Deep Q-Network Agent

Project Navigation

As part of Udacity Deep Reinforcement Learning Nanodegree

Sourav Babi Seal (sourav.seal@gmail.com)
June 2025

Summary: Train a Deep Q-Network Agent (a value-based RL method) to navigate around in a square world and collect yellow bananas. The environment provided by Unity is provided by Unity Machine Learning agents.

Environment

State Space

The state space has **37 continuous variables** that includes the agent's velocity and a ray-based object perception in the agent's forward direction.

Action Space

The agent has **four discrete actions** it can impact the environment by:

- Move Forward
- Move Backward
- Turn Left
- Turn Right

Rewards

- +1 for collecting a yellow banana
- -1 for collecting a blue banana

To solve this environment the agent must get an average of +13 over 100 consecutive episodes.

Methods

DQN Model Architecture

Employs a Deep Neural Network to approximate the Q-value optimal action-value function and map the input dimension of 37 variables to an output dimension corresponding to Q-value for each of the 4 actions in a single forward pass.

It has 3 fully connected (FC) layers

- FC 1 dimensions: input state size 37, and the output 64
- FC 2 dimensions: input state size 64, and the output 64
- FC 3 dimensions: input state size 64, and the output 4

FC 1 and FC2 have ReLU activation functions.

Training algorithm for deep Q-networks

We implement the training algorithm of deep Q-learning with experience replay and fixed Q Targets as outlined in the DQN paper.

There are two main parts to the Deep Q-network algorithm:

Sample: The Agent samples the environment by performing actions (action A from state S using an epsilon-greedy policy and observing the reward R and the next state S', and storing the observed experience in a replay memory.

Learn: A batch of experiences tuples is selected from this memory, and learn from the batch. The Agent learning step here does a gradient descent to minimize the MSE loss between the expected Q-value and the target Q-values.

There are two key enhancements the algorithm took to address the divergence and instability known to Reinforcement Learning:

1. **Experience Replay:** that randomizes over the data to remove correlations between consecutive experience tuples in the observation sequence. The experience tuples (S, A, R, S) are stored in a replay buffer. A batch of experience tuples are randomly sampled to calculate the expected value function.
2. **Fixed Q-targets:** An iterative update that adjusts the action-values (Q) towards target-values only periodically to reduce correlations with the target. The main motivation behind this is to avoid potential correlations that are possible in vanilla Q-learning when we update a guess with another guess which can lead to harmful correlations. Two networks are used: *local* and a *target* Qnetwork. The local network weights are updated at every gradient step, while the target network is updated with the weights of the local

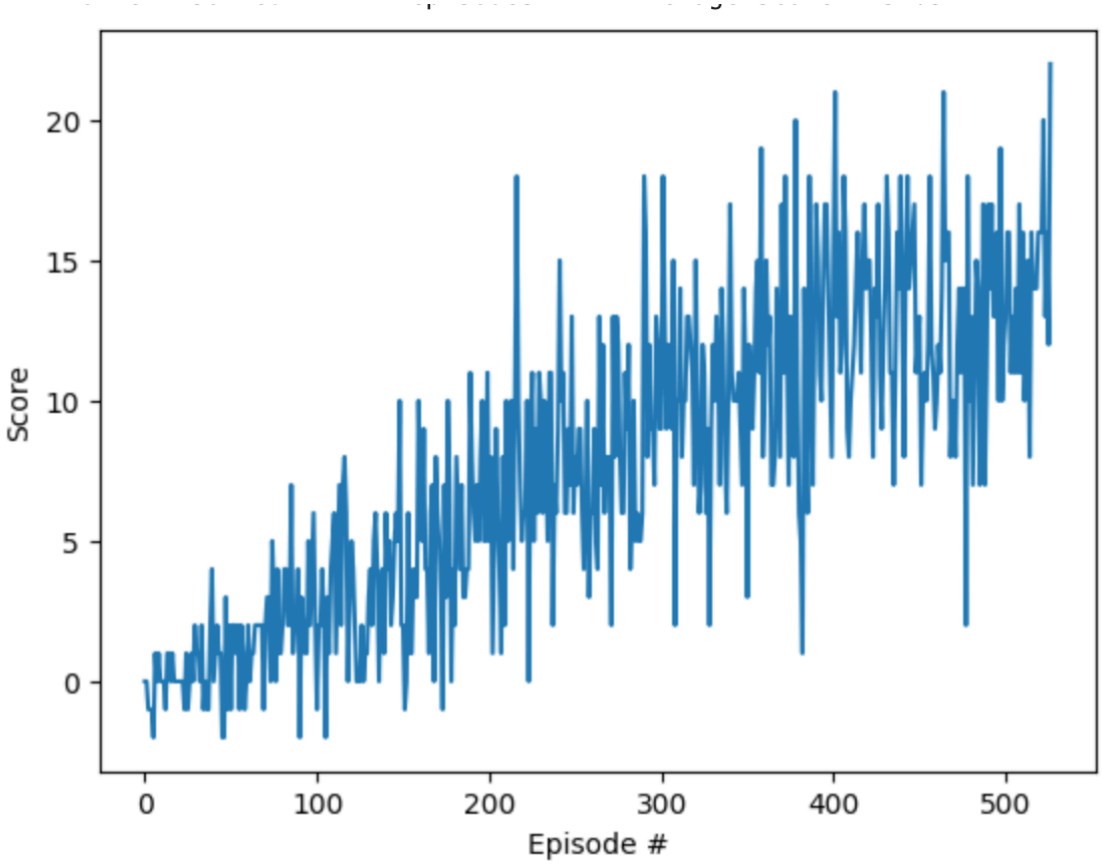
network at periodic intervals. Note this is a soft update where the target network parameters are updated gradually using an exponential moving average for increased stability and a more smoother learning curve.

Hyperparameters

Hyperparameter	Value
Batch Size	64
Discount Factor (GAMMA)	0.99
Learning Rate (LR)	5e-4
Soft Update of target network parameters (TAU)	1e-3
Replay Buffer Size	int(1e5)
Learn every UPDATE_EVERY time steps	4

Results

We solved the environment in 527 episodes, sustained an average of 13 score



Future Work

There are three different improvements that we can make to the above DQN algorithm

1. **Double DQN** Address issues with over-estimation of action values due to mistakes with the arg max operation for selecting the best action using one-set of parameters w but evaluating with a different set of parameters (w^-). We reuse the frozen set of parameters with fixed targets for this evaluation.
2. **Prioritized Experience Replay** In the experience replay when we pick an experience instead of randomly sampling the experiences we prioritize more important experiences to learn from. The priority is based on the magnitude of the TD error which is stored in the replay buffer and the priority influences the sampling probability taking care to avoid over-fitting to subset or adding an epsilon for zero errors and correcting for sampling bias.
3. **Dueling DQN** Use two streams in a Dueling DQN where the initial convolution layers are shared, but then it forks into two streams, one to estimate the state value function V(s) and the other to estimate the advantage for each action A(s,a) and combining the two values for improvements over vanilla DQNs.

In addition the paper references optimizations such as reward clipping, error clipping, storing past actions as part of the state vector, and dealing with terminal states, etc.

References

[Human-level control through Deep Reinforcement Learning]

Udacity Deep Reinforcement Learning Course.