Deep Latent Competition - V1: Decentralized multi-car racing with LSTM-Dreamer-V3

Fei Yu Guan
Department of Mathematical and Computational Sciences
University of Toronto
`vi.guan@mail.utoronto.ca`

Gabriel Paulos
Edward S. Rogers Department of Electrical Engineering
University of Toronto
`gabriel.paulos@mail.utoronto.ca`

Abstract

One of the most important challenges in autonomous driving is enabling an agent the ability to interact with other agents. This involves three steps: understanding the other agents' behavior, predicting the actions of the other agent(s), given the current state and a certain policy, and using these predictions to choose an action that maximizes its probability of fulfilling its long-term objective. As such, we propose Deep Latent Competition-v1, a modified version of the previous work, Deep Latent Competition [5]. We have included some changes to enable faster training and better performance than the traditional Dreamer [3].

1 Introduction

Autonomous car racing is an ideal test environment for evaluating the performance of different interaction algorithms between vehicles and how changes in their parameters affect that performance. In this project, we are particularly interested in investigating the effect of using LSTM as the recurrent model as opposed to GRU on the training results. We have also modified the environment to include functionalities such as throttle-brake dead zone, and brake speed threshold to improve training.

We model this racing interaction as a Multi-Agent Reinforcement Learning problem, where the state of the other agent(s) is partially observable. As such, it is ideal to use an RNN-based model such as the RSSM. To improve sample efficiency, we use the same baseline as the previous work, DLC, which is the Dreamer RNN model. To better the agents' ability to plan, we used an LSTM as opposed to a GRU as the recurrent model in Dreamer.

2 Background and related work

In recent years, significant progress has been made in many areas of autonomous driving and related control tasks. Many of these breakthroughs have occurred with the help of Reinforcement Learning. Before diving into some relevant papers on the matter and our contributions we would like to refresh the reader on some important Reinforcement Learning concepts.

2.1 Reinforcement learning concepts

Agent is an entity outside of the environment that interacts with the environment. Its policy model uses a store of states and observations to choose an action with the highest reward. The agent will store the new state and observations and repeat the process.

States and observations are derived from the state space of the environment. If the environment is not known, an agent can take partial states known as observations. The agent uses states and observations to take an action (make a decision).

Action Space holds all possible actions that an agent can take given its state and observations. These can be either continuous or discrete spaces depending on the context. In our context, we are working with a continuous action space (as there are countably infinite actions an agent can take as a car).

Policy is a rule used by an agent to decide what actions to take. It is a heuristic used to find the maximum expected cumulative reward. In our case, it is stochastic: $\theta(\cdot|s_t)$, where s_t is the state at timestep t. However, this means that we cannot directly predict the optimal value but instead must estimate which actions will give the highest likelihood of producing the maximum reward.

Reward is the feedback the agent receives after performing an action. It is always scalar. The reward is given by $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$, where γ dampens the rewards, to keep the cumulative reward finite.

Trajectory is the evolution of the states and actions of the agent as it goes through T timesteps. In a stochastic use-case such as ours, the next state is determined by $s_{t+1}\ P(\cdot|a_t, s_t)$.

A Reinforcement Learning problem is modeled as a Markov Decision Process (MDP). This formulation is useful in obtaining the best policy for our objective for a given environment. We can model these problems as $M(S, A, T, r)$ where S is the state space, A is the action space, r is the reward, and T is the transition operator. The transition operator is a tensor that determines the conditional probabilities for moving from one state to another. Each state is assumed to be Markovian, making each transition dependent only on the previous state.

We can maximize the cumulative expected reward using the MDP formulation. Assuming that we have gone through a trajectory of T-length $\tau = (a_0, s_0, ..., a_T, s_T)$, we can model the probability of going through such a path as $p_\theta(\tau) = p(s_1)\prod_{t=1}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$. The expected return of the reward is the optimal policy parameter $\theta* = argmax_\theta E_{\tau\ p_\theta(\tau)}[R|\theta]$ [4].

