Deep Learning laboratory Report

Assignment 5

Mohamed AbouHussein
Louay Abdelgawad

Department of Machine Learning
University of Freiburg
November 2016

Summary

This report is related to the work of implementing and comparing a deep Q-learning DQN neural network, a double Q-learning network DDQN, Duel Q-Learning network DUEL DQN and A duel double Q-learning network DUEL DDQN. The implementation is done using `tensorflow as tf` and `keras` package for motion planning. The report is divided into 5 sections as follows: section 1 represent a simple explanation of the deep Q-learning network implementation for solving the problem, section 2 represent a simple explanation of the deep Q-learning network implementation, section 3 represents the several approaches we investigate within this report, section 4 represents the selection performance metric, section 5 represents the results and discussion.

Report

1 Convolutional Neural Network Implementation

In this algorithm we do not load any data. This is because in deep Q-learning networks, the agent/solver react through exploring the space and getting rewards as explained later. In the `train_agent.py` the `Simulator` and the `TransitionTable`. The functions of both objects will be used later. We begin by initializing the `model` using `keras` package that acts over `tensorflow`. The functions we utilized from `keras` are `Convolutional2D`, `Activation`, `MaxPooling2D`, `Flatten`, `Dense` and `Dropout`. After that, we initialize the number of learning steps to 1 million iterations. The algorithm begins by randomly defining actions and updating the `TransitionTable`. There's two variables, `explore` and `observe` that define when to generate random actions, when to generate action through the network and when to train. through time, the algorithm reduce the rate of random actions and increase the rate of network actions and starts iteration after a certain amount of population the `TransitionTable`. Afterwards, a `batch` of 32 samples are selected from the `TransitionTable`. Q-values from the network are generated from the network for the batch. From the `TransitionTable` we retrieve `state_batch`, `action_batch`, `next_state_batch`, `reward_batch` and `terminal_batch`. We compute the `next_batch_actions`. The loss function is computed and feed to the optimizer based on the mean square error function

$$L = \frac{1}{2}[r + max_{action\_next}Q(state\_next, action_next) - Q(state, action)]^2 \quad (1)$$

The exact CNN architecture will be illustrated in the next section.

2 Designing Deep Q-learning Networks

For the design of the Deep Q-learning network, the procedure utilized was to create a single convolutional layer at a time. The layer parameters was randomly tried out from a range of values. The best estimated random values was the one implemented. Afterwards, another layer is added and the same procedure is tried out. In addition, the hypermaraters is also set in similar manner. The previous procedure has resulted in the following: The first conv layer included 32 `Covolutional2D()` filters, 8x8 `filter_size`. The activation function for the first layer was `relu`. Another `Convolutional2D()` layer is added that has the same specs and activation function but of size 64 instead, after that a third one

was added with 64 filters and 3x3 `filter_size`. Finally, the data is `Flatten()`ed and then it was reduced to 32 units using `Dense()`, afterwards it was reduced to the number of actions. Moreover, `relu` function is used on the output layer and Mean squared error `mse` is selected using Adam optimizer as the optimization technique for loss correction. The `batch_size` selected was 32. The results of the DQN are discussed in section 3.

3 Proposed Approaches

3.1 Double DQN

The DQN method is known to overestimate the results, and this is shown in many paper. One of the solutions to this problem was to use the Double DQN (DDQN). The max operator in standard Q-learning and DQN uses the same values both to select and to evaluate an action which leads to overestimating the Q-values. That's why Double DQN was proposed to solve such problem by decoupling the selection of the maximum Q value from the value it self. In other words, the same Q-function is used as that in the DQN, however, in the DDQN, the position of the maximum value is taken from the first network which is the network with the Q values, and then this position is used to get the corresponding value of it in the Target Network. Moreover, the Target Network is updated as before after a certain fixed number of steps.

3.2 Dueling DQN

This section presents the Dueling architecture for the deep Q-Learning network. The concept of dueling is motivated by the fact that for several states, the estimated action won't make much difference. Therefore avoiding overestimation of such states would be useful. The architecture of the network is similar in the beginning to the normal Q-learning network of pooling and filtering for feature extraction. The difference is that this approach splits the network before the output layer into two streams. One value stream and one advantage stream. The final layer combines the output of both streams in a single output layer of n outputs for n number of actions. The architecture is be depicted in figure 1.

