Landing on the Moon with Deep Deterministic Policy Gradients

Adam Gjersvik
gjersvik@stanford.edu

***Abstract*—Ask Niel Armstrong, landing on the Moon is no easy feat - which is why ideally we would like to have computers do it for us. This work will explore an *extremely* simplified lunar environment and attempt to train a computerized agent to perform a successful moon landing. The action space in this environment consists of continuous control variables for 3 engines onboard a lunar lander that must be utilized by the computer agent in order to safely bring the lander to rest on the landing pad. The approach to solve this decision-making/continuous control problem is the application of a Deep Deterministic Policy Gradient (DDPG) algorithm. The objective of the DDPG agent is to learn an optimal state-action value function during mission training and then use it to attempt a successful lunar landing. Two different techniques for value function exploration are tested and the best one is chosen for additional training and evaluation. Ultimately, the DDPG agent is able to make successful lunar landings under favorable initial conditions, however, it was not able to gain mastery over all possible circumstances. It did, on the other hand, learn how to cheat in certain scenarios.**

***Index Terms*—Deep Deterministic Policy Gradient, Q-Learning, Decision Making Under Uncertainty**

I. Problem Description

The objective is to solve the 2D LunarLanderContinuous-v2 environment in the OpenAI Gym. in the environment, a landing pad is always located at coordinates $(0, 0)$ and the surrounding lunar surface is randomly generated with each attempt. The lander has perfect observability of the state-space and 3 engines to control: a main engine on its under-belly and two engines on each side.

Fig. 1. LunarLanderContinuous-v2 environment render.

A. Actions

The action space consists of two continuous variables both on the interval [-1, 1]. The first real-valued continuous action variable controls the main engine, where [-1, 0] results in engine-off and (0, 1] throttles from 50% to 100% power. The second real-valued continuous action variable controls both of the side thrusters where [-1, -0.5] fires the left thruster, [0.5, 1] fires the right thruster, and (-0.5, 0.5) is engine-off.

B. Rewards

The typical reward for moving from the top of the screen to the landing pad and achieving zero speed is between approximately 100 and 140 total points. Firing the main engine results in -0.3 points for each frame and additional reward is lost if the lander moves away from the landing pad. The episode finishes when the lander either crashes or comes to rest, receiving -100 or +100 points respectively, and +200 points is earned for landing on the pad. Landing outside of the landing pad is possible and fuel is infinite.

II. Deep Deterministic Policy Gradient

The Deep Deterministic Policy Gradient (DDPG) algorithm was chosen as the method to solve this problem because it is uniquely equipped for environments with continuous action spaces [1] [2]. To demonstrate how and why the DDPG algorithm is suitable for these types of problems, let us begin with the basic theory and motivation behind Q-Learning. If an optimal state-action value function is known, then in any given state the optimal action is:

$$a^*(s) = \arg\max_a Q^*(s, a) \tag{1}$$

When there is a finite set (of reasonable size) of discrete actions, then computing the maximum over the action space can be done directly in a reasonable amount of time. However, when the action space becomes continuous, the value of every state-action pair cannot be directly computed efficiently in order to compute the maximum. Thus the assumption is made that the optimal state-action value function $Q^*(s, a)$ is differentiable with respect to the action variable because the action space is contunous. This allows for an efficient gradient-based learning rule for a policy over the state space.

The DDPG algorithm consists of two concurrent learning tasks: learning a Q function and learning a policy. Before describing the first learning task, recall the Bellman equation which describes the optimal state-action value function for Q-Learning problems:

where s' P represents samples of s' drawn from the environment with probability distribution $P(\dot{|}s, a)$.

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s'\sim P}\left[R(s,a) + \gamma \max_{a'} Q^*(s',a')\right] \tag{2}$$

We wish to learn an approximator of $Q^*(s,a)$ with a dataset D of transitions (s,a,r,s') which we will refer to as $Q_\phi(s,a)$ with parameters ϕ. To measure how closely our approximator comes to satisfying the Bellman equation, the following mean-square Bellman error (MSBE) can be computed for any given approximation of the Q-function:

$$L(\phi,D) = \mathop{\mathrm{E}}_{(s,a,r,s')\sim D}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right] \tag{3}$$

The objective is then to minimize the MSBE loss function in order to approximate the true state-action value function. In DDPG, deep feed-forward neural networks are utilized as Q-function approximators and the network parameters are updated through backpropogation of the error gradient.

