Learning to Run with Deep Reinforcement Learning

Alexander Freeman
Department of Computer Science
University of Bath
Bath, BA2 7AY
agf28@bath.ac.uk

Abstract

In this project we use the deep deterministic policy gradient algorithm (Lillicrap et al., 2015) to train a two-dimensional cheetah to run. We examine the effect of a selection of hyperparameters, leading to results comparable with existing literature. During training we observe the behaviour of the agent, finding that it discovers multiple strategies for achieving forward movement. Ultimately we see a smooth, natural, running gait. Finally, we propose extensions to the HalfCheetah environment that could be explored in future work.

1 Introduction

In this project we investigate how reinforcement learning can be applied to continuous control problems, in particular, with the HalfCheetah environment. Widely used in benchmarks for continuous control methods, the aim of this environment is for an agent to learn to control a two-dimensional 'cheetah' such that it moves forwards as fast as possible.

The environment we use is provided by the OpenAI Gym (Brockman et al., 2016), and runs in the MuJoCo physics simulator. While DeepMind offer their own suite of environments, including a similar 'cheetah' problem (Tassa et al., 2018), we choose the OpenAI version due to the volume of research that has been carried out in this setting. This will allow us to easily compare our agent with previous work when evaluating its success.

The HalfCheetah environment is closely based on the scenario described by Wawrzynski (2007), which features a two-dimensional cheetah with six degrees of freedom. The cheetah is controlled by applying torque to the joints labeled 1 to 6 in Figure 1, and the resulting reward is a function of the cheetah's forward velocity. Although previous work has shown that it is possible to learn successful policies from raw pixel inputs using convolutional neural networks (Lillicrap et al., 2015; Mnih et al., 2016), we will consider the simpler alternative, where our agent can directly observe the state of the environment. In the case of HalfCheetah, these observations consist of the cheetah's joint angles and angular velocities. This allows us to use simpler fully-connected network architectures which will be much faster to train on our relatively limited hardware. Because OpenAI Gym is only supported on Linux and MacOS, training has to be performed on a MacBook Air with a dual-core processor and no GPU.

With six continuous actions, it it not possible to solve this problem using methods that rely exclusively on an action-value function, such as Q-learning. Even if each action is discretised to one of three possible values, $[min, 0, max]$, the resulting action space still contains $3^6 = 729$ possible actions. This is already too large to realistically calculate $max_a Q(s, a)$, before even considering that a good policy will likely require much more precise control than this extremely coarse discretisation can provide (Lillicrap et al., 2015). We therefore require an algorithm that can directly produce continuous actions.

Project report for CM50270 Reinforcement Learning at the Department of Computer Science, University of Bath, UK. May 2019.

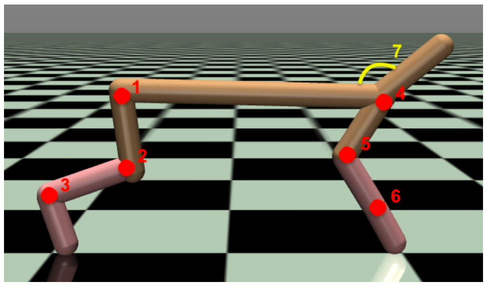


Figure 1: A screenshot of the OpenAI Gym HalfCheetah environment. The cheetah is controlled by applying torque to joints 1-6. The angle of the cheetah's head, 7, is fixed.

2 Approach

Although there now exist numerous methods for continuous control, our algorithm of choice for this problem is the deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015). One factor in this decision is its empirical performance on HalfCheetah (Duan et al., 2016; Henderson et al., 2018), in which it demonstrates faster convergence than other approaches "due to its greater sample efficiency" (Duan et al., 2016). Another notable benefit of DDPG is its relative simplicity in comparison to algorithms such as trust region policy optimisation (TRPO) (Schulman et al., 2015).

Although many alternative approaches can be applied to both continuous and discrete actions spaces (Mnih et al., 2016; Schulman et al., 2015, 2017), DDPG is limited exclusively to environments with continuous actions. However since we are focusing on one specific domain, this potential drawback does not change our evaluation that DDPG is the most suitable algorithm for the task.

