Project 1: Navigation

Kaushik Balakrishnan

October 31, 2019

1 Problem Statement

For the first project, we will attempt to solve the banana picking environment in Unity. This environment involves an enclosure where yellow and blue bananas are scattered around. For picking up every yellow banana, a reward of +1 is achieved; on the other hand, for picking up every blue banana, a -1 reward, i.e., a penalty, is realized. The goal is to collect as many yellow bananas as possible. In particular, the problem is considered solved when the average reward over the last 100 episodes is 13. See Fig. 1 for a schematic of the banana environment in Unity. This environment has a state space of size 37 and an action space of size 4.

2 DQN

For the problem of banana picking, we use the Deep Q-Network (DQN) [1], and this report will summarize the algorithm as well as the different aspects of it. Reinforcement Learning (RL) is a Markov Decision Process (MDP) where the agent takes an action depending on the immediate state it is in, and does not depend on the past states. RL involves a tradeoff between exploration and exploitation, where exporation is to take random actions to understand the surroundings better, and exploitation is to take a greedy action based on policy or value. In particular, the use of a neural network in RL as a function approximator is called Deep Reinforcement Learning (DRL). Furthermore, a replay buffer is also used.

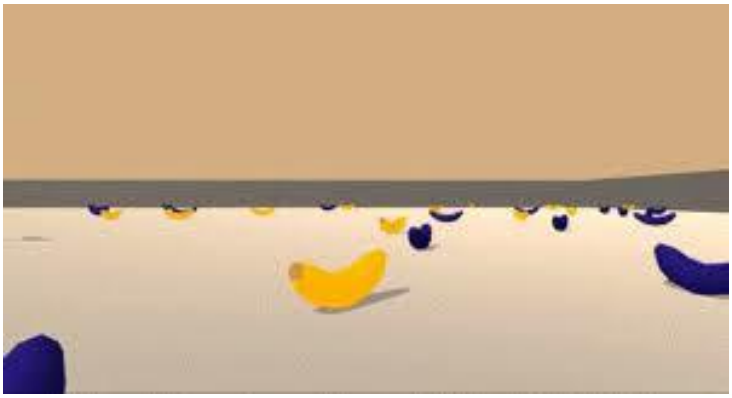

Figure 1: Banana environment in Unity.

2.1 Q-network

A neural network is used as the function approximator to obtain Q(s,a), since the number of states is very large and a classical tabular approach is infeasible. We use a neural network with two fully connected hidden layers and an output layer as our Q-network. All hidden layers have 128 neurons per layer and use the Relu() activation function. The output layer is linear and outputs 4 Q-values, i.e., one for each action. The network is updated using the Adam optimizer [2].

2.2 Target network

The target network is used for stability reasons as the primary Q-network cannot be used directly in the Bellman equation as that will not be stable enough to estimate Q values as it is changing continuously. The target network is updated slowing using a variable $\tau = 10^{-3}$, with a weighted average of the new and old parameter values weighted with τ and 1-τ, respectively. Thus, the target network slowly follows the primary Q-network.

2.3 Replay buffer

We desire i.i.d. samples to train the Q-network and so to avoid correlated samples, a replay buffer is used where past samples are stored and a mini-batch of experiences is sampled from it at every update. This mini-batch of experiences is used to update the Q-network using Adam [2]. Note that by

experience, we refer to the tuple: (state, action, reward, new state, done). We consider a standard replay buffer for this project where all the samples in the replay buffer have equal probability of being sampled, although other approaches such as prioritized experience replay (PER) [3] can also be considered; however, PER is beyond the scope of this project and not considered here.

2.4 ϵ-greedy

RL is a tradeoff between exploration and exploitation. ϵ-greedy is a common exploration strategy widely used in RL. Here, a random number, r, is generated at every time step, and only if $r \geq \epsilon$, a greedy action is taken; on the other hand, if $r < \epsilon$, a random action is taken to explore. Note that early in the training, we desire more exploration in order to better understand the environment. But in the later stages of the training, we must exploit more as we have a reasonable understanding of the environment. Thus, ϵ is sufficiently large early on, but decreases to a smaller value in the later stages of the training.

In particular, we consider two different strategies for decreasing ϵ: (1) linear annealing; (2) exponential decay. For linear annealing, ϵ is linearly annealed from 0.75 to 0.02 over the course of 1500 episodes and held fixed once its value reaces 0.02.

$$\epsilon = max\,(0.75 - i/1500\,(0.75 - 0.02), 0.02), \tag{1}$$

where i is the current episode count.

For exponential decay, we consider a half-life of 250 episodes:

$$\epsilon = max\,(0.75\,e^{-i/250}, 0.02). \tag{2}$$

Both these strategies will be considered in this report.

