Reinforcement Learning with Initialized Policy

Zihan Ding

March 23, 2019

Contents

Abstract

How can we apply reinforcement learning algorithms for robot control in most efficient way?

1 Background

2 Reinforcement Learning Algorithms

2.1 Discrete and Continuous Action Space

2.2 Deterministic and Non-deterministic Policy

There are two types of action choice in a RL policy, the deterministic and the non-deterministic. The deterministic policy is that the action values are directly determined by the outputs of neural networks. Algorithms including DQN and DDPG are deterministic policy. The non-deterministic policy is that the action values are sampled from a distribution parametrized by the direct outputs of neural networks, therefore the action values themselves are not directly determined by the outputs of neural networks. Algorithms including REINFORCE, actor-critic, TRPO and PPO are non-deterministic policy.

2.3 Present Algorithms

2.3.1 REINFORCE

2.3.2 Actor-Critic

2.3.3 Q-learning

2.3.4 DQN

2.3.5 DDPG

2.3.6 TRPO

2.3.7 PPO

3 Efficient Reinforcement Learning

3.1 Environment

In this work, we apply reinforcement learning algorithms on the *Reacher* task. As shown in Fig.1, the goal of *Reacher* task is to let the end of 'reacher' agent to be as close as possible to the positions with highest rewards.

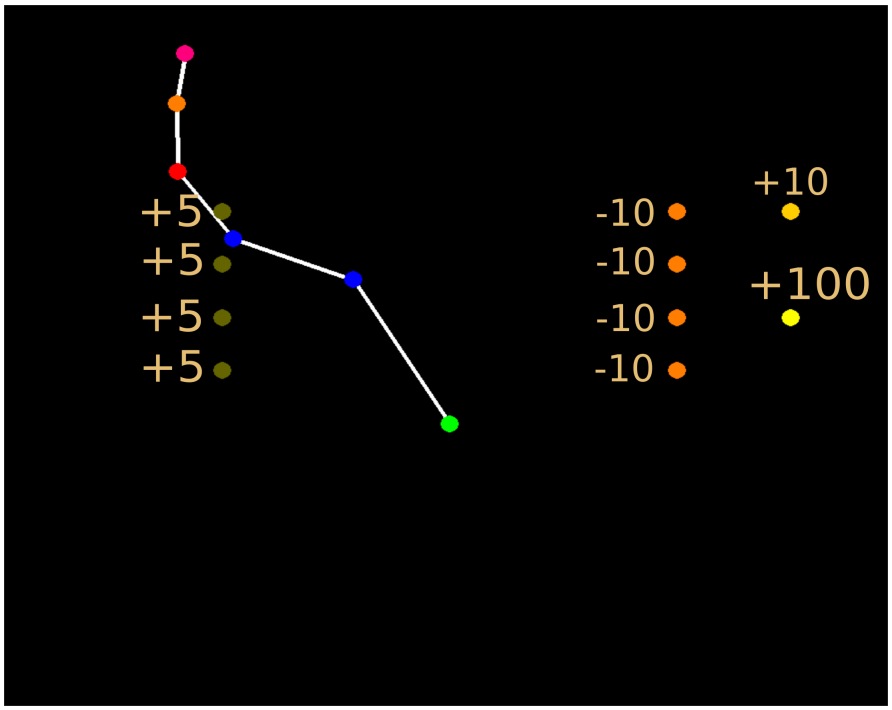

Figure 1: *Reacher* task with 5 joints and several reward/penalty positions

3.2 Initialized Reinforcement Learning with Supervised Learning Policy – Policy Replacement

Reinforcement learning algorithms usually take a long time to achieve a good policy for a specific task. And the policy always needs to be re-trained if the settings of the task are changing, even if slightly.

There are several ways to improve the learning efficiency of reinforcement learning algorithms. One promising way is to initialize the reinforcement learning policy with a pre-trained policy. And this is especially effective when the pre-trained policy is trained on a similar task with the one the reinforcement learning is trying to solve. Moreover, if the pre-trained policy is trained across the task space containing all possible settings of a specific task that the RL is solving, it can leverage the learning performance of RL for all tasks defined in this task space.

3.2.1 Task Specification

In our model, we try to initialize RL with sub-optimal policy pre-trained before the RL process, and the policy is trained with supervised learning with expert trajectories sampled from inverse kinematics on *Reacher* tasks.

The task space in our model is defined to be the same 'reacher' structure with one target point at different positions. The number of joints of 'reacher' is chosen to be 3 with length between each pair of joints to be 200, 140 and 100, respectively. The screen size is 1000 and the fixed start point of the 'reacher' is at the center of the map. The target position is sampled from a random distribution within the allowed area. The reward function for each step is defined as:

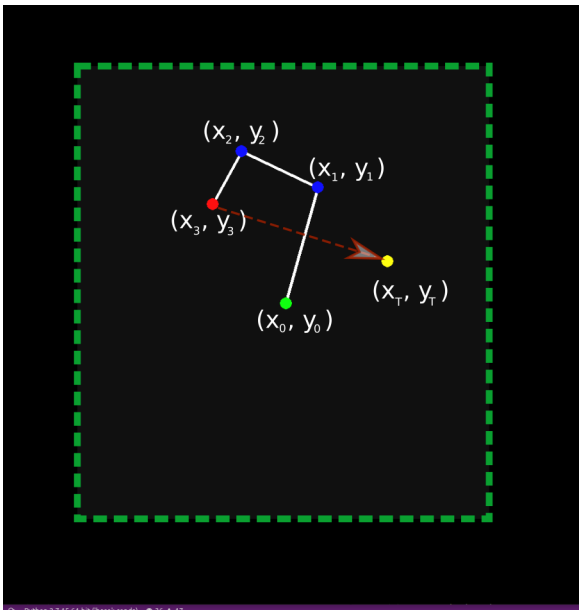

Figure 2: *Reacher* task with three joints and one target point of different possible positions. The green dashed line is the area of potential target positions in the task space.

$$R = \frac{R_0}{\sqrt{(x_3 - x_T)^2 + (y_3 - y_T)^2} + 1}, \tag{1}$$

where $R_0 = 100$ is the maximum of reward value, so $R \in (0, 100]$

3.2.2 Inverse Kinematics

Inverse kinematics is a inverse process of forward kinematics, to derive the joint angle values with respect to the end position. Considering the end position as a transformation function of joint angels: $\mathcal{T}(\theta, L)$, where $\{\theta_i\}$ are joint angles and $\{L_i\}$ are link lengths, the forward kinematics is:

$$(x_{end}, y_{end}) = \mathcal{T}(\theta, L) \tag{2}$$

while the inverse kinematics needs the Jacobian of end position with respect to the joint angles parameters $\{\theta_i\}$, and to reach a specific target position (x_T, y_T), the expected change of angle values are:

$$(\Delta(x_{end}), \Delta(y_{end})) = \alpha(\frac{\partial(\mathcal{T}(\theta, \mathrm{L}))}{\partial\theta})^{-1} \cdot (x_T, y_T) \tag{3}$$

where α is the step size.

And in practice, we usually use Moore–Penrose pseudo-inverse for calculation of the inverse Jacobian, which will induce approximation error in trajectories.

