Navigation

Introduction

The objective of this project is to test various deep reinforcement learning methods on a navigation task in a randomized environment with no prior information. Essentially, the agents proposed in the various experiments will have to explore a virtual room during a fixed period trying to maximize its score without knowing the rules of the game.

With each action the agent takes, the environment will return some quantitative feedback and the agent will update its underlying model in an iterative training process.

The Environment

In this project, we will be using a Unity Machine Learning Environment where the player can navigate a square room where bananas keep randomly spawn. There are 2 types of bananas that can spawn, and the objective is to collect the maximum number of yellow bananas while avoiding the blue bananas.

Each yellow banana counts as 1 point, each blue banana counts as -1 point and the experiment is considered successful when the agent gets 100 consecutive episodes where the final score is over 13.

Algorithms

Deep QNetwork

Deep QLearning is a refinement on QLearning where we use a neural network to compute Q values instead of a lookup table. This is especially useful when the state space is large or non-discrete.

In essence, QLearning tries to determine the best action to take given a state. The training process consists, in layman terms, taking a pair of action and state and the best possible outcome for the next state, checking if the reward received is positive and updating the model (according to *Equation 1*, where *s* is the current state, *a* is the action taken, *s'* is the posterior state and *γ* is a tunable hyperparameter).

$$Q(s,a) = r(s,a) + \gamma \cdot \max(Q(s',a))$$

Equation 1

Double QNetworks

Double DQN is a solution proposed by *Hado van Hasselt* in 2010 to try to mitigate the inherit overfitting problem of Deep QNetworks.

The idea is that we are trying to estimate what will be the best position of the model for the next state using the same model we are training and, therefore, it will be a noisy measure (especially in the beginning of the training process) and the accuracy will depend greatly on how deep we have explored our neighbors. To try and circumvent that, van Hasselt proposed to decouple the problem and use two different QNetworks.

We will be using a target network to actively decide the next action given the current state and a local learning network to compute the improvements on the model. After each training step, we will interpolate between the target network and the local network to effectively update the model.

Prioritized Replay Buffer

The objective behind Prioritized Replay is assign an importance value to each stored learning frame so training process prioritizes selecting a set of frames that can better teach the model. In layman terms, we want the model to learn from its worse mistakes, so we rate each iteration on how bad they were.

In this case, we will be assigning maximum priority newly stored frames and use a weighted loss (squared error) to update the priorities on used examples.

$$P_i(x) = \frac{p_i(x)^\alpha}{\sum_{i=0}^{N} p_i(x)^\alpha}$$

Equation 2

In the sampling process we will assign the probability of being picked as a function of the percentage of the priority of a frame over the sum of all priorities in the container as seen in *Equation 2,* where P_i represents the probability of picking the i-th example, p_i is the priority assigned to the i-th example and α is a tunable parameter to determine the weight of the priority system (α=0 generates an equiprobable distribution).

As a result of cherry picking the examples deviating from the underlying distribution leant from the environment, we need to adjust how we assign the importance of every frame by computing weight (W_i) for every selected example to later update its priority after the learning step. We will be using a function of the probability assigned, the number of elements in our buffer (N) and a tunable parameter β.

$$w_i = \left(N \cdot P_i(x)\right)^{-\beta}$$

$$W_i = \frac{w_i}{\max_{t=0}^{N}(w_t)}$$

Equation 3

Method

Model Comparison

In the first segment of the project, we will explore the behavior of the different combinations of Networks and Buffers during 1000 episode using the default, to get an initial intuition on which model could produce the better results under the constraints of the environment.

Models:

- Deep QNetwork with Replay Buffer
- Deep QNetwork with Prioritized Experience Replay Buffer
- Double Deep QNetwork with Replay Buffer
- Double Deep QNetwork with Prioritized Experience Replay Buffer

Parameter Tunning

Once we have determined the combination that yields the best results under the default configuration, we proceed to analyze if there is a concrete combination of hyperparameters that can improve what we have seen previously. The tunable hyperparameters that we will consider in this experiment will be the learning rate of the optimizer, the smoothing parameter *Tau* and the Gamma parameter.