2.5 Setting up the environment

Download the four Python files from this project:

1. dqn_agent.py This file contains the DQN Agent class, which combines the QNetwork, the target network and the replay buffer

description	name of hyperparameter	value
discount factor	γ	0.99
max number of episodes	nepisodes	5000
batch size	batch_size	64
buffer capacity	buffer_capacity	50000
start value of ϵ	eps_start	0.75
end value of ϵ	eps_min	0.02
learning rate for Adam	lr	1e-3
target net update parameter	τ	1e-3

Table 1: Table of hyperparameters

2. model.py This file contains the QNetwork class, which is the neural network function approximator for Q(s,a)

3. replay_buffer.py This file has the standard ReplayBuffer class, which stores and samples experiences

4. train.py This file combines all the others and the main() function is here for training the DQN agent

Copy the Banana_Linux folder from the Udacity website to this directory. Set the path to the Banana.x86_64 file in train.py in the mypath variable:

```
mypath = "/home/kb/rlnd/p1/Banana_Linux/Banana.x86_64"
```

Install MLAgents using pip, and also download unityagents and communicator_objects from the Udacity project website, creating symbolic links to them like so:

```
pip install mlagents
ln -s <path to unityagents> .
ln -s <path to communicator_objects> .
```

2.6 Hyperparameters

The hyperparameters used are summarized in Table 1.

2.7 Start the training

To start the training, run the command:

```
python train.py
```

2.8 Saving the model weights

The final trained model weights are saved in model_qnet.pt, which is a PyTorch file.

3 Results

As aforementioned we consider two cases where the ϵ value is decreased either linerly or exponentially. The banana picking problem is considered solved when the average episodic reward for the last 100 episodes is 13.

When ϵ is linearly annealed, the problem is solved in 1452 episodes. The episodic reward is shown in Fig. 2. On the other hand, when ϵ is exponentially decreased, the problem is solved in 584 episodes. The episodic reward is shown in Fig. 3. Thus, the rate at which the exploration-exploitation tradeoff is varied affects how fast the problem is solved, which is intuitive in a way. For ϵ slowly decreased, the agent explores much more than what is required; thus, one mut find a good ϵ variation strategy: for the banana picking task, the exponential decay in ϵ with a half-life of 250 episodes was found to be good as the agent learned in 584 episodes. This training was repeated again from scratch and the results were repeatable.

3.1 Future directions

Double Deep Q-Network (DDQN) [4] can be considered in the future, including also Dueling network architectures [5], as well as the Rainbow network [6].

4 References

1. V. Mnih, K. kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller (2013). Playing Atari with Deep Reinforcement Learning. Technical Report arXiv: 1312.5602, DeepMind Technologies

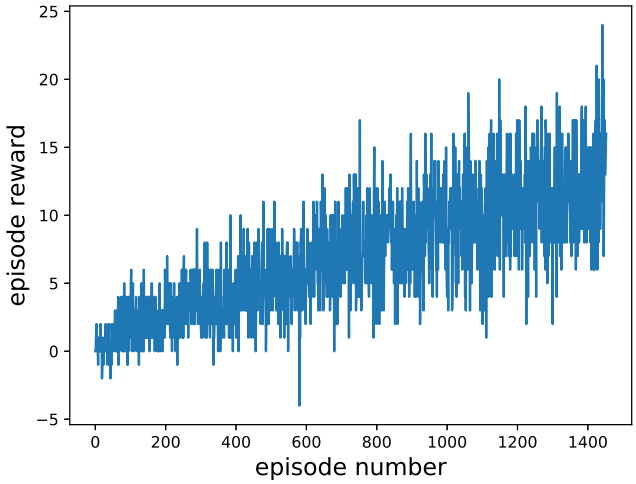

Figure 2: Episodic rewards with linear ϵ annealment.

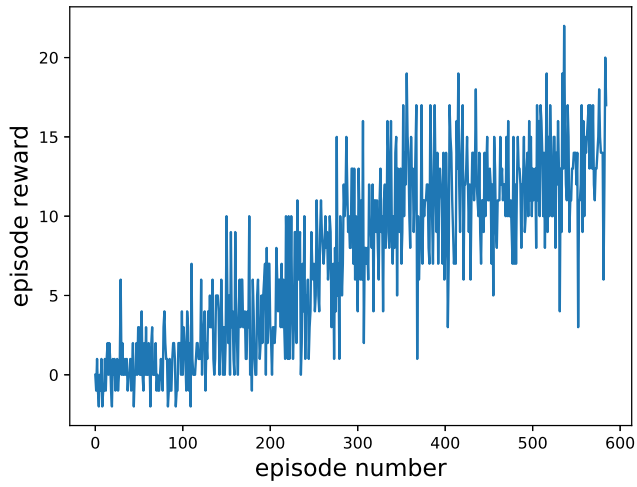

Figure 3: Episodic rewards with exponentially decaying ϵ.

2. D. P. Kingma and J. Ba (2014). Adam: A Method for Stochastic Optimization. arXiv: 1412.6980

3. T. Schaul, J. Quan, I. Antonoglou and D. Silver (2015). Prioritized Experience Replay. arXiv: 1511.05952

4. H. van Hasselt, A. Guez and D. Silver (2015). Deep Reinforcement Learning with Double Q-learning. arXiv: 1509.06461

5. Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot and N. de Freitas (2015). Dueling Network Architectures for Deep Reinforcement Learning. arXiv: 1511.06581

6. M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar and David Silver (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv: 1710.02298