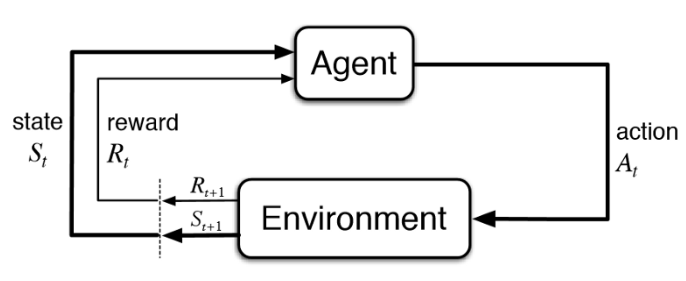
Reinforcement Learning:

Below is the flow diagram which reinforcement learning.



1. Action (A): All the possible moves that the agent can take
2. State (S): Current situation returned by the environment.
3. Reward (R): An immediate return send back from the environment to evaluate the last action.
4. Policy (π): The strategy that the agent employs to determine next action based on the current state.
5. Value (V): The expected long-term return with discount, as opposed to the short-term reward R.*Vπ(s)* is defined as the expected long-term return of the current state sunder policy π.
6. Q-value or action-value (Q): Q-value is similar to Value, except that it takes an extra parameter, the current action *a*. *Qπ(s, a)* refers to the long-term return of the current state’*s*, taking action *a* under policy π.

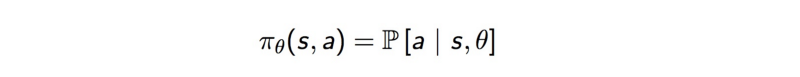
Using DDPG to write code to solve pendulum environment

**Abstract:**

Deterministic Policy Gradients, they have produced a policy-gradient actor-critic algorithm called Deep Deterministic Policy Gradients (DDPG) that is off-policy and model-free, and that uses some of the deep learning tricks that were introduced along with Deep Q-Networks (hence the “deep”-ness of DDPG). In this blog post, we’re going to discuss how to implement this algorithm using Tensorflow and tflearn, and then evaluate it with OpenAIGym on the pendulum environment. In this project, people realized that deep learning methods could be used to solve high-dimensional problems. One of the subsequent challenges that the reinforcement learning community faced was figuring out how to deal with continuous action spaces.

**Background:**

DDPG depends on the actor-critic architecture with two eponymous elements, actor and critic. For tuning the parameter 𝜽 an actor is used for the policy function, i.e. to gain the best action for a specific state.



For evaluating the policy function a critic is used to estimated by the actor according to the temporal difference (TD) error.

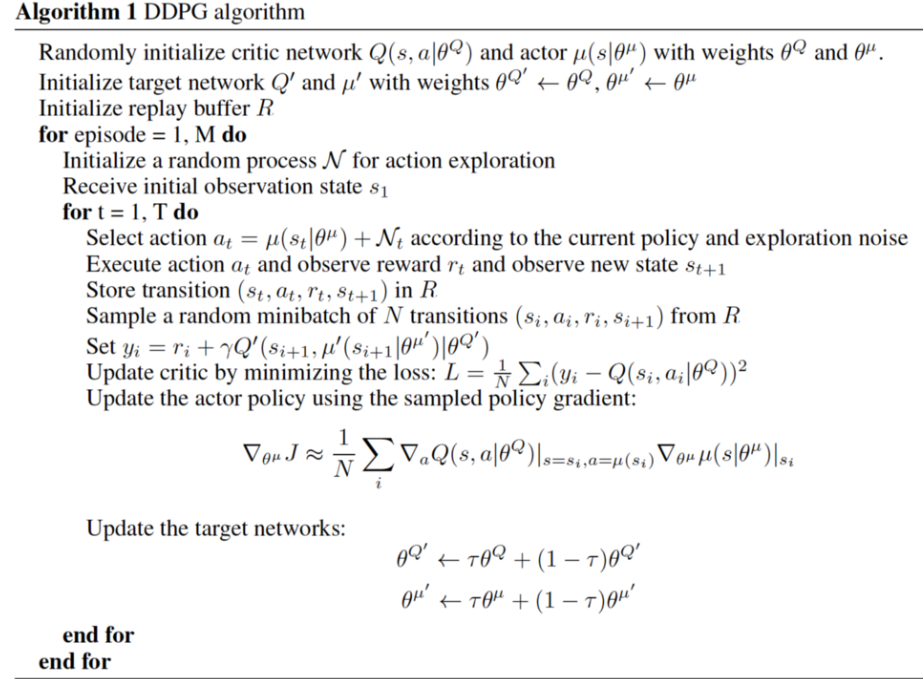
https://cdn-images-1.medium.com/max/800/1*-LcAiv5h_LEVdqIwkPNaUA.png

In the above formula *v* denotes the policy that the actor has confirmed. This is little similar to the Q-learning update equation. TD learning is a one of the learn way to predict a value based on future values of a given state. Q-learning is a specific type of TD learning for learning Q-value.