3.2.3 Supervised Learning Policy as Initialization

We apply a neural network of 5 layers to train an initialization policy, with the data generated from inverse kinematics. The inputs of neural network are positions of joints and position of the target.

Each hidden layer contains 100 nodes and uses ReLu as activation function. The output layer is Tanh activated with a scaling factor of 360, to make outputs range from -360 to 360 degree as a reasonable action value of the joints. The AdamOptimizer is applied with stochastic gradient descent in the training process. The training curve is shown in Fig. 3. Note that the y-axis is transformed to be in range [-1,1] instead of [-360, 360] as the mean squared error per action value .

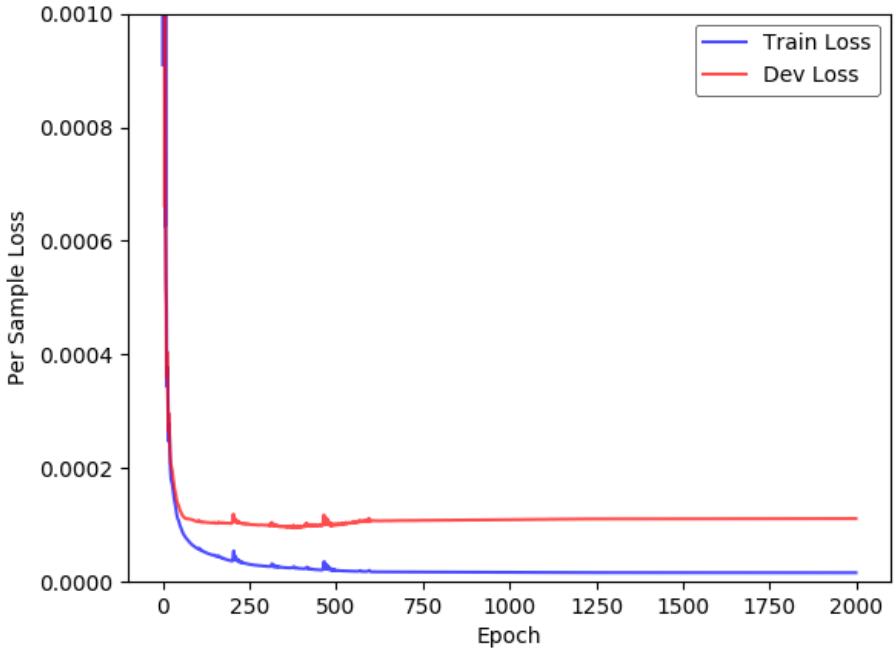

Figure 3: Training curve of supervised learning with data from inverse kinematics.

Here are several specifications about following experiments: (1). whenever there is a preheating process, the initial 600 steps in the learning curves are only for updating the critic (no actor update), so there is very small variance in learning curves when the noise applied is small; (2). as we are exploring the initialized policy for RL learning process, we only show experiments about the initial training steps (e.g. 2000 steps) for displaying the initialization effects.

3.2.4 Policy Replacement

Policy replacement is one approach to apply the policy from supervised learning in initialization of RL. The basic idea is to use the same network structure in supervised learning and the actor in RL, so that a copy can be made from the supervised learning policy to the actor as an initialization of RL. Some tricks can be applied in this process to guarantee a performance improvement, like the preheating process.

However, potential problems exist in policy replacement approach for initialization of RL, e.g. the initialized policy is fragile in the initial training process even with small exploration noise scale, which means the initialized good policy may degrade to some worse polices (the decrease on learning curve at initial training stage) and get improved through a re-learning process. We analysed the reason for

the problems and proposed some strategies for solving or alleviating it. Further analysis seen in the following subsections.

3.2.5 DDPG with Policy Replacement – General Process

We apply the DDPG algorithm for the task, and show the comparisons with or without initialization for the policy.

The DDPG algorithm contains four neural networks: the actor network, the critic network, the target-actor network, and the target-critic network. For initializing the DDPG, we define the actor network and target-actor network to have exactly the same structure as the network in supervised learning.

Initialization process of DDPG with supervised learning policy are as follows: (1). the weights of pre-trained supervised learning policy are loaded into the actor network and the target-actor are updated immediately; (2). samples are generated with initialized but frozen actor network with noise to pre-train the critic network; (3). train both the actor and critic network (also the target-actor and target-critic networks). Experiments show that the second step is non-trivial process for effective training DDPG with initialization. Details are explained in Sec. 3.2.7.

Comparison of initial learning performances of DDPG with and without initialization policy on *Reacher* task are shown in Fig. 4. The DDPG with initialization policy could significantly outperforms the one without initialization.

There are several key components in initializing the RL with supervised learning policy affecting the performance of initialized RL (DDPG), including the scale of action noise in initialized RL, different ways of normalization in supervised learning and RL, learning rates of the actor and the critic in initialized RL, number of steps of single episode and so on. Improper settings of all above factors will affect the initialization policy to be effective for RL process. Some general intuitions about making the initialization more effective for RL are like: ensuring the inputs and outputs of the replaced policy from initialization to be as similar as possible between the supervised learning and RL, therefore applying the same input and output normalization to make sure they are of the same ranges of value, the same initial positions for each episode in 'Reacher' and the same maximum length of single episode in supervised learning and RL, etc.

3.2.6 DDPG with Policy Replacement – Choice of Noise

Noise is an important factor for RL learning process. For initialized policy π_i trained with expert trajectories, the optimal trajectory for the same task should be some neighboring trajectories of the initialized trajectory, which means the difference of optimal policy π^* and π_i should be within a small range Δ,

$$D_{KL}(\pi^*||\pi_i) < \Delta \tag{4}$$

This can be derived easily by $(\pi^*(a^*|s_t) - \pi_i(a^*|s_t)) < \delta$, where a^* is the optimal action for state s_t. However, if we consider another initial policy π_i' which is not as good as π_i, then $(\pi'^*(a^*|s_t) - \pi_i(a^*|s_t)) < \delta'$ and $\delta' > \delta$, also we have $D_{KL}(\pi^*||\pi_i') < \Delta'$ and $\Delta' > \Delta$. In order to handle different initialization policies like π or π', we need to apply different noise for a better learning performance.

