RL Project Group 5: Pacman

Champ Worajitwannakul
Department of Computer Science
University of Bath
Bath, BA2 7AY
sw2780@bath.ac.uk

Chai Worajitwannakul
Department of Computer Science
University of Bath
Bath, BA2 7AY
nw600@bath.ac.uk

Dev Prajapati
Department of Computer Science
University of Bath
Bath, BA2 7AY
dp820@bath.ac.uk

Sarnaz Hossain
Department of Computer Science
University of Bath
Bath, BA2 7AY
sh2868@bath.ac.uk

1 Problem Definition

This research project aims to create an agent that plays the popular Atari 2600 game "Pacman". The goal for this game is to collect as many pellet as possible while avoiding 4 ghosts; each of which has an individual characteristic. The reason why we chose Pacman is due to the stochastic nature of the environment and the complex interaction between classical AI from ghosts, based on fixed rules. Each pellet is worth 10 scores; however, the power pellet allows Pacman to eat ghosts temporarily. For each ghost that is eaten, the score is doubled where the first ghost is worth 200 scores. Therefore, the second ghost is worth a 400 score, and so forth. We use Gymnasium - an updated version of gym - for Pacman game. The problem is summarized as follows:

Observation space: The observation is a three-dimensional 250x160 RGB image of a Pacman game. This can be expressed as follows:

$$O_t = (250, 160, 3) \in O \subset \mathbb{R}^3 \tag{1}$$

State space: The state is a tuple that consists of current observation, action and reward, which can be mathematically expressed as follows:

$$S_t = (O_t, A_t, R_t) \in S \subset \mathbb{R}^3 \tag{2}$$

Action space: There are 5 discrete actions; each corresponds to the following: Noop (0), Up (1), Right (2), Left (3), Down (4); Noop stands for No operation which means to do nothing. This is given below:

$$A = \{0, 1, 2, 3, 4\} \tag{3}$$

$$A_t \in A \subset \mathbb{R} \tag{4}$$

Transition Dynamics: The environment is deterministic in the sense that Pacman moves according to given actions. Pacman is stationary when hitting the wall without any penalty. When Pacman heads toward the path across the border, it will appear on the opposite side. The position of the pellet and power pellet remains the same across all lives. However, the movement of ghosts are also

deterministic with the except of orange which is stochastic. Each has its own characteristics; Red chases Pacman, and Pink and Blue ambush Pacman.

Reward: There are two ways to obtain rewards:

- 10 points for each pellet Pacman collects
- 200 points for first ghost, 400 points for second ghost, and so forth only when power from power pellet is on

Each episode has three lives for Pacman where the final score is accumulated across all lives. The pellets and power pellets are reset for each live as well as the initial positions of ghosts and Pacman.

2 Background

Classical AI approaches to Pacman have been attempted for decades. Researchers and developers have explored various algorithms to enhance Pacman's complex solution. In the paper by Yang Zou [10], an analysis is presented on some of these classical methods. The paper examines algorithms such as Depth-First Search (DFS), and A* Search. For the more complex multi-agent dynamics, techniques like Minimax Search and Alpha-Beta Pruning are also explored, offering insights into strategic decision-making in competitive environments. This exploration of classical AI techniques provides a foundation for understanding the evolution towards reinforcement learning approaches in gaming.

The transition from these classical AI methods to reinforcement learning in particular model-free methods, represents a significant shift in AI strategy and capability. This evolution is exemplified by the development of Deep Q-Networks (DQN). DQN is an example of a model-free reinforcement learning method used by Google DeepMind for various Atari 2600 games, achieving performances surpassing human players [6]. This marked a change from the predetermined models of classical AI, allowing agents to be more adaptable when it comes to learning about the dynamic of complex environments.

Alongside the advancements in model-free RL, there has been a large exploration towards model-based reinforcement learning (RL) also. This approach represents a more sophisticated framework, where an internal model of the environment is constructed and used to predict future actions. Notable developments in model-based RL have been made by algorithms like AlphaZero and MuZero. AlphaZero, combines Monte Carlo Tree Search with neural networks which resulted in incredible improvements especially in environments with discrete states [8]. MuZero extends these capabilities to continuous environments, learning a model that predicts vital decision-making factors without needing knowledge of the environment's dynamics [7]. This advancement broadens the applicability of model-based RL, allowing it to tackle a wider array of complex tasks.

One subsection of model-based Reinforcement Learning that is very interesting to us is imagination-based learning. Instead of learning from experience or planning through tree search, these models learn from imagination in the form of latent states. Such a model is known as a world model [1], a mental representation of the environment, which allows humans to simulate actions. Hafner makes use of this approach by introducing an architecture, PlaNET [2], which utilizes the recurrent state space model for latent space planning. Building upon this concept, the Dreamer series of models, including Dreamer V1 and Dreamer V2. The original Dreamer model integrates an actor and value network, learning to navigate and make decisions continuously within a world model. Dreamer V2 [3] takes this further by employing categorical variables for the latent representation instead of Gaussian variables, similar to the approach used in Vector Quantized-Variational AutoEncoders (VQ-VAE) [9].

