

Background

1. Reinforcement Learning

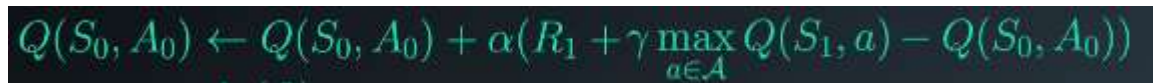
Reinforcement learning is a set of techniques to train a system (i.e., an agent) to respond adaptively within a task context through interaction. If the task context can be represented through a Markov Decision Process – where the task context has a defined set of possible states, actions, and rewards – the agent will learn to choose the best set of actions for a given state that maximizes cumulative, *future* reward (defined by task context such as winning = 1, losing = -1 in a game of pong) obtained during the interaction. Here, the best possible action is solely determined by the current environment state.

An excellent resource to learn more about reinforcement learning is Richard Sutton and Andrew Barto's Reinforcement Learning: An Introduction (1) (<http://incompleteideas.net/book/bookdraft2017nov5.pdf>).

2. Value-Based Methods and Deep Q-Networks

A method for an agent to select the best possible action for a given state is through the maintenance of a Q-table. Q-tables can be thought of as a state-action lookup table that the agent views in determining an action. For a given row (i.e., state), the agent will select the column (i.e., action) that contains the largest value (i.e., leads to greatest future reward).

Q-Learning is one of several algorithms used to construct the Q-table. During training, the agent updates the Q-table with the following:


$$Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha(R_1 + \gamma \max_{a \in A} Q(S_1, a) - Q(S_0, A_0))$$

Screen grab taken from Udacity's Deep Reinforcement Learning Course (<https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893>)

Here, $Q(S_0, A_0)$ represents the value of the given state-action pair at the current time step. This value is updated by taking the immediate reward (R_1), the maximum cumulative reward that is expected from the next state onwards – denoted by $Q(S_1, a)$ (where a is the best action one can take) – and subtracting the Q-table's current estimate of the cumulative reward. The resulting effect is that, through interaction, the value at any given state-action pair (i.e., S_0 - A_0 pair) will converge to the expected cumulative future reward the agent should expect from taking the action A_0 in the state S_0 . Parameter γ is the discount factor (set below 1, with a range typically between 0.9-0.99) which has the effect of “pressuring” the agent to always select the best action *now* by punishing the agent for waiting to take the action later (as $0.99^{\text{time steps waited}}$ will rapidly decrease the value of the action in the future). Parameter α is the learning rate which dictates how much the Q-table values are influenced by the cumulative reward estimates of the next state (too small a value and learning is slow, too large and learning can be unstable).

For successful learning, agents will adopt an ϵ -greedy policy, which first selects random actions so that more states can be explored, but over time the degree of randomness is reduced so that there becomes an increased likelihood of selecting the best *estimated* action

for a given state. Once training is done, you then proceed to only select the best action for each given state (this resulting set of “best” actions is called your policy, defined as π).

Deep Q-Networks (DQN) is a small modification to Q-Learning that replaces the Q-table with a neural network (2). Here, the inputs to the neural network is the state of the system, and the output is the value of *each possible action* for that given state. The famous research paper utilizing this approach is the following:

<https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf> where the researchers were able to train a computer to play various Atari games (such as Pong) (2).

Here is a video demonstrating the agent across multiple games:

<https://www.youtube.com/watch?v=iqXKQf2BOSE>.

Learning Algorithm

The algorithm in this repository implements DQN with the additions of a separate *local* and *target* neural network (to provide more stable learning) and an experience replay buffer (where randomly selected batches of experiences are used to train the neural networking. These experiences are randomly selected to remove the correlational structure in environment exploration that the neural network may accidentally learn to control its actions).

1. Neural Network Architecture

A neural network with two fully connected hidden layers were used to control the learning agent. An input later of 37 nodes represented the state dimensions the agent can perceive in the Unity environment. The input layer was fully connected to hidden layer 1 with 128 input nodes, which was then connected to hidden layer 2 with 128 input nodes. The output layer then was reduced to 4 output nodes that represented the actions the agent could make. The first three layers utilized a ReLU activation function, while the output function utilized a linear activation function.

2. Replay Buffer

During reinforcement learning, each (S_0, A_0, R_1, S_1) experience tuple is added to a queue (with maximum length **buffer size** [see below]). Given the **learning frequency**, a mini-batch of experiences are sampled at random and used to train the neural network for one iteration.

