Artificial Intelligence Nanodegree
Reinforcement Learning

Project Navigation

Author: Marvin Lüthe

Introduction

The aim of this project is to use **deep reinforcement learning**, a combination of **reinforcement learning** and **deep learning**, to 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. Accordingly, the goal of the 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 the agent's forward direction. Given this information, the agent shall learn how to best select actions out of four possibilities:

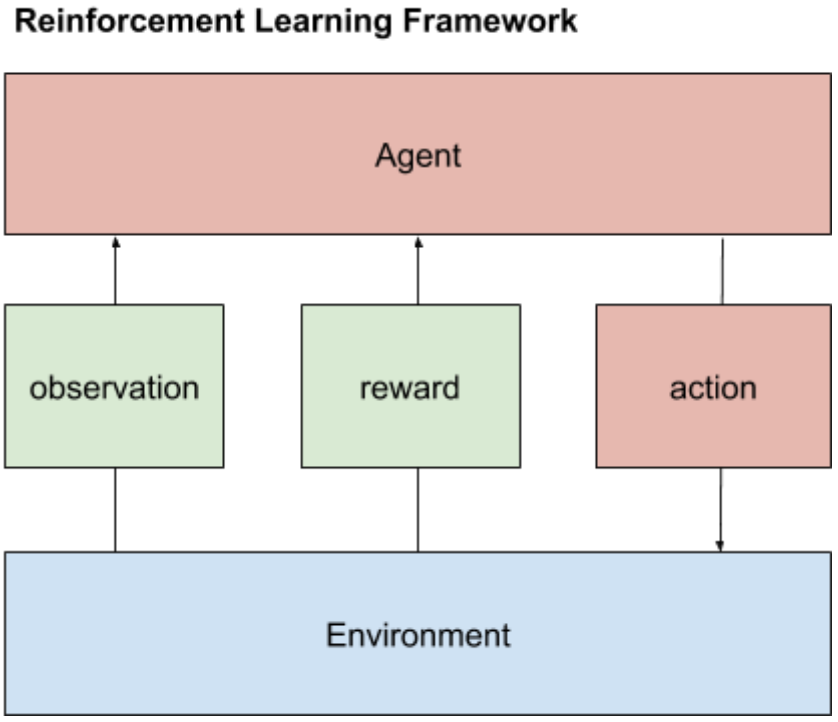
- 0: move forward
- 1: move backward
- 2: turn left
- 3: turn right.

This task is episodic. In order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

Learning Algorithm

Framework

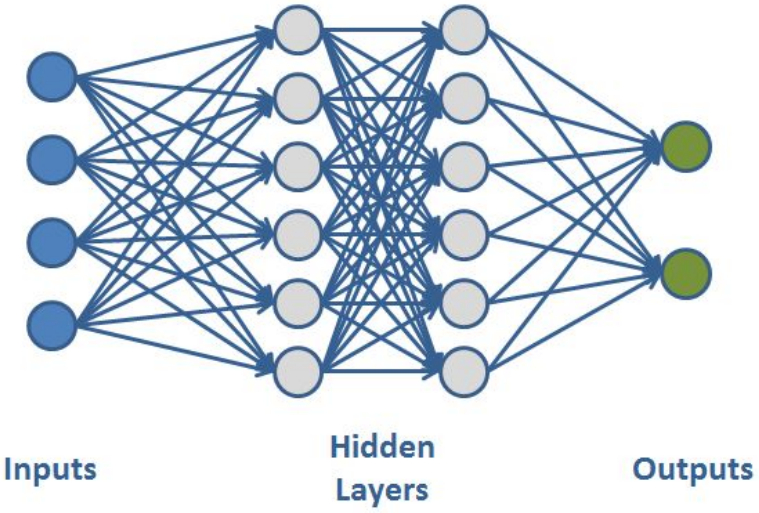
We train our agent using a combination of **reinforcement learning** and **deep learning**, called **deep reinforcement learning**. Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions. The focus is on finding a balance between exploration and exploitation. It is characterized by an agent learned to interact with its environment. The agent first observes the environment and takes action. Then, the environment changes and the agent observes again and receives a reward which indicates whether the agent has responded appropriately to the environment.