2.1 DDPG background

Unlike a stochastic policy which maps a state to a probability distribution over possible actions, $P(a|s) = \pi(a|s)$, DDPG (as the name suggests) learns a deterministic policy, $a = \mu(s)$. As an actor-critic method, DDPG also learns an action-value function, $Q(s, a)$, which can be trained using the standard Q-learning update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \text{max}_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

As previously mentioned though, it is intractable to calculate $\text{max}_{a'} Q(s', a')$ over continuous actions. Instead, this is approximated with $\text{max}_{a'} Q(s', a') \approx Q(s', \mu(s'))$. That is, we use the actor to predict the best action, then the critic to obtain its value.

The actor and critic each use a deep neural network to represent the policy function and action-value function respectively, parameterised by θ^μ and θ^Q. We update the critic network by performing gradient descent to minimise the Q-learning temporal difference error. Just like stochastic policy gradient methods, the actor is trained by gradient ascent on the policy's performance objective, J. When using a deterministic policy however, the deterministic policy gradient theorem (Silver et al., 2014) states that the gradient of J is equal to the gradient of the action-value with respect to actions, multiplied by the gradient of the policy with respect to its parameters:

$$\nabla_{\theta^\mu} J = \nabla_a Q(s, \mu(s|\theta^\mu)|\theta^Q) \nabla_{\theta^\mu} \mu(s|\theta^\mu) \tag{2}$$

Silver et al. (2014) explain that as a result of this representation, "the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient".

Making use of function approximation, bootstrapping, and off-policy training, DDPG contains all three of the elements of what Sutton and Barto (2018) call 'the deadly triad'. This makes it prone to instability and prevents guarantees of convergence. To mitigate this, Lillicrap et al. (2015) implemented several techniques that were previously used to increase the stability of the DQN algorithm (Mnih et al., 2015).

The first of these is the addition of 'target networks' for approximating $\max_{a'} Q(s', a')$. We now maintain a second actor and critic network, parameterised $\hat{\theta^\mu}$ and $\hat{\theta^Q}$, which are slow-moving copies of the original networks. These are updated at each training step according to equations 3 and 4, with τ determining how fast the values change $(0 < \tau \ll 1)$. Equation 5 shows how the Q-learning update is modified to use these target networks.

$$\hat{\theta^\mu} \leftarrow \tau\theta^\mu + (1-\tau)\hat{\theta^\mu} \tag{3}$$

$$\hat{\theta^Q} \leftarrow \tau\theta^Q + (1-\tau)\hat{\theta^Q} \tag{4}$$

$$\max_{a'} Q(s', a') \approx Q(s', \mu(s'|\hat{\theta^\mu})|\hat{\theta^Q}) \tag{5}$$

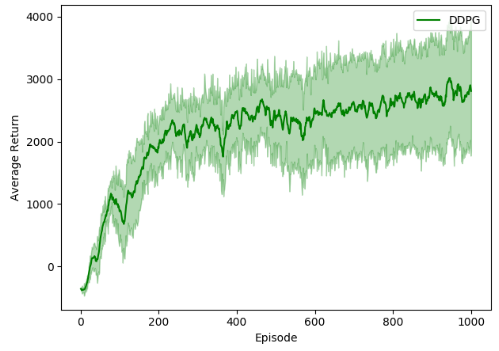
Instead of training online, using the most recent transition, the second modification is to train the networks from an experience replay memory. This is a buffer that stores a large collection of the most recent transitions, (s, a, r, s'). Training is now performed on a minibatch of transitions uniformly sampled from this experience memory, with no correlation between transitions.

In discrete action spaces it is common to enforce exploration though an ϵ-greedy policy, which occasionally selects a random action instead of the one with the greatest Q-value. With continuous actions though, this is clearly not possible. Instead we follow the approach of Lillicrap et al. (2015) and add temporally correlated noise to the output of the actor network. This can be seen in Algorithm 1 in Appendix A, which describes the complete training procedure for DDPG.

