EN.520.637
Project Report: Training Bipedal Walker In Latent Space

Jiarui Wang (jwang486)

1 Introduction

In reinforcement learning, model-free methods has been extensively studied. These model-free methods can be divided into three categories: value function-based methods [1, 2, 3], policy-based methods [4, 5], and hybrid methods [6]. These model-free methods have achieved huge success across all kinds of tasks, including continuous control, atari games and chess playing. Model-free methods, however, have some drawbacks.

Partially Observed Problem The first drawback of model-free methods is the full-observation assumption. In standard Markov Decision Process, it's assumed that the current situation of the agent can be fully described by the state s. But in lots of real-world applications, this assumption is not true. For example, in video games, we can only observe the current video frame which is static, but a single frame cannot precisely describe the state since it lacks some information like the motion. There have been works to overcome this problem by stacking several past frames together as the current state, but it highly depends on the granularity of the dynamics of the task.

Planning The second drawback of model-free methods is that model-free methods cannot perform planning. In most model-free methods, an actor is trained to map from the state to an action. Once an action is chosen, it has no idea what could happen next. In some real-world applications, for example, safety-critical tasks like self-driving, planning is important to avoid potential lethal states.

RSSM To overcome these problems, there have been works focusing on model-based reinforcement learning methods where the dynamics are explicitly modeled. [7] proposed Recurrent State Space Model (RSSM), where the observation is mapped into a latent space, the dynamics in the latent space are modeled with both deterministic and stochastic transition components, and the model is learned with maximum likelihood estimation by maximizing the evidence lower bound. The author also proposed a planning algorithm where trajectories are simulated up to a horizon by the learned model, and the action that maximizes the accumulated return in the simulated trajectory is chosen.

Dreamer In [7], even though they maximized the accumulated reward in a horizon, it's still not enough for infinite-horizon tasks or long-horizon tasks. In [8] combine RSSM with reinforcement learning algorithm and proposed Dreamer, where the algorithm switches between model learning and actor-critic learning [9] so that the actor learns a policy to maximize the long-term accumulated reward, not just the accumulated reward up to a fixed horizon. The experiment showed that Dreamer surpassed lots of state-of-the-art methods in terms of sample efficiency.

This project, as a studying project, aims to investigate the theory of RSSM and Dreamer, and reproduce the result of Dreamer on walker task in DeepMind Control Suite[10]. For the purpose of learning, I make a light-weight and readable implementation for Dreamer, the code can be found at `https://github.com/bstars/Dreamer-Walker`

2 Recurrent State-Space Model

Recurrent State-Space Model(RSSM), proposed in [7], tends to learn the dynamics in latent space from (partial) observations. The model consists of the following parts:

- $h_t = f(h_{t-1}, s_{t_1}, a_{t-1})$: The deterministic state model, represented by a recurrent neural network.
- $P(s_t|h_t)$: The stochastic state model (prior), represented by a diagonal Normal distribution where the mean and diagonal variance are computed by a small neural network from h_t.
- $q(s_t|h_t, o_t)$: The representation model (posterior), represented by a diagonal Normal distribution where the mean and diagonal variance is computed by a small neural network from h_t and the CNN embedding of o_t.
- $P(o_t|h_t, s_t)$: The observation model, represented by a Normal distribution with identity variance and the mean is computed by a transpose convolution network from h_t, s_t.
- $P(r_t|h_t, s_t)$: The reward model, represented by a Normal distribution with identity variance, and the mean is computed by a small neural network from h_t, s_t.

The structure of RSSM is shown in

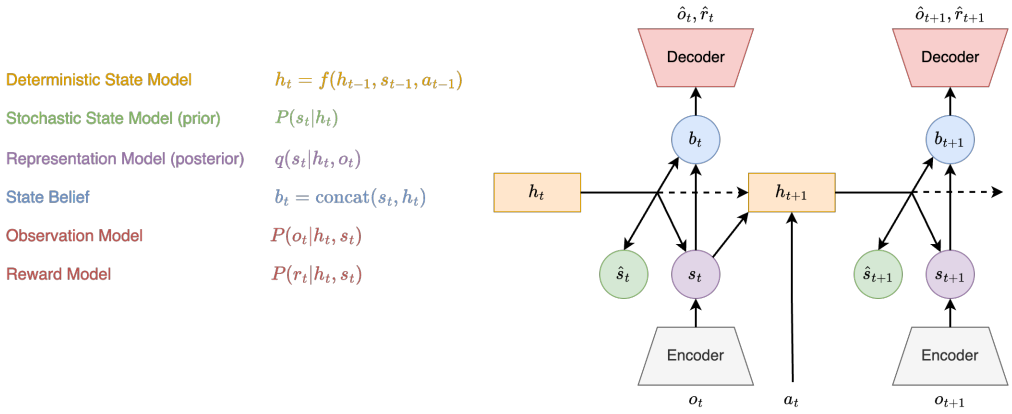

Figure 1: Structure of Recurrent State-Space Model

Intuitively, h is the deterministic part of the state, for example, the position and velocity of an item. After an action is applied, h_t transitioned to h_{t+1} in a deterministic way (for example, by Newton's law). However, due to the randomness of the environment, the actual state is not exactly h_{t+1} but follows a prior distribution $P(s_{t+1}|h_{t+1})$. After observing the observation o_{t+1} produced by the actual state, we can update our belief about the actual state and get a posterior distribution $q(s_t|h_t, o_t)$.

