Artificial Intelligence and Robotics

“La Sapienza” University of Rome

Faculty of Information Engineering, Information Technology and Statistics

Department of Informatics, Automation and Control Engineering

"ANTONIO RUBERTI"

Deep Deterministic Policy Gradient Algorithm on Open-AI gym Humanoid Environment

GARBA Mariam Olajumoke [1871871]

MAKINWA Sayo Michael [1858908]

31.03.2019

1.0 Introduction

Today, reinforcement learning in Artificial Intelligence is an exciting field of study. In recent years, it has grown exponentially, starting with Robotics Arm Manipulation, Google Deep Mind beat a professional Alpha Go player, and recently the OpenAI team beat a professional DOTA player. While reinforcement learning agents have achieved some success in various domains, they are only limited to domains with full observability and low dimensional state spaces. There have been major developments in this field, one of which is deep reinforcement learning.

When humans face difficult tasks, they seem to solve this problem by combining reinforcement learning and hierarchical sensory processing systems harmoniously. Agents on the other hand must derive efficient representations of the environment from high-dimensional sensory inputs (e.g. image pixels) and use these to generalize past experiences to make informed decisions to new situations. An advanced technique called the Deep Q Network (DQN) is used to learn successful policies using end - to - end reinforcement learning directly from high - dimensional sensory inputs.

Many tasks of interest (e.g. physical control tasks) have continuous and high-dimensional action spaces. DQN cannot be applied to a continuous domain because it only handles discrete and low-dimensional action spaces. Also, since the action spaces are continuous, it is significant to take into account the optimization process at every iteration.

To perform reinforcement learning in continuous action spaces, Deterministic Policy Gradient algorithm (DPG) is more efficient than the usual stochastic policy gradients. DPG is quite unstable when applied to challenging problems. However, a hybrid of DQN and DPG can learn competitive policies for both discrete and continuous tasks domains. This is called the Deep Deterministic Policy Algorithm (DDPG).

This study shows the implementation of DDPG and the model was evaluated on a physical control task (making a humanoid robot walk without falling) that involve complex multi-joint movements, unstable and rich contact dynamics. For this physical control task, we also compare our results over several episodes.

2.0 State of the Art

The goal of reinforcement learning is to find an optimal behaviour strategy for an agent to obtain optimal rewards. The agent's policy $\pi(s)$ provides the guideline on what is the optimal action to take in a certain state with the goal to maximize the total rewards. Each state is associated with a value function $V(s)$ predicting the expected amount of future rewards it is able to receive in a given state by acting the corresponding policy. In other words, the value function quantifies how good a state is. Fig 1 shows the summary of approaches used in reinforcement learning.

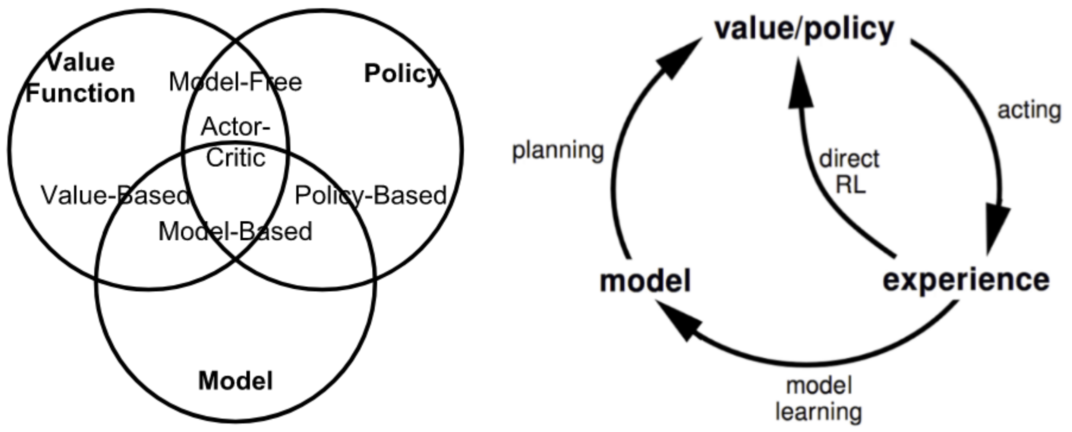


Fig.1 Summary of approaches in RL based on whether we want to model the value, policy, or the environment.

- **Model-based**: Rely on the model of the environment; either the model is known or the algorithm learns it explicitly.
- **Model-free**: No dependency on the model during learning.
- **On-policy**: Use the deterministic outcomes or samples from the target policy to train the algorithm.
- **Off-policy**: Training on a distribution of transitions or episodes produced by a different behaviour policy rather than that produced by the target policy.