The advantage stream is formally defined as follows:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2}$$

The value function V measures the importance of being in a particular state s. The Q-function measures the importance about the value of choosing each possible action when in this state. The advantage function, subtracts the value of the state from the Q-function to obtain a relative measure of the importance of each action. Thereby avoiding overestimation of actions.

3.3 Dueling Double DQN

The Dueling Double DQN approach is simply combining both the Duel Q-learning and the Double Q-learning architectures presented in the previous sections.

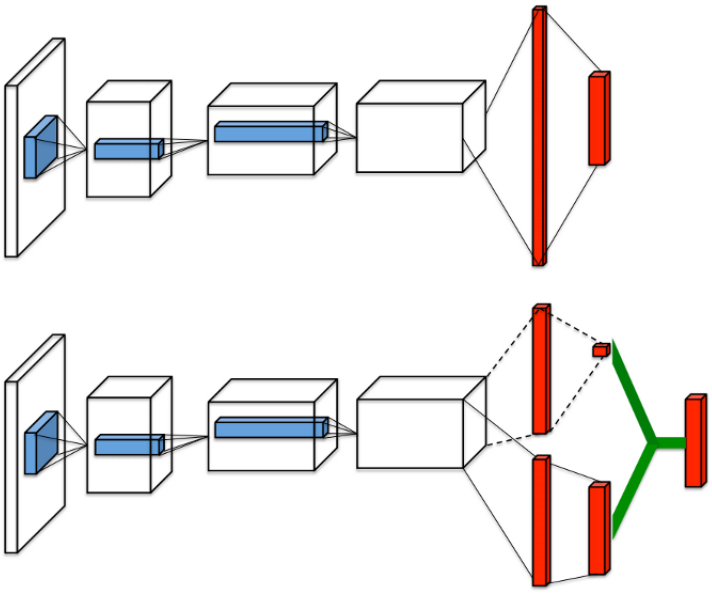


Figure 1: Duel Q-learning network architecture.

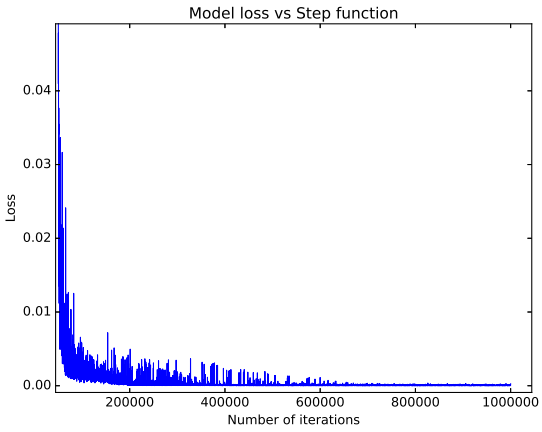
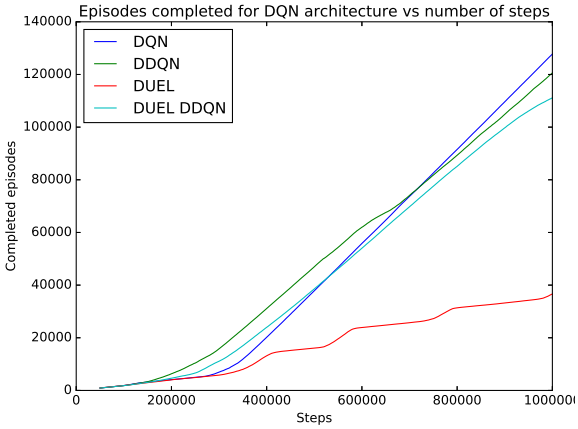