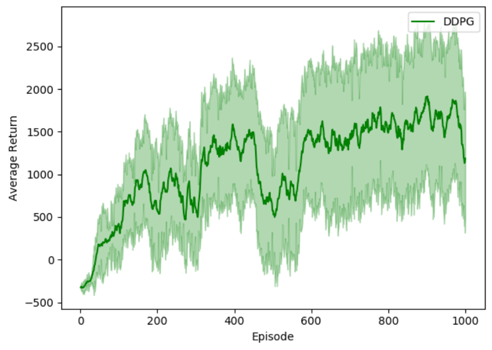
3 Experiments

We now examine the performance of our agent in the HalfCheetah environment. For most hyperparameters we use the values chosen in the original implementation of DDPG (see Appendix B), which are commonly used, and empirically show good performance.

One value we experiment with though is the reward scale σ. This number is multiplied with the reward obtained from the environment so that the reward observed by the agent is σr. While some work claims that using $\sigma = 0.1$ improves the stability of DDPG (Duan et al., 2016; Gu et al., 2016), others have found that using no scaling "yields much higher returns" (Islam et al., 2017). Henderson et al. (2018) explain that implementation details often vary, which can have a significant impact on performance, and may be a factor in this lack of consensus. Since we are using our own implementation of DDPG the best option must be determined experimentally. In Figure 2 we see that scaling the reward results in significantly lower returns, and actually appears to reduce the stability of learning in this environment. Figure 2b displays large sudden drops in performance and greater standard errors than Figure 2a, hence we use unscaled reward in future training runs.



(a) Original reward.



(b) Reward scaled by $r = 0.1r$.

Figure 2: Mean return $\pm$ standard error, averaged over 5 agents. For clarity, curves are smoothed with a 5-point moving average. The original unsmoothed learning curves can be found in Appendix C.

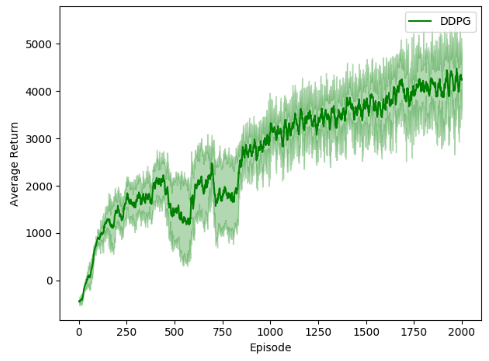


Figure 3: Retraining agents from Figure 2a over twice as many episodes.

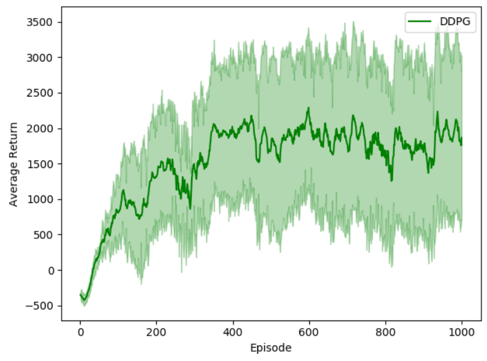


Figure 4: Retraining agents from Figure 2a with the batch size increased to 128.

Since the curve in Figure 2a has not plateaued, it is reasonable to believe that these agents will continue to improve given more training. We test this hypothesis by retraining over twice as many episodes. Figure 3 shows that the agents have indeed continued learning, and now reach an average return of 4180 (averaged over the last 100 episodes). While it is likely that we would see further improvement with even more training, this would be prohibitively time-consuming; generating Figure 3 took 35 hours.

Instead, we try to accelerate learning by using a larger batch size. This causes the agent to replay more transitions, hence potentially leading to higher returns. To quickly see if this approach has merit we train 5 agents over 1000 episodes (Figure 4). Surprisingly, we find that learning fails to progress much beyond an average return of 2000, yielding far lower returns than before. Although it is possible that adjusting the learning rate of the actor or critic would result in higher returns, this kind of hyperparameter tuning is unfeasible to perform in the time available on our hardware.

4 Observations

While average return gives a numerical measure of performance, this does not directly give us any insight into the actual behaviour of our agent. We therefore render the environment to observe and record what is happening.