When the model is fully known, following Bellman equations, we can use Dynamic Programming (DP) to iteratively evaluate value functions and improve policy. Bellman equations refer to a set of equations that decompose the value function into the immediate reward plus the discounted future values.

Policy Evaluation is used to compute the state-value $V\pi$ for a given policy π:

$$Vt+1(s) = \mathbb{E}\pi[r+\gamma Vt(s)|St=s] = \sum a\pi(a|s)\sum s',rP(s',r|s,a)(r+\gamma Vk(s'))$$

Based on the value functions, Policy Improvement generates a better policy $\pi'\geq\pi$ by acting greedily

$$Q\pi(s,a) = \mathbb{E}[Rt+1+\gamma V\pi(St+1)|St=s, At=a] = \sum s',rP(s',r|s,a)(r+\gamma V\pi(s'))$$

Policy-based methods are more useful in the continuous space. Because there is an infinite number of actions and (or) states to estimate the values for and hence value-based approaches are way too expensive computationally in the continuous space. For example, in Q-learning (Watkins & Dayan, 1992), the policy improvement step $\text{argmax}_{a\in Q}\pi(s,a)$ requires a full scan of the action space, suffering from the curse of dimensionality.

The policy gradient methods target at modelling and optimizing the policy directly. The policy is usually modelled with a parameterized function respect to θ, $\pi\theta(a|s)$. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize θ for the best reward.

In order to make effective use of large neural networks as function approximators, Mnih et al. 2015 greatly improved and stabilized the training procedure of Q-learning by using two innovative mechanisms:

- Experience Replay: All the episode steps et=(St,At,Rt,St+1) are stored in one replay memory Dt={e1,…,et}. Dt has experience tuples over many episodes. During Q-learning updates, samples are drawn at random from the replay memory and thus one sample could be used multiple times. Experience replay improves data efficiency, removes correlations in the observation sequences, and smooths over changes in the data distribution.
- Periodically Updated Target: Q is optimized towards target values that are only periodically updated. The Q network is cloned and kept frozen as the optimization target every C steps (where C is a hyper-parameter). This modification makes the training more stable as it overcomes the short-term oscillations.

These mechanisms are used in the implementation of DDPG in this next section.

3.0 Approach

Deep Deterministic Policy Gradient (DDPG) is a model-free off-policy actor-critic algorithm for continuous action space exploration, combining DQN with DPG (Lillicrap, et al., 2015). The original DQN works in discrete space, and DDPG extends it to continuous space with the actor-critic framework based on DPG (Silver et al., 2014). while learning a deterministic policy. The critic Q(s,a) is learned using the Bellman equation as in Q-learning. The actor is updated by applying the chain rule to the expected return.

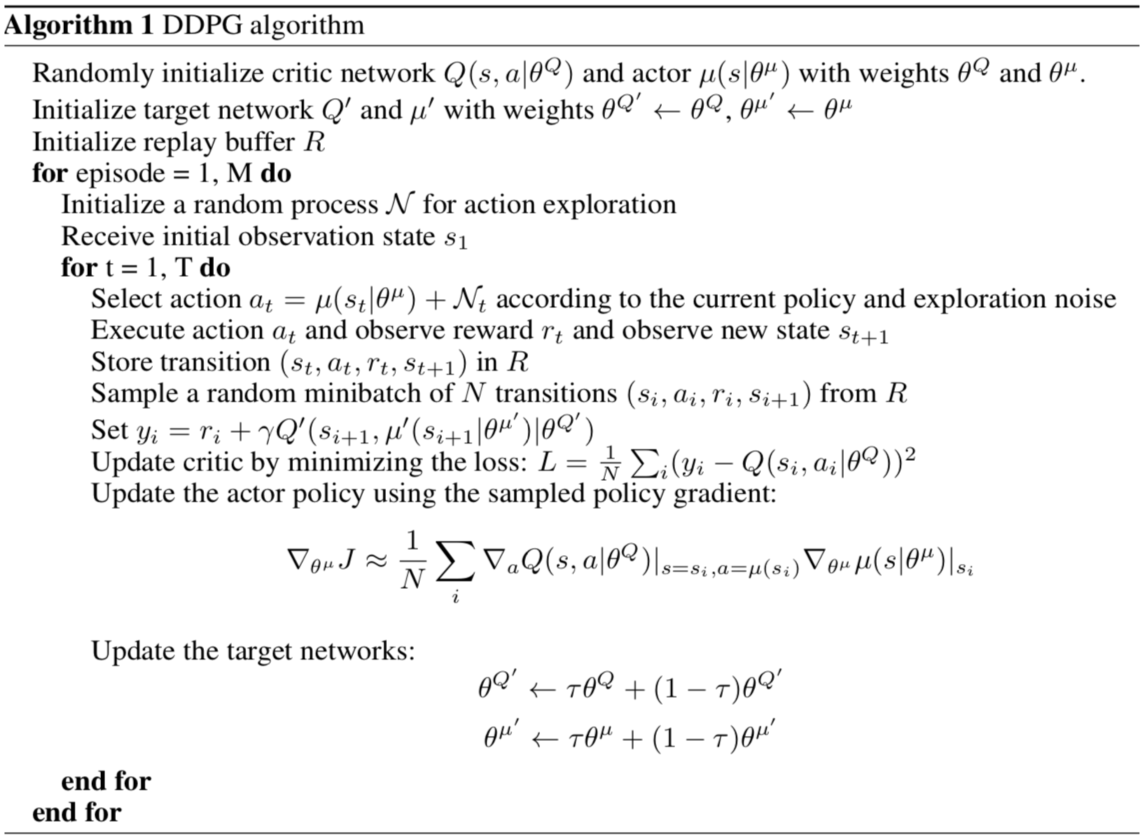
In order to do better exploration in DDPG, an exploration policy μ′ is constructed by adding noise N: $\mu'(s)= \mu\theta(s) + N$. N can be chosen to suit the environment. For the humanoid environment used in this study, an Ornstein-Uhlenbeck process (Uhlenbeck & Ornstein, 1930) is used to generate temporally correlated exploration for exploration efficiency.