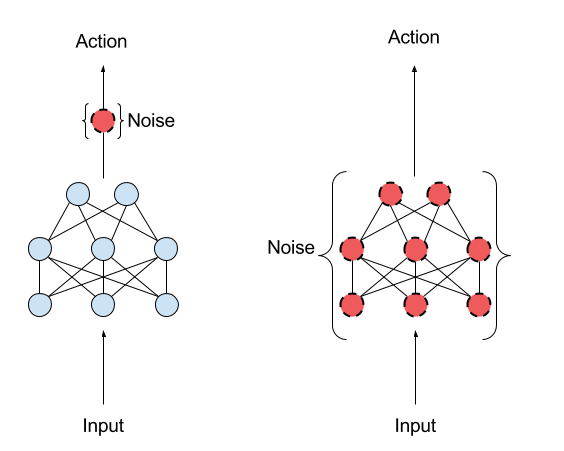
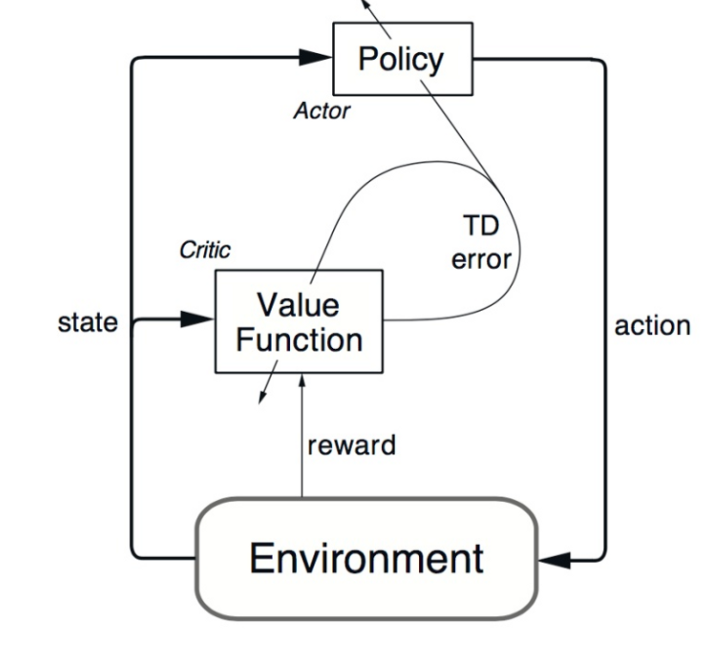
**Related Work:**

Policy-Gradient (PG) algorithms optimize a policy end-to-end by processing noisy estimates of the gradient of the expected reward of the policy and then modifying the policy in the gradient direction. Traditionally, PG methods have assumed a stochastic policy μ(a|s)μ(a|s), which gives a probability distribution over actions. This algorithm understands lots of training examples of high rewards value from good actions and negative rewards value from bad actions. Then, it can improve the chances of the good actions. In practice, you move to run into lot of problems with vanilla-PG; example, getting one reward signal at the last of a long episode of interaction with the environment creating more difficult to ascertain exactly which action was the good. This is known as the *credit assignment problem*.

**DDPG Algorithm:**



**Architecture for Actor critic and Action Noise :**



DDPG also has the ideas of experience replay and separate target networkdetails from Deep Q Network**.** Issue for DDPG is that it seldom does exploration for its actions. Solution for this issue is adding noise on the parameter space or on the action space.

Reinforcement Learning algorithms which are characterized as off-policy usually creates a separate behavior policy which is independent of the policy being improved on usage. This behavior policy is used to re-create trajectories. One of the important benefit of this separation is that the behavior policy can operate by sampling all the actions, on the other side the estimation policy can be deterministic. Q-learning is an off-policy algorithm, it modifies the Q values without any assumptions about the actual policy. The Q-learning algorithm simply states that the Q-value to state’a(t)a(t) and action a(t)a(t) is updated using the Q-value of the next state’s(t+1)s(t+1) and the action a(t+1)a(t+1) that maximizes the Q-value at state’s(t+1)s(t+1)

During training, On-policy algorithms use the policy that is estimated to sample trajectories.

