Cartpole Swing-Up using Deep Q Network and Deep Deterministic Policy Gradient

Dominik Urbaniak, Mayuran Surendran

***Abstract*— We solve the cartpole swing-up problem using a Deep Q Network and an actor-critic using deep function approximators called Deep Deterministic Policy Gradient. We demonstrate that both algorithms can learn the optimal policy. Furthermore, we point out the close connection between both algorithms and list their respective advantages and disadvantages.**

I. INTRODUCTION

In recent years, researchers in the field of artificial intelligence have made substantial progress in solving high-dimensional complex tasks using large, non-linear function approximators to learn value or action-value functions. Much of this progress has been achieved by combining advances in deep learning with reinforcement learning. Mnih et al. introduced an algorithm called "Deep Q Network" (DQN) that uses a deep neural network to estimate the action-value function, which can solve problems with high-dimensional observation spaces and discrete low-dimensional action spaces [1]. Prior to this achievement, non-linear function approximators were avoided as guaranteed performance and stable learning was deemed impossible. By adapting advancements made by Mnih et al., Lillicrap et al. introduced a model-free, off-policy actor-critic named "Deep Deterministic Policy Gradient" (DDPG), which expands the idea of "Deterministic Policy Gradients" (DPG) by Silver et al. and can operate in continuous action spaces [1][2][5].

In this work we apply DQN and DDPG to solve the cartpole swing-up problem. In section II, we provide background knowledge to understand the theory behind DQN and DDPG. In section III, we describe the cartpole environment used for the experiments. In section IV and V, the algorithms of DQN and DDPG are explained and in section VI the respective results are compared and discussed. Section VII concludes this paper.

II. BACKGROUND

A. Reinforcement Learning Setup

The standard setup of reinforcement learning consists of an agent that interacts with the environment ε by selecting actions over a sequence of time in order to maximize a cumulative reward. The environment ε may be stochastic. Therefore, we model this as a finite Markov decision process (MDP) that comprises of a state space *S*, an action space *A*, an initial distribution with density $p_1(s_1)$, a stationary transition dynamics distribution with conditional density $p(s_{t+1}|s_t, a_t)$ that satisfies the Markov property $p(s_{t+1}|s_1, a_1, \dots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$ and a reward function $r(s_t, a_t)$ which maps a state-action pair to a scalar [2][5].

In general, the agent follows a stochastic policy π that maps a state to a set of probability measures on *A*. The agent uses π to interact with the MDP to generate trajectories of states, action and rewards. In each step these rewards are discounted by a factor $\gamma \in [0, 1]$ such that the future discounted return at time t can be defined as $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. The goal of the agent is to learn a policy that maximizes the expected return from the start distribution $J = E_{r_i, s_i \sim \epsilon, a_i \sim \pi}[R_1]$ [2][5].

Moreover, $Q^\pi(s_t, a_t) = E_{r_{i>t} s_{i>t} \sim \varepsilon, a_{i>t} \sim \pi}[R_t | s_t, a_t]$ is defined as the action-value function and describes the expected return after taking an action a_t in state s_t and following π after. It follows an identity known as Bellman equation which is given as

$$Q^\pi(s_t, a_t) = \\ = E_{r_t, s_{t+1} \sim \varepsilon}\left[r(s_t, a_t) + \gamma E_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]\right].$$

The action-value function is learned by constructing a function approximator parameterized by θ^Q and minimizing the mean-squared Bellman error (MBSE) which can be formulated as

$$L(\theta^Q) = E_{s_t \sim \rho^\pi, a_t \sim \pi, r_t \sim \varepsilon}[(Q^\pi(s_t, a_t | \theta^Q) - y_t)^2]$$

where $y_t = r(s_t, a_t) + \gamma Q^\pi(s_{t+1}, a_{t+1} | \theta^Q)$ and ρ^π refers to the discounted state visitation distribution for a policy π. Differentiating the MSBE with respect to θ^Q yields the gradient of the action-value function.