Final model

In this section, we will train the best performing network using the best performing parameters from the last section during 5000 episodes to check the score evolution on a long training session

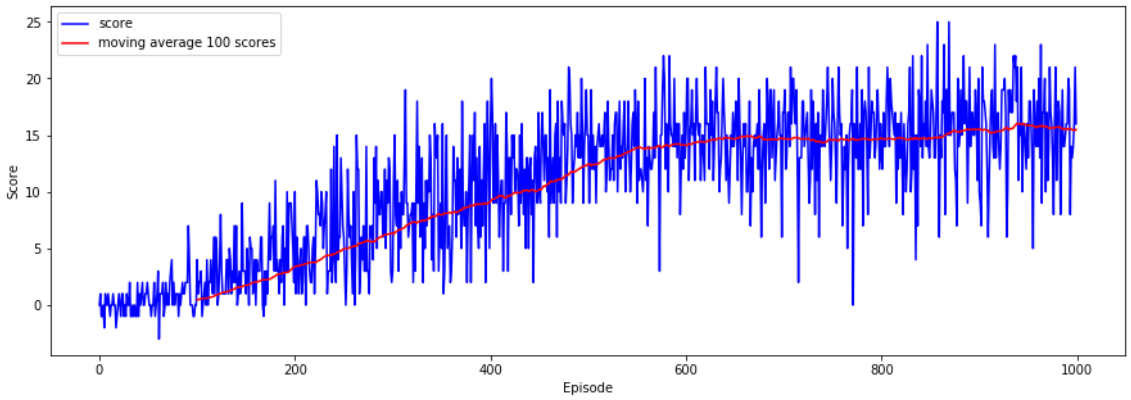
Results

Model Comparisson

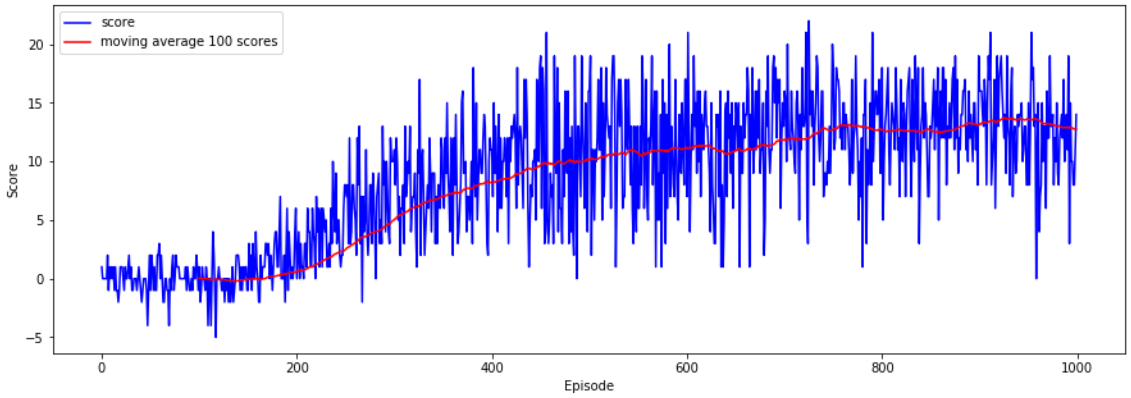
Default Parameters

- Optimizer: Adam (learning rate 10^{-3})
- Buffer Size: 10^5
- Training batch: 128 examples
- Gamma: 0.99
- Tau: 10^{-3}