Also, just like in DQN, we stabilized the learning of Q-function by experience replay and the frozen target network. The replay memory is a finite sized cache. Transitions were sampled from the environment according to the exploration policy and all the episode steps et=(St,At,Rt,St+1) are stored in one replay memory Dt={e1,…,et}. Dt has experience tuples over many episodes. When the replay memory was full the oldest samples were discarded.

At each timestep the actor and critic are updated by sampling a minibatch uniformly from the memory. The samples are drawn at random from the replay memory and thus one sample could be used multiple times. The replay memory is made to be large in order to allow the algorithm to remove correlations in the observation sequences, and smooths over changes in the data distribution.

In addition, DDPG does soft updates on the parameters of both actor and critic, with $\tau \ll 1$: $\theta' \leftarrow \tau\theta+(1-\tau)\theta'$. In this way, the target network values are constrained to change slowly, different from the design in DQN that the target network stays frozen for some period of time.

The DDPG algorithm can be seen in Fig 2. below:



```
Algorithm 1 DDPG algorithm
  Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
  Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
  Initialize replay buffer $R$
  for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
      Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
      Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
      Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
      Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
      Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
      Update the actor policy using the sampled policy gradient:
          $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
      Update the target networks:
          $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
          $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
    end for
  end for
```

Fig 2. DDPG Algorithm. (Image source: Lillicrap, et al., 2015)

4.0 Results

We conducted experiments of the performance of DDPG algorithm on the OpenAI Humanoid-v2 environment based on what we learned from Lillicrap et al. (2016) and Plappert et al. (2018). Lillicrap et al. (2016) is about the DDPG algorithm, with noise, sampled with the Ornstein-Uhlenbeck process, in the action space for exploration, while Plappert et al. (2018) is about possible improvements with noise in the parameter space.

The experiments have been carefully organised in the subsections that follows as the performance increases. The parameters used in this study can be found in the table below, gotten from the state of the art.

Parameter	Value	Parameter	Value
BUFFER_SIZE	1,000,000	LR_ACTOR	0.0001
BATCH_SIZE	128	LR_CRITIC	0.001
GAMMA	0.99	WEIGHT_DECAY	0.0001
TAU	0.01	ACTOR NETWORK	2 Layers
		CRITIC NETWORK	4 Layers