2.2 Problems in RL

We can now look at some of the challenges faced when implementing Reinforcement Learning in various use cases:

- The observations of the agent depend on its actions and the current timestep
- The policy has no ground truth and must be inferred through the reward, its only source of feedback in the training stage. A good policy potentially requires millions of training episodes and many computing resources. Both of which can make a problem intractable to solve.
- Agents must account for long time horizons as a necessary component of finding the optimal policy. The effect of a present action may only be seen many steps later. This is known as the credit assignment problem [1].

2.3 Related work

Reinforcement learning [6] is a seminal paper that pioneered the basic architecture of a Reinforcement Learning solution. The action model and the need to optimize a policy originate from this paper, as it is the first model to do so successfully.

Multi-agent interactions **Learning Latent Representations to Influence Multi-Agent Interaction** [7] and previous work **DLC** [5] suggest that learning an abstract representation of multi-agent environments produces policies that sufficiently achieve their objective, especially when the state space is high dimensional.

Latent Imagination [3] and [2] both represent recent approaches towards using latent imagination in model-based Reinforcement Learning. Latent imagination decides an action that maximizes the reward based on its ability to predict multiple future trajectories based on it's current observations and policy. In the case of Dreamer, this is achieved by back-propagating through the gradients of the latent imagination (the predicted states after an action).

2.4 Our contributions

- Investigate the effect of using LSTM instead of GRU to enable longer-term imagination.
- Apply an entropy regularization on the actor model to improve the policy
- Add functionality to the environment, throttle-brake dead zone scheduling, and brake speed threshold to improve training
- Add a termination predictor to truncate the return

3 Data

The data presented in this work was generated using a multi-car racing gym environment [5]. Each run was randomly generated using an inputted random seed. Two agents (the race cars) will race each other to the finish line within the environment. Each car stops when it finishes the race or its center leaves the road. The throttle-brake dead zone of each agent is between -0.5 and 0. An agent will brake if it receives an input action of ≤ -0.5 and will accelerate to a velocity of the magnitude of the input action given that it is > 0.

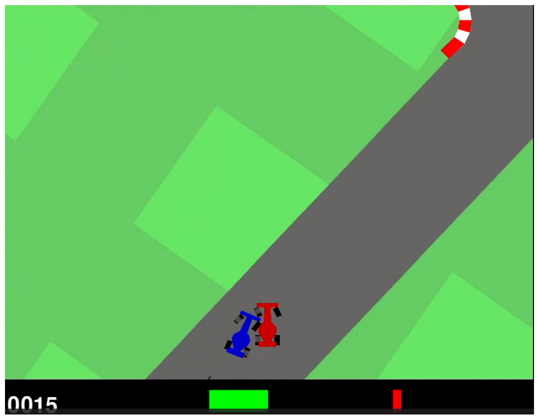


Figure 1: Multi-car racing environment

Each agent will deal with a state space of 96x96 pixels in a gym environment. The gym environment will hand out rewards according to three heuristics: tile progress, speed, and the number of timesteps it takes. A run ends once the race is complete or both agents leave the road and enter the grass areas. These reward heuristics have two goals: to promote competition between the two agents and to teach the agents effective driving. The specifics of the heuristics are shown in the table above.

Our model consists of several parts: a world model, an LSTM, an actor model, an encoder, a target encoder, a decoder, a dynamics model, a target dynamics model, a reward model, and a critic model. The model architecture is described pictorially below.

