight-through gradients could be a major point of failure. Some of these methods need deep knowledge about the underlying internal functioning of the TensorFlow framework and the library TensorFlow-Probability.

For the sake of completeness there are multiple techniques showing minimal improvement like binary latent representations, long-term entropy, mixed actor gradients, scheduling or layer normalisation. Nevertheless, it should be noted that even the authors themselves described these to "not help substantially" Hafner et al. (2022).

Also, it would be interesting to change the latent representations z_t and $\hat{z}_t$. Hafner et al. (2022) implemented them as 32x32 categorical distributions. These could be changed to either smaller or larger categorical distributions or even multiple Gaussian distributions like Figure 4 suggests. The latter would be a lot more complex to implement.

The researchers Bécsi et al. (2018) who published the highway environment tested the environment with Vanilla Policy Gradient (VPG). They used a discrete observation space with a state space of 17 and an action space of 25. To predict their action, they used one neural network with two layers and 512 neurons per layer. For their training, a total of 250,000 episodes were generated. Each episode is limited to 500 steps in the environment. In Figure 13 we can see the agent learning and reaching average rewards of 480. This is close to the maximum reward of 500 (Bécsi et al. (2018)).

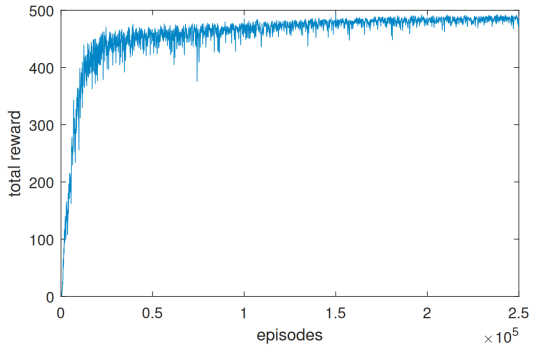


Figure 13: Averaged Total Reward per 100 Episodes in the Highway Environment with VPG

Bécsi et al. (2018)

Comparing the score of our implementation (Figure 10a) with Bécsi et al. (2018) in Figure 13, it becomes clear that their implementation of VPG reached a far larger reward than our DreamerV2 implementation. This is despite us using a smaller and easier-to-learn action space and a larger network.

Hafner et al. (2022) showed that DreamerV2 outperformed other well-established model-free approaches like Rainbow and IQN. In Figure 1 we can see that DreamerV2 reached

performances twice as good as a human. In our experiment (Figure 10a) we barely exceed the random baseline (Figure 10b).

5 Conclusion

Deep reinforcement learning is always delicate to train and sensitive to hyperparameter adjustments. It is therefore not completely unexpected (even though it is frustrating) that our implementation does not work.

Looking at Figure 3, one can see the complexity of the world model alone. The A2C adds another layer of complexity on top with the actor having three hyperparameters of which two are scaled during training. So in short, DreamerV2 has a lot of hyperparameters to fine tune. This in addition to the complex network architecture with a unique set of techniques like straight-through gradients or KL-balancing, results in a lot of possible error sources.

It would have been a better ending to report on all the upsides of the implementation. Nevertheless, science is about approximating truth and not about what is good. It might be possible to detect the problem in the implementation by trying further changes and hyperparameter tuning. But our spent computing and time resources already exceeded what was planned and expected within this course.

Even though we experienced challenging issues trying to re-implement DreamerV2, we still think it is an impressive model and pushes the boundaries regarding the capabilities of model-based reinforcement learning. We are optimistic regarding the future of world models in model-based DRL and are keen to see how this relatively new field will develop in the future.

Acknowledgements

No funding was received.

We thank our tutor Leon Schmid for the support and inspiration we received during this project.

References

Bécsi, T., Aradi, S., Fehér, Á., Szalay, J., and Gáspár, P. (2018). Highway Environment Model for Reinforcement Learning. *IFAC-PapersOnLine*, 51(22):429–434.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.

Gläscher, J., Daw, N., Dayan, P., and O'Doherty, J. P. (2010). States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 66(4):585–595.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2020). Dream to Control: Learning Behaviors by Latent Imagination.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning Latent Dynamics for Planning from Pixels. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2555–2565. PMLR.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2022). Mastering Atari with Discrete World Models.

Hestenes, D. (1992). Modeling games in the Newtonian World. *American Journal of Physics*, 60:732–748.

Khemlani, S. S., Barbey, A. K., and Johnson-Laird, P. N. (2014). Causal reasoning with mental models. *Frontiers in Human Neuroscience*, 8.

Lee, H.-M., Henze, D. K., Alexander, B., and Murray, L. T. (2014). Investigating the sensitivity of surface-level nitrate seasonality in Antarctica to primary sources using a global model. *Atmospheric Environment*, 89:757–767.

Leurent, E. (2018). An Environment for Autonomous Driving Decision-Making. GitHub.

McGrath, T., Kapishnikov, A., Tomašev, N., Pearce, A., Hassabis, D., Kim, B., Paquet, U., and Kramnik, V. (2021). Acquisition of Chess Knowledge in AlphaZero.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

OpenAI (2020). Part 2: Kinds of RL Algorithms — Spinning Up documentation. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

Orban de Xivry, J.-J. and Ethier, V. (2008). Neural Correlates of Internal Models. *The Journal of Neuroscience*, 28(32):7931–7932.

Racanière, S., Weber, T., Reichert, D. P., Buesing, L., Guez, A., Rezende, D., Badia, A. P., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. (2017). Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5694–5705, Red Hook, NY, USA. Curran Associates Inc.

Seel, N. M. (2017). Model-based learning: A synthesis of theory and research. *Educational Technology Research and Development*, 65(4):931–966.

Wiering, M. and Schmidhuber, J. (1999). Efficient Model-Based Exploration.

6 Appendix

6.1 A. List of all Hyperparameters

Name	Symbol	Value
World Model		
Buffer size	-	100000
Batch size	-	50
Discrete latent dimensions	-	32
Discrete latent classes	-	32
RSSM units	-	400
KL loss scale	-	0.1
KL balancing	-	0.8
World model learning rate	-	2×10^{-4}
Reward transformation	-	Linear
Actor-Critic		
Imagination horizon	H	15
Discount factor	γ	0.995
$\lambda - targetparameter$	λ	0.95
Actor gradient mixing	ρ	1
Actor entropy loss scale	η	$1 \cdot 10^3$
Actor learning rate	-	$4 \cdot 10^5$
Critic learning rate	-	$1 \cdot 10^4$
Slow critic update interval	-	5
Common		
Gradient clipping	-	100.0
Adam epsilon	-	$1 \cdot 10^5$