This leads us to the final iteration and the algorithm that interested us the most when reviewing the literature around RL algorithms - Dreamer V3. This was first introduced as a state of the art reinforcement learning model able to collect diamonds in Minecraft. However the main innovation lies with the ability to tackle tasks across many domains without extensive resources and task-specific tuning. In the paper Mastering Diverse Domains through World Models we can see the notable achievements that demonstrate this aim [4]. Dreamer V3 was applied to different environments, including those with continuous and discrete actions, visual and low-dimensional inputs, and 2D and 3D worlds. It achieved "strong" performance on all environments that were tested including tasks

such as robot locomotion, Atari games, and complex 3D domains like Minecraft. It even set new state-of-the-art performances on Proprio Control Suite, Visual Control Suite, BSuite, and Crafter.

Given these impressive results, Dreamer V3 was chosen as algorithm to implement and test out with the challenges presented by Pacman.

3 Method

We based our implementation on the Deepmind version of Dreamerv3 in JAX [5]; on the contrary, we wrote our version in Pytorch. In summary, DreamerV3 can be categorized into three parts: Replay Buffer, Actor Critic, and World Model. Firstly we will mention techniques that we implemented that are unique to Dreamer v3 and then get into the main implementation.

3.1 Techniques

The main distinguishing difference from Dreamerv2 to Dreamerv3 is the introduction of Symlog Distribution and Two Hot Encoding.

3.1.1 Symlog Distribution

Dealing with diverse environments is difficult; reward scores can be sparse - an agent only receives a non-zero reward at the end of the game. Current techniques such as normalization lead to non-stationary optimization,therefore the model does not converge. As a result, Hafner introduces a Symlog function to deal with diverse reward distribution.

$$symlog(x) = sign(x) \times ln(|x| + 1) \tag{5}$$

This is possible since the function is logarithmic thus large value is truncated while the small value is maintained; we deployed this in Critic, Decoder, and reward predictor.

3.1.2 Two Hot Encoding

The return distribution for critics can be widespread, thus by solely outputting the expected mean value, it does not represent the variance of such distribution. Therefore, Hafner introduces two-hot encoding; a discrete regression approach that is built upon one-hot encoding. In addition, since some environments can be stochastic, predicting by distribution allows a model to capture some randomness in the return.

$$\text{twohotencoding}(x) = \begin{cases} \dfrac{|b_{k+1} - x|}{|b_{k+1} - b_k|} & \text{if } i = k \\ \dfrac{|b_k - x|}{|b_{k+1} - b_k|} & \text{if } i = k+1 \\ 0 & \text{else} \end{cases} \tag{6}$$

3.2 Replay Buffer

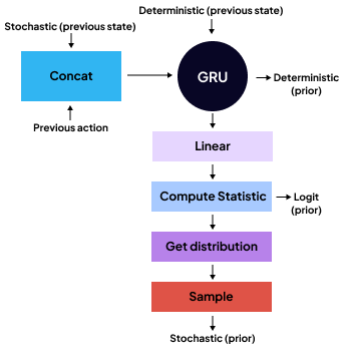
Replay buffer acts as a storage of experiences that the agent plays during training and evaluation. These experiences are then used to train different components of Dreamerv3, by retrieving batches. To get batches, We first segment trajectories into a desired form. This is done by ignoring the remainder of batch length on the right end so that stored replayed steps can be segmented. We uniformly choose a segment and repeat it batch size number of times. After that, we transform the list into a list of tensors where the first two dimensions are batch size and batch length, expressed as follows: $(batchsize, batchlength, *)$. The training ratio dictates number of replayed step over number of environment steps at which the default value is 1024.

3.3 World Model

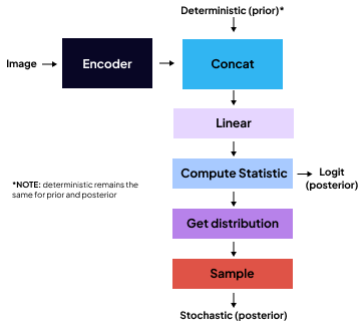
The World Model serves as the agent's mental representation of the environment, this utilizes the Recurrent State-Space Model (discussed in more detail next) to predict the next state based on its

understanding of the world, rather than solely on the input. It's designed to forecast not just the next state but also other crucial environmental variables like reward and continuation probability. Using different loss functions for both: improving the accuracy of the direct next prediction (with the rewards and continuity) and, to improve the representation of its current current state in order to improve its own representation of the world. This understanding is to be then used by the Actor-Critic model to 'dream' or simulate future events, enhancing its ability to forecast without the need for real data.

The World model is composed of 3 neural nets: Recurrent State-space Model, Encoder and Decoder.



(a) Process of creating prior



(b) Process of creating posterior

3.4 Recurrent State-Space Model (RSSM) World Model

Recurrent state-space model (RSSM) is the state space model that allows the model to have an ability to plan and predict the next states by using previous actions and state. However, this representation may be oversimplified, thus may reduce its effectiveness in certain circumstances. The main working principle of RSSM consists of two losses which are *representation loss* and *dynamic loss*, KL divergence has been applied both in posterior and prior.