Model-free Reinforcement Learning algorithms are that make less effort to learn the important dynamics that govern how an agent works with the environment. In some cases where the environment has a discrete state space and the agent has a discrete number of actions to choose from, in that environment a model of the dynamics of the environment is the 1-step transition matrix: T(s(t+1)|s(t),a(t))T(s(t+1)|s(t),a(t)). This stochastic matrix provides all of the probabilities for getting at a expected state prvided the current state and action. For this problems with high-dimensional state and action spaces, this matrix is highly expensive in space and time to compute. If your state space is the set of all possible 64 x 64 RGB images and your agent has 18 actions available to it, the transition matrix’s size is|S×S×A|≈|(68.7×109)×(68.7×109)×18||S×S×A|≈|(68.7×109)×(68.7×109)×18|, and at 32 bits per matrix element, thats around 3.4×10143.4×1014 GB to store it in RAM!

Model-free algorithms directly estimate the optimal policy or value function through algorithms such as policy iteration or value iteration. This algorithm is computationally more efficient. Obtaining and using a good approximation of the underlying model of the environment can be beneficial. Be wary- using a bad approximation of a model of the environment will only bring you misery. Hence, model-free methods require a huge number of training examples to get good benefits.

DDPG is a policy gradient algorithm that uses a stochastic behavior policy for good exploration but estimates a deterministic target policy, which is much simple to learn. Policy gradient algorithms uses a form of policy iteration and evaluate the policy, and then follow the policy gradient to increase performance. Since DDPG is off-policy and uses a deterministic target policy, this thing allows us for the use of the Deterministic Policy Gradient theorem. This algorithm is an actor-critic algorithm as well and it basically uses two neural networks, one for the actor and another for the critic. These networks compute action uses for the current state and generate a temporal-difference error signal each step. The input of the actor network is the called current state, and output is calculated single real value representing an action taken from a continuous action space. The critic’s output is the estimated Q-value of the current state for of the action given by the actor. The deterministic policy gradient theorem gives the new rule for the weights for actor network. The critic network is modified from the gradients gathered from the temporal-difference error signal.

It turns out that tossing neural networks at DPG results in an algorithm that results very less performance, resisting all of your most efforts to get it to not good output. The following aresome of the key conspirators:

1. In general, training and evaluating your policy and/or value function with thousands of temporally-correlated simulated trajectories leads to the introduction of enormous amounts of variance in your approximation of the true Q-function (the critic). The TD error signal is excellent at compounding the variance introduced by your bad predictions over time. It is highly suggested to use a replay buffer to store the experiences of the agent during training, and then randomly sample experiences to use for learning in order to break up the temporal correlations within different training episodes. This technique is known as experience replay. DDPG uses this.
2. Directly updating your actor and critic neural network weights with the gradients obtained from the TD error signal that was computed from both your replay buffer and the output of the actor and critic networks causes your learning algorithm to diverge (or to not learn at all). It was recently discovered that using a set of target networks to generate the targets for your TD error computation regularizes your learning algorithm and increases stability. Accordingly, here are the equations for the TD target yi and the loss function for the critic network:

yi=ri+γQ′(si+1,μ′(si+1|θμ′)|θQ′)