In order to train the Q-function deep neural network, the DDPG algorithm utilizes a replay buffer of past experiences. It is a dataset D of variable size containing previous transitions, making DDPG an off-policy algorithm. This simply means that the experience history data was collected using an outdated policy - in other words, using a policy different than the current behavior policy. It is important to note that a replay buffer that is too small can result in over-fitting of the most-recent data, and a replay buffer that is too large can significantly slow down training.

Another attribute of DDPG which is common with other Q-learning algorithms is the use of target networks. When minimizing the MSBE loss, we are trying to get the Q-function closer to following target:

$$r + \gamma(1-d)\max_{a'} Q_\phi(s',a') \tag{4}$$

This target, however, depends on the parameters of the network that are being updated, ϕ. To improve the stability of training, the target network is updated once per update by polyak averaging of the updated network and previous target network parameters:

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi \tag{5}$$

This results in minimizing the following MSBE loss:

$$L(\phi,D) = \mathop{\mathrm{E}}_{(s,a,r,s')\sim D}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_{\phi_{\text{targ}}}\left(s',\mu_{\theta_{\text{targ}}}(s')\right)\right)\right)^2\right] \tag{6}$$

where $\mu_{\theta_{\text{targ}}}(s')$ is the target policy. Now that we have established the means of learning an optimal state-action value function $Q_\phi(s,a)$ for continuous actions, we still need to learn a deterministic policy which gives the action that maximizes $Q_\phi(s,a)$. Recalling that we have made the assumption that the value function is differentiable with respect to the continuous action variable, we can simply perform gradient ascent with fixed Q-function parameters to find an optimal policy:

$$\max_\theta \mathop{\mathrm{E}}_{s\sim D}\left[Q_\phi(s,\mu_\theta(s))\right] \tag{7}$$

III. Exploration vs. Exploitation

As mentioned above, DDPG is an off-policy algorithm, meaning it learns from previous experiences collected from previous policies. In order to help the algorithm explore better during training, we can add noise to the output of the Q-function network or the network parameters themselves in order to experience more randomized transitions. These two techniques have both been demonstrated to improve training performance [3] [4].

A. Action Noise

Applying noise to the output of the Q-function network, the actions, is one technique of promoting exploration during network training. This allows the model to select a wider distribution of actions and experience a broader range of transitions. Various distributions can be used when applying noise to actions and it is common to use noise generated by a Ornstein-Uhlenbeck process. However, it has been shown that mean-zero Gaussian noise can work just as well [4]. It is also a possibility to scale down the noise as training progresses in order to collect more accurate training data.

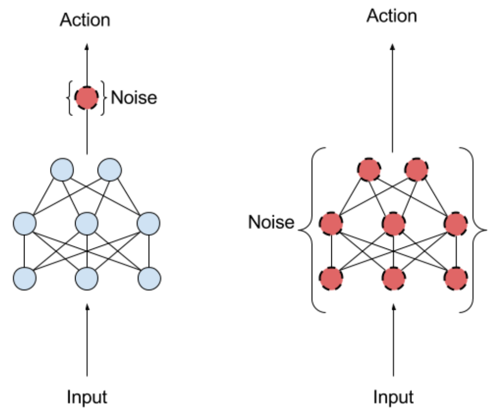

Fig. 2. Left: Application of action noise. Right: Application of parameter noise. Image taken from https://openai.com/blog/better-exploration-with-parameter-noise/ [4]