Generally, a larger noise scale or larger probability to have a noise action will help worse policy π' to have larger chances to sample an optimal action. We can testify the case of larger probability of have

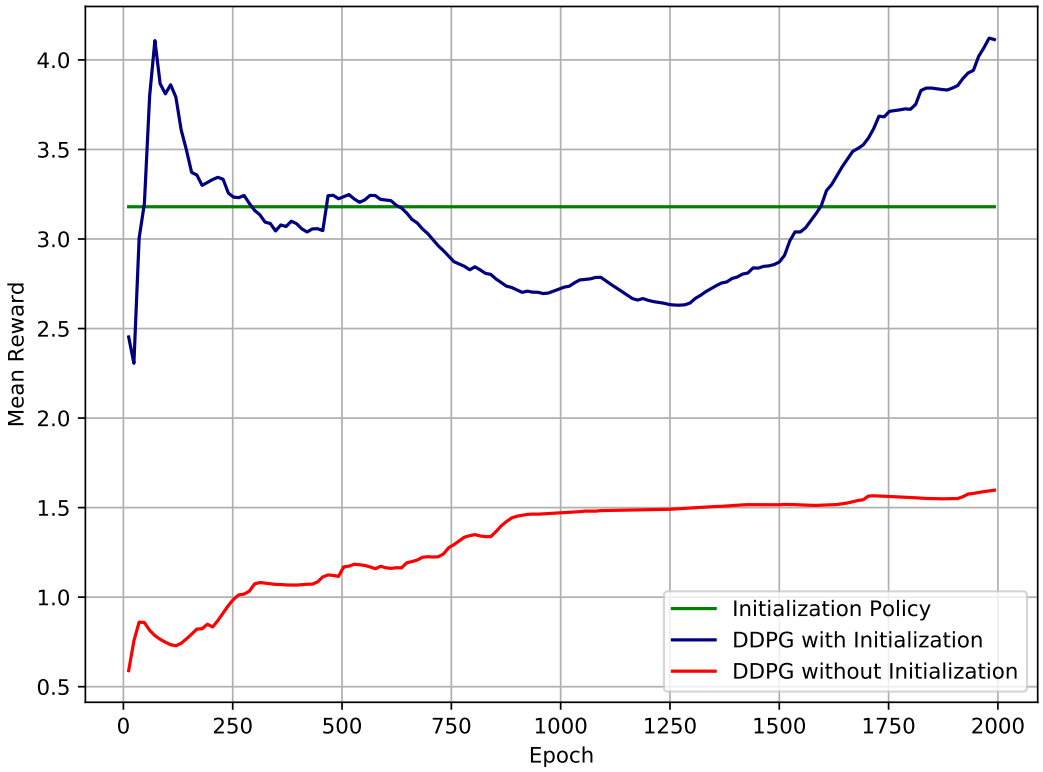

Figure 4: Comparison of initial learning performances of DDPG with/without initialization policy on *Reacher* task. The green line is the mean reward of initial policy trained with supervised learning. The blue line and red line are mean rewards of DDPG with and without initialization in initial 2000 training epochs .

a noise through ϵ-greedy policy, as shown in Fig. 5. The ϵ-greedy policy is defined as follows:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \dfrac{\epsilon}{|A(s)|}, & a = a* \\ \dfrac{\epsilon}{|A(s)|}, & a \neq a* \end{cases} \tag{5}$$

In Fig. 5, the ϵ-greedy policy can be regarded as a uniform distributed noise on the possible range of action value. It shows that the noise helps the initial policy to have a larger probability to sample the optimal action a^*, when the initial policy has a relatively large difference from the optimal policy, which is the case of π' with the KL-divergence upper bound $\Delta' > \Delta$. Fig. 6 shows that for the same optimal distribution and different initial distributions, same ϵ will cause different effects for noisy policy. Generally, it shows that large ϵ value (0.8) only helps with large divergence of initial policy and optimal policy. And this is the reason to apply decayed ϵ in practice for ϵ-greedy policy.

Fig. 7 shows that when two policies have large divergence Δ', larger ϵ in will help more for the initial policy to sample optimal actions with ϵ-greedy policy. And larger ϵ generally means larger chances to have noisy actions.

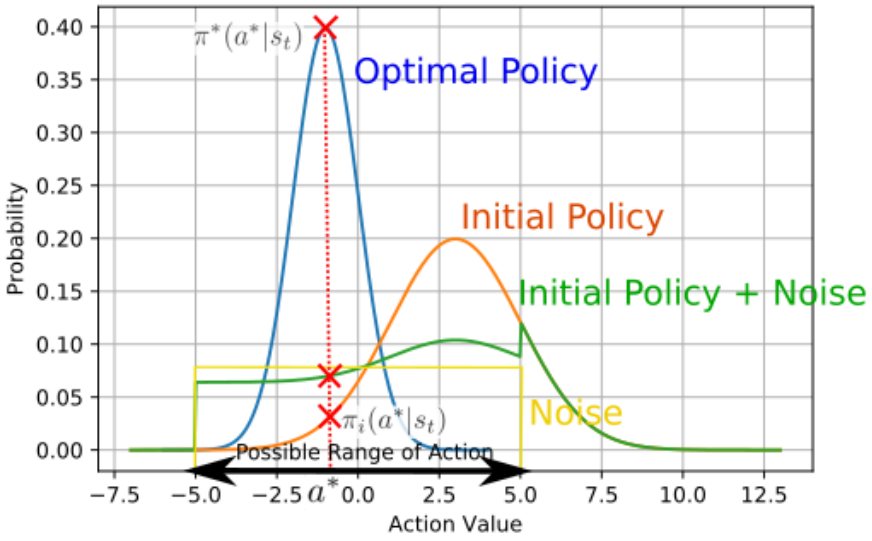

Figure 5: A figure showing why adding noise would help for exploring better actions, with noise type of ϵ-greedy policy. Suppose the initial policy and the optimal policy are Gaussian distributions with different means and variances, and the noise is a uniform distribution within specific range (range of possible action value). The figure shows how the noise of ϵ-greedy helps to increase the possibility for non-optimal policy to sample the optimal action

Above analysis is based on ϵ-greedy policy, which represents the variety of probability to apply the noise in actions. However, the case of the variety in scale of noise has similar effects for different distributions. Therefore, we can derive some conclusions that for better initial policy, it means the smaller the divergence it is from the optimal policy, therefore it needs to apply noise with a smaller probability or apply smaller scale noise. So, how does it work in initialized RL?

In initialized RL, the cases are the same. Experiments with different initialized policy for DDPG on *Reacher* task are shown in Fig. 8, in which the relatively good initial policy π is called from expert,

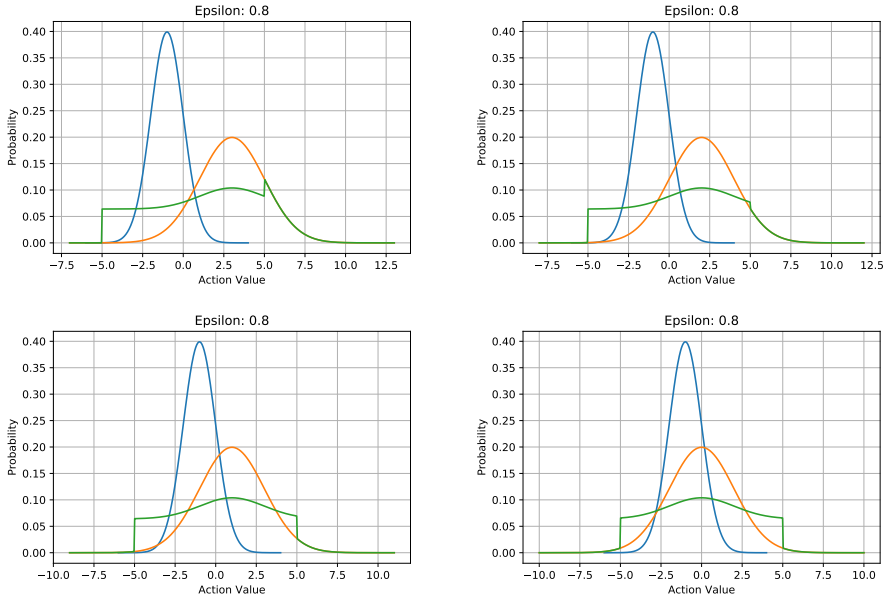

Figure 6: ϵ-greedy policy with different initial distributions and same ϵ value. It shows the same ϵ will cause different effects for noisy policy, it benefits the top two initial distribution and hurts the bottom two distributions. So large ϵ value (0.8) only helps with large divergence of initial policy and optimal policy.