The various parts of the model can be assumed:

$$\text{World Model: } h_t, c_t = f(z_{t-1}, a_{t-1}, h_{t-1}, c_{t-1})$$
$$\text{Encoder Model: } z_t \sim e(x_t, h_t)$$

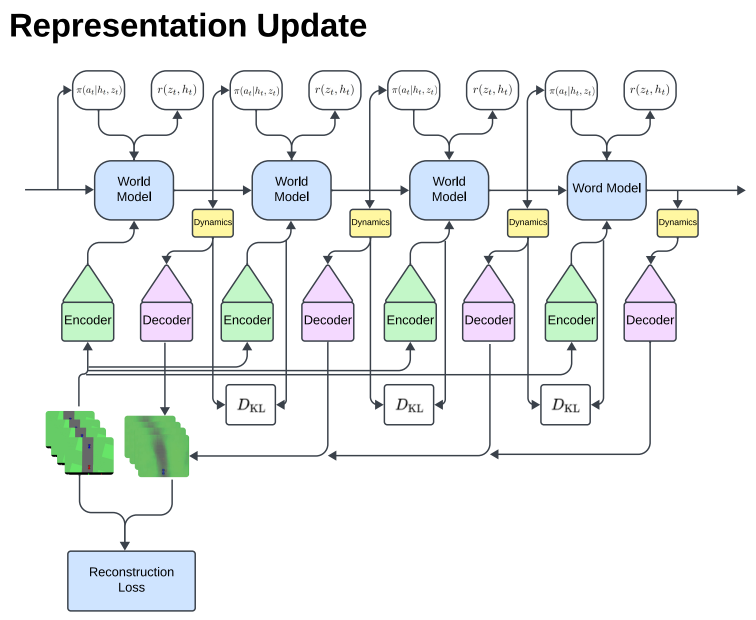


Figure 2: Representation Update

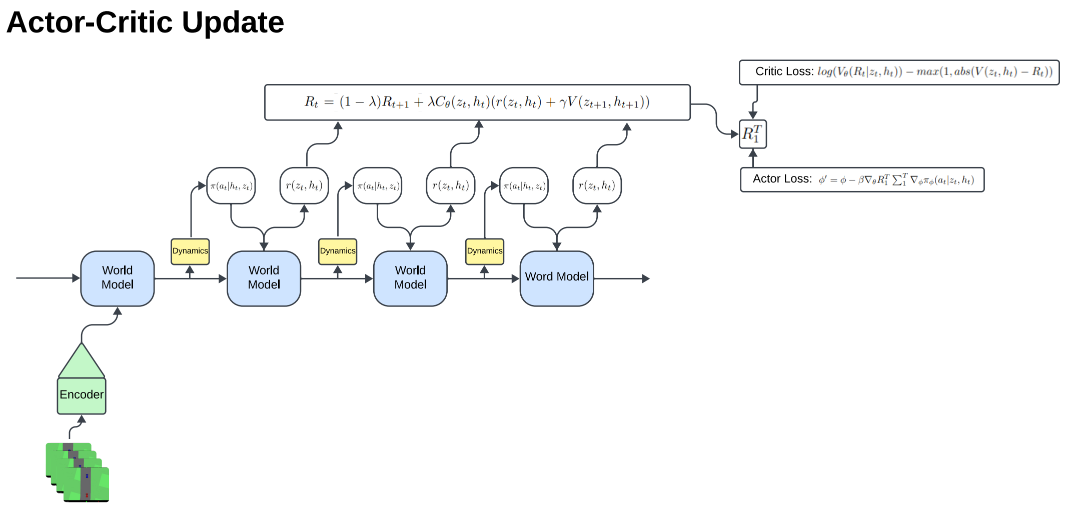

Actor-Critic Update
$R_t = (1-\lambda)R_{t+1} + \lambda C_\theta(z_t, h_t)(r(z_t, h_t) + \gamma V(z_{t+1}, h_{t+1}))$
Critic Loss: $log(V_\theta(R_t|z_t, h_t)) - max(1, abs(V(z_t, h_t) - R_t))$
R_1^T
Actor Loss: $\phi' = \phi - \beta \nabla_\theta R_1^T \sum_1^T \nabla_\phi \pi_\phi(a_t|z_t, h_t)$
$\pi(a_t|h_t, z_t)$
$r(z_t, h_t)$
Dynamics
World Model
Word Model
Encoder