B. Experience Replay

Both DQN and DDPG use neural networks as nonlinear function approximators for which many optimization algorithms require independently and identically distributed samples. However, this is not true for most applications in the field of reinforcement learning as samples are generated from exploring sequentially. To tackle the problems of correlated data and non-stationary distributions, an experience replay mechanism based on a replay buffer is used. Consequently, the use of the experience replay mechanism results in removing correlations in the observation space and smoothing over changes in the data distribution. The replay buffer is a finite sized cache storing previous transitions as tuples. The experience replay mechanism randomly samples minibatches of previous transitions in order to train the function approximators. The size of the replay buffer is one of the hyperparameters which must be tuned: using most recent data or too much experience may result in overfitting and slow training, respectively [3][5][6].

C. Target Networks

Using nonlinear function approximators such as neural to approximate the Q-function results in unstable and divergent behavior during training which is mainly because the network is also used to calculate the target value during minimization of the MSBE. Therefore, the update is likely to diverge. Hence, DQN uses a target network which has a set of parameters θ' similar to the parameters θ of the Q-network. The target network used in DQN is updated every C steps and is fixed between updates. DDPG uses one target network for its actor and critic function, respectively, which are updated every step using "soft" target updates. Using the target network to calculate the MSBE, it can be reformulated as

$$L(\theta^Q) = E_{s_t \sim \rho^\pi, a_t \sim \pi, r_t \sim \varepsilon}[(Q^\pi(s_t, a_t|\theta^Q) - y_t)^2]$$

where $y_t = r(s_t, a_t) + \gamma Q^{\pi'}\left(s_{t+1}, a_{t+1}|\theta^{Q'}\right)$ [3][5][6].

III. Cartpole swing-up problem

The system consists of a cart running on a track which has a pole mounted on it. The goal is to balance the pole in the upright position by only applying external force F to the cart and requiring the cart to be in the center of the track.

The state space is four-dimensional and consists of position x, cart velocity $\dot{x}$, angular position θ as well as the angular speed $\dot{\theta}$. Consequently, the state vector is given as $\mathrm{s} = \left(x, \dot{x}, \theta, \dot{\theta}\right)$. The reward signal is given as

$$r(s, a) = -\left(1 - e^{-0.5(j - j_{target})T^{-1}(j - j_{target})'}\right)$$

where $\mathrm{j} = (x, \sin\theta, \cos\theta)$ is equal to the current position and $\mathrm{j}_{target} = (x, \sin\theta, \cos\theta)$ is equal to the goal position of the cart and pole. The inverse of T is called precision matrix and is given as

$$T^{-1} := A^2 \begin{bmatrix} 1 & l & 0 \\ l & l^2 & 0 \\ 0 & 0 & l^2 \end{bmatrix}$$

where l is the length of the pole and A^2 is a scalar that controls the width of the reward parabola which was set to 1 for our experiments. We use the equations of motion derived in the dissertation of Deisenroth [4]. Additional information can be found in the appendix.

IV. Deep Q Network

A. Algorithm

In conventional Q-Learning, the approximation of the action-value function Q proved to be a main challenge. Deep neural networks emerged as a powerful function approximator and Mnih et al. effectively applied them to solve complex reinforcement learning tasks [1][3]. Contrary to Mnih et al., we work with the cartpole swing-up environment. The main difference is the state space. Atari games provide images as inputs which lead to a much larger state space. Hence, instead of convolutional neural networks we utilize a fully connected feedforward network with three hidden layers. We use this neural network with weights θ^Q to approximate the optimal action-value function Q*:

$$Q(s, a|\theta^Q) \approx Q^*(s, a)$$

DQN is an online method that performs a step of gradient descent in order to minimize aforementioned MSBE with respect to the network weights. The network is initialized to take an input consisting of the state variables of the environment as well as the action a. Furthermore, a discrete set of eight actions, evenly distributed in the range of the action interval, is available for the DQN agent to choose its action from. During training we weighted the current and goal function accordingly such that the position x of the cart is less important than the angle θ:

$$j_{DQN} = (0.5x^2, sin\theta, 2cos\theta)$$

$$j_{DQN,target} = (0,0,2)$$

This was done due to the limited number of discrete actions that the agent can select. Furthermore, we use a replay buffer to perform the experience replay mechanism and initialize a target network in order to prevent divergence during the training. For the DQN agent we chose an ε-greedy strategy with the following threshold.

$$\varepsilon_{thresh} = \frac{mean(\boldsymbol{e})}{w}$$

Here, **e** represents the array of the last ten rewards which is updated only when another full episode was achieved, w = -500. To comply with the task specifications, we clip $\dot{\theta}$ and $\dot{x}$ to its limits and terminate a game for the DQN agent if it exceeds the boundaries of x. Additional information about the training procedure and environment setup are detailed in the appendix.

