Report Navigation Project

In this report I describe how I solved the first project. I focused on solving only the mandatory part and the report does not include a solution for the optional pixel challenge.

Learning Algorithm

The learning algorithm was based on the exercise about the Deep Q-Network, as suggested by the instructions. Once I modified and cleaned up the code, I ran a training with the model which immediately gave me in average good results of 16.17 after testing. The code can be found in "models/linear_model.py" as class called "QNetwork".

However, I decided to give the neural network my personal touch, so I extended the file "models/linear_model.py" with another class called "DropoutNetwork". Here I first just wanted to test if a Dropout layer at the end of the regular Network might help with the training. There I managed to get an average score of 15.36 after testing.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Linear: 1-1                            [-1, 64]                  2,432
├─Linear: 1-2                            [-1, 64]                  2,432
├─Linear: 1-3                            [-1, 4]                   260
├─Linear: 1-4                            [-1, 1]                   65
├─Linear: 1-5                            [-1, 4]                   24
==========================================================================================
Total params: 5,213
Trainable params: 5,213
Non-trainable params: 0
Total mult-adds (M): 0.01
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.02
Estimated Total Size (MB): 0.02
==========================================================================================
```

Figure 1: Torch Summary of the solution architecture, DuelingQNetwork

Still eager to improve above the performance of the standard QNetwork, I decided to implement the duelling network architecture from the paper of Wang et al. [1]. Since we had no need to extract the state itself from the game by using a convolutional neural network, the implementation was simple. The input state is forwarded into two separate fully connected with 64 units each. Then one of these layers is connected to a fully connected layer of size 4, which represents action advantages described in the paper[1]. The other layer is then connected to a fully connected layer of size 1, which represents the state value. As a last step state value and action advantages were concatenated into a single layer of size 5 which we then connected with the output layer (size 4) to which torch applies a softmax layer per default. For a torch summary see Figure 1. The code can be found in "models/linear_model.py" as class called "DuelingQNetwork". With the exception of the output layer, each fully connected layer was also passed through a relu-function.

As hyper-parameters for the "DuelingQNetwork" I have chosen replay buffer size to be 10000, minibatch size to be 64, gamma 0.99, tau 0.001, learning rate 0.0005. The model was updated every 4 steps done by the agent. Epochs were defined to be of size 100 episodes. I also implemented an

early out policy for the training. If the agent failed to improve for 3 consecutive epochs, the training was terminated and the weights of the last saved checkpoints were chosen to be the ones for saving for the model.

Plot of Rewards

In Figure 2 I show that my agent is able to outperform the required score of 13 on average for 100 episodes. The agent was trained for 1600 episodes. The weights can be found either in the project root folder under "solution_model.pth" or in "saved_models/dueling_net_avg16_24.pth".

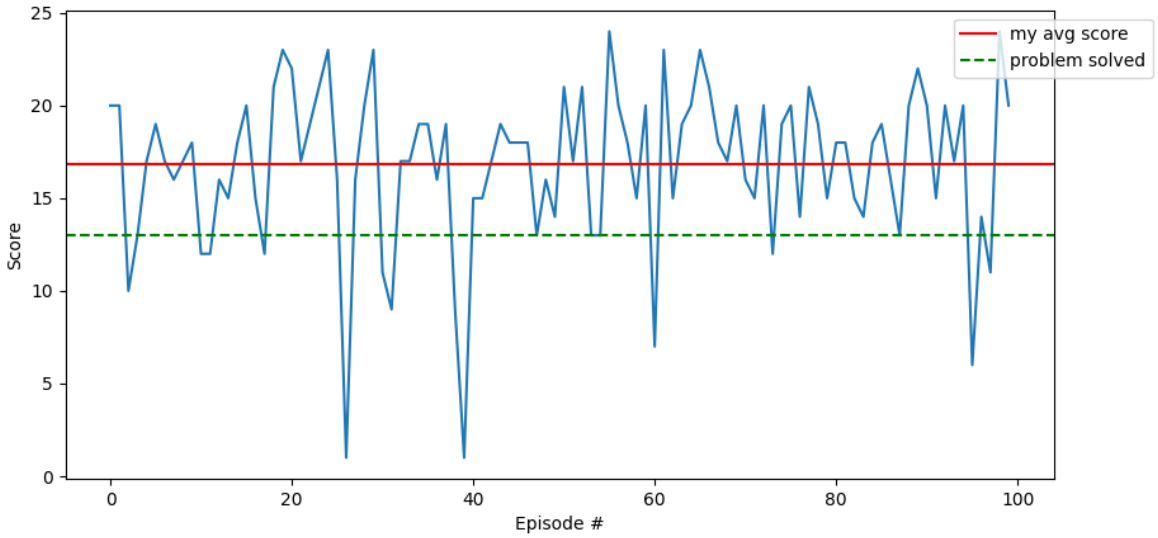

Figure 2: Plot of the average score of the DuelingQNetwork model. The average score lies at 16.85.

Ideas for Future Work

As future work one could also implement the double DQN[2], the prioritized experience replay[3] and also the rainbow paper[4].