Project 2: Continuous Control
Udacity Deep Reinforcement Learning

Bastian Broecker

Abstract. *This document shows the reinforcement learning approaches appied in the second project "Continuous Control" (Policy-Based Methods) as part of the udacity nanodegree: Deep Reinforcement Learning. I decided to implement two different actor-critic based approaches, the stochastic on-policy method A2C, introducted as A3C by [Mnih et al. 2016] and the deterministic off-policy method DDPG [Lillicrap et al. 2016]. I give a short overview over the basic principles of these methods and the impact of different hyper parameters settings.*

1. Project Environment and Goal Description

The training environment consist of a double-jointed arm and a moving spherical target location. The environment returns a reward of $+0.1$ as long as the tip of the arm is inside the target area (see Figure 1). Therefore, the agent has to learn a policy which is capable of maintaining a position inside the goal for as long as possible. The environment comes in two versions, a single robot arm environment and a 20 robot arm environment. The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector is bound by the values -1 and 1. In order to speed up the training, by collection experiences/roll-outs in parallel, I use exclusively the 20 robot version. The task is episodic and terminates after 1000 time steps. In order to solve the task the agent has to average 30+ points over the last 100 episodes.

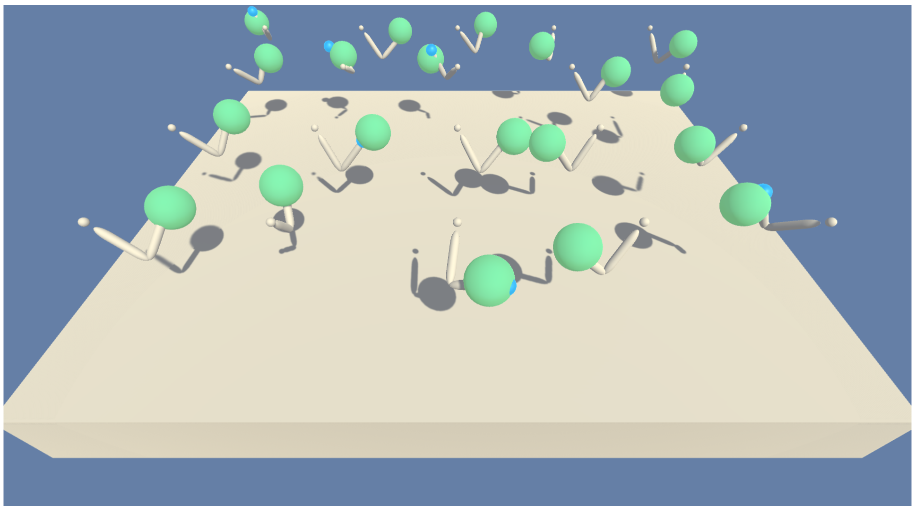

Figure 1. Environment

2. Actor-Critic

Policy based reinforcement methods are able to determine a valid action or action distribution directly from the input state. They don't not using the intermediate step of a state-action evaluation in their policy (like for example DQN), this makes these methods suitable for continues action spaces. In vanilla policy gradient based method e.g. REINFORCE [Sutton et al. 2000], we require a full episode (trajectory), to determine the policies gradient in regards to the weights θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t] \tag{1}$$

With G_t being the long-term reward over the hole trajectory τ, starting at time-step t, $\pi_\theta(s, a)$ being the probability of choosing action a in state s and θ being the weights of the policy network. These methods have a lot of noise and high variance in the gradient, this leads to a slow convergence. An other problem of these kind of methods is that they require episodic environments, in order to determine the long-term (summed up) reward G_t.

The idea of the actor-critic approach is to introduce an additional estimator/critic, reducing the variance by estimation the long term reward and therefore guiding the policy network during training. In regards to REINFORCE, this could a be the use of a Q-value estimation of the long-term reward, see the following equation:

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) Q(s, a, w)] \tag{2}$$

With w being the parameterization of the critic, provided the Q-value. During this project I chose the deterministic off-policy based method DDPG (Section 4) and the on-policy probabilistic method A2C (Section 3).

3. A2C

Asynchronous advantage actor-critic A3C was introduced by [Mnih et al. 2016], it uses N-step bootstrapping, this means it uses N-steps of a policy's trajectory to update the gradient. DQN or DDPG are considered 1-step bootstrapping. A2C uses the same techniques as A3C, but the workers, which collect the rollouts are synchronized, this makes the programming easier. A2C has two networks, an actor-network, with weights θ' providing a distribution over action values based on the current state and a critic network (with weights θ_v) providing the estimate of the value-function $V(s)$. The update is done by following gradient (see Equations 3 and 4) in respect to the networks weights:

$$\nabla'_\theta \log \pi(a_t|s_t; \theta') A(s_t, a_t; \theta_v) \tag{3}$$

$$A(s_t, a_t; \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{i+1} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{4}$$

k is upper bound by t_{max} which is the maximal number of steps considered for the n-step bootstrapping. In A2C the trained policy is also the exploration policy, which makes this approach an on-policy approach and uses therefore no replay buffer. In order to speed up the training, roll outs for the current policy can be done in parallel and collectively used to update the networks.