RSSM model can be trained with maximum likelihood estimation by maximizing the evidence lower (ELBO) bound [11]. The ELBO can be computed as

$$
\begin{aligned}
\log P(o_{\leq T}|a_{<T}) &= \log \sum_{s_{\leq T}} P(o_{\leq T}, s_{\leq T}|a_{<T}) \qquad (2.1)\\
&= \log \sum_{s_{\leq T}} q(s_{\leq T}|o_{\leq T}, a_{<T}) \frac{P(o_{\leq T}, s_{\leq T}|a_{<T})}{q(s_{\leq T}|o_{\leq T}, a_{<T})} \\
&\geq E_{s_{\leq T} \sim q}\left[\log \frac{P(o_{\leq T}, s_{\leq T}|a_{<T})}{q(s_{\leq T}|o_{\leq T}, a_{<T})}\right] \\
&= E_{s_{\leq T} \sim q}\left[\log P(s_{\leq T}|a_{<T}) + \log P(o_{\leq T}|s_{\leq T}, a_{<T}) - \log q(s_{\leq T}|o_{\leq T}, a_{<T})\right] \\
&= \sum_{t=1}^{T} E_{s_t \sim q(s_t|h_t, o_t)}\left[\log P(o_t|s_t, h_t)\right] - \sum_{t=1}^{T} KL\Big(q(s_t|h_t, o_t) \| P(s_t|h_t)\Big)
\end{aligned}
$$

In the above computation, we assume the dynamic in latent space is Markov $(s_t \perp s_\tau | s_{t-1}, a_{t-1})\ \forall \tau < t$, and we use h_t to summarize the history $(o_{<t}, a_{<t})$. Intuitively, at each time step, the RSSM formulation is similar to variational auto-encoder (VAE), where the observation o_t is the observed variable, s_t is the latent variable, and everything is dependent on the history h_t. We can also see that the ELBO is very similar to the ELBO in VAE.

For implementation, since $P(o_t|s_t, h_t)$ has identity variance, maximizing log-likelihood is implemented as minimizing $L2$ distance to avoid unnecessary computation.

3 Dream To Control

Dreamer, proposed in [8], combined RSSM with actor-critic learning. The dreamer model consists of

- A RSSM model
- $\pi_\phi(s_t, h_t)$: The actor outputs a tanh-transformed Normal distribution

$$a_t = \tanh(\mu_\phi(s_t, h_t) + \sigma_\phi(s_t, h_t) * \epsilon),\ \epsilon \sim \mathcal{N}(0, I)$$

- $v_w(s_t, h_t)$: The critic estimates the value function at state (s_t, h_t)

At each iteration, it learns the RSSM as in the previous section, rolls out a trajectory in latent space (as shown in Figure 2), and performs actor-critic learning with the imagined trajectory as in (3.1).

$$\max_\phi . \sum_{\tau=t}^{t+H} G_\tau^\lambda \qquad \min_w . \sum_{\tau=t}^{t+H} \left(v_w(s_\tau, h_\tau) - G_\tau^\lambda\right)^2 \tag{3.1}$$

where G_τ^λ is the $\lambda-$return at time t [12]. And since we have a model, we can back-propagate from the $\lambda-$return to the policy to maximize the $\lambda-$return instead of using other methods like policy gradient. The complete Dreamer algorithm is outlined in Algorithm 1.

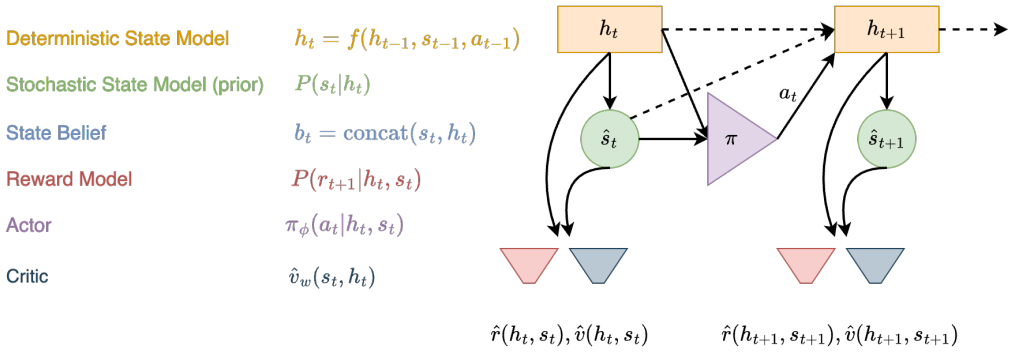

Figure 2: Roll out a trajectory in latent space

Algorithm 1 Dreamer

Initialize a replay buffer $\mathcal{D}$ with random episodes
while not converged **do**
  for t = 0,1,...T **do**