3.4.1 Encoder and Decoder

The convolutional architecture of the Encoder is comprised of multiple layers, each consisting of a Convolutional layer, layer normalization and activated using a Sigmoid Linear Unit (SiLU). The design maintains fixed settings, with a stride of 2 and a kernel size of 4. This architecture reduces the resolution by a factor of 2 with each layer, continuing until the output resolution is 4x4. Upon reaching this size, the process is followed by flattening and then passing through two linear layers. The detailed calculation of the number of layers utilized, based on the initial input size, is elaborated in the appendix.

The Decoder adopts an inverted approach to the Encoder's convolutional architecture, comprising layers of Transpose Convolution, layer normalisation and activated via Sigmoid Linear Units (SiLU). This reverse structure deconstructs and reconstructs the encoded features, aiming for precise input restoration. Following the transposed convolutional sequence, the architecture extends into two linear layers. These layers refine the reconstructed features, ultimately producing two distinct outputs: the "MLP output" from the multi-layer perceptron and the "CNN output" from the convolutional layers. This output mechanism enhances the Decoder's ability to accurately and efficiently regenerate the original input features.

3.4.2 Multi-layer perceptron (MLP)

Multi-layer perceptron (MLP) are predictors that is based on linear layer architecture. MLP can be categorised into two types that has the difference in term of output distribution which are *reward predictor* and *continue predictor*. *Reward predictor* is mainly worked by using *Symlog Distribution* while whereas *continue predictor* uses Bernoulli distribution.

3.5 Actor Critic

For the actual RL algorithm, we retained the core architecture of the Actor Critic while incorporating features from the World Model instead of raw observations. Both components are built on a multi-layer perceptron architecture, where the Actor is updated via the REINFORCE algorithm, which learns to select actions that maximize expected returns from a discrete action space, represented by a one-hot distribution. Whereas the Critic, employs cross-entropy loss to assess the value of those actions, guiding the actor to better decisions in the form of policy refinement. The combination of the two ensures our agent's behavior is continually improving and balanced between exploration and exploitation. In addition to this, we've also implemented bootstrapped lambda returns using recursion for the critic, as suggested by Hafner. While the precise reasoning for this is unclear to us, after research it is predicted to be for smoothing future predictions.

4 Results

The following section contains several ablation studies on different components and mechanisms of DreamerV3. Our approach to evaluating the score is the following: we calculate the mean score of three episodes for every 10 iterations of training. We do this as we observed that the score distribution is generally of high variance meaning that if we were to take the score from a single episode, it would likely be biased and would not be a good representation of the agent's performance.

For the baseline, we decided to use our own baseline which was the mean score of random play for 10000 steps with complete episodes. The score was 234.

4.1 First experiment

Our initial assumption was that the Decoder does not play a vital role in improving the performance of DreamerV3 since it is not utilised in the learning process.

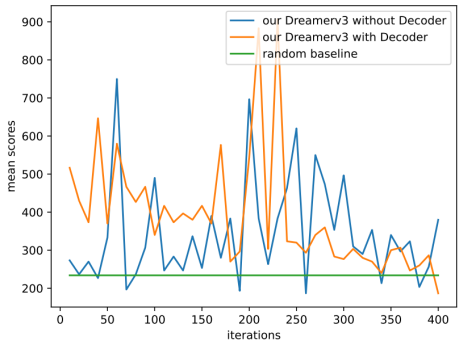
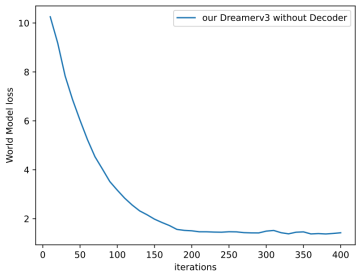


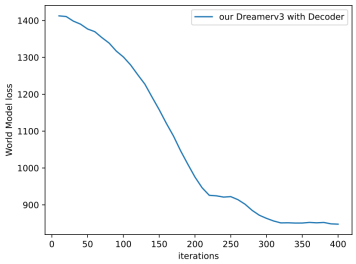
Figure 2: Mean scores of decoder and without decoder

According to figure 2, we can deduce that the decoder does stabilize the learning process to a certain extent since the spikiness is less than the one without a decoder. Moreover, the decoder does improve the performance. We hypothesise that by adding a decoder, the loss of the world model decreases slowly, thus having more time to learn from more scenarios. Additionally, it forces the model to learn features better since the decoded images stem from using features as input.

From geometric interpretation, we can observe that the loss curve for the model without and with decoder are convex and concave; this means that the change of loss increases with the decoder while decreasing without the decoder, therefore, the world model learns slower at the start with the decoder, which may result in better performance as shown in figure 3. It is noteworthy to state that the value of loss does not affect the performance but rather the change of loss, since back propagation relies on the gradient of loss concerning weights instead of its magnitude.