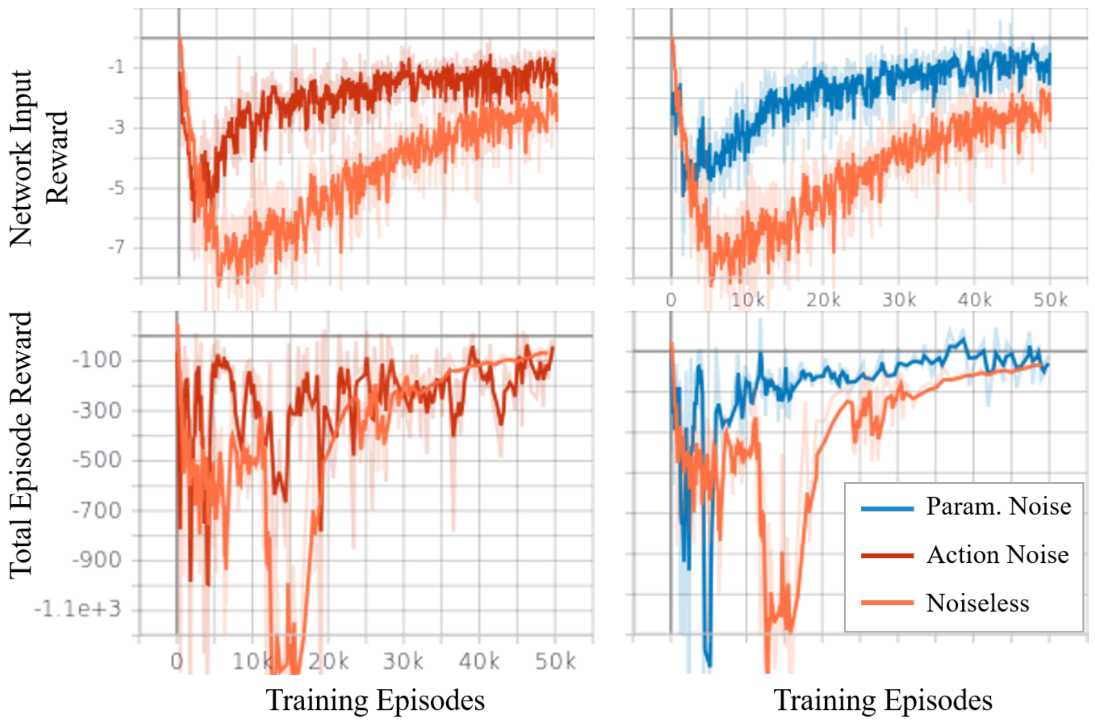

Fig. 3. Left column: Training performance with action noise. Right column: Training performance with parameter noise. Top row: Network input reward vs training episode. Bottom row: Total episode reward vs training episode.

B. *Parameter Noise*

In contrast to applying noise directly to the actions output from the learned policy, noise can instead be applied directly to the parameters of the learned policy. This type of exploration technique has been shown to accelerate training and improve performance in comparison to action noise. Often times the standard deviation of the parameter noise is scaled adaptively during training because it's unknown how the magnitude of the noise influences the policy. Adaptive noise scaling typically depends on how much the parameter perturbations affect the actions chosen.

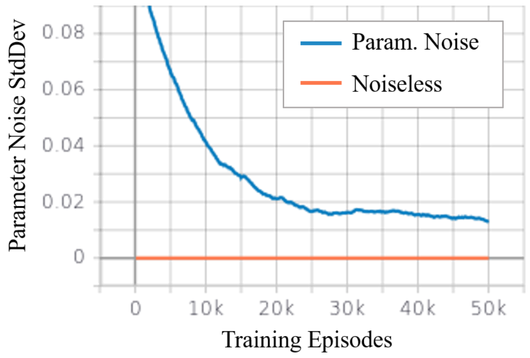

Fig. 4. Progression of adaptive parameter noise standard deviations during training.

IV. Performance Results

A. *Training*

To understand the impacts of the two exploration techniques described above, the Lunar Lander agent was trained for 50,000 episodes with no noise, action noise, and parameter noise applied separately. Figure 3 shows the training results for the three cases with the performance of the two noise techniques overlayed on noiseless training.

As seen in the training performance plots, both action and parameter noise achieve higher reward inputs to the network than the noiseless case as training progresses, with parameter noise slightly beating out action noise. In regard to the total episodic reward during training, the action noise network appears to acheive similar and sometimes lower reward than the noiseless case, while parameter noise results in consistently higher reward with less variance. This is not incredibly surprising considering that applying noise to the actions likely results in more random exploration, whereas adaptive parameter noise correlates exploration to the actual network parameters. Figure 4 additionally shows how the standard deviation of the adaptive parameter noise changes as training progresses.

It may be tempting to apply both exploration techniques during training in an attempt to accelerate learning even further, however, Figure 5 demonstrates that that may not be the case. Interestingly, the training performance could even be considered to have degraded when both noise techniques are incorporated. This is likely due to too much error in the model which ultimately degradeds training accuracy.