Before training we see erratic and uncoordinated actions, often resulting in a net movement backwards. This explains the negative return seen in the first few episodes of all previous learning curves. By 50 episodes the cheetah is making forward progress, but not in the way we had expected. The agent firstly performs a somersault, landing on its back, then simply waves its legs in the air (Figure 5). Amazingly, this results in forward movement. Over the next 50 or so episodes the agent obtains greater returns just by perfecting the somersault and resuming the same behaviour thereafter. We hypothesise that this behaviour is seen early on since it does not require the fine control of a usual running gait, hence the agent quickly discovers that this strategy yields positive rewards. Since the agent's sole objective is to maximise return, and this is the only way it knows how to obtain a positive reward, the agent keeps doing this until a better alternative is found after sufficient exploration. This is a nice demonstration of how reinforcement learning agents can discover novel strategies for achieving a broadly-defined goal.

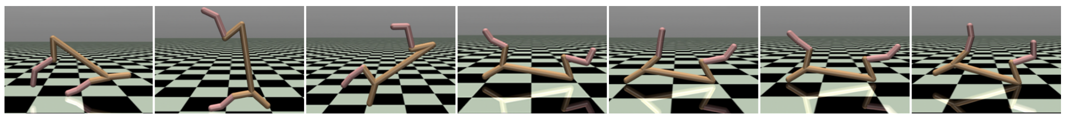


Figure 5: A sequence of frames after 50 episodes.

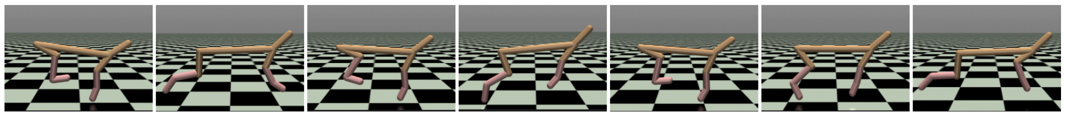


Figure 6: A sequence of frames after 1000 episodes.

By 1000 episodes the cheetah is displaying the expected running gait, using its hind leg to propel itself forwards much faster than before (Figure 6). It is not perfect though; we find the agent often loses balance and falls forwards into the ground. The greater returns seen in subsequent episodes come as a result of increasing stability and smoothness.

5 Evaluation

We compare our results to those of Henderson et al. (2018), who investigated multiple different algorithms for continuous control. This includes DDPG, meaning we can examine variations due to implementation in addition to how our approach compares to alternative methods. Although Henderson et al. (2018) report average return as a function of timesteps instead of episodes, the OpenAI HalfCheetah environment automatically ends an episode after 1000 steps and has no condition for ending sooner. Since we have used the same environment, we can easily compare their results with our own.

In Figure 7 we overlay our learning curve from Figure 3 onto those created by Henderson et al. (2018). We see that our DDPG agent performs similarly to theirs, achieving almost identical returns at the end of training. The most notable differences are the two dips in our learning curve; these were caused by two agents which saw sharp declines in performance for a few episodes. Duan et al. (2016) also observed that with DDPG "the performance of the policy can degrade significantly during training". Most importantly, we find that our implementation of DDPG performs better than we could have achieved with alternative algorithms.

In addition to the numerical results, our observations have revealed that the cheetah learns a natural, smooth, running gait as one would expect to see in the real world. This is important since return alone can be difficult to interpret and does not give any insight into the behaviour of the agent. As we saw previously, increased return does not necessarily mean the agent is doing what we would expect.