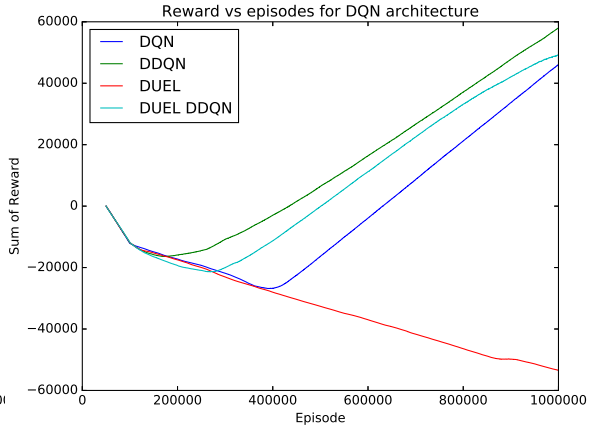
Figure 2: Loss of other approaches

4 Comparison Metrics

Usually, different approaches are compared to each other using loss vs steps, or the reward (highest score). For the loss vs steps as seen in figure 2, the result is not descriptive enough as all of the methods above yield approximately the same graph except for the Dueling network which yields a different result, and the reason behind that is explained in 5. Moreover, the loss vs step method isn't informative enough as it doesn't actually show



(a) Completed Episodes vs Steps



(b) Sum_Reward vs steps

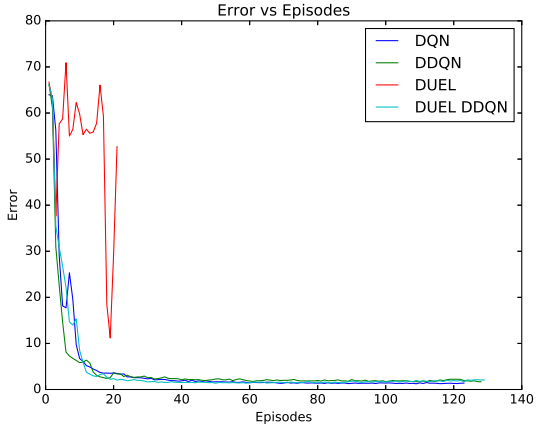


Figure 4: Error vs Episode

the performance of each method, it just shows whether the loss of the network converges or increases. In addition, the reward in our game is as well not good enough as it has a maximum value of 1 and minimum value of -75, so if the game converged, the score of all the methods will always just slightly below 1 depending on the random initial position. That's why we tried three other methods for comparing the different approaches. Firstly, we plotted the number of successfully completed episodes vs the training step, to show how many episodes were completed on the same number of steps during training, which can be seen in figure 3a. Secondly, we tried showing the sum of all rewards since the start of training till the end, where it can be seen that it starts with a negative value which means it doesn't reach the terminal, or reaches in many steps, however after a while the sum of reward starts to increase which means it gains more positive rewards. This can be seen in figure 3b. Lastly, we used a model which was trained before for 2 million steps

as a reference, to get the difference between the optimal (trained model) and the model we are trying to train. This method is the same as using the Manhattan distance to get the optimal. However, it was not suitable for our game/maze since there are walls/blocks that's why the trained model was used. Moreover, the results of this method is shown in figure 4, where the error is the difference between the path taken by the bot and the optimal path.

5 Results and Performance Evaluation

This section represents the results for different architectures. The results show that DDQN is the best architecture for our domain/game. This is because it converged before the other architectures. This can be depicted in figure 3a. The DUEL DDQN architecture is the second best architecture, The normal DQN is third while the worst is the DUEL network. The closeness of results for the first three ranked architectures was expected due to similar loss convergence results. As for the Duel architecture, the low performance was due to the non-convergence of the loss function. Figure 3b shows `Sum_Reward` vs steps. The decrease in the values at the beginning is due to the fact that at first, random exploration of the space causes negative reward sum values. After a while, the loss converges and the Q-values starts mimicking the true optimal values. Thereby, the sum of rewards starts to increase due to reaching the goal faster and adding the reward of +1. Figure 3b supports the results previously mentioned showing the `Sum_Reward` vs steps scored higher sum rewards for the DDQN, DUEL DDQN and DQN respectively. The DUEL network also did not not converge as the loss value did not converge as well. Our investigation in why DUEL network performs poorly is that DUEL network architecture is suitable for games which have states that taking no action or taking any action is indeed favorable. This is not the case with our problem as standing still is not favorable and taking an action is a must for solving the problem. Another reason is that DUEL architecture was mainly used for games or problems which has time as a variable. In our case, the problem is not affected by time. An example of a game which is affected by time is Atari game Enduro or Breakout, where the value of staying in the same position will eventually decrease or increase with time, as the states continually change, unlike our domain/game, where staying in position for infinite time doesn't lead to any positive or negative rewards as the environment is constant and not dynamic.

To sum up, choosing the right network architecture is dependant on the game parameters.