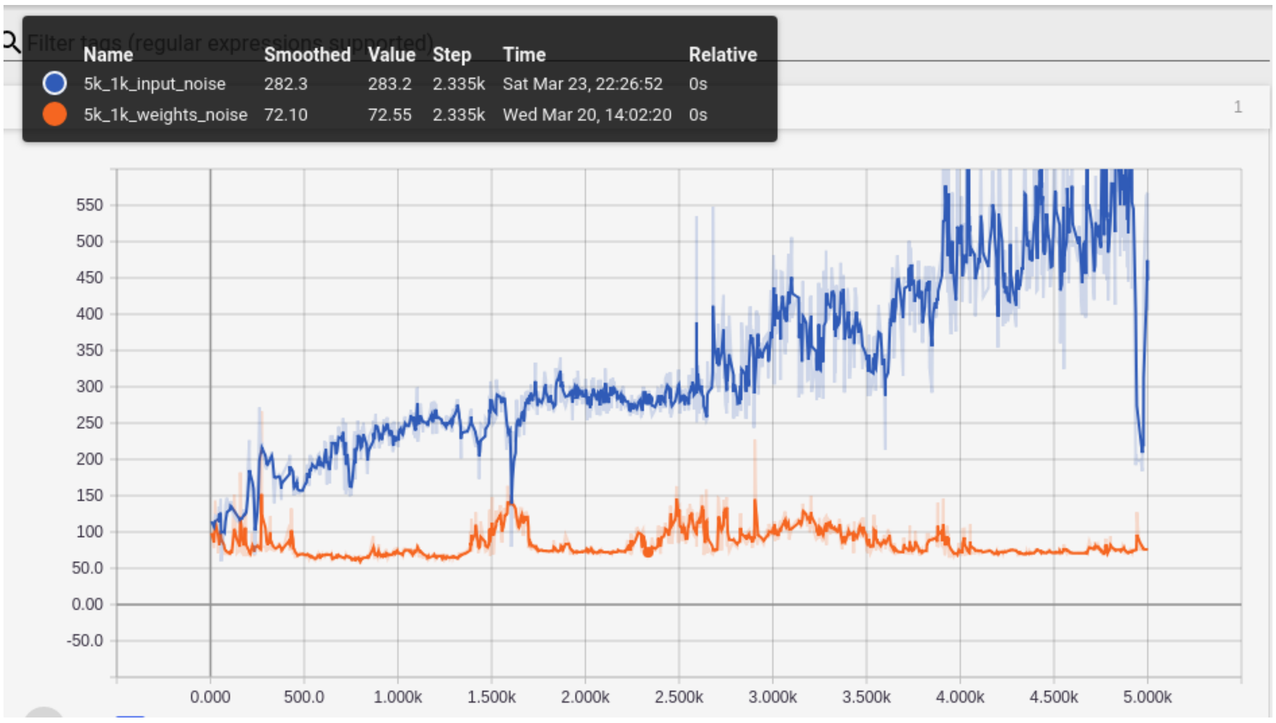
Table 1. Parameters used in the algorithm

Furthermore, the number of episodes and the number of maximum iteration for each episode was arbitrarily set to 5,000 and 1,000 respectively for each run. All experiments were conducted on OpenAI gym humanoid environment.

Please note that the charts of all the experiments in the following subsections have the number of episodes on the x-axis and the average score per episode on the y-axis.

4.1 Experiment 1 - Parameter space noise: Noise in weights vs Noise in inputs

Here, we explored how noise can be added in the parameter space, so the effect can be part of what the network learns. But noise in the parameter space can mean adding noise to the weights or to the input, hence, we studied how each helps the network learn policies in a continuous action space. We sampled Gaussian noise with mean zero and standard deviation of 1 and added it to the weights being learned by the neural network, and the inputs (the current state), separately, each time an action is to be generated. For the noise added to weights, we sampled Gaussian noise separately for each layer of the actor network.



	Name	Smoothed	Value	Step	Time	Relative
🔵	5k_1k_input_noise	282.3	283.2	2.335k	Sat Mar 23, 22:26:52	0s
🟠	5k_1k_weights_noise	72.10	72.55	2.335k	Wed Mar 20, 14:02:20	0s