(a) Loss value without decoder



(b) Loss value with decoder

Figure 3: Loss of decoder and without in world model

4.2 Second experiment

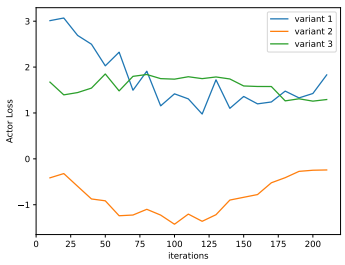
We noticed that the model performance decreases over a certain period of time; we hypothesise that this was due to a lack of exploration because the same batches from replay buffer are used repeatedly. Moreover, the start of the batch is always at the start of a game, leading to a lack of a diverse environment. Therefore, for these experiments, we explored different mechanisms of replay buffers and their impact on performance.

There is limited information concerning replay buffer from paper so we came up with different new solutions and compared the performances:

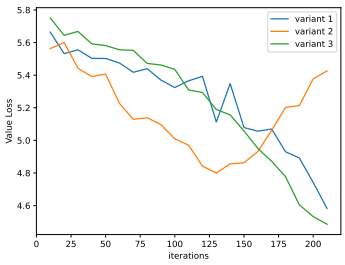
Mechanism	Description
Evaluated Step Inclusion	Adding a replayed step from the evaluation while training
Episodic Batch	Storing trajectories from the whole episode rather than a fixed number of steps
Uniform Sampling	Uniformly sample a batch based on a random starting point rather than from a collection of batches

Table 1: Mechanisms table

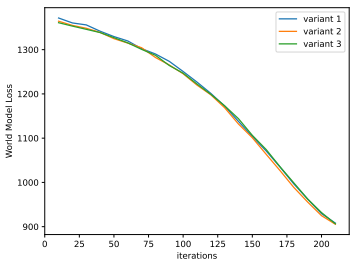
From these mechanisms, we came up with three new variants that utilize a combination of the mechanisms: Variant 1 is a combination of evaluated step inclusion and non-episodic batch, Variant 2 combines uniform sampling and episodic batch and Variant 3 combines episodic batch.



(a) Actor loss



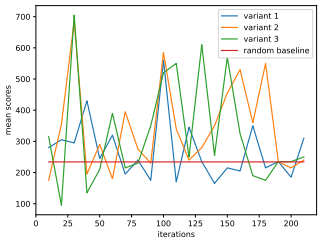
(b) Value loss



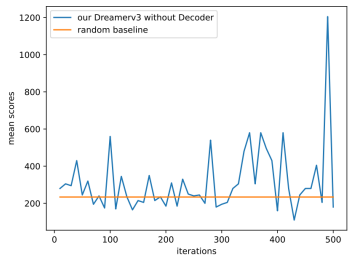
(c) World-model loss

Figure 4: Loss results

The figures above show the change of world model, actor and value loss respectively across all iterations. According to figure 4, the loss for world model is nearly identical in all three. In contrast, the actor and value losses change drastically. The main point of interest is that the value loss for variant 2 increases over time; this is a clear sign of overfitting which may stem from the fact that batches may have overlapped steps since it has random initialised positions, which then leads to models focusing on certain scenarios, thus reducing its ability to generalise, in particular the complex movement of ghosts. The actor loss does not improve; we hypothesise that this may be due to the small decrease in continue and reward loss and also its dependency on critic head.



(a) Comparison of all variants



(b) Variant 1 across 500 iterations

Figure 5: Final performance

4.3 Final result

We have compared the performance of all three variants of DreamerV3 across 210 iterations. As shown in figure 5a, all variants perform above the baseline most of the time; variant performance is the most stable with the least spikiness.

According to the figure 5b, the performance peaks at 1250 scores which is when the agent develops a strategy to collect a power pellet when ghosts are nearby. The high variation is due to the unpredictability of ghost movements which affects the agent greatly. Since the agent has not yet adapted or learned to predict the movement of each ghost, it is a difficult task.

5 Discussion

From these experiments, we have shown that DreamerV3 has shown a competitive performance in the Pacman game despite different variants. It is noteworthy that different approaches do not yield much improvement to the models; this may be due to external problems such as the size of the models.

What we observe is that throughout the training process, the model is always stuck with a single strategy over time, which in some cases, leads to poorer performance during evaluation. This may be due to the skewness of policy distribution in which the mode stays the same across an episode. However, such a phenomenon does not occur during training since action is taken by sampling from distribution; in fact, the model shows an improvement. In the future, we would love to see how distribution would change given more training time.

The learning curve is surprisingly fast with DreamerV3 in comparison to other Reinforcement Learning algorithms; this is also evident in DreamerV3 paper. This may be due to the two hot encoding mechanisms which allow neural nets with symlog distribution to infer at a much faster rate than the usual approach which utilizes all outputs from a final layer.

6 Future Work

There are lots of things we wanted to do, other than just train for longer with more computation. eg:

• **Experiments with recurrent neural nets:** Given that the current recurrent state space model uses a Gated Recurrent Cell, we really would like to explore the impact of using transformer architecture. Perhaps, the scaling of parameters in a transformer is a great direction to explore.