3.1. Generalized Advantage Estimation (GAE)

Depending on the environment, different numbers of t_{max}, max number of steps in n-step bootstrapping, are converging quicker than others. Using this principle Generalized Advantage Estimation (GAE) [Schulman et al. 2016] is a extension to all n-step bootstrapping approaches. Using a λ-return it combines all n-step return to one (see Equation 5).

$$A_t^{GAE(\gamma,\lambda)} = (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + ...) \tag{5}$$

Setting λ to 0 results in a 1-step bootstrapping and setting λ to 1, only considers the "infinite"-bootstrapping, a λ between 0 and 1 combines all sequences, with a different weighting.

3.2. Model Architecture

Both actor and critic-model have two hidden-layers with 128 nodes and ReLu activation and linear-activation on the output.

3.3. Hyperparameters and Convergence

I used a simple grid-search to identify the most significant hyper-parameters. The test showed that the number of steps (n-step bootstrapping) influence the training. Where a number of 3 results in a fast max score, the training is not stable and drops many times. The step count of 4 and 5 are more stable, but differ in convergence rate (see Figure 2). I achieved the best result with the use of GAE with a $\lambda = 0.95$ (see Section 3.1) and 5-step bootstrapping, the agent solves the environment after 60 episodes, see Figure 3. The best performing parameters are as follows:

- Learning Rate: $1e^{-4}$
- n-steps: 5
- GAE: active and $\lambda = 0.95$
- γ: 0.95 (reduction of future rewards)

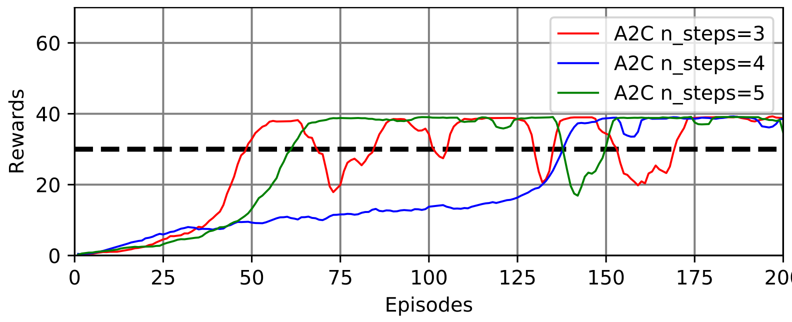

Figure 2. A2C Training

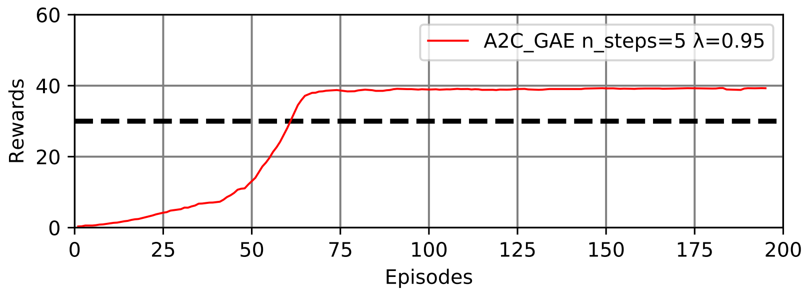

Figure 3. A2C Training with GAE

4. DDPG

DDPG was introduced by [Lillicrap et al. 2016] as an actor-critic method, but some researchers think it's best classified as DQN ([Mnih et al. 2013]) for continuous actions spaces. DDPG is a deterministic a approach where the actor/policy network returns the best action $\mu(s;\theta_\mu)$ for state s. The actor is basically learning the $\arg\max_a Q(s,a)$ of the equivalent DQN. The second part of the DDPG is the critic, which learns to estimate the optimal action-value function by using the actors best believed action $Q(s,\mu(s,\theta_\mu);\theta_Q)$. The actor is updated by applying the chain rule to the expected return with respect to the actors parameters [Lillicrap et al. 2016]:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}] \tag{6}$$