Figure 3: Actor-Critic Update

Decoder Model: $\hat{x_t} \sim d(z_t, h_t)$
Actor Model: $a_t \sim \pi(a_t|z_t, h_t)$
Critic Model: $v_t \sim V(z_t, h_t)$
Dynamics Model: $\hat{z_t} \sim m(h_t)$
Reward Model: $\hat{r_t} \sim r(z_t, h_t)$

where

z_t denotes the embedding of the encoder at time t.
h_t is the hidden states of the LSTM.
c_t is the gate-states of the LSTM.
$\hat{r_t}$ is the predicted reward at time t.
a_t is the action taken at time t.
x_t is the pixel input at time t.

3.1 World Model

The world model captures the underlying changes and predicts the upcoming changes in the latent state. In this project, it is implemented as an LSTM, and the information generated is used for learning the representations for itself, and for updating the actor and predicting the reward, through latent imagination.

3.2 Encoder and Decoder Model

The encoder model captures observations from the environment and feeds them into the world model. The decoder reconstructs the embeddings so that reconstruction loss can be calculated and minimized to ensure the embedding and hidden state are meaningful.

3.3 Actor Model

The actor model converts a meaningful, embedding sampled from the encoder, as well as the hidden state generated by the world model, to an action that maximizes the long-term return to the best of its knowledge. Our actor is stochastic, meaning it outputs a Gaussian distribution rather than a vector from the NN.

3.4 Critic Model

The critic model estimates the long-term value from the hidden state and the embedding. We use the values of the critic model and the policy of the reward model to update the policy.

3.5 Dynamics Model

The dynamics model converts a hidden state generated by the world model into an embedding. This embedding should contain meaningful information so that it resembles the input at the current timestep.

3.6 Reward Model

The reward model uses the world model and its state to determine the predicted reward it will receive.

3.7 Termination Predictor

The termination predictor outputs 0 if the next state of the current state is terminal, and 1 otherwise. We get this information from the replay buffer.

3.8 Update Rules

We define the parameters of the world model as: θ. Then

$$\mathcal{L}_\theta = \sum_{t=1}^{T} \mathcal{L}_{\theta;pred} + \mathcal{L}_{\theta;dyn} + \mathcal{L}_{\theta;rep}$$

where

$$\mathcal{L}_{\theta;pred} = -log(d_\theta(x_t|z_t, h_t)) - log(r_\theta(r_t|z_t, h_t) - log(C_\theta(c_t|z_t, h_t))$$
$$\mathcal{L}_{\theta;dyn} = max(1, D_{KL}(sg(e_\theta(z_t|x_t, h_t))||m_\theta(\hat{z}_t|h_t))$$
$$\mathcal{L}_{\theta;rep} = max(1, D_{KL}(e_\theta(\hat{z}_t|x_t, h_t)||sg(m_\theta(\hat{z}_t|h_t)))$$
$$\mathcal{L}_{\theta;crit} = -log(V_\theta(v_t|z_t, h_t))$$

$sg(\cdot)$ is the stop-gradient operator and here it means $\hat{H}$ where its parameters $\hat{\psi}_t = \rho\psi_t + (1-\rho)\hat{\psi}_t$, with ρ being a float number close to 0. $sg(H)$ is also called the "target network" of H . For updating the actor and critic, we let the world model imagine 50 steps, only giving it the initial h, initial c, and the action input for each step, and let it unroll. This is called Latent Imagination. Specifically, we define $R_t = (1-\lambda)R_{t+1} + \lambda C_\theta(z_t, h_t)(r(z_t, h_t) + \gamma V(z_{t+1}, h_{t+1})) - \alpha H(\pi(a_t|z_t, h_t))$ where $\lambda = 0.95$, and $R_T = R_{T-1}$ whereas $T > 0$ is the terminal step and $H(\cdot)$ is the entropy regularizer described in the next subsection. Below is the pseudo-code for our algorithm.