Deep Learning

Deep learning uses artificial neural networks, inspired by the brain, to train models in classification and regression tasks. These neural networks are composed of layers of nodes and weights that connect these nodes. Below we can see an example with one input layer, two hidden layers and one output layer.

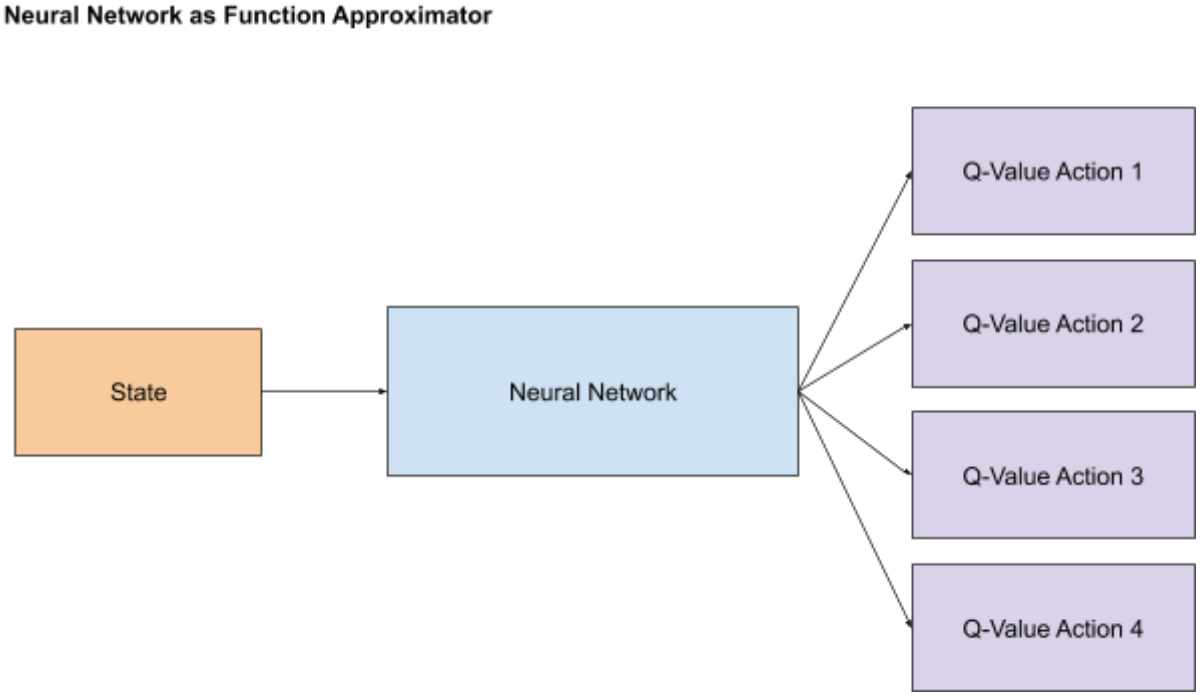
Neural Network



The shown neural network is very similar to the chosen neural network for this project. The implemented neural network consists of:

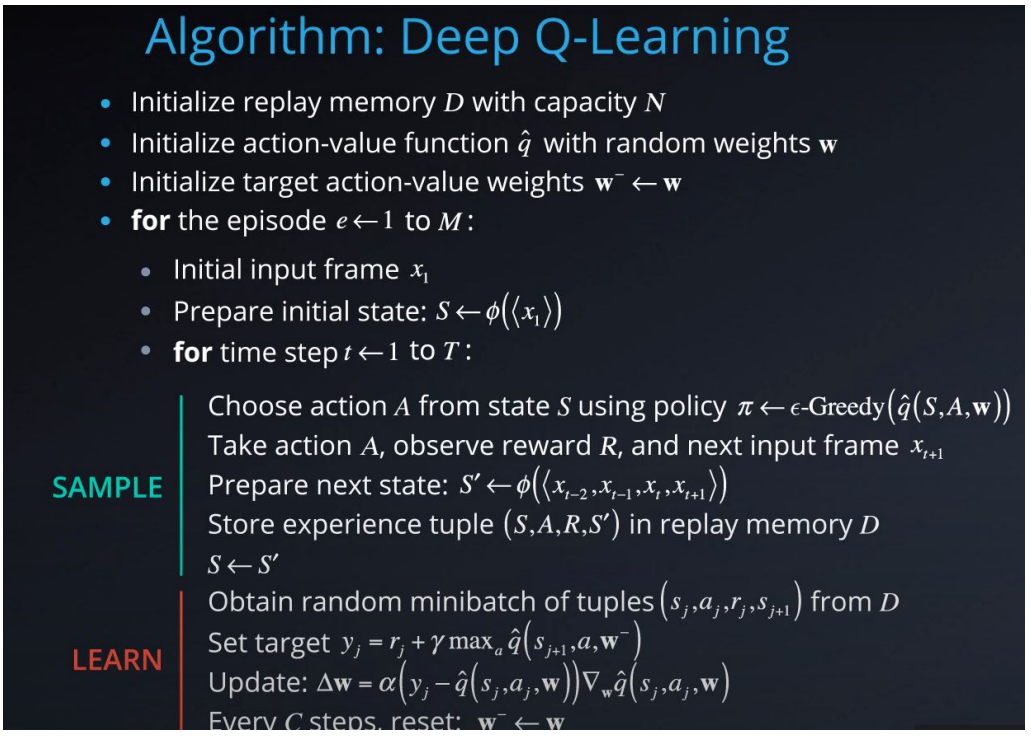
- one input layer with 37 nodes (state size)
- two fully connected hidden layers with 64 nodes
- one output layer with 4 nodes (action size)

The neural network acts as a function approximator which receives the state (37 parameters) as input and returns the approximated Q-Values for each action.



Deep Q-Network

The full DQN algorithm consists of the steps shown below:



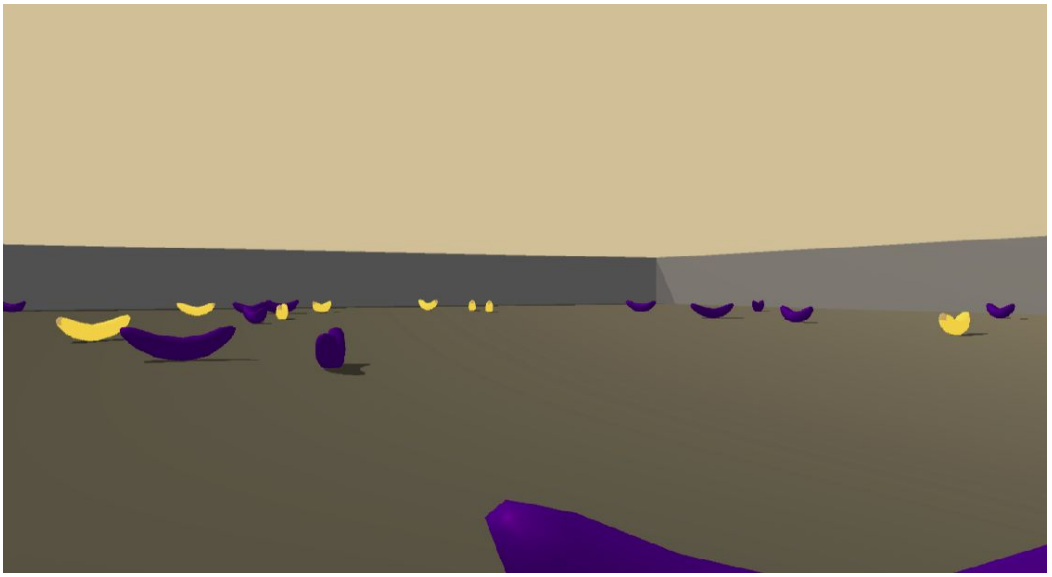
Source: Udacity

We initialize a buffer for experience replay and two neural networks. The local network will be used for the sample phase. During sampling, the agent interacts with the environment and stores these experiences in the buffer. The actions are selected based on an epsilon-greedy policy and the local network being the Q-table which approximates the action-value function.
During the learning phase, we update the weights of the neural networks based on the magnitude of error between the expected and target values. The expected values result from the local network while the target network provides the target values. Using gradient descent, we update the values of the local neural net and softly transfer this update to the target network to ensure smooth learning.

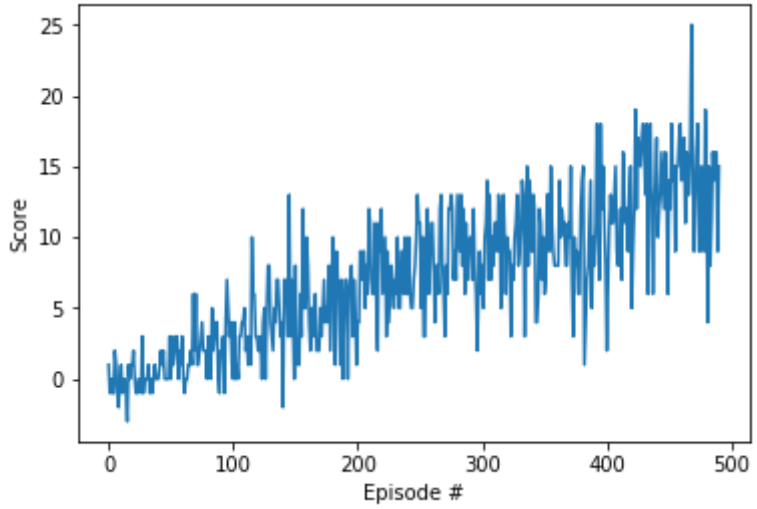
Performance

The goal of the project is to train a deep reinforcement learning agent that receives an average score of +13 over 100 consecutive episodes.

The square world looks like this:



The target of +13 average score was reached after 491 episodes and we saved the weights of the neural network.



I selected the following hyperparameters to train the agent:

Parameter	**Value**
Replay Buffer Size	100.000
Batch Size	64
Gamma	0.99
Tau	0.001
Learning Rate	0.0001
Update Every	4
Epsilon Init	1.0

Epsilon Decay Rate	0.995
Minimum Epsilon	0.01
Max Time Steps t	1000
Max Number of Episodes	2500

Conclusion

In this project we used deep reinforcement learning to train an agent navigate in a square world and collect yellow bananas. We combined the building blocks of deep learning and reinforcement learning to learn how to act in a given environment in order to maximize the reward received.

The implementation can be enhanced by techniques such as **Double DQN**, **Prioritized Experience Replay** and **Dueling DQN**.

DQN tends to overestimate action values. In the early episodes, the argmax function for the Q-values of the target network can take “wrong” actions as the Q-values are evolving. Accordingly, the idea behind **Double DQN** is to select one action with one set of parameters but evaluate with another set of parameters. The two different function approximators must agree on the same action.

During the learning phase, random samples of experience tuples are selected to update the network’s weights. **Prioritized Experience Replay** is a technique which adds a sampling probability to the experience tuples so that the agent can learn more effectively from certain (tricky) situations.

Currently, in order to determine which states are (or are not) valuable, we have to estimate the corresponding action values for each action. However, by replacing the traditional DQN architecture with a **dueling architecture**, we can assess the value of each state, without having to learn the effect of each action. The intuition behind **dueling networks** is that the state values do not vary a lot across actions. We can try to directly estimate them and combine them with the advantage values to get the Q-values.