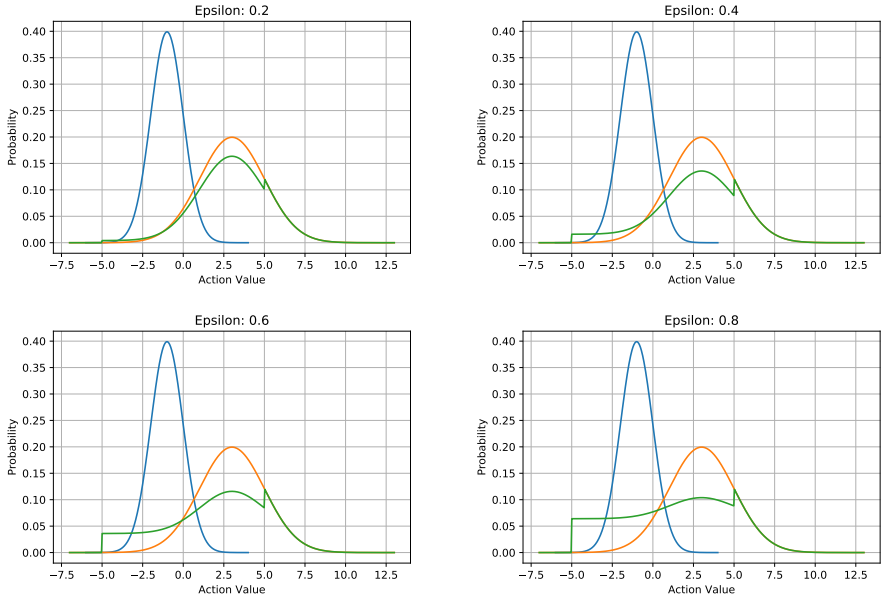

Figure 7: ϵ-greedy policy with different ϵ value and same initial and same optimal distribution. Larger ϵ value benefits more for large divergence between the optimal policy and the initial policy.

and the relatively bad but better than random initial policy π' is called the half-expert. Fig. 8 shows that for a bad initial policy, which has a relatively large divergence from the optimal policy, larger noise scale works better for improving the learning performance. And this results testify the above analysis of the case of ϵ-greedy policy. Note that the experiments shown in Fig. 8 are all without preheating process, which is the second step described in the general process section. However, if we take experiments on the policy initialized from a better policy, Fig. 9 shows smaller noise is better for DDPG initialized with an expert policy, which also testifies above analysis results.

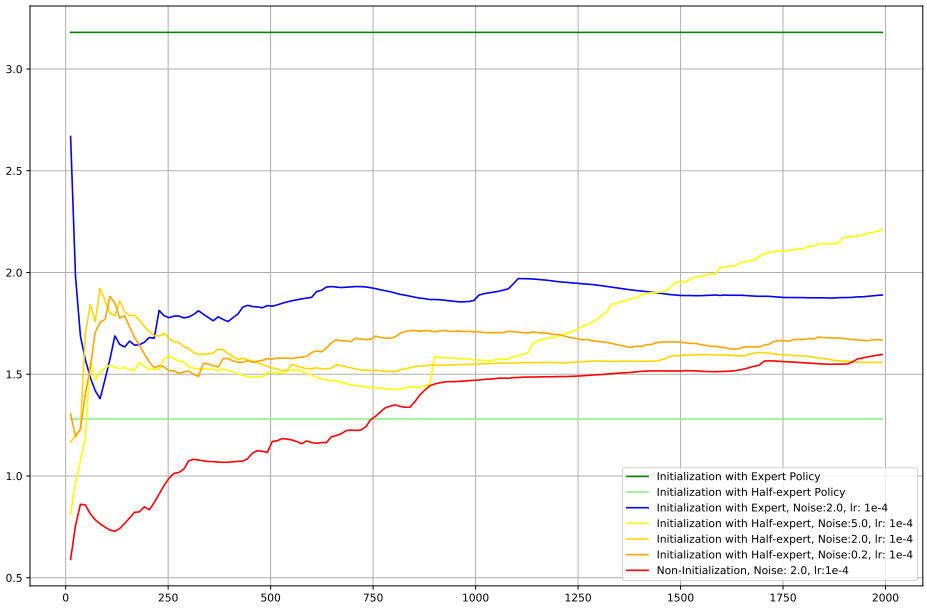

Figure 8: Experiments with different initialized policy for DDPG on *Reacher* task

As for the learning rate settings, Fig. 10 shows a comparison of different actor learning rate for initialized DDPG with a proper noise scale. For initialized DDPG from an expert policy, we actually need only tune the parameters in the actor neural networks in a small range, which requires a small scale of actor learning rate (e.g. 10^{-4}). A larger learning rate could change the networks dramatically and also the outputs. However, when the actor learning rate is too small, it will not help with learning process as shown in Fig. 10.

3.2.7 DDPG with Policy Replacement – Pre-train the Critic

The actor-critic scheme is important in DDPG, as the critic instructs the actor's choise of action values through evaluating the Q-value of each action. Therefore, a good critic is critical for the DDPG to show great learning performance in a task. However, in initialized policy for RL, the supervised learning with inverse kinematics could only mapping from the input states to the output actions, without any reference of the estimated value of each action. It means the critic of DDPG cannot be

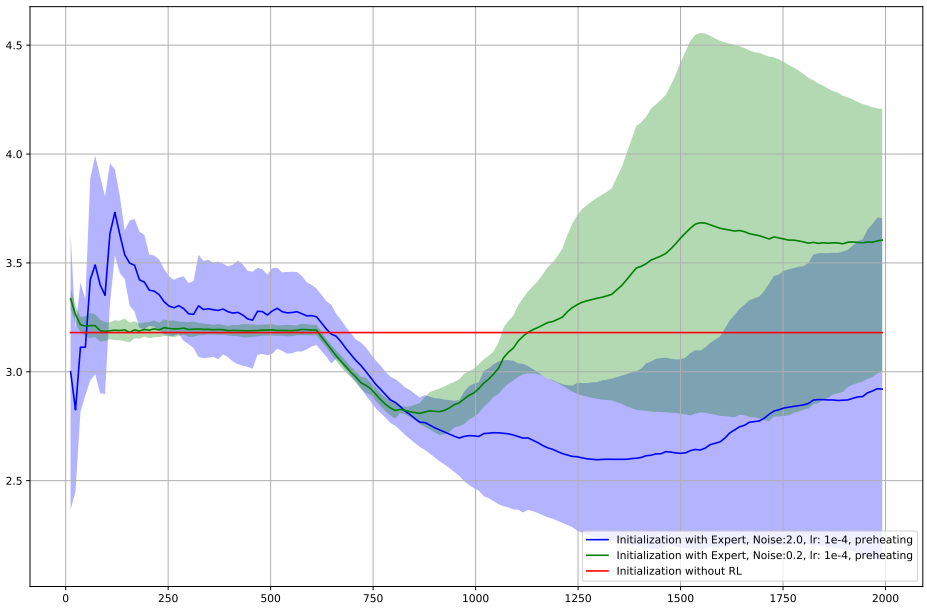

Figure 9: Experiments with different initialized policy for DDPG on *Reacher* task

