DRLND-P1-Report

Ali Alizadeh

April 26, 2020

1 Introduction

In this report, I briefly summarize my final modeling decisions as to succeed in solving the Navigation project. The implementation of using Deep Q-Network (DQN) for reinforcement learning is described which was used to solve a navigation problem in Banana-collector environment. For the instructions of how to use the code. please refer to the *README.me* file.

I have used standard DQN with experience replay and fixed Q-Targets as described in [1] as well as its variant Double DQN, which improves the performance of standard implementation through avoiding overestimation of the action values.

2 Deep Q-Network Architecture

When we talk about Deep Neural Network (DNN) models and number of parameters, we mean architectures with multiple stacked layers of neurons and large number of parameters, in an order of couple of thousands or millions and above. Here, with state space dimension of 37 and action space dimension of 4, the problem does not need a model with high capacity to capture the dynamics of the environment. therefore, I kept the number of layers low, particularly 3 to 4, and with small number of neurons for each layer, ranging from 64 to 128, I came to the decision of choosing the following architecture which solves the environment under 500 episodes. As it can be seen in Fig. 1, the model is not so deep and it contains 4 hidden layers of $64 - 128 - 64$ with total parameters of around $19K$.

3 Q-Learning models

I utilized both *standard DQN* as in [1] and *Double DQN* [2]. *Double DQN* outperformed the standard one and I decided to proceed with it. Notice that my default mode in my implementation in *Navigation.ipynb* is the Double DQN.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [-1, 64]           2,432
            Linear-2                  [-1, 128]           8,320
            Linear-3                   [-1, 64]           8,256
            Linear-4                    [-1, 4]             260
================================================================
Total params: 19,268
Trainable params: 19,268
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.07
Estimated Total Size (MB): 0.08
----------------------------------------------------------------
```

Figure 1: The summary of the utilized model showing the architecture and number of parameters

4 Hyper-parameter Optimization

In deep learning/deep reinforcement learning realm, correctly deciding on hyper-parameters plays significant role in quickly converging to the solution, stability and robustness of the model performance and so on. There are a large number of hyper-parameters that can be tuned in order to control the performance of the agent. I will describe some important one here such as *learning rate*. There are some other parameters such as the buffer size, discount factor, gain for soft update, and the update frequency of the local and target model which I didn't play with them so much and got an intuition from previous implementations and fixed them in my code.

learning rate, ϵ is defined dynamic and it changes as the learning progresses. I chose $\epsilon_{decay} = 0.99$ and $\epsilon_{end} = 0.007$ as it will determine the learning rate through maximum number of episodes of 2000 with $\epsilon = max(\epsilon_0 \times \epsilon^i_{decay}, \epsilon_{min})$ relation, where $\epsilon_0 = 1$. As it can be noticed, the learning rate will decrease gradually with the number of episode and will be fixed after some time to the miimum epsilon value.

5 Results and Discussion

All competing trained agents in this work could solve the environment in much less number of episodes mentioned in the project instruction. Here in Fig. 2, I present the performance of an agent with decided hyper-parameters as defined in previous sections. I have contended with the success goal of this project which is defined to be +13 and terminate the training progress of the agent once achieved this score. As to give exact number I could achieve the required performace in 364 episodes which is acceptable.

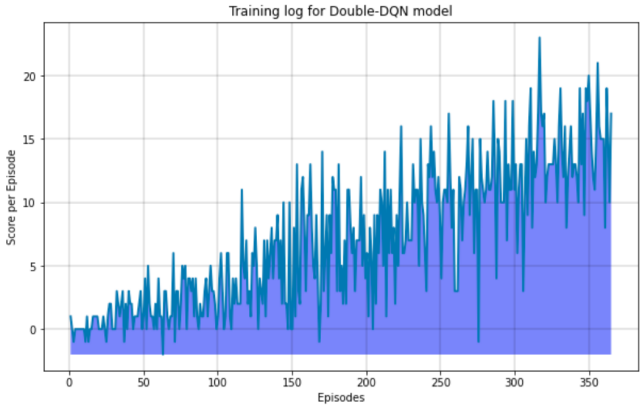

Figure 2: Training log of the agent's performance in banana-collector environment

6 Future Work

Through my experimentation to find the best solution, I found out how hyper-parameter optimization can boost the training progress significantly, increase the performance and stability of the trained agent. Exploring almost all hyper-parameters manually is a tedious job and tracing the combinations is so difficult, so I think my next step can be an automated hyper-parameter search. Another so improtant step will be to utilize the prioritized experience replay instead of the random one used in this work. Using prioritized experience replay can significantly stabilize the model. Also using other variant of DQN such as Dueling DQN is a good move.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.