Ablation: Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning

Chao Yu Huang, Yu Lun Wang, Guan Ting Liu
Department of Computer Science
National Chiao Tung University
b608390.cs08g@nctu.edu.tw, useseldom@gmail.com, bts89602@gmail.com

1 Problem Overview

The implementation task for our team is the ablation study of the paper "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning" [Nagabandi et al. [2017]], in the first section, we are going to provide a problem overview of our project. For our project code, please reference https://github.com/brian220/neural-mpc

- The main research problem tackled by the paper
  Locomotion task is to simulate the motion of animals moving from one place to another, it contains the continuous states and actions. This task can be solved by model-free deep reinforcement learning, but in order to achieve good performance, the algorithm need a very large number of samples.
  In this paper, the author want to achieve excellent sample complexity and produce stable and plausible gaits to accomplish various of locomotion tasks in the Mujoco environment.

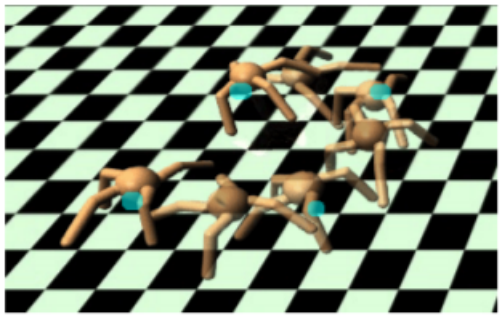


  Figure 1.1 One of the locomotion task is to make the robots learned the walking motion and walk through the user-defined points.

- High-level description of the proposed method
  This paper apply the combination of model-based and model-free approaches. The algorithm use the model-based approaches to initialize a model-free learner, for the model-based approach, a dynamic model is used to simulated the environment and the model predictive control (MPC) is used to choose action based on a random-sampling method. For the model-free approach, Trust Region Policy Optimization (TRPO) [Schulman et al. [2015]] is used to find the optimal policy based on the trajectories learned from the model-based approach. This combination can benefit from the sample efficiency of model-based methods and the high task-specific performance of model-free methods.

Preprint. Under review.

2 Background and The Algorithm

- Optimization Goal
  The goal in this paper is to learn a policy that maximize the sum of future rewards. At each time step t, the agent is in state $s_t \in \mathcal{S}$, (different motions of animals in the locomotion task), executes some action $a_t \in \mathcal{A}$, receives reward $r_t = r(s_t, a_t)$, and the discounted sum of future rewards is $\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_t, a_t)$, where $\gamma \in [0, 1]$. The reward function r in this paper is predefined according to the different Mujoco locomotion tasks.

3 Detailed Implementation

- The system architecture

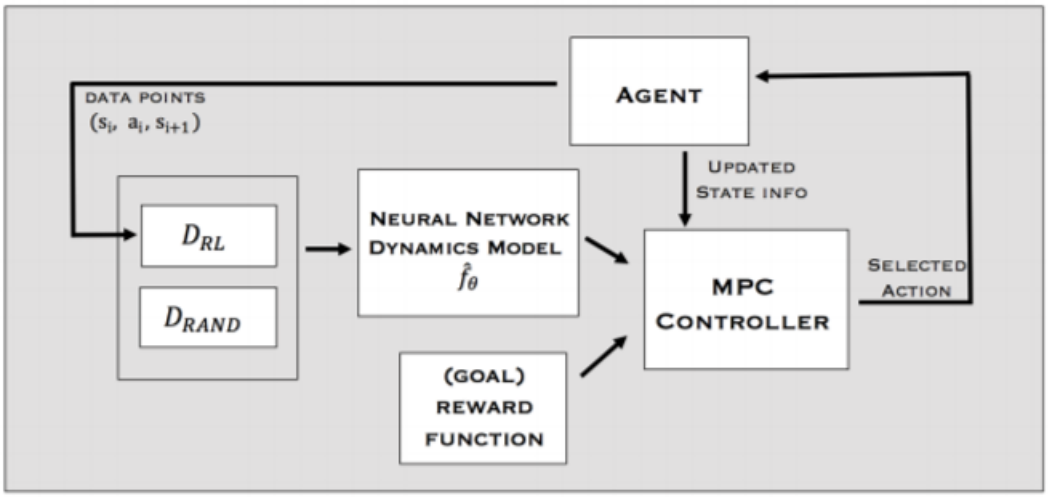


Figure 3.1

Figure 3.1 is the system architecture of the algorithm and following is the short descriptions for each component:

$\mathbf{D_{RAND}}$ The state-action pairs (s_t, a_t) are split from the random trajectories. The label of this data set is s_{t+1}, which is the next state obtained from interacting with the environment.

$\mathbf{D_{RL}}$ The state-action pairs (s_t, a_t) learned by the MPC controller. The label of this data set is s_{t+1}, which is the next state obtained from interacting with the environment.

Neural Network Dynamics Model The neural network trained for simulating the environment.

Reward Function Predefined reward function, return the reward based on (s_t, a_t).

MPC Controller Model predictive controller, which select the action based on Dynamics Model and reward function.

Agent The robot we want to control and the Mujoco environment.

- Model-based deep reinforcement learning

Dynamic model

The dynamic model is the neural network which is used to simulate the environment. A straightforward implementation is making the input of model be the information of current state and current action (state-action pair (s_t, a_t)) and predict the information of next state ($\hat{s}_{t+1}$). However, this function can be difficult to learn when the states s_t and s_{t+1} are too close, so the effect of the action is seemingly little on the output. So in this paper, instead of predicting the next state, the dynamic model predict the difference between the current state and next state. Which is:

$$\hat{s}_{t+1} = s_t + \hat{f}_\theta(s_t, a_t)$$

The $\hat{f}_\theta(s_t, a_t)$ represent the dynamic functions learned by the dynamic model. Based on this function , the training error will be:

$$\varepsilon(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}} \frac{1}{2} ||(s_{t+1} - s_t) - \hat{f}_\theta(s_t, a_t)||^2$$

And for our Tensorflow implementation:

```
tf.square(self.delta_pred - self._deltas)
```

The `self.delta_pred` is the output of the model and `self._deltas` is the ground truth difference between states. We will summarize this term and compute the mean value, the mean value will be the error of our model. Then use tf.sess.run() function in Tensorflow to train the model. And In section 4 of the report, we will evaluate and compare the performance of predicting s_{t+1} and predicting $s_{t+1} - s_t$.

MPC controller

In order to learn model $\hat{f}_\theta(s_t, a_t)$, together with the reward function $r(s_t, a_t)$ the author formulate a model-based controller, the controller try to optimize the sequence of actions $A_t^{(H)} = (a_t, .., a_{t+H-1})$ over the horizon H, using the learned dynamic model to predict the future states:

$$A_t^{(H)} = \arg\max_{A_t^{(H)}} \sum_{t'=t}^{t+H-1} r(\hat{s_{t'}}, a_{t'}) :$$

$$\hat{s}_t = s_t, \hat{s}_{t'+1} = \hat{s}_{t'} + \hat{f}_\theta(\hat{s}_{t'}, a_{t'})$$

Because the dynamic function and reward function are linear, so the optimum for this equation is difficult to calculate, but an approximate solution will be sufficient for the disired task. The author use a simple random sampling method, in which K candidate sequences are randomly generated. And the next state can be predicted by the dynamic models we talked about in the previous subsection, so the approximate accumulated reward can be computed. Then the model predictive control (MPC) will be used: The policy executes only the first action a_t, receives updated state information s_{t+1}.

Data aggregation

The model predictive control in the previous subsection will choose action for the each state in every iteration. After each iteration, these state-action pairs (s_t, a_t) will be appended into the training data set for the next iteration, these data is called D_{RL}, these appending work is called data aggregation in the original paper. And we will evaluate this part in section 4.

