Project report

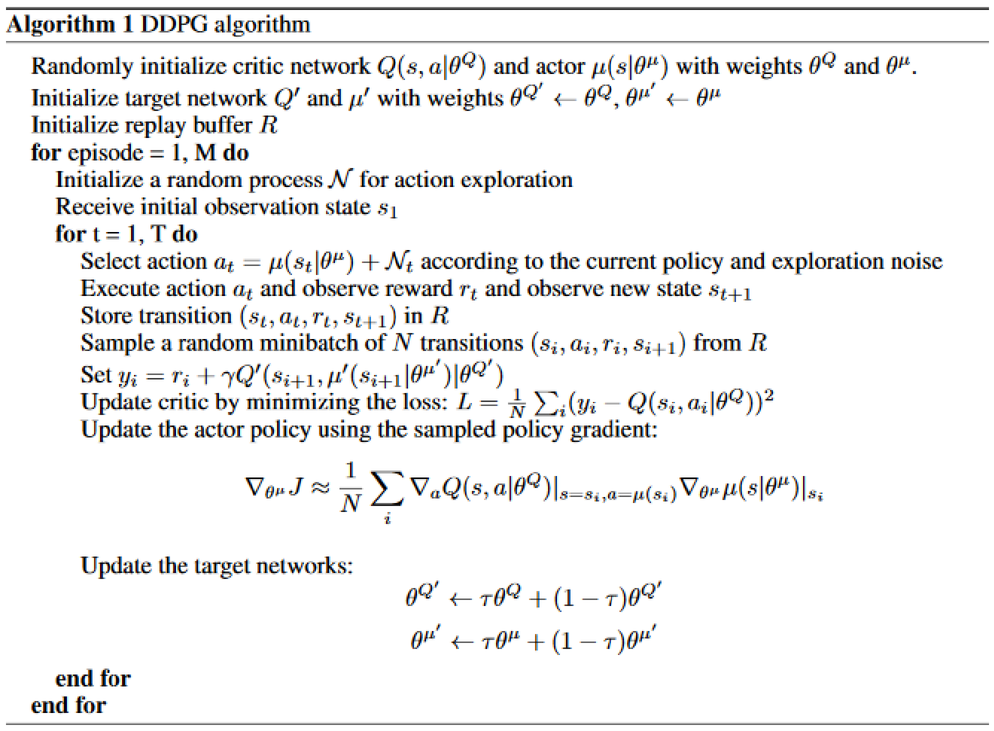
Learning algorithm

The learning algorithm used is Distributed Distributional Deterministic Policy Gradient, D4PG, as described in original paper https://arxiv.org/pdf/1804.08617.pdf

Base of this alghoritm is DDPG alghoritm described in https://arxiv.org/abs/1509.02971 , that can be seen as Deep Q learning for continuous action spaces. Main part of this algorithm are four neural networks:

- local and target network for actor **μ** and **μ′**
- local and target network for critic **Q** and **Q′**

This approach is just like one in Deep Q learning, where to stabilize whole alghoritm we are using local network that was updated each step, and target network that was updated each , fixed amount of steps. Detailed implementation details can be seend in mentioned paperwork, so I`ll just put alghoritm here:



```
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
```

As we can see we have replay buffer, where we store transitions, we're calculating same TD and critic loss as in DQL. Main difference is that for action prediction we are using critic network instead of actor tarrget network jus like we did in DQN.

In DQN we had following y update:

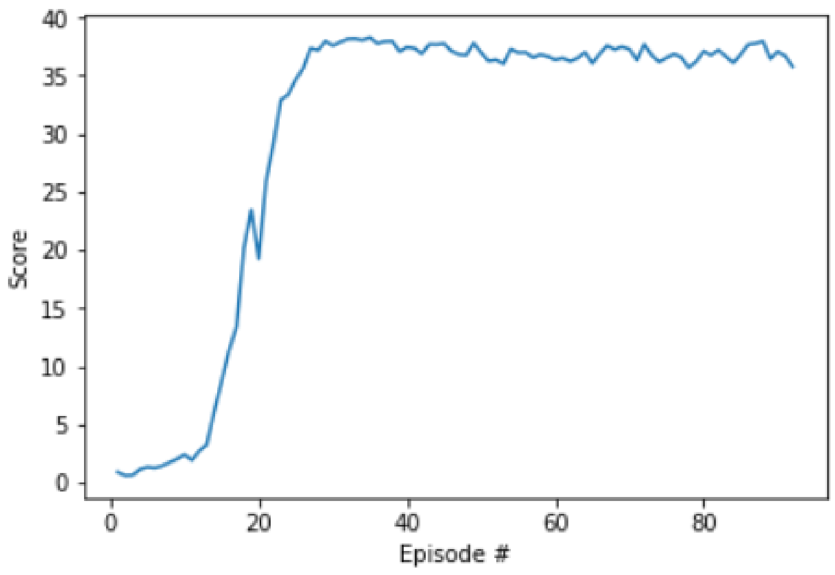
$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax} Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