B. Results

The algorithm is expected to reach a threshold of accumulated rewards for ten consecutive games in order to trigger the early stopping condition that determines a successful training session. In the beginning of the training the agent fails to stay within the boundaries of x for an average of 170 games and learns the policy after an average of 250 episodes. This fast convergence is mainly the result of the combination of the chosen reward function and the exploration-exploitation strategy.

Fig. 1 illustrates the fast convergence once the agent achieved the first full episode. This results in a shorter runtime, since full episodes execute more computations. Fig. 2 shows the development of the cumulative reward for all games that were not terminated early averaged over the ten trained models also indicating when a model achieved the early stopping condition. The training is executed with an action space of eight actions. The action space parameter influences the runtime as the DQN agent computes a Q-value for each action and afterwards decides to take the action which received the maximum Q-value. However, this attribute also gives our DQN agent more flexibility. As shown in Fig. 3 the agent can handle different action spaces. Hence, with a small decrease in performance a better runtime in the simulation can be achieved. Our DQN algorithm proves to robustly and reliably solve the cartpole task.

V. Deep deterministic policy gradient

A. Algorithm

In the last section DQN successfully solved the problem.

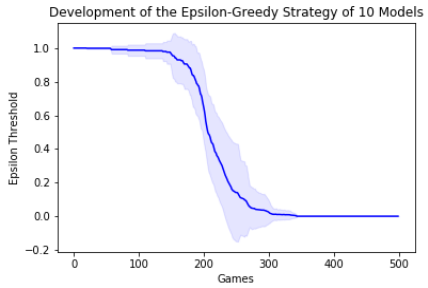

Figure 1. Cumulative rewards for different action spaces

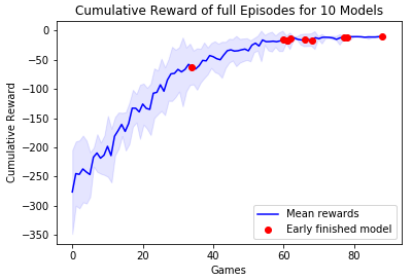

Figure 2. Cumulative rewards for different action spaces

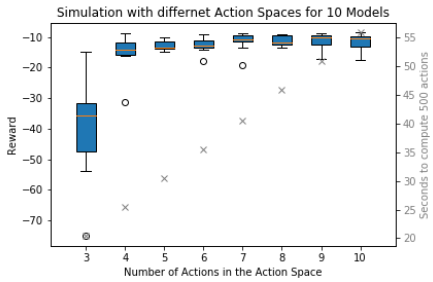

Figure 3. Cumulative rewards for different action spaces

Nevertheless, most tasks involve continuous high-dimensional action spaces. One approach to applying DQN to problems with continuous action spaces is the use of finer discretization of the action space. However, this method leads to an exploding number of discrete actions known as the curse of dimensionality. An optimization of action a_t at every timestep using the greedy policy would be too slow and impractical for unconstrained function approximators and nontrivial, high-dimensional action spaces. Therefore, we use an actor-critic based on the DPG algorithm called DDPG that can operate in continuous action spaces and concurrently learns the deterministic policy and action-value function given by $\mu(s_t|\theta^\mu)$ and $Q^\mu(s_t, a_t|\theta^Q)$, which are approximated by neural networks parameterized by θ^μ and θ^Q, respectively. $\mu(s_t|\theta^\mu)$ maps a state directly to the action that maximizes action-value function $Q^\mu(s_t, a_t|\theta^Q)$ such that $\max_a Q(s, a) \approx Q(s, \mu(s))$. Here, the superscript μ in $Q^\mu(s_t, a_t|\theta^Q)$ states that the action-value function is dependent on the deterministic policy μ [5][6].