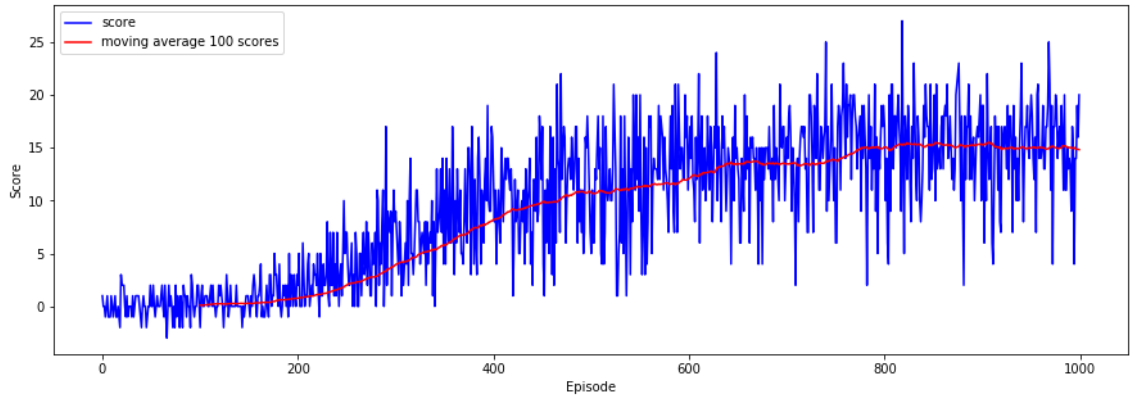
QNetwork with Replay Buffer



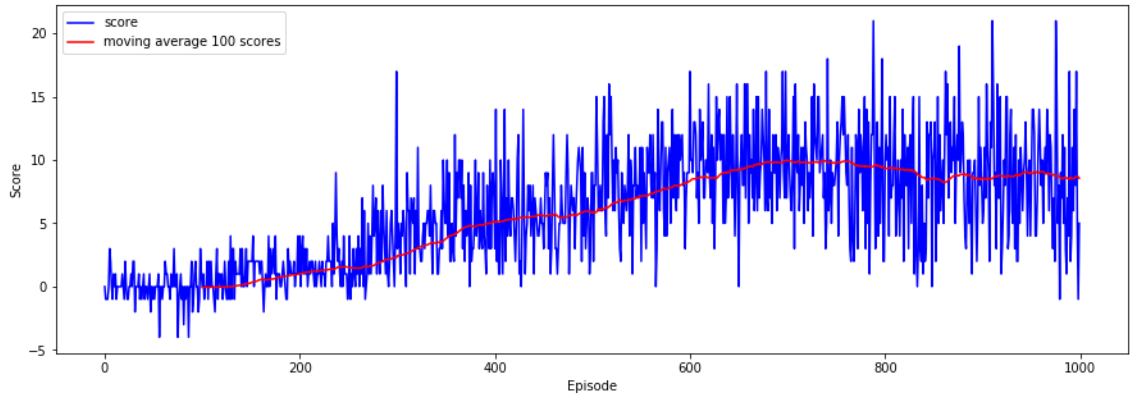
QNetwork with Prioritized Experience Replay Buffer