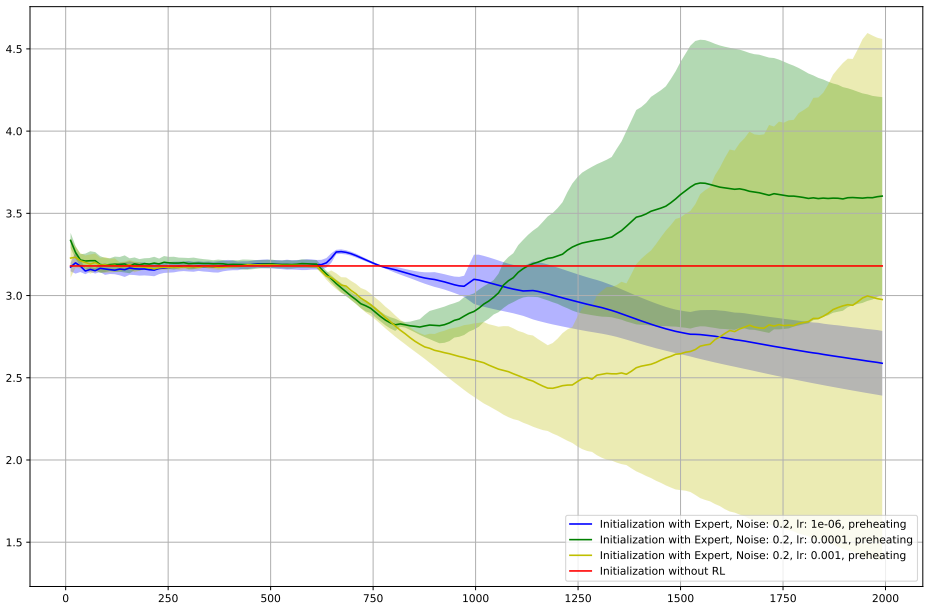

Figure 10: Experiments with different initialized policy for DDPG on *Reacher* task

directly initialized with policy from supervised learning. Therefore, the role of the critic is amplified in the learning process of initialized DDPG, which is testified in experiments of this section. To solve the problem that the supervised learning policy can only initialized the actor networks, we propose a pre-training process of the critic in initialized DDPG called the 'preheating', with which the initialized policy could be more effective than without it.

The 'preheating' process is conducted as follows: after loading the weights from pre-trained policy to the actor networks in DDPG, we sample from the frozen actor to generate near-expert samples with noise scale σ and feed them into the memory (DDPG is off-policy learning), then train the critic networks with generated samples for N_{pre} epochs. The frozen actor can be achieved through setting the actor learning rate to be 0 in practice. After the 'preheating' step of N_{pre} epochs, we train the initialized DDPG in a general way.

As shown in Fig. 11, without pre-training the critic, there is always a severely decrease in performances of the initialized DDPG algorithm, no matter what the scale of exploration noise is. The decrease phenomenon for initialized DDPG without preheating can be explained as follows: the main problem of initialized policy for RL without preheating is that the virtue of initialized policy is ruined too quickly through changing of the weights of the initialized actor, and this is always the case as long as the learning rate is of a relatively considerable value. Because no matter how much the exploration noise is, the weights of the actor will be changed according to the learning rate of the actor, and neural network as an estimator has property of sensitivity on weights, which ends up with a dramatic changing in output policy of the actor. Therefore the actor will diverge fast from the initial near-expert policy, without a good critic. A good critic means it shows positive instructions for the actor to update weights in the correct direction. On the other hand, a good actor here in initialized DDPG will have great chances to decrease its performance with a random initialized critic. Therefore the 'pre-heating' process actually prevent the actor from updating to hurt the performances, or at least alleviate it. This is shown clearly in Fig. 11 that the lower bound of initialized DDPG with preheating is much higher than without the preheating process.

3.2.8 DDPG with Policy Replacement – Conclusions

Some conclusions can be derived from above analysis and experiments about DDPG with initialization policy for *Reacher* task: (1). Different initial policy requires different noise scales to achieve better performances. For expert initialization policy, smaller exploration noise for DDPG shows better learning effects; for half-expert initialization policy, larger noise shows better learning effects.

(2). The preheating process is important for initialized policy on DDPG: with the pre-training of the critic before general learning process of DDPG, the learning performance is always better than without it, It reduces the dramatic decrease at the initial training phase when the DDPG is initialized with an expert policy. The important intuition from this is that we should not use the critic for instructing the actor unless it's good enough for a good actor.

3.2.9 PPO with Policy Replacement

PPO is an on-policy RL algorithm, and its exploration is controlled by the variance σ of output action distribution from the policy neural network, which is learned during the training process. We can control the scale of exploration through adding a value to the variance, which is called noise base σ_0 here. So the actual action variance is $\sigma = \sigma_0 + \sigma_{pre}$ where σ_{pre} is trained output variance of the policy network.

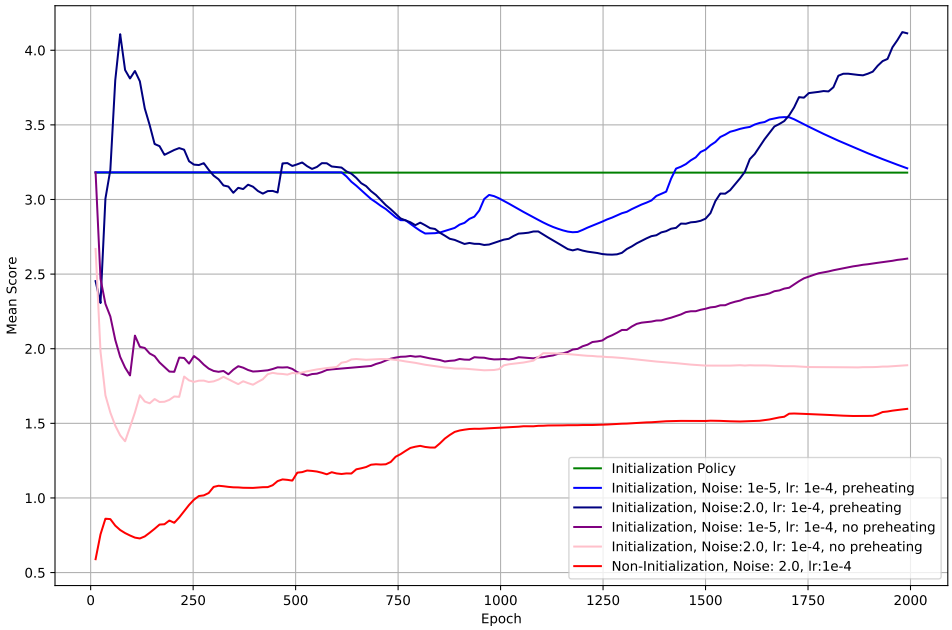

Figure 11: Comparison of initialized DDPG with/without pre-training the critic on *Reacher* task. The green line is the mean reward of initial policy trained with supervised learning.