Above we have defined the Bellman equation for an agent following a stochastic policy π. If the target policy μ is deterministic, which means that states are directly mapped to actions, the Bellman equation is given as

$$Q^\mu(s_t, a_t) = E_{r_t, s_{t+1} \sim \varepsilon}\left[(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))\right]$$

by omitting the inner expectation. Hence, the expectation only depends on the environment which makes it possible to learn the action-value function Q^μ off-policy using transitions generated from a different stochastic policy β. Supposing that action space A is continuous, Q^μ is assumed to be differentiable with respect to the parameters θ^μ of the actor function. Therefore, the actor function is updated by applying the chain rule to the expected return from the start distribution J with respect to θ^μ, which is also referred to as policy gradient [5]:

$$\nabla_{\theta^\mu} J \approx E_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] =$$

$$= E_{s_t \sim \rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t}\right]$$

DPG uses an actor target network μ' and a critic target network Q' which are used to calculate the target values for minimizing the MBSE. The weights $\theta^{\mu'}$ and $\theta^{Q'}$ of the target networks are changed using "soft" target updates by polyak averaging: $\theta' = \tau\theta + (1 - \tau)\theta'$ with $\tau \ll 1$. Consequently, the target values change slowly and track the learned networks which improves the stability of training [5].

An agent trained on-policy using the DDPG algorithm will not explore a wide range of actions due to the deterministic nature of the policy. However, the problem of exploration can be tackled by constructing an exploration policy μ' by adding noise sampled from a noise process N. The original paper of DDPG suggests using a time-correlated noise generated by the Ornstein-Uhlenbeck (OU) process, which enables efficient exploration in physical control problems with inertia:

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + N$$

We use a zero-mean noise with an initial standard deviation of 0.2 and linearly decreasing it to 0.05 during the training process. θ is fixed as 0.15 [5][6][7]. Training hyperparameters are given in Table II of the appendix.

B. Results

We tried three different training configurations: without OU noise, OU noise with constant standard deviation and OU noise with linearly decreasing standard deviation. The training of the agent is completed after either 1000 games or when reaching a threshold of accumulated rewards for ten consecutive games. In case of convergence (~80% of the trials), the agent solves the cartpole swing-up problem by successfully learning the optimal policy for all ten runs in every training configuration. The cumulative reward averaged over ten runs is illustrated in Fig. 4, where each data point is calculated by considering the running average of the last ten data points. It can be concluded that all training configurations result in the similar training behavior with models using exploration scoring slightly better during the end stage of the training. Furthermore, Fig. 5 depicts the average number of episodes it took the algorithm to reach the threshold of accumulated rewards for ten consecutive times in order to stop the training process earlier. The number of training runs that were stopped early are written in each bar. Here, we can see that training using OU noise with

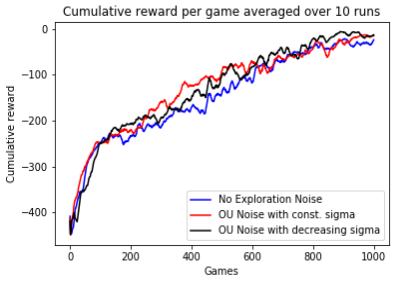

Figure 4. Cumulative reward per game averaged over 10 runs

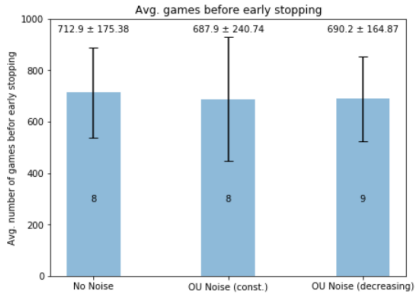

Figure 5. Statistical data concerning early stopping

decreasing standard deviation leads to a lower average number of games needed for early stopping and a total of nine early stoppings compared to the eight early stopping for the first and second configuration. Furthermore, the error bar for the second configuration is bigger which indicates that the model is not able to exploit the learned policy as much as the two other models, resulting in suboptimal actions taken by the agent and ultimately preventing it from early stopping.

