Solving CartPole-v1 with Deep Q-Learning Methods

Mohamad Nashouqu

**Abstract**

In this paper, we provide the details of implementing various reinforcement learning (RL) algorithms for controlling a Cart-Pole system. In particular, we describe various RL concepts such as Q-learning, Deep Q Networks (DQN), Double DQN, Dueling networks,. In the process, the readers will be introduced to OpenAI/Gym and Keras utilities used for implementing the above concepts. It is observed that DQN provides best performance among all other architectures being able to solve the problem within 150 episodes.

**The System:**

We use OpenAI Gym [16] to simulate the Cart-Pole system. Few snapshots of Cart-Pole states are shown in Figure 1. The left image shows the balanced state while the right image shows an imbalanced state. It consists of a cart (shown in black color) and a vertical bar attached to the cart using passive pivot joint. The cart can move left or right. The problem is to prevent the vertical bar from falling by moving the car left or right. One can see the animation of system behaviour under random action policy by executing the code given in Listing 1. The state vector for this system x is a four dimensional vector having components {x, x, θ, ˙ ˙θ}. The action has two states: left (0) and right (1). The episode terminates if (1) the pole angle is more than ±12◦ from the vertical axis, or (2) the cart position is more than ±2.4 cm from the centre, or (3) the episode length is greater than 200. The agent receives a reward of 1 for every step taken including the termination step. The problem is considered solved, if the average reward is greater than or equal to 195 over 100 consecutive episodes.

**Deep Q Network (DQN) Algorithm:**

Q-learning algorithm suffers from the Curse-of Dimensionality problem as it requires discrete states to form the Q-table. The computational complexity of Q-learning increases exponentially with increasing dimension of the state and action vector. Deep Q learning solves this problem by approximating the Q-value function Q(s, a) with an artificial neural network. This is achieved by the function build model() that uses Keras APIs to build a deep Q-network as shown below:

    def build\_model(self):

        model = keras.Sequential() #linear stack of layers https://keras.io/models/sequential/

        model.add(keras.layers.Dense(24, input\_dim=self.nS, activation='relu')) #[Input] -> Layer 1

        #   Dense: Densely connected layer https://keras.io/layers/core/

        #   24: Number of neurons

        #   input\_dim: Number of input variables

        #   activation: Rectified Linear Unit (relu) ranges >= 0

        model.add(keras.layers.Dense(24, activation='relu')) #Layer 2 -> 3

        model.add(keras.layers.Dense(self.nA, activation='linear')) #Layer 3 -> [output]

        #   Size has to match the output (different actions)

        #   Linear activation on the last layer

        model.compile(loss='mean\_squared\_error', #Loss function: Mean Squared Error

                      optimizer=keras.optimizers.Adam(lr=self.alpha)) #Optimaizer: Adam (Feel free to check other options)

        return model

    def save\_model(self):

**Listing 1: Creating a DQN using Keras APIs**

feed-forward network with 4 inputs, 2 outputs and two hidden layers each having 24 nodes. Hidden nodes use a RELU activation function while the output layer nodes use a linear activation function. Having a deep network to estimate Q values allows us to work directly with continuous state and action values. The Q network needs to be trained to estimate Q-values for a given state and action pair. This is done by using the following loss function:

Li(θi) = E(s,a)∼P (s,a) [Q ∗ (s, a) − Q(s, a; θi)]2

where the target Q value Q∗ (s, a) for each iteration is given by:

Q ∗ (s, a) = Es 0∈S[R(s, a) + γ max a0 Q(s 0 , a0 ; θi−1)|s, a] (4)

where R(s, a) is the reward for the current state-action pair (s, a) obtained from the environment and Q(s 0 , a0 , θi−1) is the Q-value for the next state obtained using the Q-network weights from the last iteration. This is implemented using the code provided in the code listing 6. It also shows the code for computing Q targets for DDQN architecture which will be explained later in this paper.