Pseudo code

Following is the pseudo code for the model-based deep reinforcement learning in this paper:

Algorithm 1 Model-based Reinforcement Learning

gather dataset $\mathcal{D}_{\text{RAND}}$ of random trajectories
initialize empty dataset $\mathcal{D}_{\text{RL}}$, and randomly initialize $\hat{f}_\theta$
for iter=1 **to** max_iter **do**
  train $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$ by performing gradient descent on Eqn. 1, using $\mathcal{D}_{\text{RAND}}$ and $\mathcal{D}_{\text{RL}}$
  for $t = 1$ **to** T **do**
    get agent's current state $\mathbf{s}_t$
    use $\hat{f}_\theta$ to estimate optimal action sequence $\mathbf{A}_t^{(H)}$ (Eqn. 3)
    execute first action $\mathbf{a}_t$ from selected action sequence $\mathbf{A}_t^{(H)}$
    add $(\mathbf{s}_t, \mathbf{a}_t)$ to $\mathcal{D}_{\text{RL}}$
  end for
end for

- Initialize model-free learner

  And to improve the performance, author also use the model-free approaches, this part can be divided to two parts, first, initialize the policy by training a deep neural network, the network can predict the policy $\pi_\phi(a|s)$ which can match the "expert" trajectories learned from the model-based method. Second, use trust region policy optimization (TRPO) to find out the best policy. Following is the introduction of these two parts:

  Policy Initialization

  After the training done for the model-based controller (a.k.a the MPC controller) We can use this controller to interact with the environment and collect the trajectories. These trajectories can be thought as a set of good motion data since the model-based controller has done well for playing this game. Now, we can initialize a Gaussian policy(a neural network) by supervised learning of these training trajectories. If trained well, this policy can do as good as the MPC controller.

  For reinforcement learning (TRPO)

  Since we have used the neural network trained by supervised learning to make this policy 'initialized' to become the implicit policy under model, we can use this policy to be trained by reinforcement learning again. Although the original paper use TRPO, but here, we implement Proximal Policy Optimization (PPO) [Schulman et al. [2017]] from scratch. Because PPO is the algorithm which is simpler to implement.

4 Empirical Evaluation

- Environment Setting
  The operating system we use is Linux ubuntu 16.04, and the GPU resources are GTX 1080Ti, GTX 1080, GTX 970. And the code we reference is `https://github.com/aravindsrinivas/neural-mpc`, which contains the implementation of the model-based deep reinforcement learning part. The most difficult part for the environment setting is to install the correct version packages to run the project, after much time of trying, we find out all the correct versions. The setting will be recorded in our github, which will help if someone want to run this project in the future.

- Hyperparameters
  Mujoco environment: "HalfCheetah-v1"
  Learning rate for dynamic models: 1e-3
  Episode Length: 1000
  Training iteration: 15
  Horizon (H): 15
  Random sampling action sequence number (K): 1000
  The H and K have been both introduced in the section 3. We use the parameters above and the method described in the paper to train our baseline model, the average reward for the baseline model in each iteration will be drawed as the blue line with the range [min reward, max reward]. And in the following section, we are going to change the training algorithm and hyperparameters then compare the results with our baseline model.

- Ablation study on the model-based approach

  Compare the different error function for the dynamic model
  In section 3, we have discussed that instead of predicting the next state data, this paper predict the difference between the current state and next state. We compare both of the error setting and the following figure is the result:

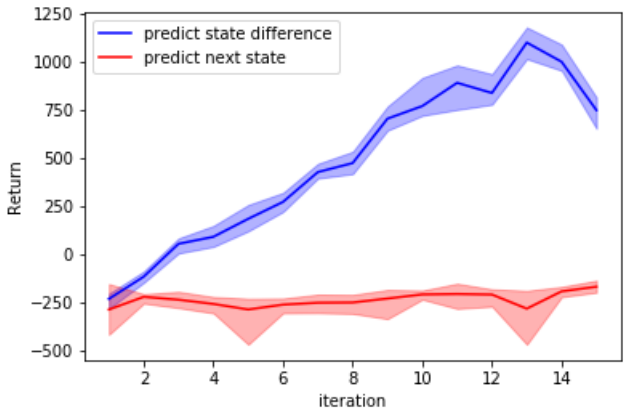


Figure 4.1

By Figure 4.1, we observe that the blue line grow faster than the red one, which shows that by predict the difference between state will give out a better result and although the red one is also growing, the growing speed is quite slow.

