DQN – Navigation

1. Introduction

In this project, we train an agent to navigate and collect bananas in a large, square world.

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of our agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around agent's forward direction. Given this information, the agent must learn how to best select actions. Four discrete actions are available, corresponding to move forward, move backward, turn left, and turn right.

The task is episodic, and to solve the environment, our agent must get an average score of +13 over 100 consecutive episodes.

2. Implementation

2.1. Learning Algorithm

The learning algorithm in this project is DQN and it was implemented using PyTorch. The image below illustrates the main components of DQN.

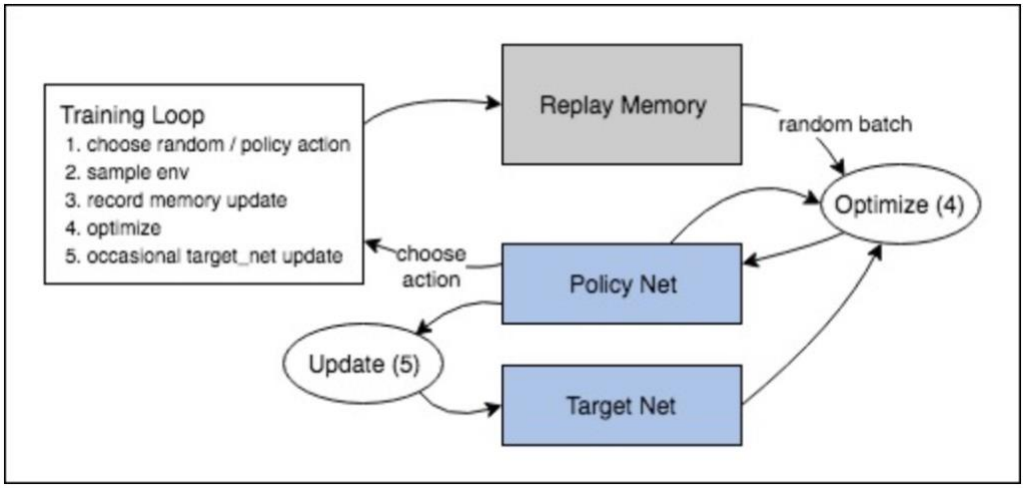


image source: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

Three helper classes are used to implement the DQN algorithm:

Class “QNetwork” implements the Neural Network which is composed by the following layers:

- 37 input neurons
- 128 Fist hidden layer neurons
- 128 Second hidden layer neurons
- 4 Output neurons

The “forward” method builds a network that maps state to action values using RELU as the activation function.

Class “ReplayBuffer” implements the experiences store and contains two main methods, “add” and “sample”, which add new experiences to the memory and randomly sample a batch of experiences respectively.

Class “Agent” interacts and learns from the environment. The class initialises the Policy and Target network and sets Adam as optimiser. Methods of this class are “Step”, “Act”, “Learn”, and “Soft update” that conclude the DQN required components.

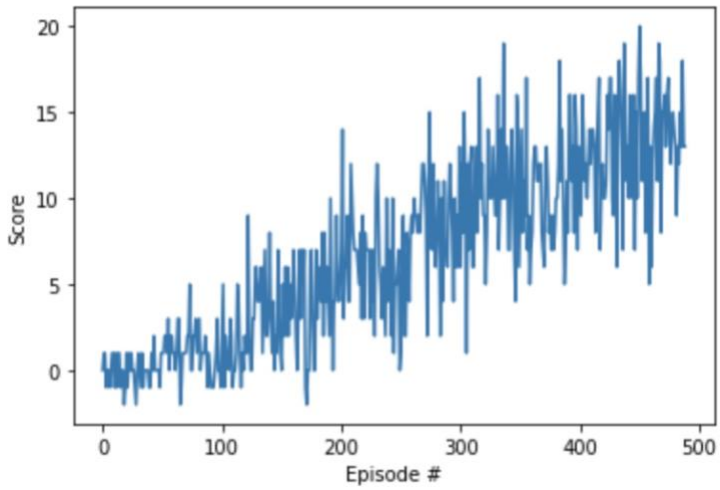
2.2. Hyperparameters

The table below shows the chosen hyperparameters, decided by a combination of trial and error and of baselines in bibliography.

Name	Value	Description
BUFFER_SIZE	1e5	Replay buffer size
BATCH_SIZE	64	Minibatch size
GAMMA	0.99	Discount factor
TAU	*1e-3*	Soft update of target parameters
LR	*5e-4*	Learning rate
UPDATE_EVERY	4	How often to update the network
EPSILON_START	1.0	Epsilon start value
EPSILON_END	0.01	Epsilon minimum value
EPSILON_DECAY	0.99	Epsilon decay value
N_EPISODES	2000	Total number of episodes
MAX_TIMESTEPS	1000	Maximum steps per episode

2.3 Results

The plot below illustrates the agent’s rewards per episode. The agent was able to meet that target of +13 average reward over 100 episodes after just above 480 episodes.



3 Future Work

Using a hyperparameters tuning tool might increase the agent's performance. For example, Tune, a Ray library, can run experiments in parallel and once it has converged in the best performing hyperparameters it can continue the training loop using them.

Additionally, this implementation is using "simple" DQN, further experimentation with more advanced versions such as prioritised experience replay, dueling DQN, RAINBOW and more.