**Double DQN:**

Taking the maximum of estimated Q value as the target value for training a DQN as per equation 4 may introduce a maximization bias in learning. Since Q learning involves bootstrapping, i.e., learning estimates from estimates, such overestimation may become problematic over time. This can be solved by using double Q learning [8] [18] which uses two Q-value estimators, each of which is used to update the other. In this paper, we implement the version proposed in [18] that uses two models Q and Q0 sharing weights at regular intervals. The network Q0 is used for action selection while the network Q is used for action evaluation. That is, the target value for network training is obtained by using the following equation:

Q ∗ (s, a) ≈ rt + γQ(st+1, arg 0 max a Q 0 (st, at))

We minimize the error between Q and Q∗ , but have Q0 slowly copy the parameters of Q through Polyak averaging: θ 0 = τθ + (1 − τ )θ 0 . The code for computing target Q value and weight update is shown in code listings 6 and 7 respectively where the ddqn flag needs to be set to true.

**Dueling DQN:**

The Q-value Q(s, a) tells us how good it is to take an action a being at state s. This Q-value can be decomposed as the sum of V (s), the value of being at that state, and A(s, a), the advantage of taking that action at the state (from all other possible actions). Mathematically, we can write this as:

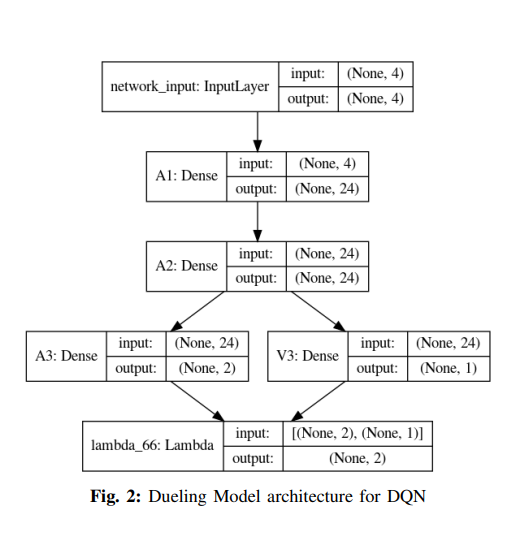
Q(s, a) = V (s) + A(s, a) ----- > (6)

Dueling DQN uses two separate estimators for these two components which are then combined together through a special aggregation layer to get an estimate of Q(s, a). By decoupling the estimation, intuitively the Dueling DQN can learn which states are (or are not) valuable without having to learn the effect of each action at each state. This is particularly useful for states where actions do not affect the environment in a meaningful way. In these cases, it is unnecessary to evaluate each action for such states and could be skipped to speed up the learning process. Rather than directly adding individual components as shown in (6), the q-value estimate can be obtained by using the following two forms of aggregation:

Q(s, a) = V (s, β) + A(s, a, α) − max a0 A(s, a0 , α) -------------- (7)

Q(s, a) = V (s, β) + A(s, a, α) − 1 |A| X a0 A(s, a0 , α) -------------- (8)

where β and α are the weights for the networks V (s) and A(s, a) respectively. The first equation (7) uses max advantage value and the second equation (8) uses the average advantage value to estimate Q(s, a) from V (s). This form of aggregation apparently solves the issue of identifiability, that is - given Q(s, a), it is difficult to find A(s, a) and V (s). The implementation of Dueling DQN architecture involves replacing the build model() function provided in Code Listing 5 with the function provided in the listing 9. A blockdiagram visualization of the dueling architecture is shown in Figure 2. It uses Keras’ Lambda function utility to implement the final aggregation layer.



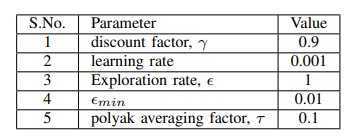


TABLE I: Values of user-defined parameters used for simulation

**EXPERIMENTS AND RESULTS:**

This section provides the details of experiments carried out to evaluate the performance of various reinforcement learning models described in the previous sections. This is described next in the following subsections.

**Software and Hardware Setup:**

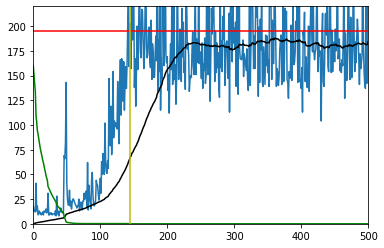
The complete implementation code for this paper is available on GitHub [12]. The program is written using Python and Keras APIs [7]. It takes about a couple of hours (2- 3 hours) for running about 1000 episodes on a HP Omen laptop with a Nvidia GeForce RTX 2060 GPU card with 6 GB of video ram. It is also possible to make use of freely available GPU cloud such as Google Colab [6] [2] or Kaggle [10] if you don’t own a GPU machine.

