For this project, you will work with the Reacher environment.

In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.

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 should be a number between -1 and 1.

The task is episodic, and in order to solve the environment, your agent must get an average score of +30 over 100 consecutive episodes.

## Background

To solve this problem I used the Deep Deterministic Policy Gradient (DDPG) algorithm.

Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

This approach is closely connected to Q-learning, and is motivated the same way: if you know the optimal action-value function $Q^*(s,a)$, then in any given state, the optimal action $a^*(s)$ can be found by solving:

$$a^*(s) = \ arg\ max_a Q^*(s,a)$$

When there are a finite number of discrete actions, the max poses no problem, because we can just compute the Q-values for each action separately and directly compare them. But when the action space is continuous, we can't exhaustively evaluate the space, and solving the optimization problem is highly non-trivial.

The way DDPG handles this, is by having two networks. One, called the actor, that approximates the best deterministic action, $a^*(s)$, for any given state. The second, called the critic, approximates the action-value function $Q^*(s,a)$.

Like DQN these networks change a lot and make learning instable, therefore local and target network with soft updates are also used.

So DQN network minimizes the following loss function:

$$L(\phi) = E[Q_\phi(s,a) - (r + \gamma(1-d)\max_a Q_\phi(s',a'))]$$

Where d indicates whether the state is terminal.

While DDQN critic network minimizes the following loss function:

$$L(\phi) = E[Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi target}\left(s', \mu_{\theta target}(s')\right)\right)]$$

Where $\mu_{\theta target}$ is the target actor network. The DDQN actor network maximizes the following function:

$$J = \frac{1}{n}\sum_{n} Q_{\phi}(s,a)$$

Which is the same as minimizing:

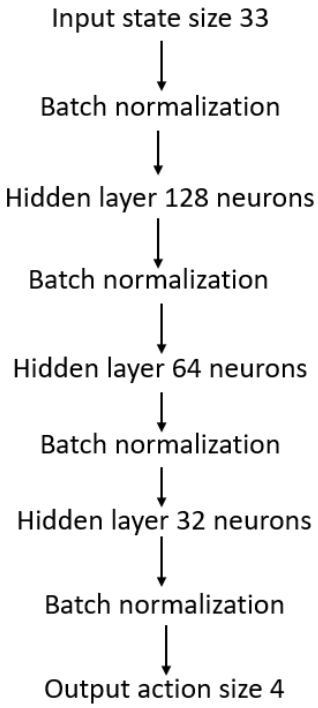
$$J = -\frac{1}{n}\sum_{n} Q_{\phi}(s,a)$$

# Experiment

## Neural Network structure

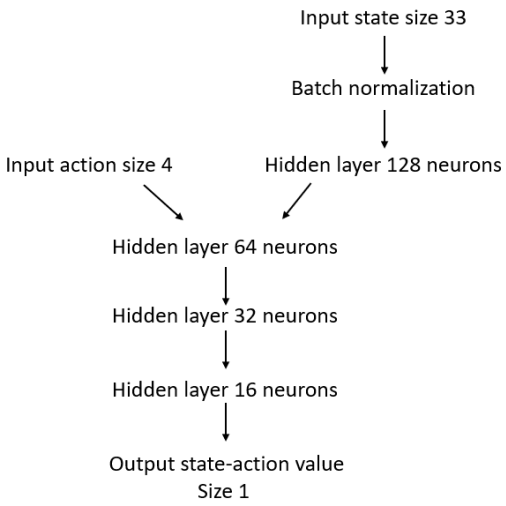
As described in the background section, DDPG has an actor network, which given an states outputs the best action and a critic network which given a state and an action evaluates how good the action was. To approximate these networks I used a Neural Network.

The actor network has the following structure:




The activation functions used were selu for all layers except the last which used tanh function since the action space was between -1 and 1.

The critic network has the following structure:




The activation function use was selu for all layers, since the output cannot be negative since all rewards are either 0 or positive.

The number of neurons in the input layer represent the state space and the number of neurons for the output layer represent the action space. For this specific problems these number cannot be changed.

### Hyperparameters

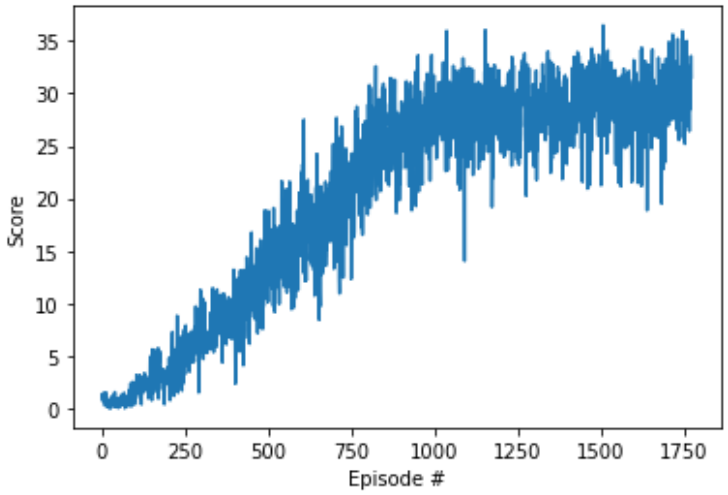
The following hyperparameters were used:

- Tau: 1e-3
- Gamma: 0.99
- Learning rate actor: 1e-4
- Learning rate critic: 1e-3
- Batch size: 128
- Replay buffer size: 1e6
- Learn every: 8, meaning that for 8 consecutive frames we collect the tuple (state, action, reward, next state, done) and then we perform one step of learning, were we calculate the gradients and update the networks.

## Results

On the following Figure we can see the obtain results. As we can see, in less than 2000 episodes we achieved the goal of an average of 30 over 100 consecutive episodes.

The trained actor and critic network can be found on the files ‘actor_local.pt’ and ‘critic_local.pt’, respectively.




## Conclusion

As I showed, by fine tuning the networks and hyperparameters convergence can be achieved for this problem.

As a way to improve the presented model, I would like to experiment which different hyperparameters and analyse how these impact the results. My guess is that increasing the learning rate would make converge slower, while decreasing it would like the converge instable and therefore slower. A better chance at improving performance would be vary the tau, batch size, and ,learning every. Bearing in mind that, decreasing the batch size and the learning every might lead to overfit and, therefore, instable and slower.

As future work I would like to implement the Proximal Policy Optimization (PPO) algorithm to solve this environment.