Dual QNetworks with Replay Buffer

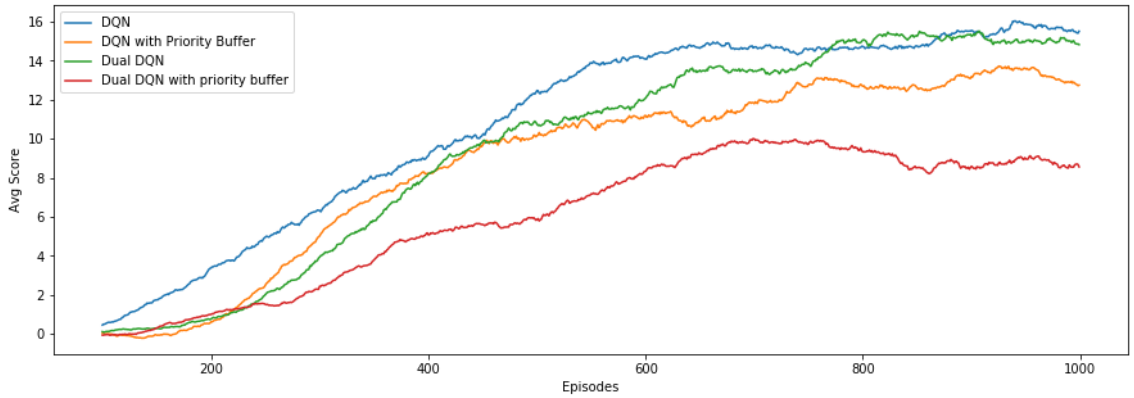


Dual QNetworks with Prioritized Experience Replay Buffer



Comparison

Moving Average (100 samples) comparison



Parameter Tunning

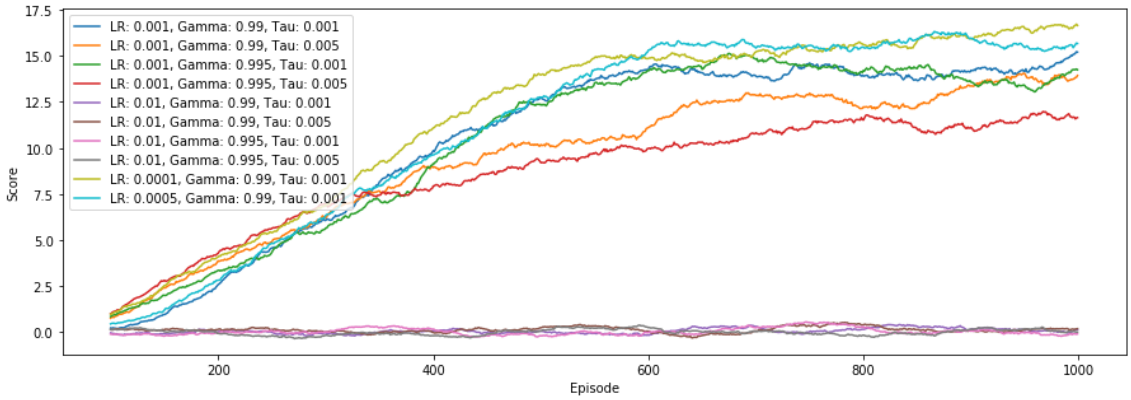
We used a QNetwork with a standard replay buffer as the underlying model for this section.

We initially checked the model behavior for any combination of:

- Learning rate: 10^{-3} or 10^{-2}
- Gamma: 0.99 or 0.995
- Tau: 0.001 or 0.005

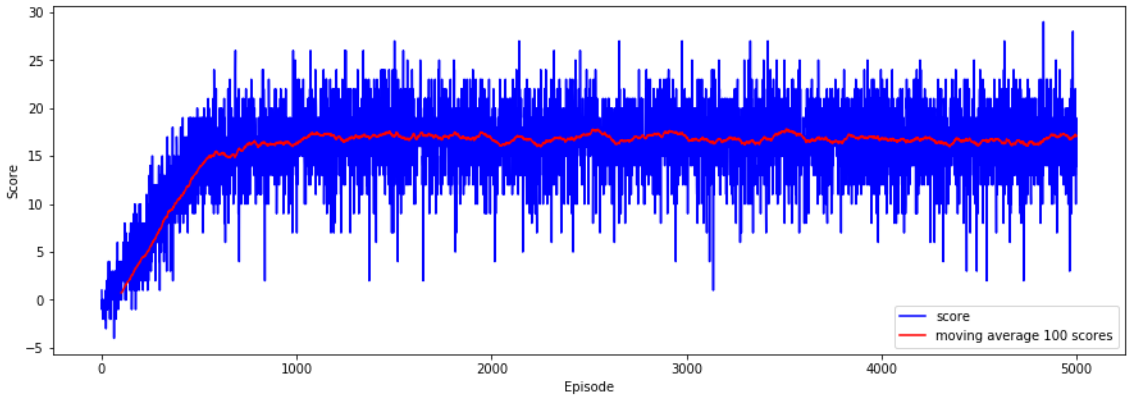
After analyzing the initial results, we decided to additionally check the behavior of the model using:

- Learning rate: $5x10^{-4}$, Gamma: 0.99, Tau: 0.001
- Learning rate: 10^{-4}, Gamma: 0.99, Tau: 0.001



Final model

- QNetwork with standard replay buffer
- Optimizer: Adam with learning rate of 10^{-4}
- Buffer size: 10^5
- Training batch: 128
- Gamma: 0.99
- Tau: 10^{-3}



Conclusions

In this case, it appears that the *simpler* solution is good enough to build a robust model capable of reliably navigate the environment with acceptable scores.

Next Steps

To further explore possible solutions to this environment, it could be interesting to implement state of the art algorithms in deep reinforcement learning like Dueling QNetworks or Rainbow.

Furthermore, another possible interesting line of investigation could be using image data to train a QNetwork based on CNNs. Up to this point, there is some work done on the Visual Navigation jupyter notebook but there are no relevant results yet.