The preheating process for PPO is conducted by pre-training the value evaluation network, similar to pre-training the actor in DDPG.

Fig. 12 shows comparisons of initialized PPO for *Reacher* task. Similar to conclusions in initialized DDPG algorithm, the preheating process is effective for reducing the decrease in initial training phase of PPO when initialized from an expert policy.

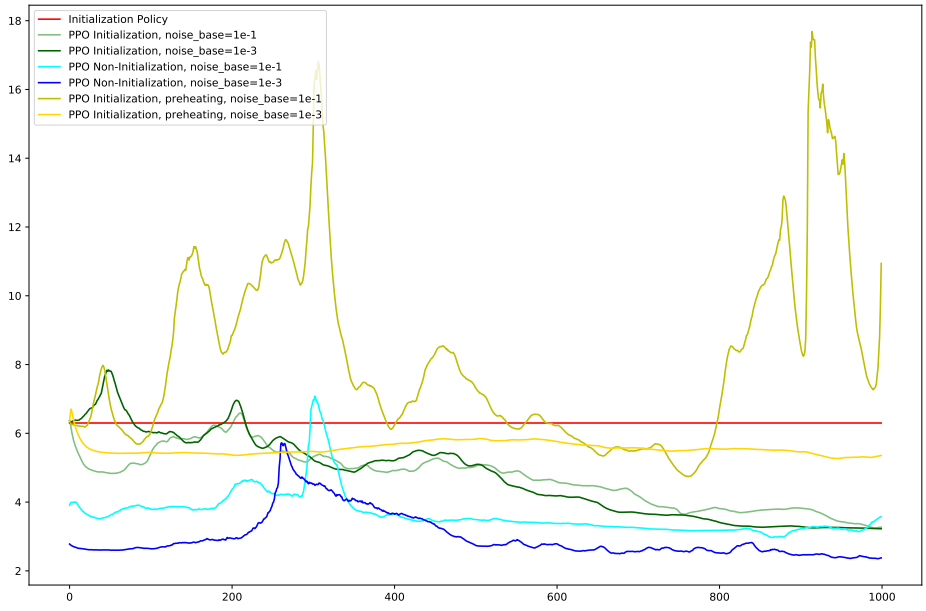

Figure 12: Comparisons of PPO with or without initialized policy and preheating process with different noise scale.

Fig. 13 shows different actor learning rates for initialized PPO. A proper range of actor learning rate is needed for effective learning.

Fig. 14 shows different noise base for initialized PPO. A proper range of noise is needed for effective exploration and learning.

3.3 Initialized Reinforcement Learning with Supervised Learning Policy – Residual Policy Learning

In addition to the policy replacement approach for initialization of RL, residual policy learning [1] is another approach to realize initialization. It s based on good but imperfect controllers for robot manipulation tasks, and to learn a residual policy on top of that initial controller. For robot manipulation in real world, the initial controller could be a pre-trained policy in simulation; and for robot manipulation in simulation, the initial controller could be from the pre-trained supervised learning with expert trajectories as in Section 3.2.

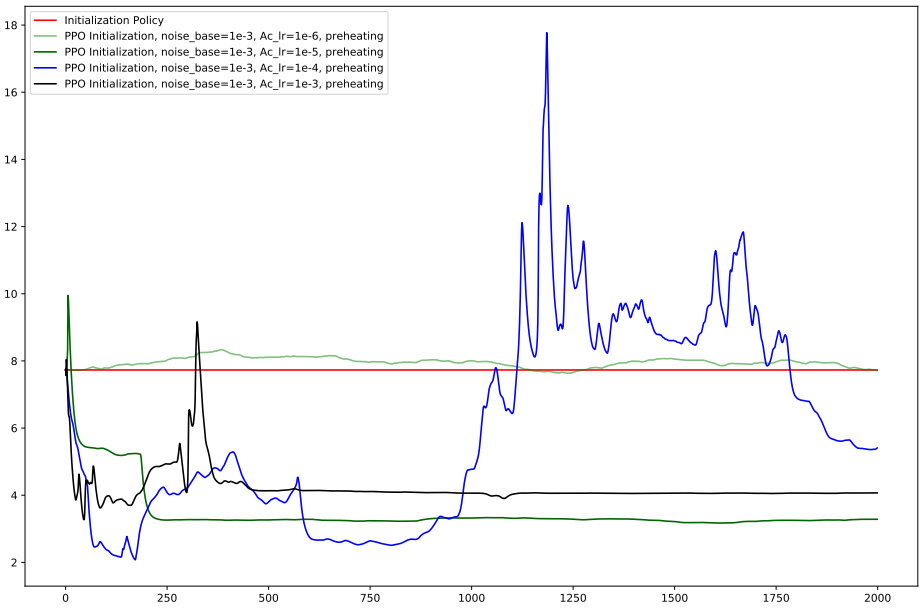

Figure 13: Comparison of different actor learning rates for initialized PPO with preheating process on *Reacher* task.

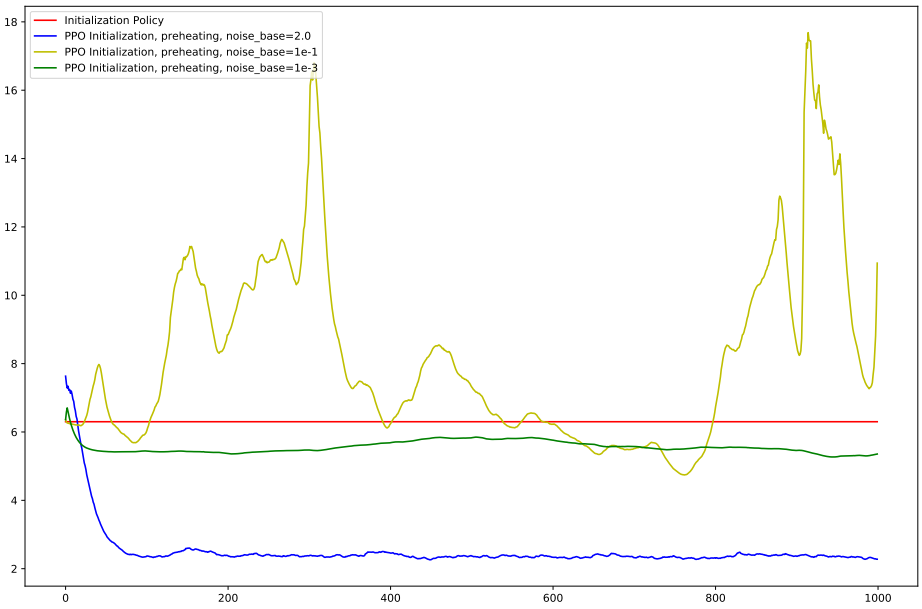

Figure 14: Comparison of different noise base for initialized PPO with preheating process on *Reacher* task.