    /* Dynamics Learning */
    Sample a batch of transitions $\{o_t, a_t, r_{t+1}\}_{t=k}^{k+L}$ from $\mathcal{D}$
    Compute model states $\{(h_t, s_t)\}_{t=k}^{k+L}$
    Learning the RSSM model by taking a gradient step to maximize ELBO

    /* Behavior Learning */
    Imagine trajectory from each (h_t, s_t)
    Predict rewards with learned RSSM, predict value with critic
    Compute $\lambda-$return with predicted value and rewards
    Update actor and critic by Actor-Critic learning
  end for
  Collect an episode by learned actor and add to buffer $\mathcal{D}$
end while

4 Model Predictive Control with Dreamer

After we train the dreamer, we have a model for the latent dynamics, then we can perform model predictive control (MPC) for planning. In [7], the author proposed to perform MPC to maximize the accumulated reward up to a fixed horizon, the problem is that we usually want to maximize the accumulated reward in long term. In Dreamer, in addition to the RSSM model with can predict the instant reward, we also have $v_w(s_t, h_t)$, an estimation for long-term return, so we can perform MPC to maximize the estimated long-term return. The MPC algorithm with Dreamer is outlined in Algorithm 2, it's very similar to Algorithms 2 in [7], except that we choose the best k actions with the highest $\lambda-$return instead of $H-$step reward.

Algorithm 2 Model Predictive Control with Dreamer

Input: Latent state h_t, s_t, planning horizon H, number of iteration T, number of samples N, number of good actions K
Roll out a trajectory with Dreamer from h_t, s_t, up to a planning horizon H and get the action $a_{t:t+H}$
Initialize a Normal distribution $q(a) \sim \mathcal{N}(a_{t:t+H}, I)$
for t = 0,1,...T **do**
  Sample N sequence of actions $\{a_{t:t+H}^{(i)}\}_{i=1}^{N}$ $q(a)$
  Roll out a trajectory $\{h_t^{(i)}, s_t^{(i)}, r_{t+1}^{(i)}, ..\}$ with the sequence of actions $\{a_{t:t+H}^{(i)}\}_i$ for each $i = 1, ..., N$
  Compute the $\lambda-$return G_i^{λ} for each trajectory at time t.
  Choose the top k sequences of actions that achieves highest $\lambda-$returns $\mathcal{K} = \text{argmax}\{G_i^{\lambda}\}_{i=1}^{N}[1:k]$
  $\mu = \text{mean}(\{a_{t:t+H}^{(i)} | i \in \mathcal{K}\}), \sigma = \text{std}(\{a_{t:t+H}^{(i)} | i \in \mathcal{K}\})$
  $q = \mathcal{N}(\mu, \sigma)$
end for
Return the action for the first time step in mean of q

5 Experiment

The experiment is performed with the Walker environment in DeepMind Control Suite [10]. Where we down-scale the raw pixel to a 64×64 RGB image as the observation.

We train the dreamer model for 100k iterations and sample a new episode every 100 iterations (1000 episodes in total). Figure 3 shows the accumulated reward averaged from 10 episodes along the training.

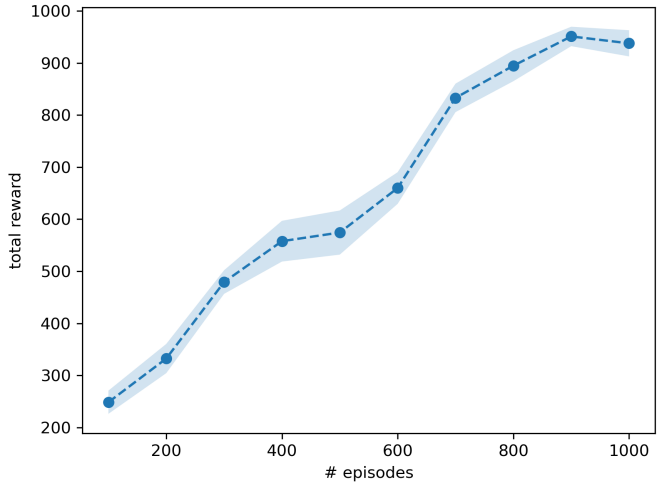

Figure 3: The accumulated rewards along the training

To demonstrate RSSM's ability to model the dynamics, we roll out a trajectory with Dreamer, plot the predicted observation, and compare it with the real trajectory produced by applying the same actions as the Dreamer's output actions. The comparison is shown in Figure 4