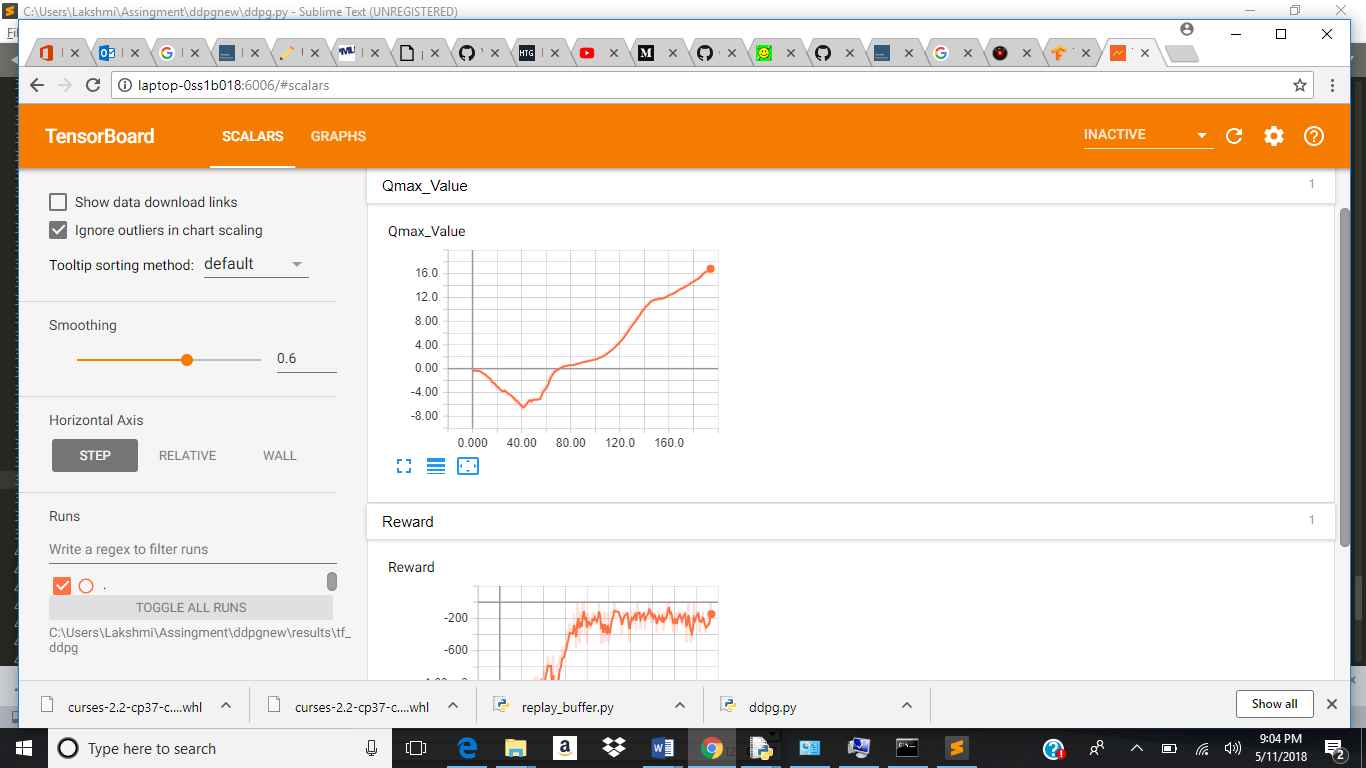
L=1N∑i(yi−Q(si,ai|θQ)

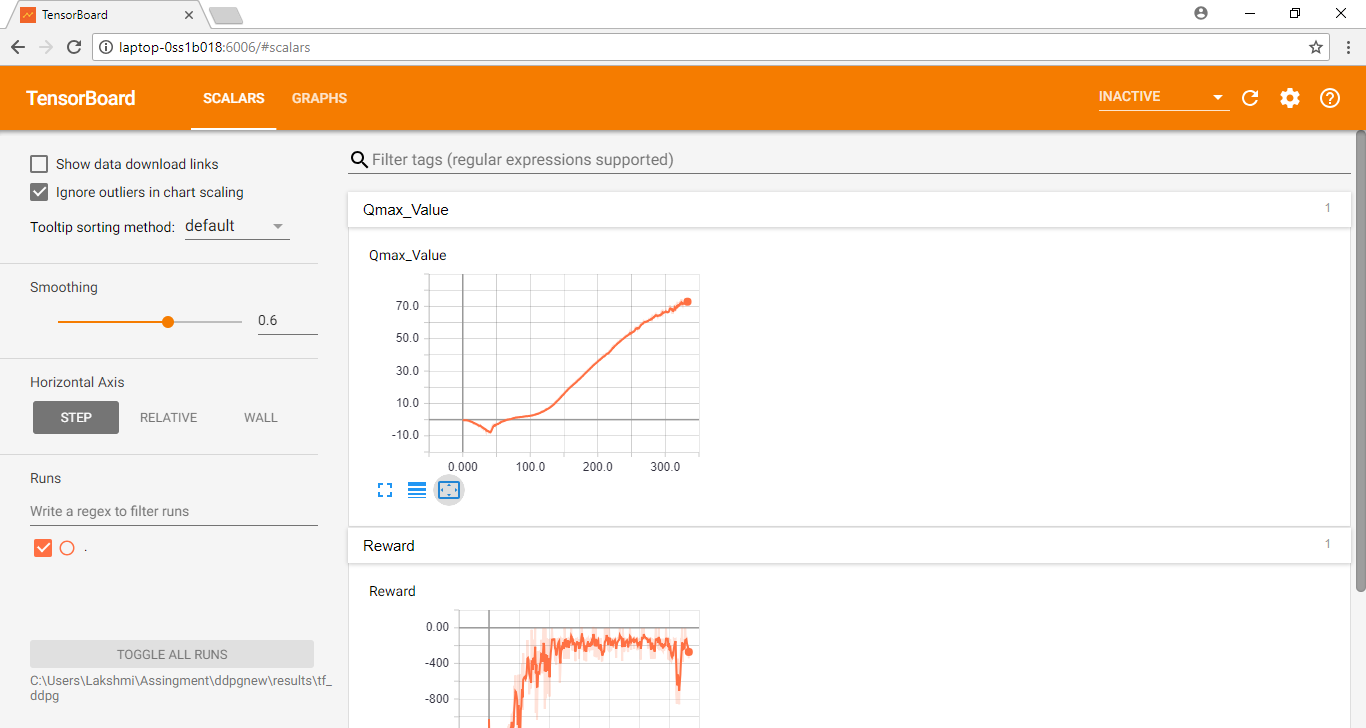
Here, a minibatch of size NN has been sampled from the replay buffer, with the ii index referring to the i’th sample. The target for the TD error computation, yiyi, is computed from the sum of the immediate reward and the outputs of the target actor and critic networks, having weights θμ′θμ′ and θQ′θQ′ respectively. Then, the critic loss can be computed w.r.t. the output Q(si,ai|θQ)Q(si,ai|θQ) of the critic network for the i’th sample.