Effect on data aggregation

In section 3, we have talked about that this paper used data aggregation to improve the performance, so we change the code by only training the dynamic model by D_{RL} get from the current iteration, so the old data will not be used, we compare this with the result with the baseline:

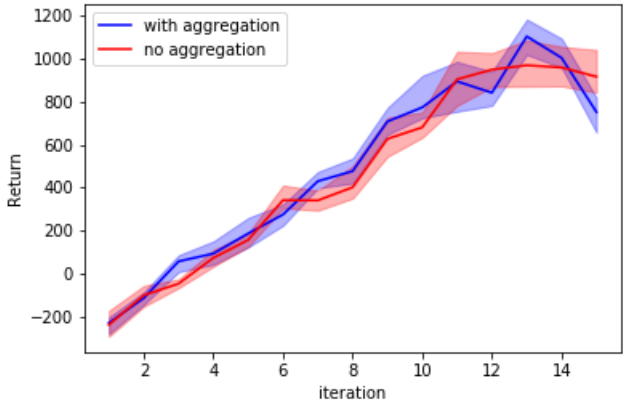


Figure 4.2

By observing Figure 4.2, the baseline (blue, with aggregation) is more stable and outperform the red line (without aggregation) in the most of iterations.

Changing the hyperparameters

We also change the different hyperparameters and compare the results with the baseline. First change the training episode length for each iteration:

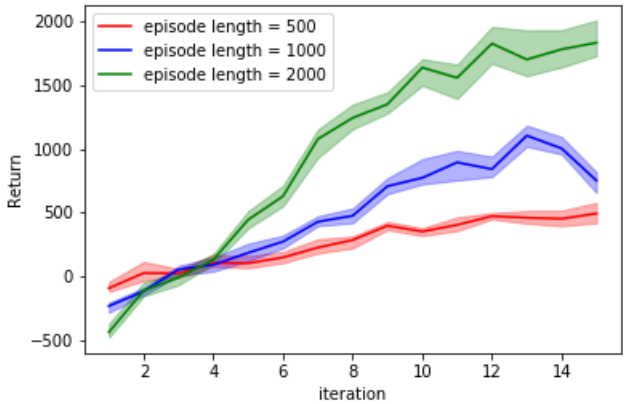


Figure 4.3

We think this result is trivial because the more training episodes means the more times we trained our dynamic models, which can bring the better result.

Then we change H and K to compare the results, first for H:

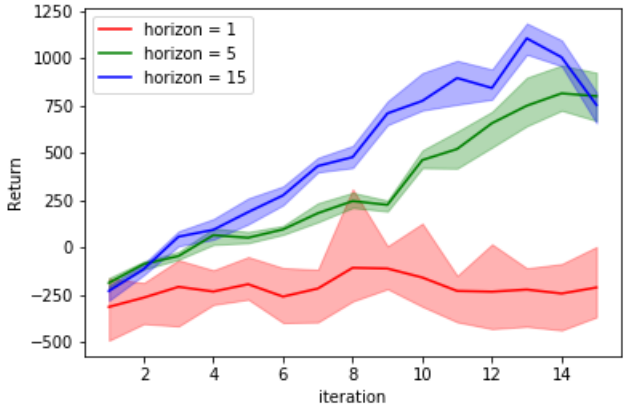
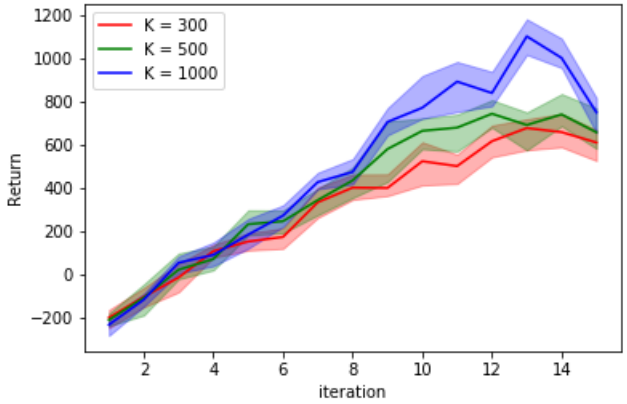