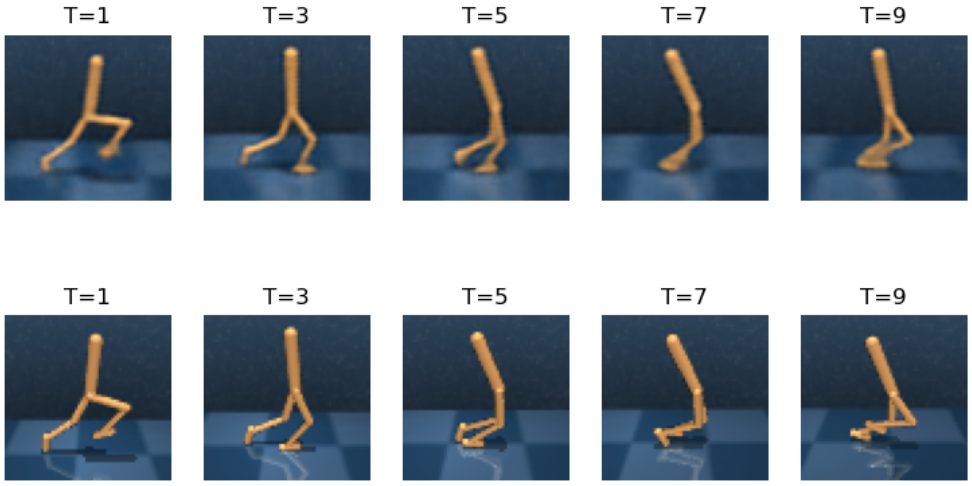

Figure 4: The first row is the imagined trajectory by Dreamer, the second row is the real trajectory produced by applying the same action as Dreamer's output action

We also compare the performance of one-step control (the actor in Dreamer) and the model predictive control (algorithm 2). Figure 5 shows the accumulated reward averaged from 10 episodes for different methods. We observed that the MPC controller improves the performance in the early training stage but

eventually the difference becomes negligible. We think that if we collect new episodes in training with an MPC controller, the algorithms will converge faster.

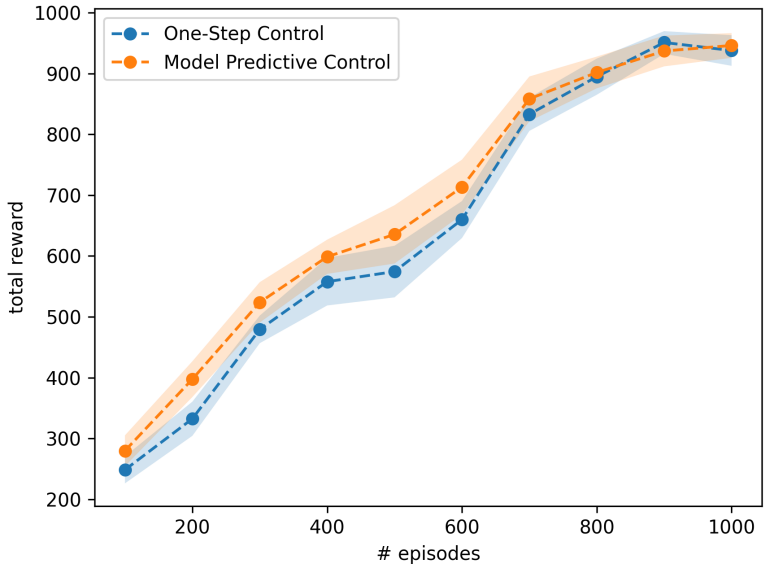

Figure 5: The comparison between one-step controller and model predictive control

6 Conclusion And Future Work

In this project, I got a solid understanding of RSSM, Dreamer, and other state-of-the-art model-based reinforcement learning algorithms. Due to the limited time and computation resources, this project lacks a comparison between Dreamer and other methods.

The future work will include the following:

- The Dreamer algorithm follows the same pattern as Bayesian Optimization: model fitting, optimizing using the model, data collection, and refitting the model. If the reward predictor outputs a confidence interval for the reward, we can try maximizing the Upper Confidence Bound of the $\lambda-$return. By doing this, we try to behave in the face of optimism, and hopefully, we can get an algorithm similar to GP-UCB and a better balance between exploration and exploitation.
- When we have full observation, we can still use the Dreamer algorithm. But we can try adding a line in each iteration to learn a control barrier function so that we'll co-learn an actor, a critic, and a control barrier function and we'll get a safe agent.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[3] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

[4] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[5] Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[7] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[8] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[9] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[10] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[12] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.