The critic $Q(s,a)$ is learned using the Bellman-equation, therefore the lost is defined as follows:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}) \tag{7}$$

$$L = \frac{1}{N}\sum_i^N (y_i - Q(s_i,a_i|\theta^Q))^2 \tag{8}$$

To avoid the moving target problem, I use a second target network denoted by $'$. In order to keep the target stable I use a soft-update with the parameter τ. This updates the target-network slowly towards the current local network weights. Since DDPG is a off-policy approach I used a replay buffer, to experience samples multiple times and use a noise process (Uhlenbeck & Ornstein) to sample noise ($\mathcal{N}$) to the actors polity to allow exploration during training:

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N} \tag{9}$$

4.1. Model Architecture

The used critic-model has two hidden-layers with 128 nodes and ReLu activation and linear-activation on the output. The actor-model has also two hidden-layers with 128 nodes and ReLu activation and a tanh-activation on the output, since the action-values of the environment are between 0 and 1.

4.2. Hyperparameters and Convergence

In order to test the impact of different hyper parameters on the training, I did a grid-search. This sections won't show all results, but give a short overview over the most significant findings. One of the most important property seemed to be the learning-rate of the actor, while a learning rate of $1e^{-4}$ requires more than 120 episodes to converge, is a setup with a learning rate of $1e^{-3}$ able to solve the environment after 38 episodes. An other important parameter is the batch size, where a batch size of 32 results in a slow convergence, a batch size of 64 or higher increases the rate of convergence (see Figure 4). The environment is considered solved if the agent's average score is $+30$ for the last 100 episodes, the DDPG agent solves the environment after 38 episodes, see Figure 5.

- Actor Learning Rate: $1e^{-3}$
- Critic Learning Rate: $1e^{-4}$
- Batch Size: 128
- Buffer Size: 10000
- γ: 0.95 (reduction of future rewards)

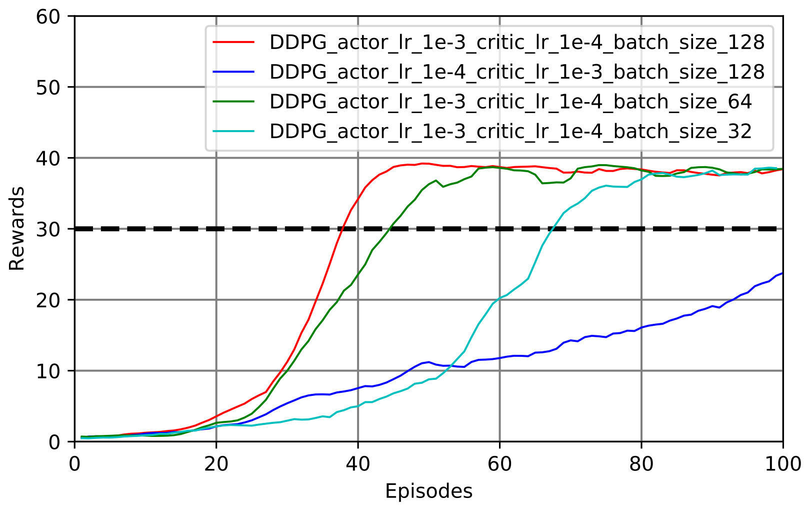

Figure 4. DDPG Hyperparameters

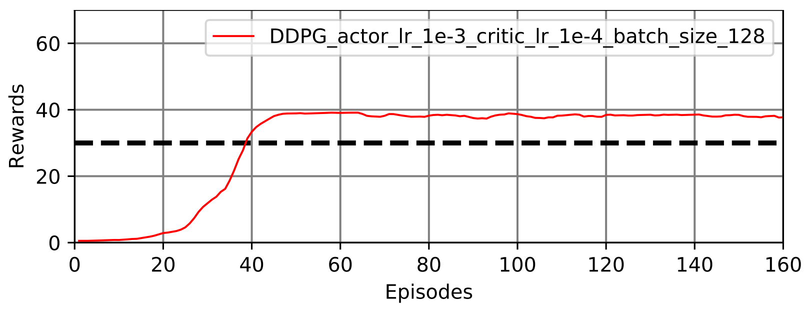

Figure 5. DDPG Solves Environment after 38 Episodes

5. Ideas for Future Work

In order to improve the performance of the two tested agent types (A2C and DDPG), I would like to try automatized hyper-parameter tuning, for example Bayesian parameter optimization and see how much this can help finding the best settings, network sizes and network activation-functions. Further more, I would like to see if extensions like, prioritized experience buffer, batch normalization and frame skipping could improve the training performance and generalize the networks for other environments.

References

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *ICLR*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. cite arxiv:1511.05952Comment: Published at ICLR 2016.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Sutton, R. S., Mcallester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, volume 12, pages 1057–1063. MIT Press.

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461.

Wang, Z., de Freitas, N., and Lanctot, M. (2015). Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581.