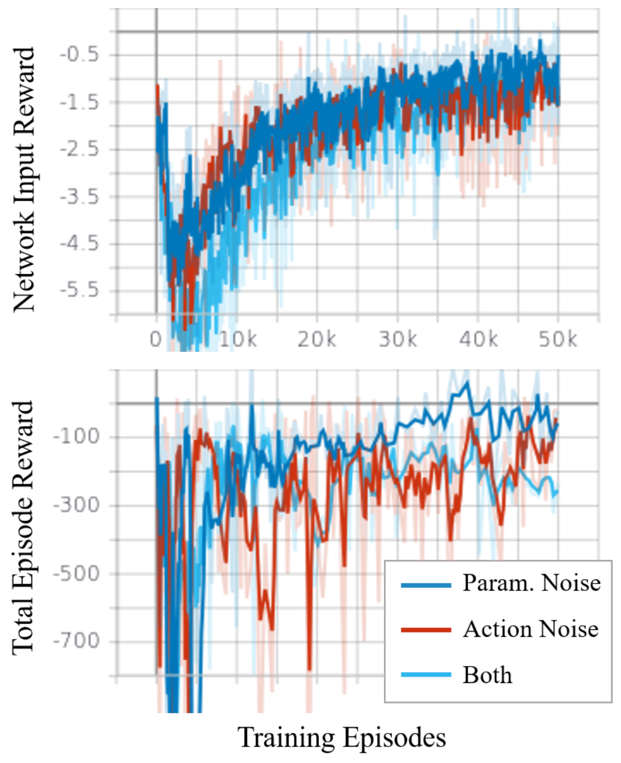

Fig. 5. Comparison of training with both action and parameter noise. Top: network input rewards during training. Bottom: Total episode rewards during training.

Notice that after 50,000 episodes, not even parameter noise training has reached the typical reward of a solved environment (100 to 140 points). From Figure 3 it appears that the networks from all training techniques may be approaching an asymptote, but there is really only one way to find out. Scaling up the number training episodes by a factor of 10 for only the parameter noise technique results in the training performance shown in Figure 6. Notice that after approximately 150k episodes some higher-reward experiences are explored that begin to drive learning up towards the rewards that are typically achieved during successful landings.

B. Evaluation

A qualitative evaluation of the learned models' behaviors after 50,000 training episodes reveals that, even though they have not solved the environment yet, they are all exceedingly good at stabilizing themselves after being initialized with some random downward/horizontal velocity. However, it appears they have not yet learned how to approach the landing pad and come to rest after stabilizing. Either they will hover to the left or right of the landing pad (depending on the direction of their initial velocities) or touch the lunar surface and then rocket away. This behavior is most likely due to the fact that they have not yet explored very many, if any, successful landings. In other words, they are stuck in a local minimum of stable flight near the landing pad with insufficient experience of actually landing on the pad to drive their learning.

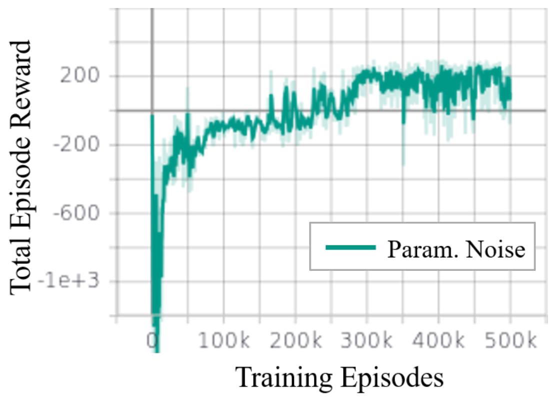

Fig. 6. Training performance with parameter noise exploration for 500,000 episodes.

Luckily, overcoming this local minimum can be achieved by simply training longer and allowing the agent to explore more. Figure 6 again demonstrates this jump in reward after discovering higher-reward episodes. With this more heavily trained neural network model, the agent is able to successfully land on the landing pad, however, only under certain initial conditions. A qualitative evaluation of the lander's performance reveals that the agent can only perform a successful landing when its initial velocity vector is favorable - and also by cheating.

If the lander is initialized with small horizontal velocities in either direction, it can gracefully maneuver down to landing pad and come to rest. When there is a large initial horizontal velocity, however, the lander tends to over-correct and is not able to slow its vertical velocity causing it to crash. It seems that with prolonged training the agent was able to learn that landing on the pad has high reward, but it forgot how to stabilize itself like it was so good at doing in the local minimum. This is a good example of how reinforcement learning algorithms don't necessarily retain all information that was previously learned when the training/exploration duration is increased.

Another interesting aspect of reinforcement learning and a strategy that this agent has picked up on is the ability to "cheat". The agent discovered that if it is able to come to rest on the lunar surface outside of the landing pad without crashing and if the lunar surface is not to steep, it can use one of its side engines to slide into the landing zone. This was obviously not the intended objective of the agent, but it was able to pick up this strategy by learning about the lunar environment - a good example of emergent behavior of reinforcement learning algorithms.