The action in residual policy learning is the sum of the initial policy π_{ini} and the residual policy π_{res}:

$$a = \pi_{ini}(s) + \pi_{res}(s) \tag{6}$$

In this way, the residual policy learning is able to preserve the initialized policy performance to the best advantage. Some tricks are applied to guarantee that:

(1). The weights of the output layer of the actor in RL are initialized to be 0;

(2). The same preheating process as in DDPG with initialization: train the critic while fixed the weights of the actor to alleviate the decrease in initial RL training stage caused by a bad critic;

(3). Apply amplification factor for noise scale during the preheating process: for pre-training the critic to have better generalization ability, apply noise multiplied by an amplification factor in the preheating process and reduce it to normal when training the actor.

Note that with above settings, the actor network outputs are actually all 0 during the preheating process, so the action values are fully attributed to the amplified noise and initial policy, which means $\pi_{res} = \epsilon$ and also:

$$a = \pi_{ini}(s) + \alpha * \epsilon, \alpha > 1 \tag{7}$$

where ϵ denotes the action noise and the α denotes the amplification factor. And the action values stored in the memory (with DDPG algorithm) are $\pi_{res} = \epsilon$ during preheating. This is not a problem as we only train the residual policy π_{res} instead of the composite policy $\pi_{ini} + \pi_{res}$, and action values stored in memory are just $\pi_{res}(s)$ instead of the real action value a. Correspondingly, the estimated Q-values learned by the critic are also only for the residual policy, which are $Q(s, \pi_{res}(s)))$.

However, there are still several freedoms to be set in residual policy learning, e.g. the scale of the output action from the residual policy can be chosen arbitrarily, which is actually a trade-off between the initialized policy and the learning policy, etc.

3.3.1 DDPG with Residual Policy Learning

3.3.2 Comparison of Residual Learning and Policy Replacement

We chose the goal position to be easy enough to reach for the agent, in order to shorten the training process and show the overall learning curves, as in Fig. 15.

With the same experiment settings: actor learning rate = 1e-4, critic learning rate = 1e-4, steps per episode = 12, same actor and critic network structures, preheating steps = 600 in initialized RL, noise scale = 2.0. The comparisons of non-initialized policy and initialized policy with residual Learning and policy replacement on DDPG for a easy-to-reach goal are shown in Fig. 16. For some easy-to-reach goals, the policy replacement approach seems to outperform the residual policy learning and non-initialization RL.

Comparison in Fig. 15 shows that, for some goals, initialization with residual learning may not learn as well as policy replacement at the beginning of training, but will eventually learn better policy. This may benefit from the performance guarantee of the initial good policy for residual policy learning, while in policy replacement the learning process may search in a larger trajectories space (testified in observing the experiments).

With above analysis, we can see that the performances of different approaches to initialize the RL actually have large variance, which are significantly affected by parameters settings and task specifications like goal position, etc.

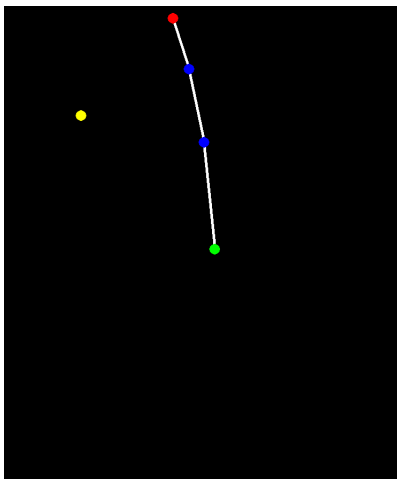

Figure 15: Easy-to-reach goal position used in comparison.

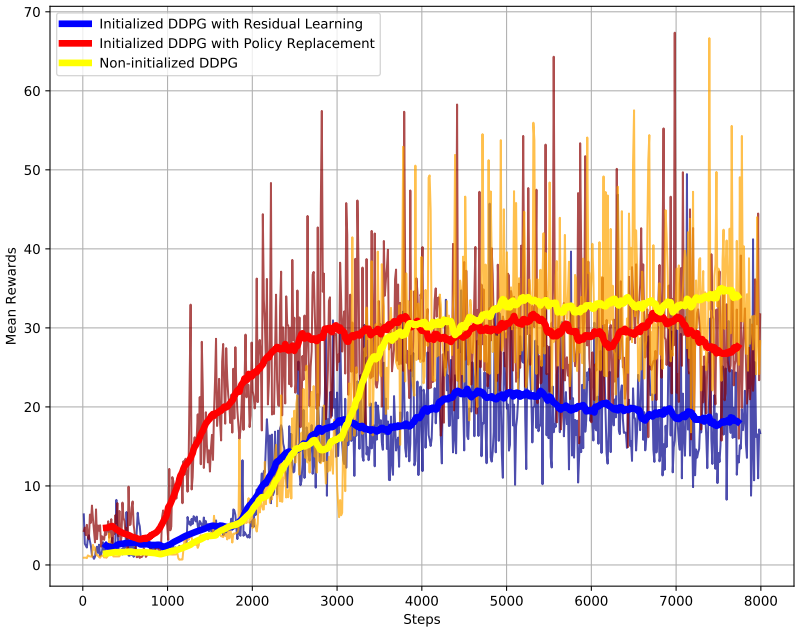

Figure 16: Comparison of non-initialized policy and initialized policy with residual learning and policy replacement for DDPG with an easy-to-reach goal position. The bold lines are moving average of episode rewards during learning process. The initialized policy has a preheating process of 600 steps.

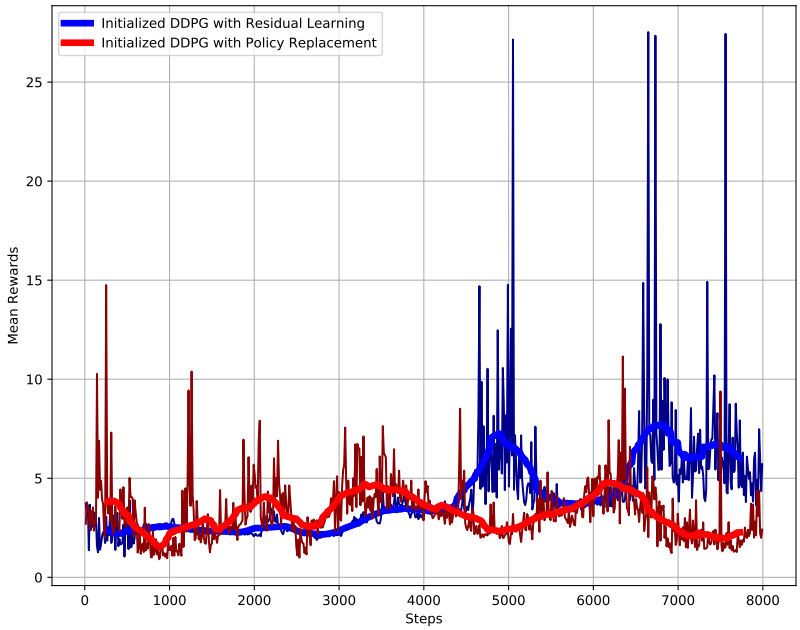

Figure 17: Comparison of residual learning and policy replacement for DDPG with another goal position.