Fig 3. Chart comparing noise in weights with noise in inputs

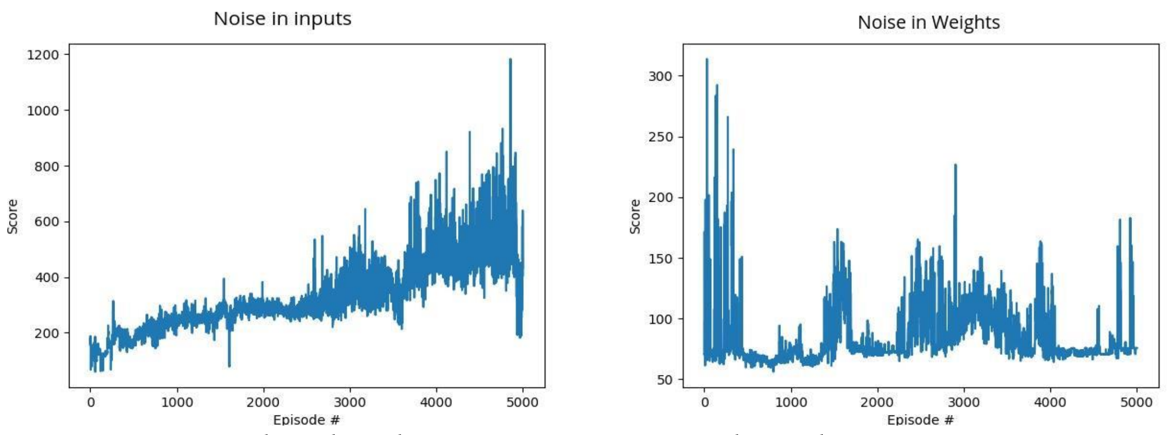
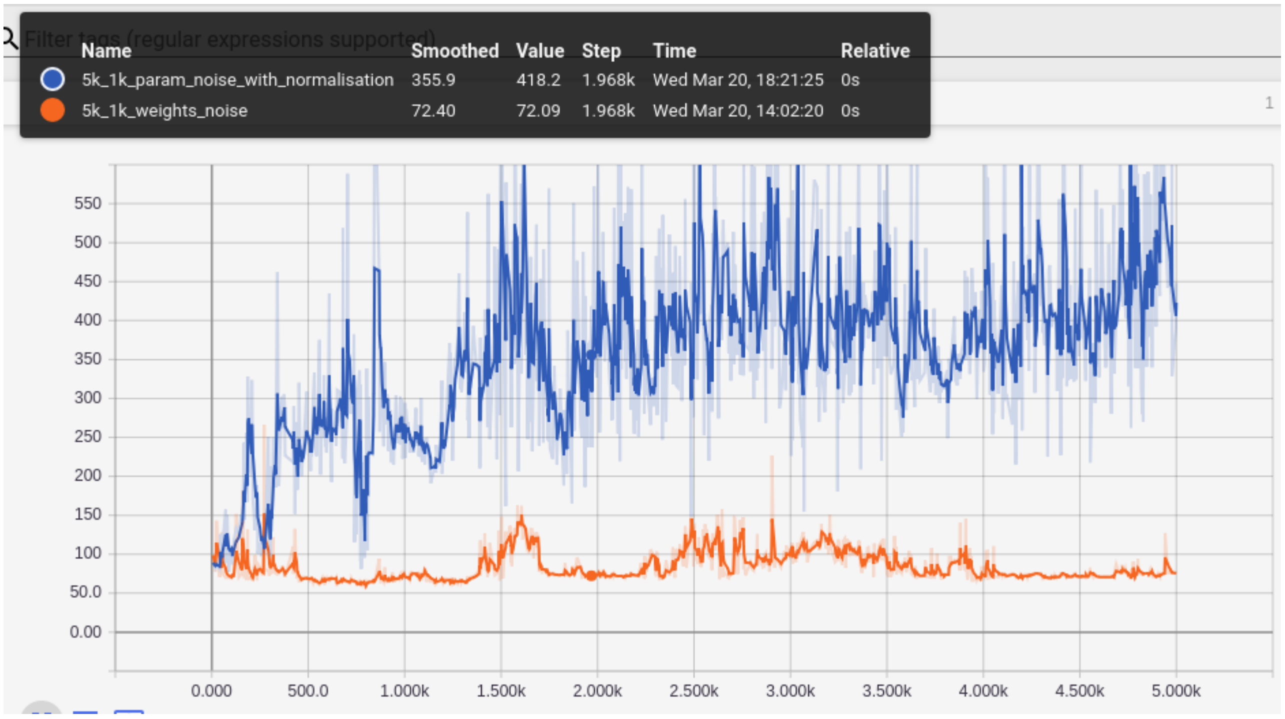


Fig 4. Chart describing scores in noise in weights and noise in inputs

As evidently visible from the chart, parameter space noise is better added to the inputs than to the learned weights, and this is in fact the case from the state of the art of neural networks. We see the scores start almost at the same value, but we continuously record a net increase in the scores for when the noise is added to the input as opposed to what happens when the noise is added to the weights, perhaps the score will increase even further if the execution goes on for more episodes.

4.2 Experiment 2 - Parameter space noise: Noise in weights; layer normalised or not

We further explored noise in weights to see if the result can be improved by normalising the layer after adding the weights. Each layer was normalised by dividing each value in the vector by the sum of all the values in the layer.