In DDPG we have similar one but instead of using argmaxQ(…) we`re using critic target network to calculate action to be taken in state t_1:

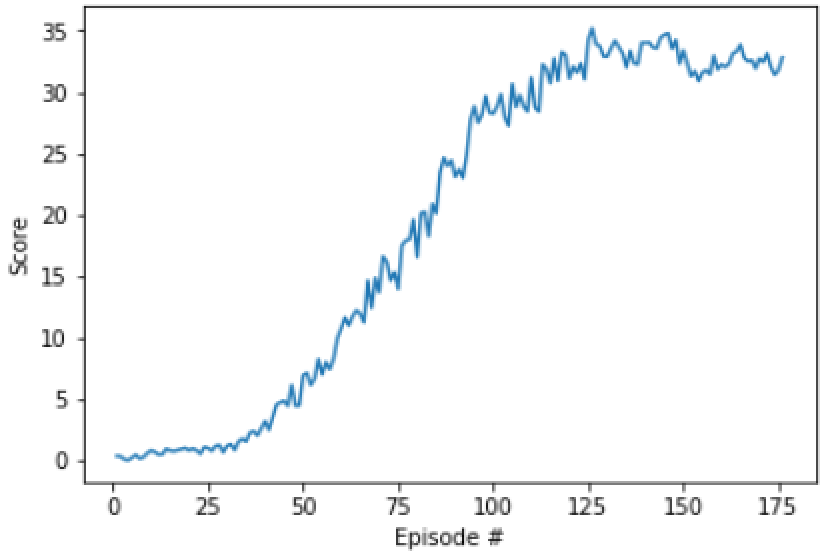
$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

During implementation I`ve made few steps:

1. I've implemented plain DDPG algorithm to have some baseline to compare other modifications with it. For exploration I`ve used Ornstein-Uhlenbeck process , with default values from paper. Parameters that were the best I`ve met during trials and errors plus each episde normal and mean score are in DDPG.pdf file. In this case alghoritm reach 30 pts average in episode 92: **Reached 30.01249932916835 in episode 92**. Below is graph from given best trial.



2. Second idea was to use weight decay on critic/actor with different mixes, but this was not so good as plain DDPG , because best atempt I get was **Reached 30.054664328225886 in episode 176**. So it took so way longer to get into 30 pts checkpoint. Aditionally, what can be seen in below chart, updates were lot smaller than in plain DDPG



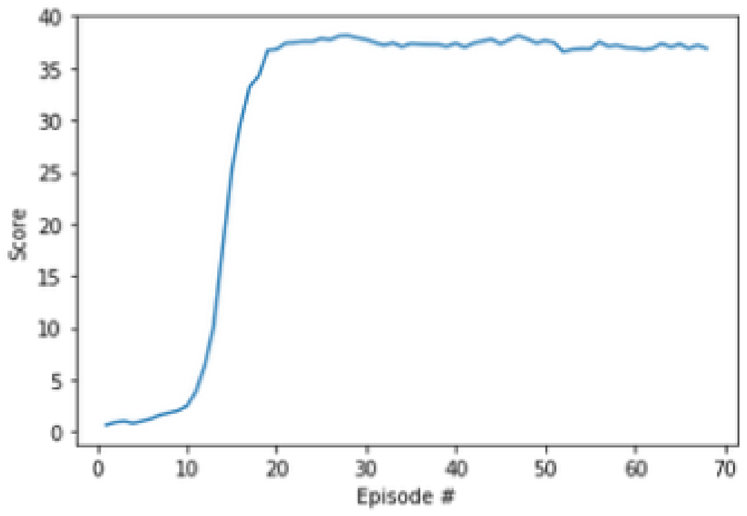
3. Now the fun part starts. I`ve tried to implement D4PG alghoritm ideas into plain DDPG. First , and easiest one is n step return. Base idea of this is that  before we put our s,a,r,s,d samples into replay buffer, we`re making n steps of real stepping, with each step multiplied by gamma to adequate power, like below:

$$\left(\sum_{n=0}^{N-1} \gamma^n r_{i+n}\right)$$

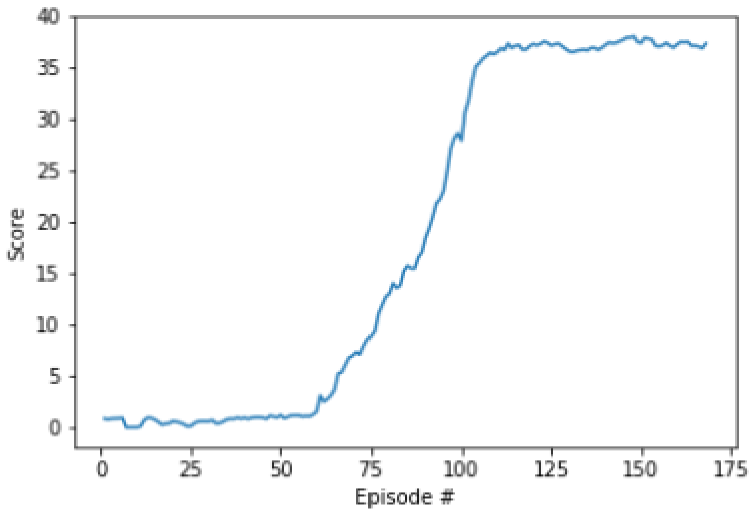
 because of that we get response of real system, not some prediction for n steps forward. This helps us to better estimate next return and helps lower variance of system.