Algorithm 1 DLC-V1

```
 1: Initialize parameters $\theta, \phi$, $\tau = 0.005$, $\lambda = 0.95$, $\alpha = 0.0006$, $\beta = 0.0001$
 2: Hard update $\hat{\theta}_e \leftarrow \theta_e$, $\hat{\theta}_m \leftarrow \theta_m$
 3: for i=1, 2,...,num_episodes do
 4:     while not $Terminate$ do
 5:         $action \leftarrow \{\}$
 6:         for x in agents do
 7:             Predict $z_t, h_t$ from the world model for each agent
 8:             $action[x] \leftarrow a_t^x \sim \pi_a(z_t, h_t)$
 9:             Add $(O_t^x, a_t^x, r_t^x, term_t^x)$ to agent's replay buffer
10:         $O_{t+1}, r_{t+1}, term_{t+1} \leftarrow$ env.step(action)
11:     for x in agents do
12:         for y=1,2,..., Training_step=200 do
13:             Sample sequences $\tau$ of batch size $B = 15$ and length $L = 50$ from replay buffer
14:             Use the encoder for each agent to predict embeddings $z_t, \hat{z}_t$
15:             Calculate $\mathcal{L}_\theta^x$ using the Update Rule
16:             Update the model by minimizing $\mathcal{L}_\theta^x$ with learning rate $\alpha$
17:             Soft-update on $sg(e_\theta(\cdot))$ by $\hat{\theta}_e = (1-\tau)\hat{\theta}_e + \tau\theta_e$
18:             Soft-update on $sg(m_\theta(\cdot))$ by $\hat{\theta}_m = (1-\tau)\hat{\theta}_m + \tau\theta_m$
19:             for imagination_step = 1,...,T=50 do
20:                 Sample the stochastic embedding $\hat{z}_t \sim m(\cdot|h_t)$
21:                 Choose $a_t^x \sim \pi(a_t|z_t, h_t)$
22:                 Propagate the deterministic component of state $h_{t+1}$ with $a_t$, $\hat{z}_t$ and $h_t$
23:                 Accumulate $\lambda$-return in Update Rule, $r_t^x \sim r^x(z_t, h_t)$ and $V_t^x \sim V^x(z_t, h_t)$
24:             Update the actor through gradient ascent, $\phi' = \phi + \beta\nabla_a R_1^T \sum_1^T \nabla_\phi \pi_\phi(a_t|z_t, h_t)$
25:             Update the critic by minimizing the critic loss using learning rate $\alpha$:
                    $\mathcal{L}_{\theta;crit} = -log(V_\theta(R_t|z_t, h_t))$
```

3.9 Entropy Regularization

We will be applying an entropy regularizer. The entropy regularizer is given as $min\ \mathbb{E}[log(\pi(a_t|z_t, h_t)) - log(1 - tanh(a_t)^2)]$, where $log(\pi(a_t|z_t, h_t))$ prevents the entropy(randomness of the action) from decreasing too quickly and encourages exploration while $log(1 - tanh(a_t)^2)$ penalizes extreme actions.

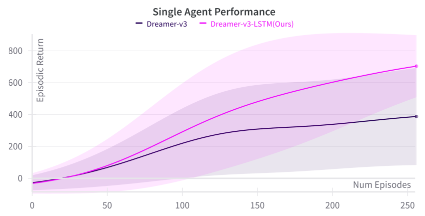
3.10 Model Dimensions

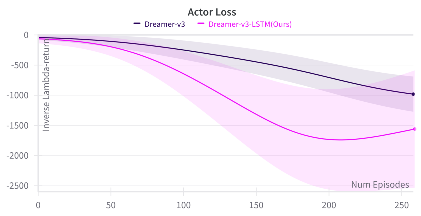
Please see the appendix.

3.11 Results

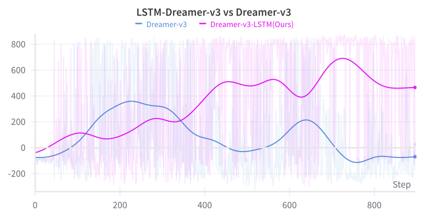
To understand the capabilities of our model (LSTM-Dreamer-v3) we have compared it against the baseline in two settings: a single-agent environment and a multi-agent environment. Each plot is generated with three seed runs and Gaussian smoothing.