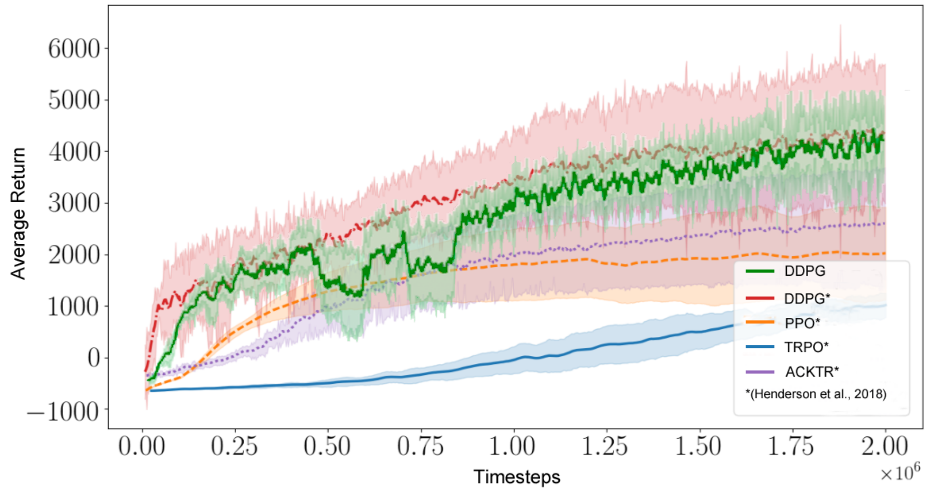


Figure 7: Comparison of deep learning algorithms in the HalfCheetah environment, adapted from Henderson et al. (2018). 2×10^6 timesteps is equivalent to 2000 episodes.

6 Conclusions

In this work we have investigated the deep deterministic policy gradient algorithm for continuous control in the HalfCheetah environment. We have achieved numerical results comparable to the literature, and shown that this yields a natural running gait.

6.1 Limitations and future work

Because of the time taken for training we had to limit our experiments to 2000 episodes, averaged over 5 agents. With more time and computational resources we would continue to train agents until we see no improvement in return, and observe the behaviour at this point. We would also average experiments over more agents for greater confidence in numerical results. Our agent could then be improved by incorporating the work of Hou et al. (2017), who found that prioritised experience replay could be used with DDPG to give faster and more stable learning.

There are many expansions to the HalfCheetah environment that would be interesting to explore in the future. The first of these would be to add two extra legs, turning the HalfCheetah into a 'FullCheetah'. With a higher-dimensional action space, this would be a more challenging problem. We would examine how well DDPG and other applicable methods scale to this new environment. Additionally, the current reward function leads to an agent that learns to run as fast as possible. In nature though, efficiency is an important consideration; animals have different gaits for walking and running, with a trade-off between efficiency and speed. We would introduce an 'energy penalty' into the reward function and observe how the policy changes as energy use becomes a consideration.

6.2 Final remarks

This project has been an exciting opportunity for us to explore a different area of reinforcement learning, working with continuous actions. Although we encountered a couple of difficulties during implementation, prior experience with TensorFlow made debugging significantly easier. Once our agent was working it was enjoyable observing it during training, particularly when we saw the behaviour in Figure 5 where it somersaulted onto its back. With reinforcement learning now able to learn good polices for challenging high-dimensional environments such as humanoid walkers, we are interested to see what problems will be tackled in the future, and how new approaches will yield better policies and more sample-efficient learning.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI gym. *arXiv preprint arXiv:1606.01540.*

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *The International Conference on Machine Learning*, pages 1329–1338.

Gu, S., Lillicrap, T. P., Ghahramani, Z., Turner, R. E., and Levine, S. (2016). Q-Prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247.*

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *The 32nd AAAI Conference on Artificial Intelligence*.

Hou, Y., Liu, L., Wei, Q., Xu, X., and Chen, C. (2017). A novel DDPG method with prioritized experience replay. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 316–321. IEEE.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. (2017). Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133.*

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971.*

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *The 33rd International Conference on Machine Learning*, volume 48, pages 1928–1937. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540).

Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. (2015). Trust region policy optimization. In *The 32nd International Conference on Machine Learning*, volume 37, pages 1889–1897. JMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *The International Conference on Machine Learning*, pages 387–395.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*, volume 2. MIT Press.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T. P., and Riedmiller, M. A. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Wawrzynski, P. (2007). Learning to control a 6-degree-of-freedom walking robot. In *EUROCON 2007-The International Conference on "Computer as a Tool"*, pages 698–705. IEEE.