Figure 4.4

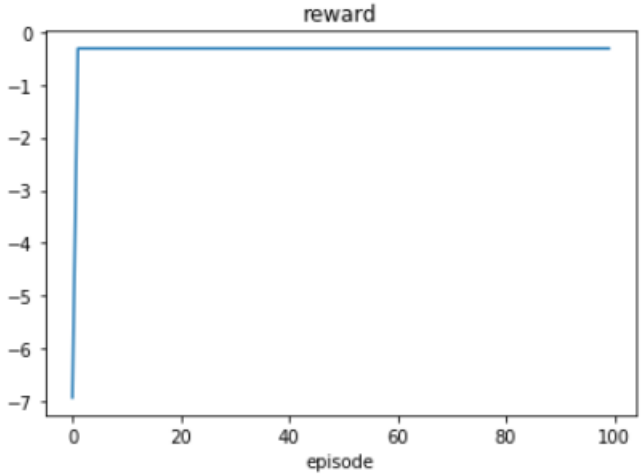
This Figure shows the usage of horizon in this work, and we can observe that, in our task the horizon from 1 to 5 will have a larger performance improvement compare to the rise from 5 to 15, so we can give a suggestion that if keep increasing the horizon number, we may achieve an optimal H that will bring the optimal performance.

Then for K:



We also change the number of action sampling sequence(K) in MPC, and we find out that for our experiment, when the K is bigger then 300, increasing the K's number can't give a significant improvement on the performance.

- Model-free reproduction
  For the model-free reproduction part, because we couldn't get the code, so we planned to implement the model free part by PPO from scratch, but unfortunately, we didn't success to get a nice result in the end. Following is the training result we get:



  The result converged very quickly and do not get a good result. Maybe it's a chance that we reproduce this part by applying the PPO baseline which provided on the internet and train our model again.

5 Conclusion

- The potential future research directions
  For our project, the first thing that can be improved in the future is the implementation of the model-free part, because we didn't finish that part in this project. And for the model- based part, there may be some method that is better then sampling K random action sequences to choose the action, other algorithm can be applied to improve the performance. And we also want to figure out under what conditions, the model-free algorithm can really outperform the model-base algorithm.
- Any technical limitations
  The technical we meet is when training the policy initialization network in the model-free method, we need to sampling the trajectories from the model-based output, but we find out that if we sample 1000 trajectories, it will take us up to two days to train the model, and if we use less trajectories, we are worried that we may not be able to get the well-trained model due to the lack of training data. So improving our GPU resource may be the solution for this problem and we need to get more understanding and training experience in training the PPO and policy gradient methods. One approach to achieve this is by changing to more simple environment with less states and actions.

- Any latest results on the problem of interest
  The locomotion problem is a challenge problem, until now, many researchers want to solve the problem by reinforcement learning. In 2018, [Yu et al. [2018]] takes the minimalist learning to the learning problem and provide a locomotion that is more symmetric and looks like human. And in 2020, instead of simulating the locomotion in virtual environment, some researchers applied the reinforce learning on the real-world robot the robot can learned the locomotion skills in 5 minutes and can navigate automatically through a curved path. Please reference to `https://ai.googleblog.com/2020/05/agile-and-intelligent-locomotion-via.html` for detail description of this project.

References

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596, 2017.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Wenhao Yu, Greg Turk, and C. Karen Liu. Learning symmetry and low-energy locomotion. *CoRR*, abs/1801.08093, 2018.