• **Complex and continuous environment:** We would love to test this algorithm on a continuous environment, in particular, robotic tasks that are available in the Gymnasium. Furthermore, we really would love to test on Minecraft, which is one of the main reasons we chose this algorithm.

7 Personal Experience

DreamerV3 algorithm is without a doubt a challenging algorithm to implement due to the complex nature of the recurrent state space model and a large number of components. Nonetheless, we have gained valuable experience in particular on Pytorch implementation of Deep Reinforcement Learning.

References

[1] Dedre Gentner and Albert L Stevens. *Mental models*. Psychology Press, 2014.

[2] Danijar Hafner et al. "Learning latent dynamics for planning from pixels". In: *International conference on machine learning*. PMLR. 2019, pp. 2555–2565.

[3] Danijar Hafner et al. "Mastering atari with discrete world models". In: *arXiv preprint arXiv:2010.02193* (2020).

[4] Danijar Hafner et al. *Mastering Diverse Domains through World Models*. 2023. arXiv: 2301.04104 [cs.AI].

[5] Danijar Hafner et al. *Mastering Diverse Domains through World Models*. https://github.com/danijar/dreamerv3. 2023.

[6] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[7] Julian Schrittwieser et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839 (2020), pp. 604–609.

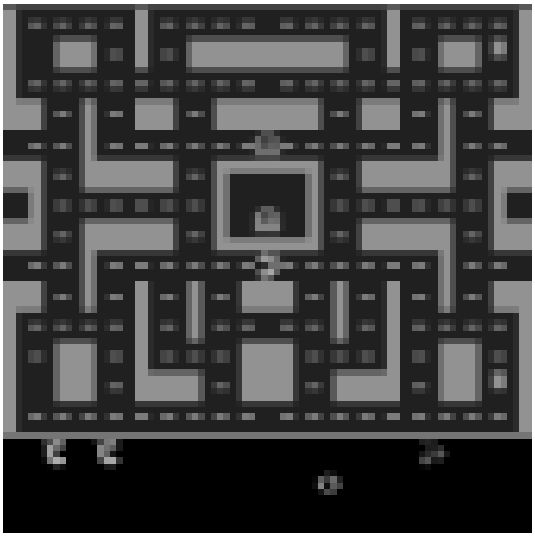
[8] David Silver et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815* (2017).

[9] Aaron Van Den Oord, Oriol Vinyals, et al. "Neural discrete representation learning". In: *Advances in neural information processing systems* 30 (2017).

[10] Yang Zou. "General Pacman AI: Game Agent With Tree Search, Adversarial Search And Model-Based RL Algorithms". In: *2021 2nd International Conference on Big Data Artificial Intelligence Software Engineering (ICBASE)*. 2021, pp. 253–260. DOI: 10.1109/ICBASE53849.2021.00053.

Appendices

Appendix A: preprocessed image



Appendix B: Encoder calculation

- kernel size and stride are initialized as 4 and 2 respectively

$$r_{out} = \frac{r_{in} + (2 \times padding) - (dilation \times (kernelsize - 1)) - 1}{stride} + 1 \tag{7}$$

$$(2 \times r_{out}) = r_{in} + (2 \times padding) - (3 \times dilation) - 1 + 2 \tag{8}$$

$$(2 \times r_{out}) - r_{in} - 1 = (2 \times padding) - (3 \times dilation) \tag{9}$$

let denote v = $(2 \times r_{out}) - r_{in} - 1$

$$v = (2 \times padding) - (3 \times dilation) \tag{10}$$

if v < 0 then

$$v + (3 \times dilation) = 2 \times padding \tag{11}$$

since padding cannot be less than zero then

$$v + (3 \times dilation) > 0 \tag{12}$$

$$d > \frac{v}{3} \tag{13}$$

$$d = round(\frac{|v|}{3}) \tag{14}$$

if v is even then assume that dilation = 2 then

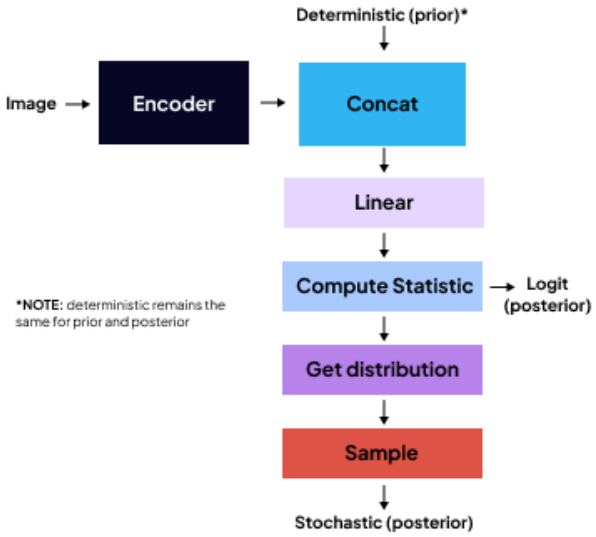
$$padding = \frac{(v + 6)}{2} \tag{15}$$