3.3.3 Modelling Analysis of Residual Policy Learning

Consider that the residual policy learning is actually using policy $\pi_{res}(s)$ to generate the residual action, whose inputs s are exactly the same as the initial policy $\pi_{ini}(s)$, the modelling of residual policy is actually may be more complicated than a general policy in RL, although the value range of outputs for residual policy may be smaller (a smaller searching space for delicate actions). In order to have a more accurate composite action, the residual policy not only needs to model the environment but also the initial policy. The modelling relationship can be interpreted as follows.

We denote the modelling of a general policy in RL as $\mathcal{M}_g$, the modellings of the initial policy and the residual policy are $\mathcal{M}_i, \mathcal{M}_r$, respectively. A modelling means a mechanism that the policy learned from interactions with its environment to represent the its environment and based on which the policy determines actions. The relationship of above modellings is:

$$\mathcal{M}_g = \mathcal{M}_i \oplus \mathcal{M}_r \tag{8}$$

which means the $\mathcal{M}_i, \mathcal{M}_r$ are actually dual modellings with respect to a $\mathcal{M}_g$. As we only train the residual policy in the learning process of residual RL, we have the modelling to construct in this training process the $\mathcal{M}_r$:

$$\mathcal{M}_r = \mathcal{M}_g \ominus \mathcal{M}_i \tag{9}$$

where the $\oplus, \ominus$ denotes a positive or negative modelling composite relationship. And the composite relationship does not determine the complexity of the composite modelling to be more or less than original constituent modellings, which means the modelling of $\mathcal{M}_r$ may not be less complicated than the general policy $\mathcal{M}_g$, and sometimes even more complicated than $\mathcal{M}_g$. This means the learning of a residual does not have to be easier than directly learning the general policy.

The *Reacher* task is made more complicated for testing effects of demonstrations, in which there are two penalty areas. Fig. shown comparisons of DDPG from demonstrations with general DDPG, both without prioritized experience replay.

3.4 Meta-learning as Initialization for Reinforcement Learning

3.4.1 MAML and Reptile

3.4.2 Reptile + PPO

3.5 Off-Policy Reinforcement Learning with Demonstrations

We show different initialization strategies for efficient RL as above, and leveraging demonstrations for deep RL is an alternative approach to learn efficiently. Both initialization approach and leveraging demonstrations approach are taking advantages of the expert trajectories, however, they treat them differently. Instead of pre-training a policy to initialize the RL policy, the approach of leveraging demonstrations directly feeds those expert trajectories into memory of off-policy RL (e.g. DDPG) to make the policy trained with both demonstrations and explorations.

We apply a more complicated environment for this experiment as shown in Fig. 18, with two penalty areas. The optimal trajectories for the agent is to go through the middle of two penalty areas. And we generate good demonstrations by manually setting an intermediate goal position at the middle of two penalty areas.

Experiment results of applying DDPG from demonstrations with different feeding approaches and vanilla DDPG are shown in Fig. 19. The prioritized experience replay is not used in this experiment.

The feeded demonstration dataset contains totally 50 episodes (20 steps each episode) of trajectories, and each feeding operation is to feed the whole demonstrations dataset. It seems the more demonstrations feeded into the memory, the worst learning performance it is.

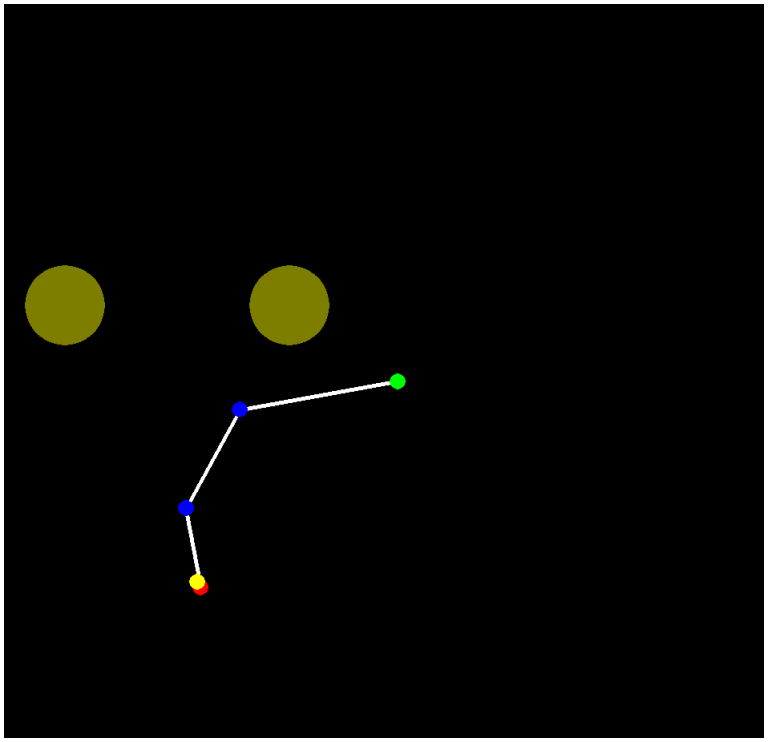

Figure 18: *Reacher* environment with two penalty areas.

Bibliography

[1] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. *arXiv preprint arXiv:1812.03201*, 2018.

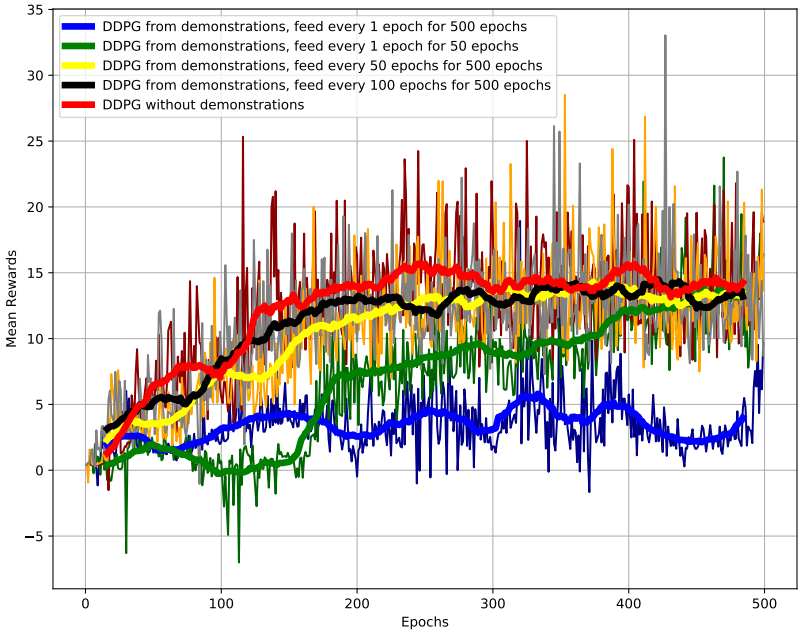

Figure 19: Comparison of DDPG from demonstrations with different feeding approaches (without prioritized experience replay) and vanilla DDPG. The blue line is feeding demonstrations for every epoch for the whole training process (500 epochs); the green line is feeding demonstrations for every epoch for the first 50 epochs; the yellow line is feeding demonstrations for every 50 epochs for the whole training process; the black line is feeding demonstrations for every 100 epochs for the whole training process; the red line is vanilla DDPG without using demonstrations.