   With this approach and plain DDPG I get really good scores , that were hard to beat :

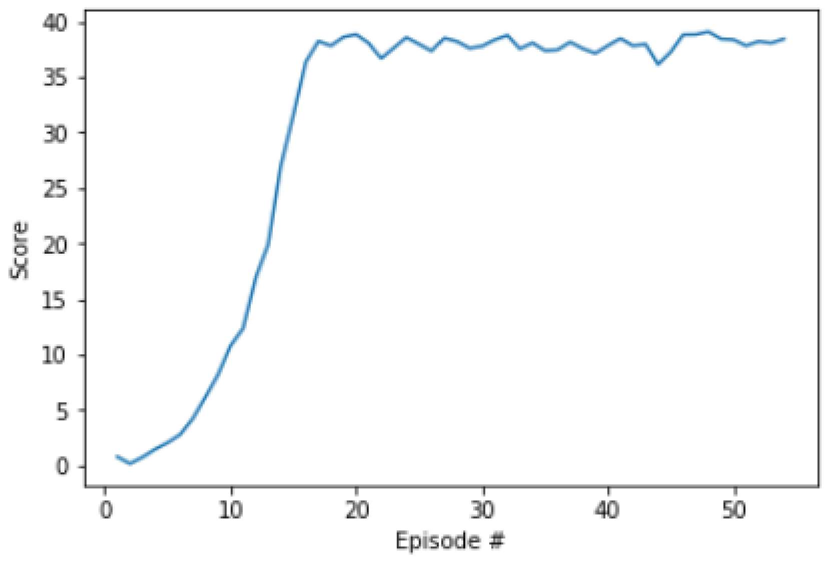
   Reached   30.011499329190702 in episode 68. So with only n step return (n = 5, but I`ve checked n from 1 to 7, and 5 was the best one) I`ve get from 92 to 68 episodes to get baseline of 30 pts. It can be seen in DDPG n_step.pdf file. Below we can see chart from this trial. As we can see there is less jumping about 35 points, and line is more smooth.



4. Another and hardest one to implement, idea was using of distributional critic, that is main part of this algorithm. For this , in mentioned paper authors used Categorical distribution. It was very hard to implement it, but I found paper ***A Distributional Perspective on Reinforcement Learning*** (https://arxiv.org/pdf/1707.06887.pdf), where categorical distribution algorithm is given. Additionally, book ***Deep Reinforcement Learning Hands-On*** there is detailed, and explained implementation of this algorithm. I've tried to implement is by myself for like 4 days, but I didn't worked like it should. Good think about those trials are, that I`ve fully understood this algorithm working mode, principles, math etc, so I took distribution creation from this book and modified it, so it could fix my code. With n step and distribution only , I`ve get not so good results : **Reached   30.15615432595741 in episode 168**. So this alghoritm reached given base point line much slower than DDPG with n step.



5. Last thing that wa missing , to get D4PG algorithm is using of Prioritized Replay memory. Becuase I had one implemented by myself for previous assignment, I've just used it with little modification. Additional thing is connected with local to target networks parameters copying. In this run I`ve tried multiple approaches, but the one that really worked is like that: Each step do soft update with tau = 0.001, but every 100 steps do hard update. And this was Jack pot. With n step/ distributional critic/ prioritized replay and noise dampening basing on current score, I've managed to get best results, that can be seen in D4PG.pdf file: **Reached   30.09634191988691 in episode 54**.



Hardest part of the assigment was understanding what Categorical distribution parameters does, how to set V_min, V_max, number of atoms and what are they for. But because many trials and errors it looks so easy now. It it far more easier to implement A3C t/A2C to continuous spaces now.

Parameters for each trial are given in pdf files mentioned in descriptions.

Saved models are in checkpoint folder. Models are saved each 20 steps.

Final hyper parameters used are :
batch_size = 256
buffer_size = 100000
tau = 0.001
actor_lr = 0.0001
critic_lr = 0.001
actor_weight_decay = 0
critic_weight_decay = 0
gamma = 0.99
max_episode_len = 1000
atoms_number = 51
V_min = -5
V_max = 5
delta = (V_max - V_min)/(atoms_number -1)
n_steps = 5

Neural network is easy one :

Actor network:

- layer 1 [33, 512]
- relu()
- layer 2 [512, 256]
- relu()
- layer 3 [256, 4]
- tanh()

Critic network:

- layer 1 [33, 512]
- relu()
- layer 2 [512 + 4, 256]
- relu()
- layer 3 [256, 51]

Improvements

- Exchange Categorical distribution to gausian distributional
- Try to implement A3C/A2C algorithms (but I think D4PG will work better, but still it is good exercise)
- Try implemented algorithm on others multi agent environments