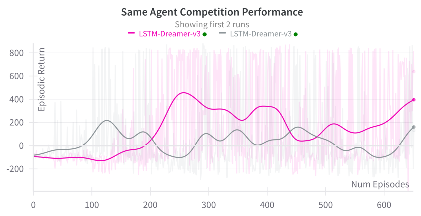
Single Agent Training Performance





Multi-Agent Training Performance





3.12 Discussion

Each agent is trained in a single-agent environment with a non-shared policy as the complexity and the dynamics of a multi-agent environment would render training a shared policy too slow in practice. The hyperparameters used by the LSTM-Dreamer-v3 are a batch size of 15, a sequence length of 50, and a horizon length of 50. The hyperparameters used by the Dreamer-v3 are a batch size of 50, a sequence length of 50, and a horizon length of 15.

Each model was trained using a different Reccurent State-Space Model: one with an LSTM cell (LSTM-Dreamer-v3) and one with a GRU cell (Dreamer-v3). The LSTM-Dreamer-v3 significantly outperforms Dreamer-v3 as its actor loss has a steeper slope. The bottom right loss plot in section 3.11 shows this. The LSTM-Dreamer-v3 can perform better because of its longer horizon imagination (it can generate longer sequences of potential future paths). The longer horizon imagination and LSTM's higher model capacity also generates better training signal for the actor.

The agents were trained in two steps. Firstly, training the model in a single-agent environment and selecting the best policy after a set number of runs. Secondly, we make the agent race against its best previous run (as a ghost). The policy model improves as better heuristics will lead to better cumulative reward – leading to the car beating its previous best time.

After training the agents, we made them compete in a race. Initially, both agents were perturbed by each other, but the LSTM-Dreamer-v3 recovered after about 120 runs and outperformed Dreamer-v3 significantly. This was due to its ability to produce longer horizon predictions than the Dreamer-v3.

3.13 Limitations

Our experiment is only limited to a two-agent competition. In the real world, each car may have a different dynamics model due to its unique design, and there might be other road users, such as bike riders and pedestrians. For self-driving vehicles to be safe and robust, its built-in model must be able to predict and interact with the environment and coordinate with the other road users safely. In this experiment, we have found that training a policy through long-term imagination outperforms short-term imagination with a higher batch size. Thus, creating such a world model is crucial in fully developing autonomous driving.

4 Ethical considerations

All the data are simulations within a gym environment. None of these simulations have access to real-world data.

The scope of the latent imagination in our project is limited to predicting the trajectories of the current car and its adversary. In real life, for the vehicle to be safe, the world model needs to be robust enough to predict the trajectories of other heterogeneous agents such as cars, bikers, or pedestrians to enable safe decision-making. This robustness needs to be sufficiently generalizable such that the model may generate out-of-distribution unsafe scenarios, evaluate those scenarios, and train the actor to prevent them from happening. This continues to be an ongoing challenge in the field.

5 Conclusion

In this work, we evaluated the performance of LSTM-Dreamer-V3 in a decentralized car racing environment. As a result, the LSTM-Dreamer-v3 outperforms the standard Dreamer-v3 model both in training as solo agents and in a competitive head-to-head race. This is because the LSTM-Dreamer-v3 has a higher model capacity, which enables it to use its longer imagination horizon to take actions that give it better long-term rewards. Thus, the LSTM-Dreamer-v3 is more robust from perturbation by other agents and dynamic components of a multi-agent environment. Additionally, penalizing each agent involved in a collision further improves the applicability of this experiment toward the development of autonomous vehicles. Thus, improving the model capacity and entropy regularization on the actor model can significantly improve the performance of a model-based Reinforcement Learning baseline involving imagination, like the Dreamer models.