Appendix A DDPG algorithm

Algorithm 1 Deep Deterministic Policy Gradient

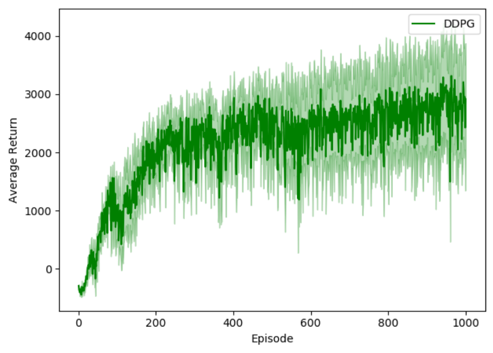
```
Initialise actor network with weights $\theta^\mu$
Initialise critic network with weights $\theta^Q$
Initialise target critic network with weights $\hat{\theta^\mu} \leftarrow \theta^\mu$
Initialise target actor network with weights $\hat{\theta^Q} \leftarrow \theta^Q$
Initialise replay memory $M$
Initialise noise generation process $N$

Observe initial state $s_1$
**for** $t = \{1, 2, ..., T\}$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + N_t$
    Execute action $a_t$, observe reward $r_t$ and state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $M$
    Uniformly sample minibatch of transitions $B = (s_i, a_i, r_i, s_{i+1})$ from $M$

    Calculate temporal difference target:
        $y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}|\hat{\theta^\mu})|\hat{\theta^Q})$

    Update critic network using gradient descent to minimise TD error:
        $$\nabla_a \frac{1}{|B|} \sum_i (Q(s_i, a_i|\theta^Q) - y_i)^2$$

    Update actor network using the sampled policy gradient:
        $$\nabla_{\theta^\mu} J \approx \frac{1}{|B|} \sum_i \nabla_a Q(s_i, \mu(s_i|\theta^\mu)|\theta^Q) \, \nabla_{\theta^\mu} \mu(s_i|\theta^\mu)$$

    Update target network parameters:
        $\hat{\theta^\mu} \leftarrow \tau\theta^\mu + (1-\tau)\hat{\theta^\mu}$
        $\hat{\theta^Q} \leftarrow \tau\theta^Q + (1-\tau)\hat{\theta^Q}$
**end for**
```

Appendix B Experimental details

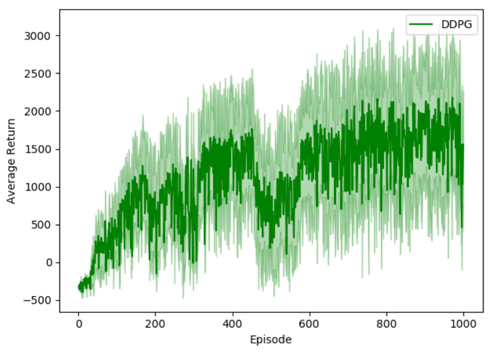
Table 1: Hyperparameter choices.

Hyperparameter	Value
Actor learning rate	0.0001
Critic learning rate	0.001
Actor network nodes	[400, 300]
Critic network nodes	[400, 300]
Experience replay buffer size	1×10^6
Batch size	64
Reward scale	1
τ	0.001
γ	0.99

Appendix C Supplementary learning curves



(a) Original reward.



(b) Reward scaled by $r = 0.1r$.

Figure 8: Mean return $\pm$ standard error, comparing the effect of reward scaling.

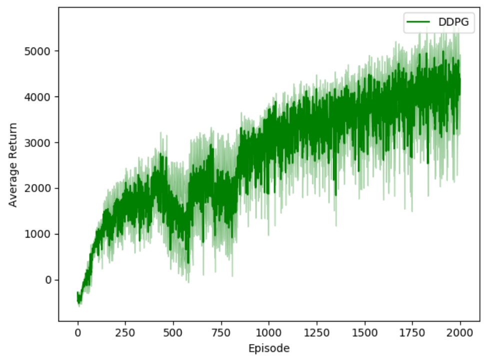


Figure 9: Retraining agents from Figure 8a over twice as many episodes.

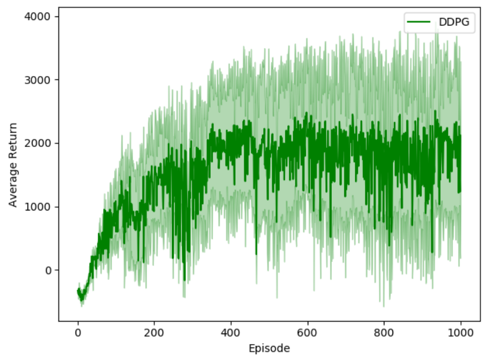


Figure 10: Retraining agents from Figure 8a with the batch size increased to 128.