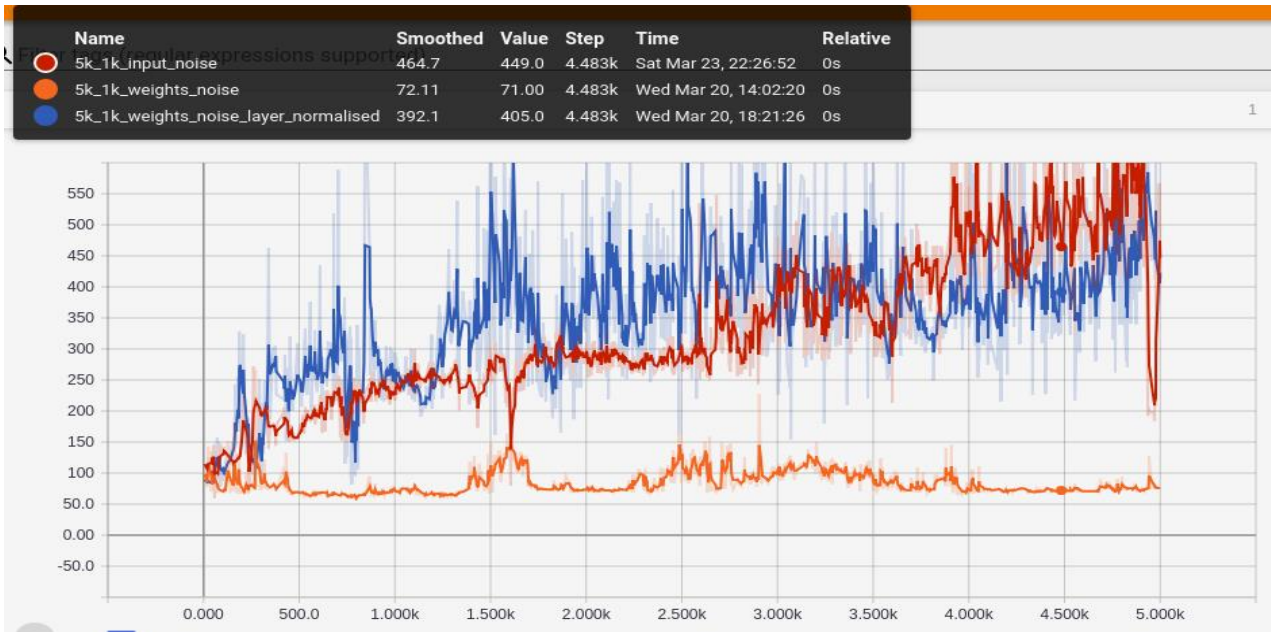


	Name	Smoothed	Value	Step	Time	Relative
🔵	5k_1k_param_noise_with_normalisation	355.9	418.2	1.968k	Wed Mar 20, 18:21:25	0s
🟠	5k_1k_weights_noise	72.40	72.09	1.968k	Wed Mar 20, 14:02:20	0s

Fig 5. Chart comparing noise in weights with layer normalised or not

Layer normalisation caused an almost dramatic improvement in the score values; there is an upward trajectory in the score values, though it later seems to stall, perhaps more episodes can cause further increase in the score values.

Comparing all the three we have seen so far, we see that noise in weights with layer normalisation started outperforming noise in inputs, however, as the layer-normalised noise started to stall, input noise climbed above it. Perhaps, it is safe to say that with more episodes, noise in input will perform even much better than layer-normalised weights noise.



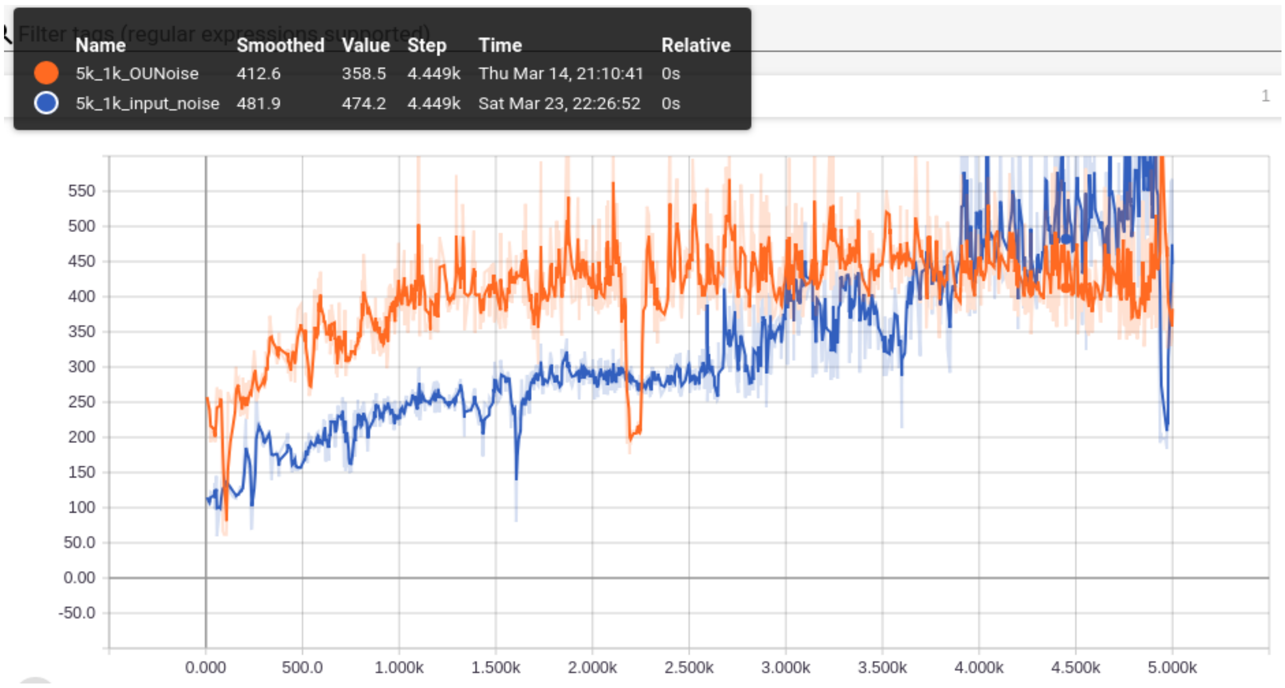
	Name	Smoothed	Value	Step	Time	Relative
🔴	5k_1k_input_noise	464.7	449.0	4.483k	Sat Mar 23, 22:26:52	0s
🟠	5k_1k_weights_noise	72.11	71.00	4.483k	Wed Mar 20, 14:02:20	0s
🔵	5k_1k_weights_noise_layer_normalised	392.1	405.0	4.483k	Wed Mar 20, 18:21:26	0s