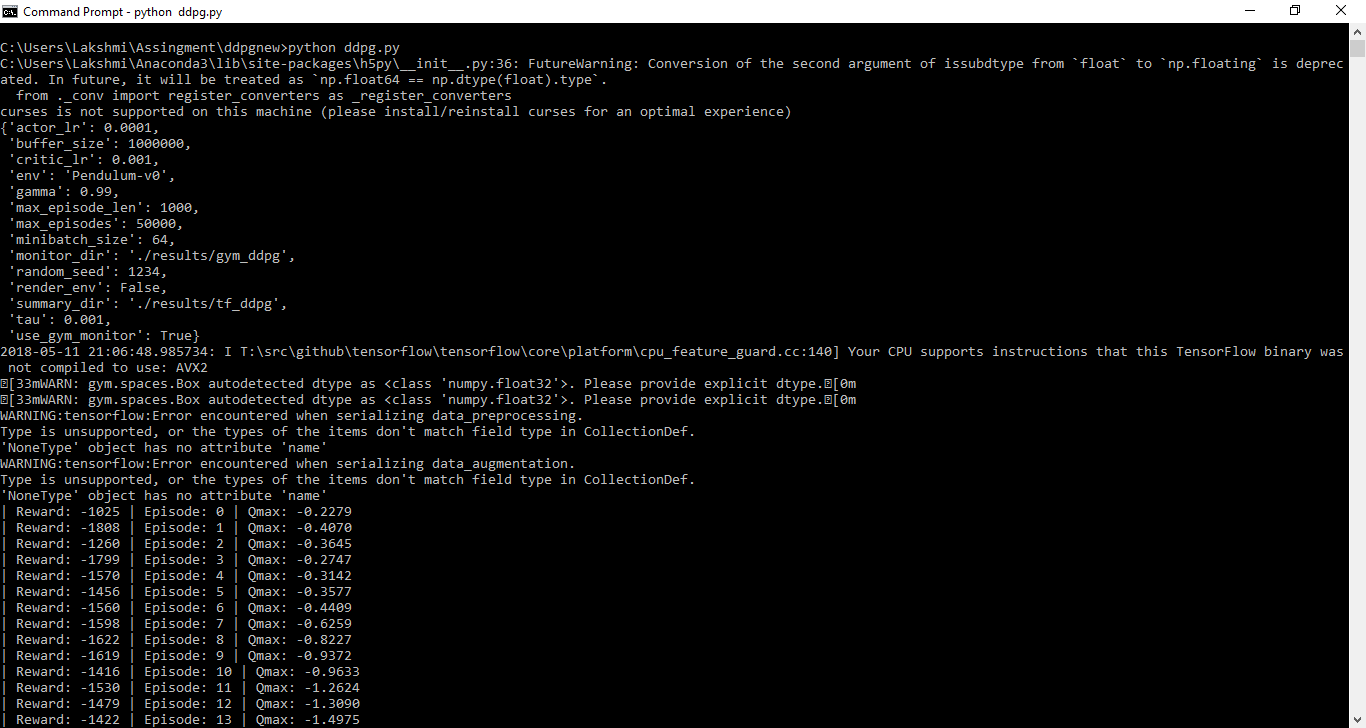
Now, as mentioned above, the weights of the critic network can be updated with the gradients obtained from the loss function in Eq. 2. Also, remember that the actor network is updated with the Deterministic Policy Gradient. Here lies the crux of DDPG! Silver, et al., [[2]](http://pemami4911.github.io/blog/2016/08/21/ddpg-rl.html#References) proved that the stochastic *policy gradient* ∇θμ(a|s,θ)∇θμ(a|s,θ), which is the gradient of the policy’s performance, is equivalent to the deterministic policy gradient, which is given by:

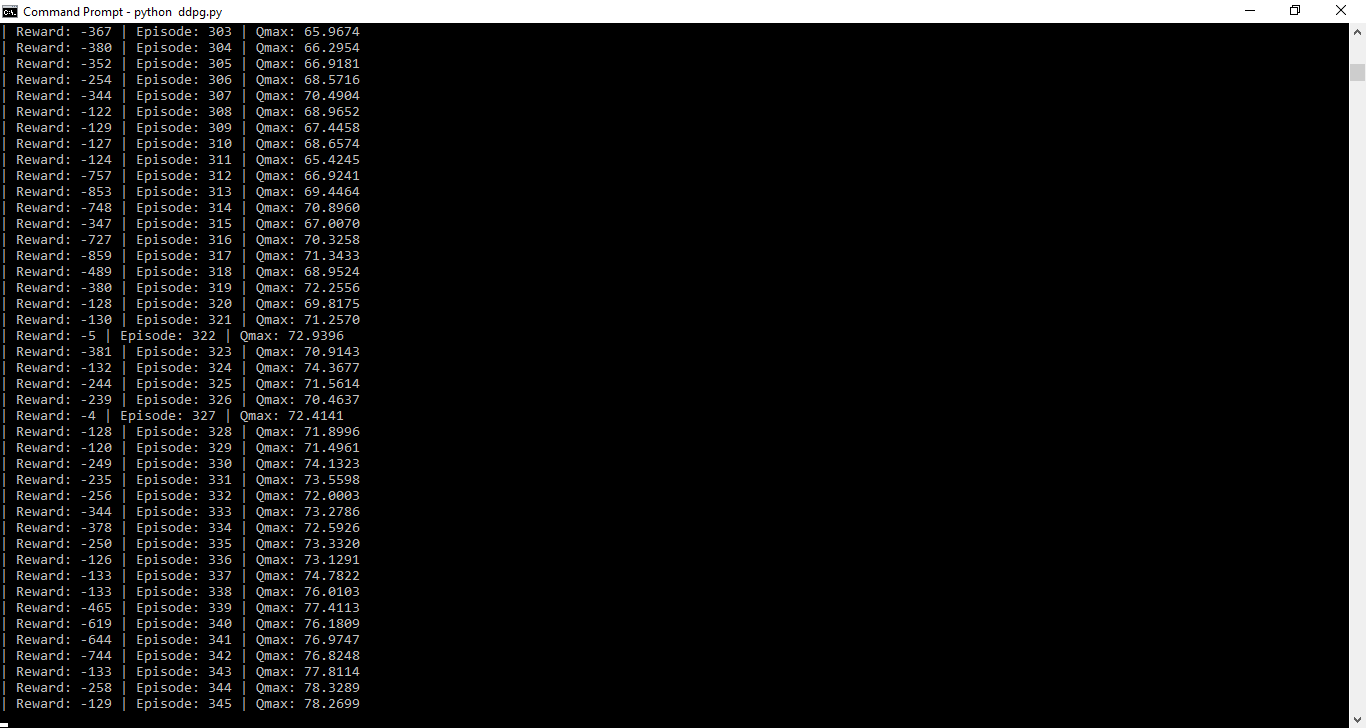
∇θμμ≈Eμ′[∇aQ(s,a|θQ)|s=st,a=μ(st)∇θμμ(s|θμ)|s=st]

**Results:**









**References:**

<https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287>

<http://pemami4911.github.io/blog/2016/08/21/ddpg-rl.html>