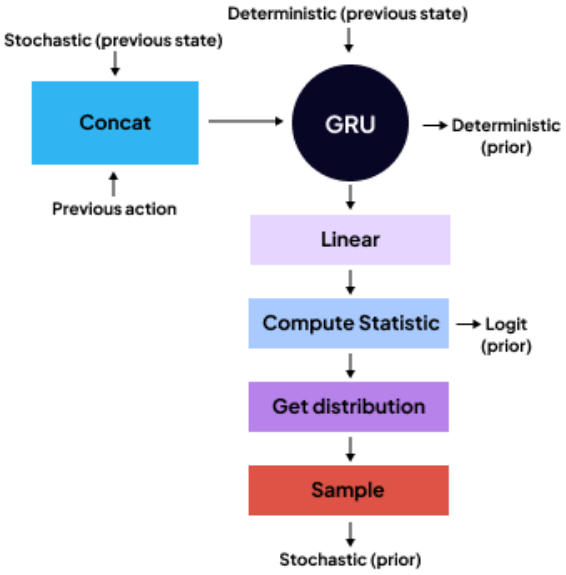
if v is odd then assume that dilation = 1 then

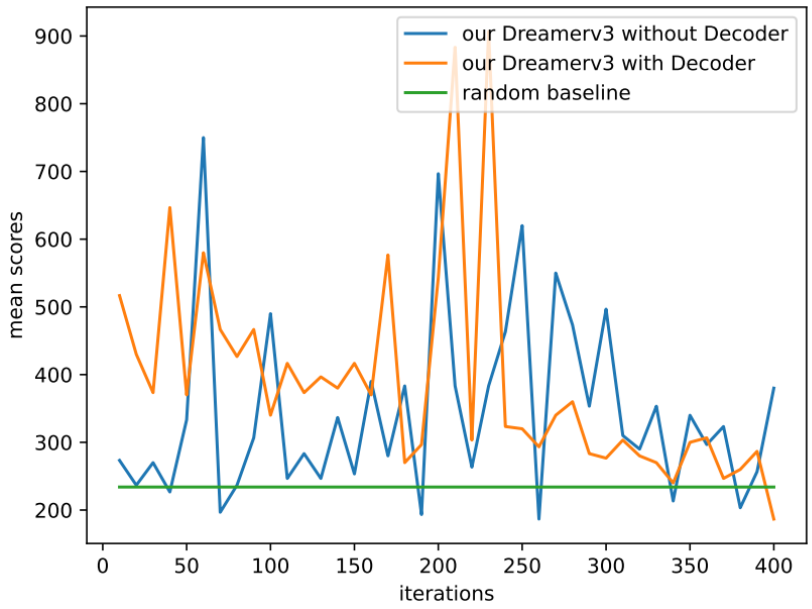
$$padding = \frac{(v + 3)}{2} \tag{16}$$