Fig 6. Chart comparing noise in weights with layer normalised or not, with noise in inputs

4.3 Experiment 3 - Action space noise vs Parameter space noise

We compared the best modification so far - noise in the network input - to noise in the action space, sampled with the Ornstein-Uhlenbeck (OU) process (Uhlenbeck & Ornstein, 1930). As noted in the state of the art, one can study and understand the effect of noise in the action space, while it is impossible to study the effect of noise in the parameter space. The state of the art describes the OU noise as a temporally correlated noise which helps to better explore in physical environments that have momentum. We used the OU noise parameter values θ = 0.15 and σ = 0.2 which are the optimal values from the state of the art.

In this experiment, OU noise in the action space started really well, seriously outperforming noise in the input space. However, OU noise stalled and stopped improving at about the 4,000th episode, noise in the input caught up and performed better.



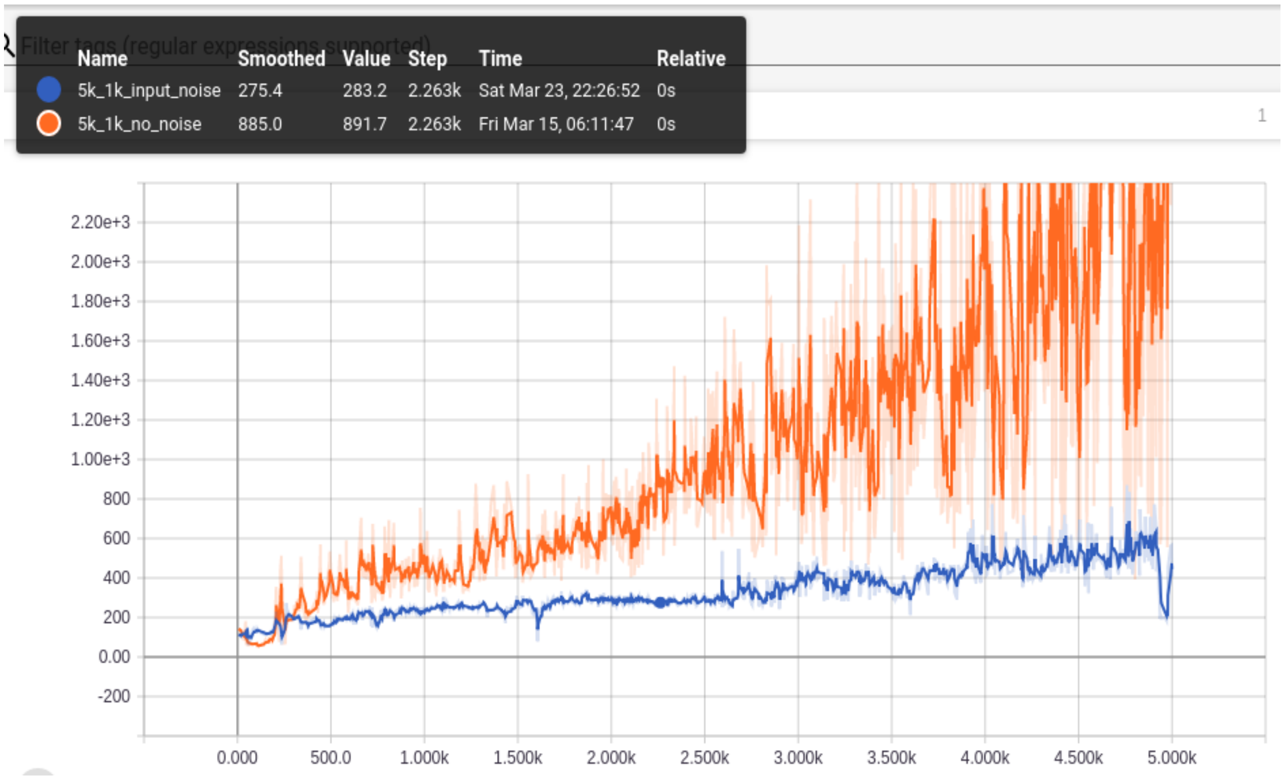
	Name	Smoothed	Value	Step	Time	Relative
●	5k_1k_OUNoise	412.6	358.5	4.449k	Thu Mar 14, 21:10:41	0s
○	5k_1k_input_noise	481.9	474.2	4.449k	Sat Mar 23, 22:26:52	0s