References

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, nov 2017.

[2] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

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

[4] Sergey Levine. Cs284 lecture 4 - berkeley.

[5] Wilko Schwarting, Tim Seyde, Igor Gilitschenski, Lucas Liebenwein, Ryan Sander, Sertac Karaman, and Daniela Rus. Deep latent competition: Learning to race using visual control policies in latent space. In *Conference on Robot Learning*, 2020.

[6] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, jul 1991.

[7] Annie Xie, Dylan P. Losey, Ryan Tolsma, Chelsea Finn, and Dorsa Sadigh. Learning latent representations to influence multi-agent interaction. In *CoRR*, 2020.

Appendices

A Network Architecture

World Model	
Network Type: LSTM	**Embedding Dim:** 1024
Initial h and c: Learned	**Output Dim:** 512
Representation Model(Sub-part of the encoder model)	
Network Type: MLP	**Embedding Dim:** 512
Activation: SiLU	**Hidden Dim:** 1024
Output Type: Gaussian	**Output Dim:** $\mu(60)$, $\sigma(60)$
Num Layers: 3	**Batch Norm:** Every Layer
Transition Model	
Network Type: MLP	**Input Dim:** 1024
Activation: SiLU	**Hidden Dim:** 1024
Output Type: Gaussian	**Output Dim:** $\mu(512)$, $\sigma(512)$
Num Layers: 3	**Batch Norm:** Every Layer
Encoder	
Network Type: CNN	**1st Layer:** Conv2D(in=3, out=32, k=4, s=2)
2nd Layer: Conv2D(in=32, out=64, k=4, s=2)	**3rd Layer:** Conv2D(in=64, out=128, k=4, s=2)
4-th Layer: Conv2D(in=128, out=256, k=4, s=2)	**5-th Layer:** Linear(in=4096, out=1024)
Activation Type: LeakyReLU	**Output Type:** Deterministic
Batch Norm: Every Layer	
Decoder	
Network Type: Transpose CNN	**1st Layer:** Linear(1024, 4096)
2nd Layer: Unflatten((-1, 8, 8))	**3rd Layer:** ConvTranspose2d(in=64, out=64, k=3, s=1)
4th Layer: ConvTranspose2d(in=64, out=32, k=4, s=2)	**5th Layer:** ConvTranspose2d(in=32, out=3, k=12, s=4)
Activation Type: LeakyReLU	**Output Type:** Gaussian
Output Dims: $\mu(3, 96, 96)$, $\sigma(1)$	**Batch Norm:** Every Layer
Note: $\sigma(1) \perp\!\!\!\perp$ input and $\sigma(1)$ is learned	$D \sim \mathcal{N}(\mu, e^{\sigma(1)})$, D is reconstructed image
Actor	
Network Type: MLP	**1st Layer:** Linear(512+60, 512)
2nd-4th Layers: Linear(512, 512)	**5-th Layer:** Linear(512, ($\mu(2)$, $\sigma(2)$))
Activation Type: SiLU	**Output Type:** Gaussian
Batch Norm: Every Layer	
Reward	
Network Type: MLP	**1st Layer:** Linear(512+60, 512)
2nd Layer: Linear(512, 512)	**3rd Layer:** Linear(512, ($\mu(1)$, $\sigma(1)$))
Activation Type: SiLU	**Output Type:** Gaussian
Batch Norm: Every Layer	
Continue Predictor	
Network Type: MLP	**1st Layer:** Linear(512+60, 512)
2nd Layer: Linear(512, 512)	**3rd Layer:** Linear(512, 2)
Activation Type: SiLU	**Output Type:** Bernoulli
Batch Norm: Every Layer	
Critic	
Network Type: MLP	**1st Layer:** Linear(512+60, 512)
2nd Layer: Linear(512, 512)	**3rd Layer:** Linear(512, ($\mu(1)$, $\sigma(1)$)))
Activation Type: SiLU	**Output Type:** Gaussian
Layer Norm: Every Layer	