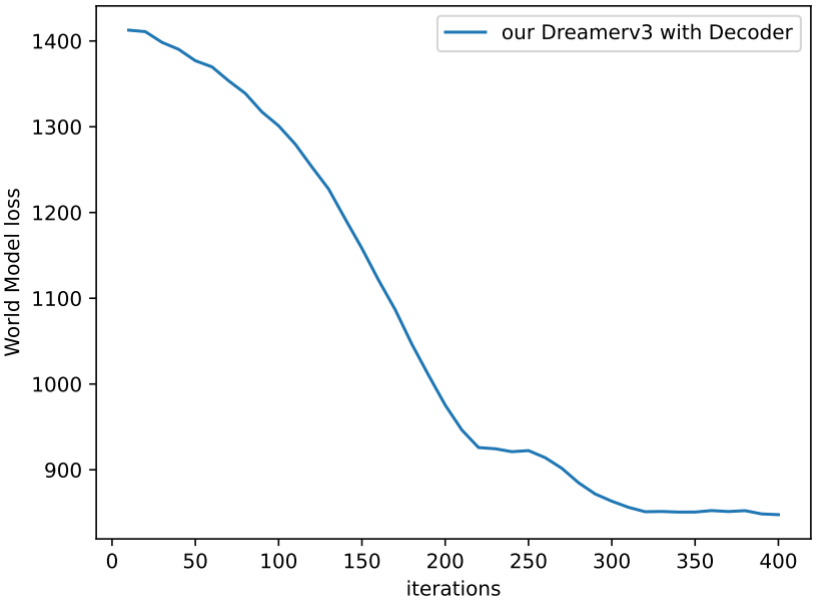
$$\tag{17}$$

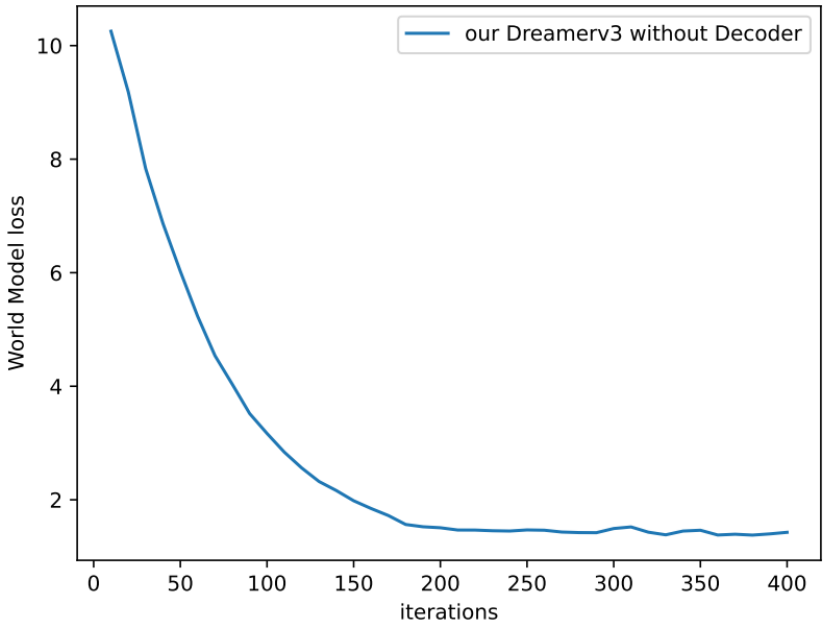
Appendix C: Figures Larger

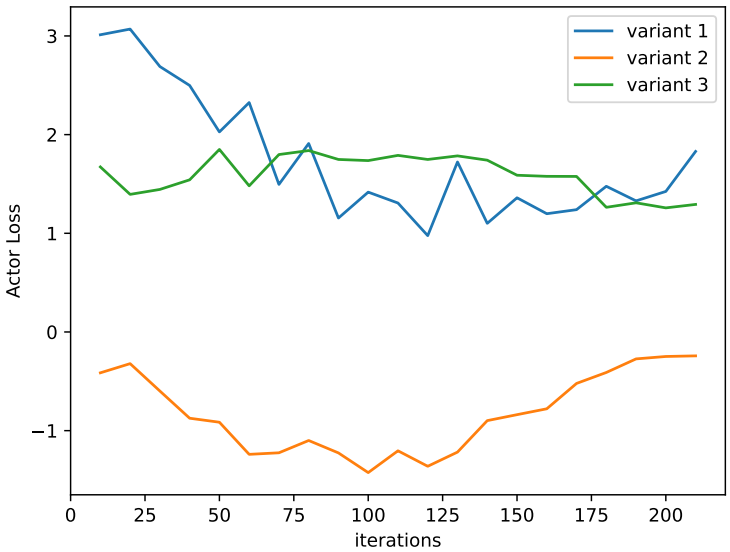

Deterministic (prior)*
Image
Encoder
Concat
Linear
Compute Statistic
Logit
(posterior)
*NOTE: deterministic remains the same for prior and posterior
Get distribution
Sample
Stochastic (posterior)

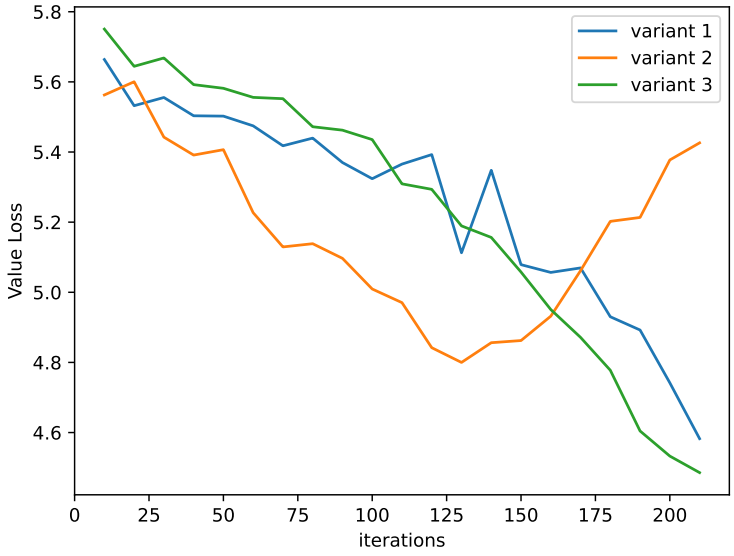

Deterministic (previous state)
Stochastic (previous state)
Concat
GRU
Deterministic
(prior)
Previous action
Linear
Compute Statistic
Logit
(prior)
Get distribution
Sample
Stochastic (prior)