VI. Discussion

In this section, we compare the performance of DQN and DDPG in the cartpole environment. Fig. 6 illustrates the cosine of the mean angle averaged for ten runs. All three configurations of DDPG learn the same optimal policy whereas DQN takes more run-up in order to swing the pole into the upright position. Furthermore, DDPG without exploration noise and DQN need longer to stabilize the pole in the goal position. However, it must be noted that DQN does not have the ability to choose the optimal action at each time step because it uses a finite number of discrete actions. To compensate for the limited actions the DQN agent moves further distances along the x-axis. Fig. 7 shows the actions performed by the best model of each training configuration. The DQN agent can only choose a minimum force of 1.4 Newton. Hence, with the pendulum in the upright position, it shows an uneven behavior, oscillating between -1.4 Newton and 1.4 Newton, whereas the DDPG agents are able to stabilize the pole using a smaller interval of actions.

VII. Conclusion

This paper uses the DQN and DDPG to train agents which solve the cartpole swing-up problem by learning the optimal policy. Both algorithms show satisfactory performance. The DQN algorithm achieves a 100% convergence rate for ten models and the DDPG agent achieves a smooth and fast task completion.

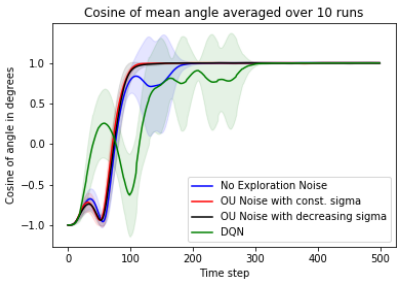

Figure 6. Comparison of the mean angle at each time step

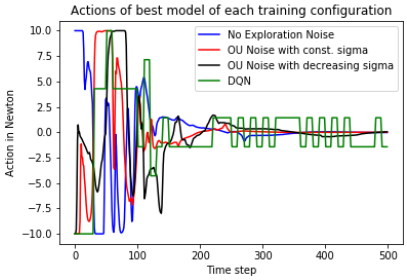

Figure 7. Actions of best model of each training configuration

VIII. Appendix

TABLE I. Cartpole Environment

Specification	***Values***	***Unit***
Bounaries for cart position *x*	[-6, 6]	*m*
Boundaries for cart velocity $\dot{x}$	[-10, 10]	*m/s*
Boundaries for anglular pos. θ	[$-\pi$, π]	*rad*
Boundaries for angular speed $\dot{\theta}$	[-10, 10]	*rad/s*
Force *F*	[-10, 10]	*N*
Gravity *g*	9.82	*Nm/s²*
Pole length *l*	0.6	*m*
Mass of pole and cart	0.5	*kg*
Friction coefficient *b*	0.1	*N/m/s*

TABLE II. Training Hyperparameters

Specification	***DQN***	***DDPG***
Maximum number of games	500	1000
Maximum number of iterations	500	500
Threshold for early stopping (for different reward functions)	-20	-55
Time step dt	0.1	0.01
Discount factor γ	0.99	0.99
Batch size for training	200	128
Target network update (Iterations C / τ)	100	0.01
Network layer sizes	[5, 120, 200, 60, 1]	A: [4, 512, 128, 1] C: [4, 1024+1, 512, 300, 1]

REFERENCES

[1] V. Mnih et al., “Playing Atari with deep reinforcement learning” in *NIPS Deep Learning Workshop 2013*, 2013.

[2] D. Silver et al., “Deterministic policy gradient algorithms” in *ICML*, 2014.

[3] V. Mnih et al., “Human-level control through deep learning” in *Nature*, 2015.

[4] M. Deisenroth, “Efficient reinforcement learning using gaussian processes”, 2009.

[5] T. P. Lillicrap, “Continuous control with deep reinforcement learning” in *ICLR*, 2016.

[6] https://spinningup.openai.com/en/latest/algorithms/ddpg.html, 09.02.2020

[7] G. E. Uhlenbeck and L. S. Ornstein, „On the theory of the Brownian motion” in *Physical review*, 1930.