V. Conclusion

In this work, a Deep Deterministic Policy Gradient algorithm was applied to solve the OpenAI Gym Lunar Lander continuous environment. The DDPG algorithm was chosen because it is uniquely equipped and suitable for environments with continuous actions spaces. Two noise-based exploration strategies were investigated in order to accelerate learning:

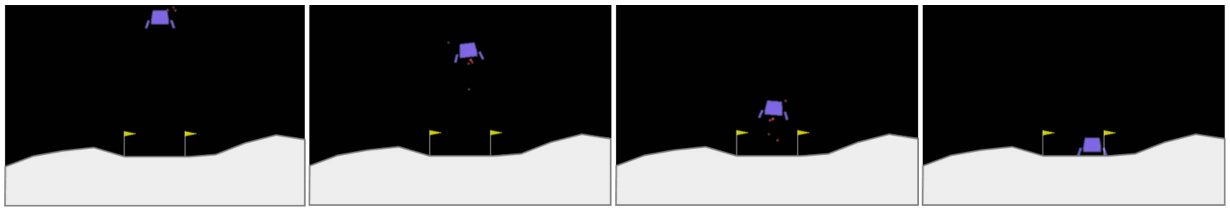

Fig. 7. Environment snapshots of a successful lunar landing. (The color of the lunar surface was modified for appearance.)

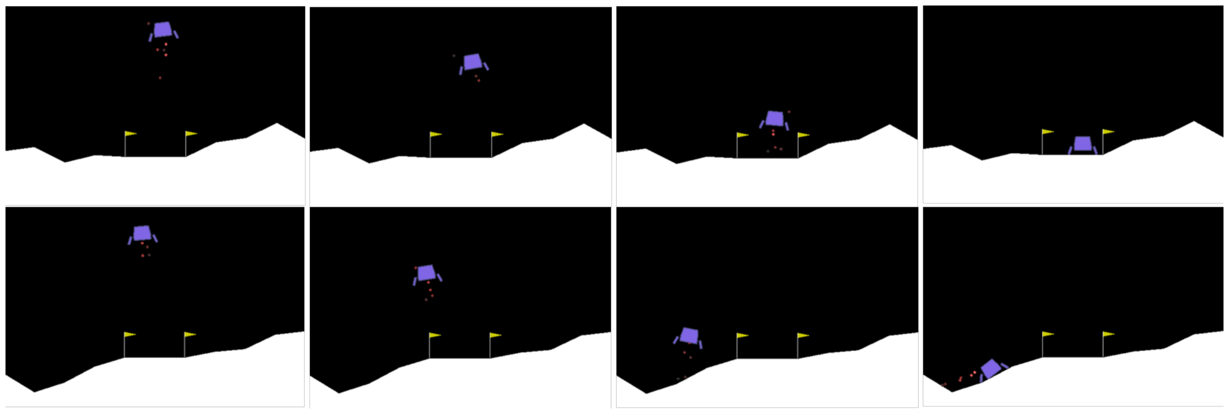

Fig. 8. Snapshots from more successful and unsuccessful landings.

noise applied to the action outputs of the value function neural network model, and noise applied to the neural network model parameters themselves. Parameter noise was demonstrated to result in better training performance and this exploration technique was chosen for further training of the agent in an attempt to solve the problem. By proloning the training duration with the parameter noise exploration strategy, the agent was able to make successful lunar landings under favorable initial conditions. Although it didn't quite master the environment with the training that it had, it could possibly achieve mastery with simply more training. It also happened to pick up some unexpected emergent behavior: cheating in order to get into the landing zone after already landing outside. Ultimately, the DDPG algorithm proved to be a very effective and efficient model for *nearly* solving the continuous Lunar Lander problem (only training for about an hour to achieve successful landings), and is demonstrably susceptible to the well known characteristics of general reinforcement learning techniques.

References

[1] D. Silver and G. e. a. Lever, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31st International Conference on MachineLearning*, vol. 32, 2014.

[2] T. Lillicarp, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Solver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," in *International Conference on Learning Representations*, 2016.

[3] OpenAI. (2018) OpenAI Spinning Up. [Online]. Available: https://spinningup.openai.com/en/latest/index.html

[4] OpenAI. (2017) Better Exploration with Parameter Noise. [Online]. Available: https://openai.com/blog/better-exploration-with-parameter-noise/