3. Local Q-Network, Target Q-Network, and Soft-Update

To provide stable learning, separate networks are kept so that the neural network is not learning to chase “a moving target.” The Q-learning update rule is adjusted so that the change in weights (Δw) is the following:

$$\Delta w = \alpha \left(R + \gamma \max_a \hat{q}(S', a, w) - \hat{q}(S, A, w) \right) \nabla_w \hat{q}(S, A, w)$$

Screen grab taken from Udacity's Deep Reinforcement Learning Course
(<https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893>)

Where $q(S',a,w)$ is the target Q-network (in red) that is updated periodically and represents the value function of the best possible action in the next step, while $q(S,a,w)$ (in blue) is the current, local value estimate.

As the local Q-network is trained (given the **learning frequency** [see below]), the target Q-network is gradually updated via the following soft update function:

$$Q\text{-network}_{\text{target}} = \tau(Q\text{-network}_{\text{local}}) + (1-\tau)(Q\text{-network}_{\text{target}})$$

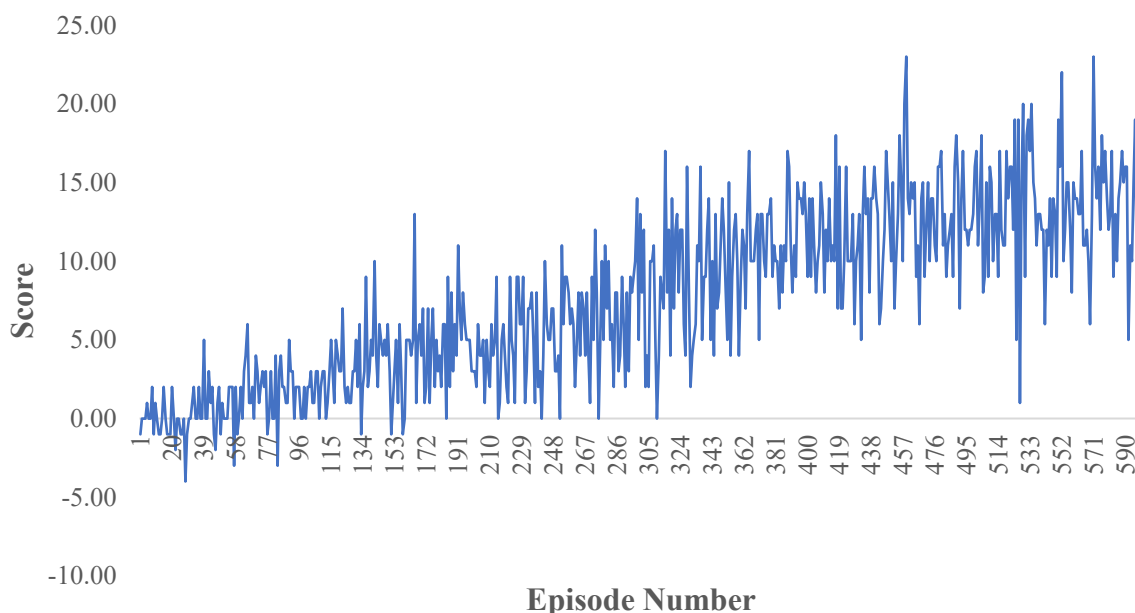
where τ is the interpolation factor. The above soft-update function was taken from a coding exercise project from Udacity's Deep Reinforcement Learning course. The resulting effect is more stable learning.

3. DQN Hyperparameters:

- **Replay Memory Buffer Size:** 1×10^5
- **Mini-batch Size:** 64
- **γ (gamma/discount factor):** 0.99
- **τ (tau/network interpolation):** 1×10^{-3}
- **α (alpha/learning rate):** 5×10^{-4}
- **learning frequency:** 4 timesteps
- **Total number of episodes:** 600
- **Maximum timestep per episode:** 1000
- **Epsilon Start / End / Decay Rate (the likelihood a random action is taken):** 1 / 0.01 / 0.995

Plot of Rewards

An example plot of scores across 600 episodes is plotted below. Within 600 episodes, the agent was able to reach a mean score of 13.37 over the last 100 episodes.



Ideas for Future Work

The included source code can be expanded to incorporate 1.) *prioritized replay*, which prioritizes the selection of rare events in the memory buffer for more robust learning, 2.) modifications of the original DQN architecture to include architectures such as *duelling DQN* (3), which has separate networks for estimation of state values (independent of action) and the action advantages (for a given state), and 3.) implementing other reinforcement learning approaches such as curriculum based learning to assist the agent in learning the appropriate actions across every increasing level complexity (to mirror developmental learning in humans).

References

1. Sutton RS, Barto AG (2016) *Reinforcement Learning : An Introduction* doi:10.1109/TNN.1998.712192.
2. Mnih V, Kavukcuoglu K, Silver D, Rusu A, Veness J (2015) Human-level control thorough deep reinforcement learning. *Nature*. doi:10.1038/nature14236.
3. Wang Z, et al. (2015) Dueling Network Architectures for Deep Reinforcement Learning. Available at: <http://arxiv.org/abs/1511.06581> [Accessed October 19, 2018].