Fig 7. Chart comparing action space noise with noise in parameter space

4.3 Experiment 3 - Noise vs No noise

We finally compared the leading modification so far (noise added to inputs) with the inputs without noise. Plappert et al. (2018) noted that DDPG has the ability to explore continuous action space without any perturbations, and indeed, we found this to be the best modification for the humanoid environment that we used in this study.

Another really important thing to note with the exploration without perturbations is how the score continuously increases with increasing number of episodes, reaching a score as high as almost 5,000 within 5,000 episodes, with occasional worse off scores which allowed for exploration. The upward trajectory of the scores are such that it is safe to say the increase will continue with increasing number of episodes.



	Name	Smoothed	Value	Step	Time	Relative
🔵	5k_1k_input_noise	275.4	283.2	2.263k	Sat Mar 23, 22:26:52	0s
🟠	5k_1k_no_noise	885.0	891.7	2.263k	Fri Mar 15, 06:11:47	0s

Fig 8. Chart comparing noise in input with no perturbations

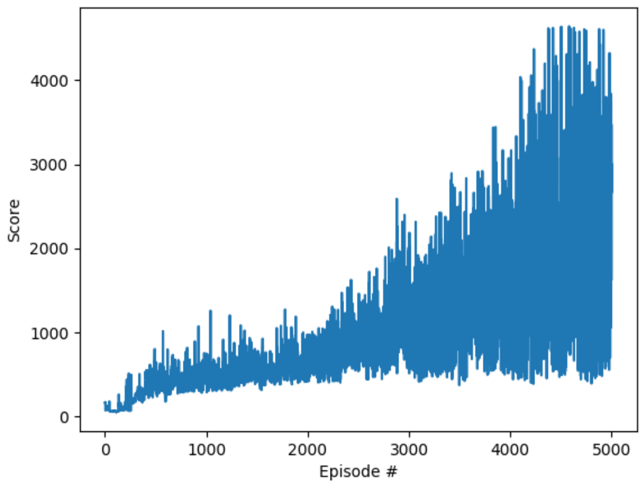


Fig 9. Chart showing the performance of DDPG algorithm with no perturbations

5.0 Conclusion

With the experiments we conducted, we saw that the DDPG algorithm without noise is by far the best performer in learning policies for the OpenAI gym Humanoid-v2 environment. We could not train the model for more than 5,000 episodes, which at times runs for about 48 hours on our computer; this limitation prevented actively testing even many more modifications, and it also means we could not allow models with upward trajectory to run for even longer to see what the end of the performance will be after training for an even much longer number of episodes. We expect that training for long number or episodes should allow for much better results since the Humanoid environment has a continuous action space, with a size of 17, making it impossible to be able to either exhaustively try out every action or narrow down to specific actions with some parameters other than the reward.

References

1. Hado van Hasselt and Marco A. Wiering (2007). Reinforcement Learning in Continuous Action Spaces. *IEEE.*
2. Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz (2018). Parameter Space Noise for Exploration. *ICLR*
3. Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wier-stra, Daan, and Riedmiller, Martin (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
4. Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare,Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al (2015). Human-level control through deep reinforcement learning. *518(7540):529–533*.
5. Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin (2014). Deterministic policy gradient algorithms. *InICML*.
6. Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess,Tom Erez, Yuval Tassa, David Silver & Daan Wierstra (2016). Continuous control with deep reinforcement learning. *arXiv:1509.02971v5 [cs.LG] 29 Feb 2016*.
7. Uhlenbeck, George E and Ornstein, Leonard S (1930). On the theory of the brownian motion. *Physical review, 36(5):823*.
8. Watkins, Christopher JCH and Dayan, Peter (1992). Q-learning. *Machine learning. 8(3-4):279–292*.
9. https://github.com/openai/baselines
10. https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-bipedal