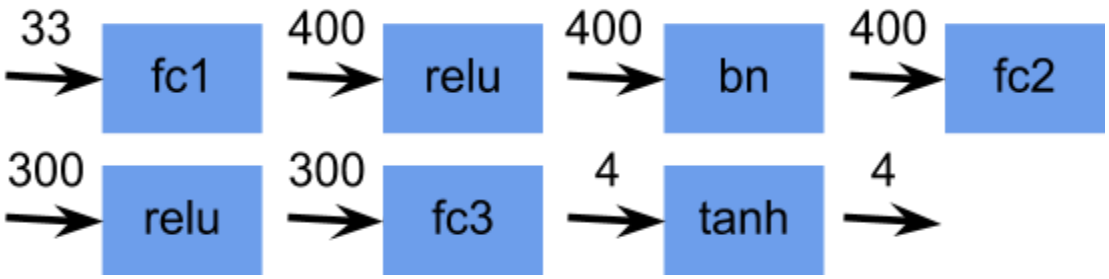
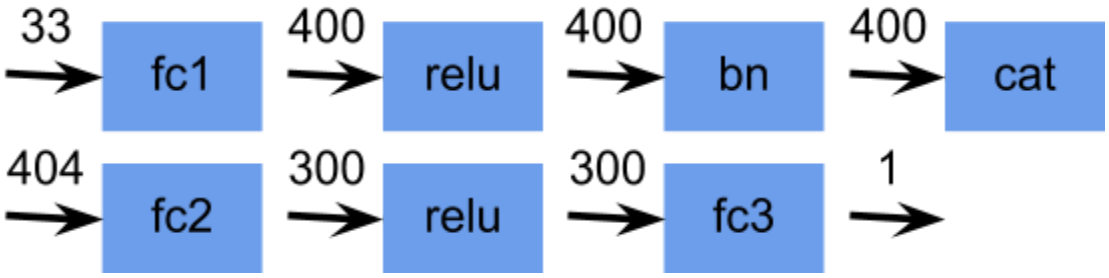
I. Models

Actor



Critic



II. Agent Structure (DDPG - Deep Deterministic Policy Gradient)

actor_local - predict the current state using the Actor model
actor_target - predict the target state using the Actor model
actor_optimizer - Adam optimizer
critic_local - predict the current state using the Critic model
critic_target - predict the target state using the Critic model
critic_optimizer - Adam optimizer
noise - OUNoise
memory - array that holds past state, action, reward, next state, and done values

Parameters

n_episodes	400	Number of times the environment is run
BUFFER_SIZE	1e6	Replay (memory) buffer size
BATCH_SIZE	128	Minibatch size
GAMMA	0.99	Discount factor
TAU	1e-3	Soft update of target parameters
LR_ACTOR	2e-4	Learning rate of the actor
LR_CRITIC	2e-4	Learning rate of the critic

WEIGHT_DECAY	0	L2 weight decay
mu	0	Noise parameter
theta	0.15	Noise parameter
sigma	0.1	Noise parameter

Overview

For each episode:

1) Reset the environment
2) Get the state of the environment

While the environment is running:

1) Use the agent to **predict action** based on the environment's state
2) Pass the predict action into the environment
3) Retrieve the next state, reward, and done variable
4) **Agent step**. Record next state, reward, and done variable to **memory** in agent
5) Set the current state variable to next state
6) Add reward to score
7) Break out of while loop if done is true

Predict action

1) Use **actor_local** to predict action with state as input
2) Re-train **actor_local**
3) Add noise to **action** and clip values between -1 and 1

Agent step

1) Add experience to memory

   If number of past values is greater than BATCH_SIZE

   a) Randomly sample experiences from **memory**
   b) Use **actor_target** to get the next action (**actions_next**) with next state as input
   c) Use **critic_target** to get the next Q target (**Q_targets_next**) with next state (**next_states**) and next action (**actions_next**) as input
   d) Computes the Q target with the following formula

   $$Q\_targets = reward + (gamma * Q\_targets\_next * (1 - dones))$$

   e) Get the expected Q values (**Q_expected**) from local model (**critic_local**) with state and action as input
   f) Use **critic_loss** for **Q_expected** and **Q_targets**
   g) Predict actions (**actions_pred**) using **actor_local** with state as input
   h) **actor_loss** is the negated mean of output of **critic_local**
   i) Update the weights of the target network (**critic_target**) from the local network (**critic_local**) using the following formula

   $$critic\_target_{weights} = tau * critic\_local_{weights} + (1 - tau) * critic\_target_{weights}$$

j) Update the weights of the target network (**actor_target**) from the local network (**actor_local**) using the following formula

$$actor\_target_{weights} = tau * actor\_local_{weights} + (1 - tau) * actor\_target_{weights}$$

OUNoise

1) The noise is generated by

$$dx = theta * (mu - x) + sigma * rand(size)$$

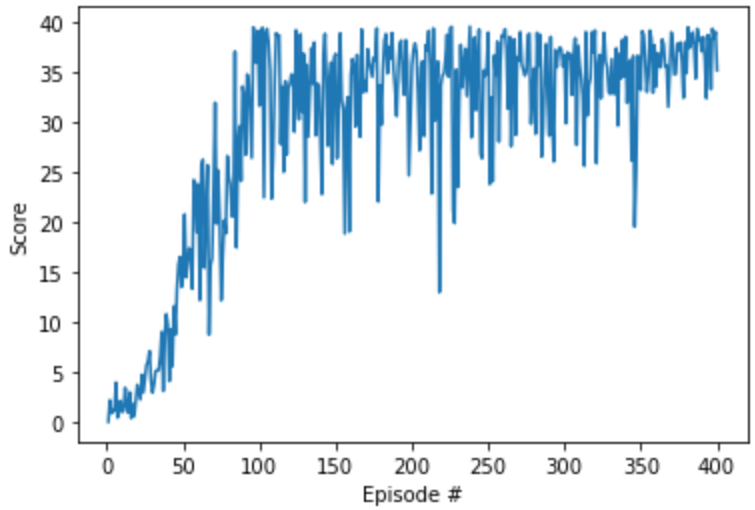
$$x = x + dx$$

Where

x	State of the noise
rand(size)	*Size(4)* number of random values

III. Results

Here's a graph of the episodes vs scores. As the episodes get closer to 400, we see the scores concentrate between 31 and 38.



IV. Ideas for Future work

Other improvements to the algorithm I would like to pursue:

1) Prioritized Experience Replay