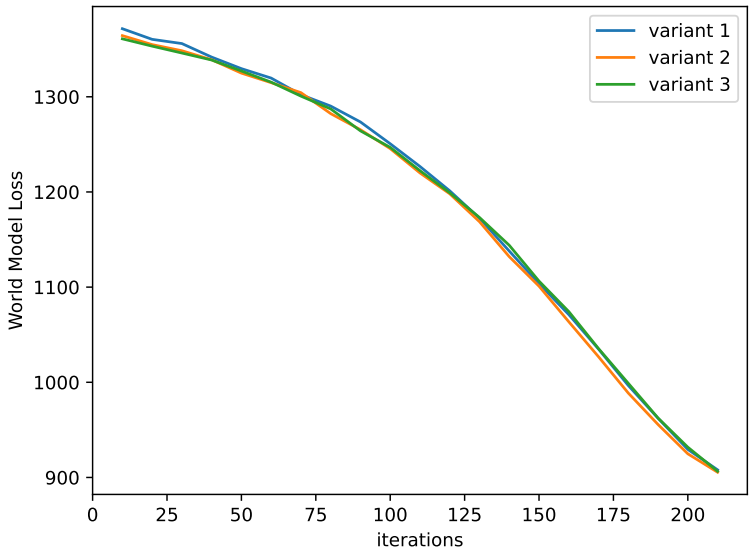

our Dreamerv3 without Decoder
our Dreamerv3 with Decoder
random baseline
mean scores
iterations
900
800
700
600
500
400
300
200
0
50
100
150
200
250
300
350
400

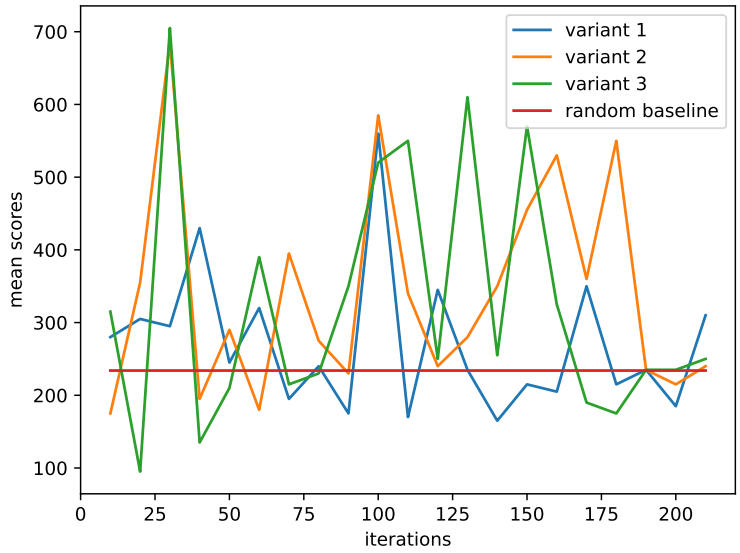

our Dreamerv3 with Decoder
World Model loss
iterations
1400
1300
1200
1100
1000
900
0
50
100
150
200
250
300
350
400

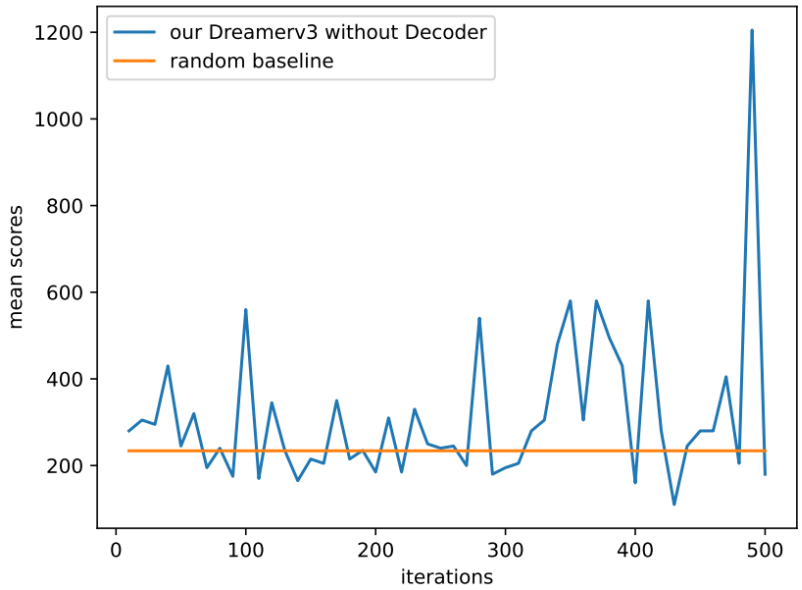

our Dreamerv3 without Decoder
World Model loss
iterations
10
8
6
4
2
0
50
100
150
200
250
300
350
400


variant 1
variant 2
variant 3
Actor Loss
iterations
3
2
1
0
−1
0
25
50
75
100
125
150
175
200


variant 1
variant 2
variant 3
Value Loss
5.8
5.6
5.4
5.2
5.0
4.8
4.6
0
25
50
75
100
125
150
175
200
iterations


variant 1
variant 2
variant 3
World Model Loss
1300
1200
1100
1000
900
0
25
50
75
100
125
150
175
200
iterations


variant 1
variant 2
variant 3
random baseline
mean scores
iterations
700
600
500
400
300
200
100
0
25
50
75
100
125
150
175
200


our Dreamerv3 without Decoder
random baseline
mean scores
iterations
1200
1000
800
600
400
200
0
100
200
300
400
500