**Performance of various RL models:**

The performance of DQN Algorithm with experience replay is shown in Figure 4.

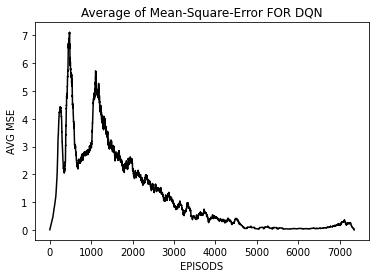
It found to solve the problem with 200 episodes. The performance comparison for DQN, Double DQN (DDQN) and DDQN with Polyak Averaging (PA) is shown in Figure 5. While all of them are able to solve the problem within 300 episodes, DQN is clearly the fastest. DDQN and DDQN-PA do not provide any perceptible advantage over DQN. This could be because the problem is itself too simple and does not require these complex architectures. The replay memory size of 2000 and batch size of 24 is used for producing the result shown in 5. Polyak Averaging (PA) tends to slow down the learning process and it is more commonly known as the soft method for updating target model. Similarly, the dueling versions of DQN or DDQN architectures fail to provide convergence within 300 episodes as shown in Figure 6. The problem might be too simple to make use of these complex architectures. Dueling architectures with Prioritized Experience Replay (PER) has been shown to provide remarkable improvement in ATARI games. It can be seen that Dueling-DQN is faster than Dueling-DDQN as it uses less number of parameters. The performance of DQN algorithms is also affected by changing the values of parameters such as the replay memory size (MS) and batch size (BS) selected for experience replay.

DQN Performance Diagrams:



Here is a graph of the results. If everything was done correctly you should see the rewards over the red line.

Black: This is the 100 episode rolling average  
Red: This is the "solved" line at 195  
Blue: This is the reward for each episode  
Green: This is the value of epsilon scaled by 200  
Yellow: This is where the tests started.

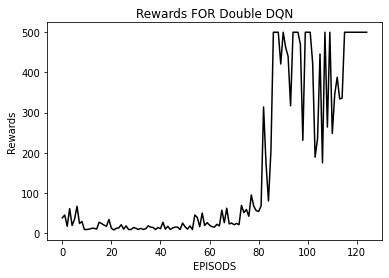
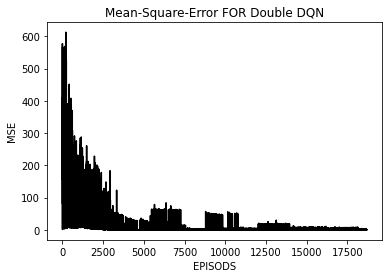
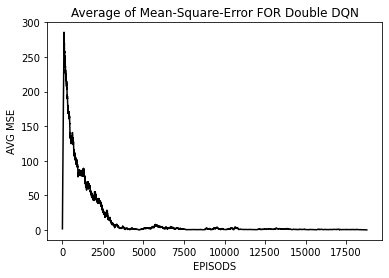
 A picture containing text, candelabrum, chime

Description automatically generated

A picture containing text, chime

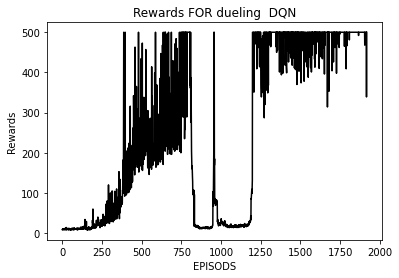
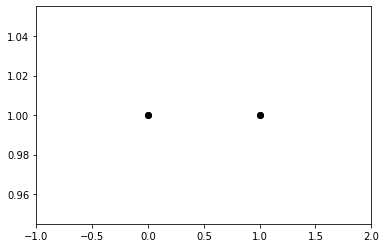
Description automatically generated

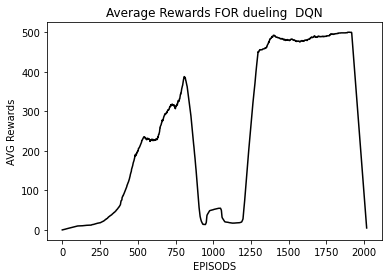
Double DQN performance Diagrams:



DQN needs 120 episodes to reach the solution.

Dueling DQN Performance Diagrams:





**Actor Critic Performance Diagrams:**

Chart, bar chart

Description automatically generated

Chart, line chart

Description automatically generatedChart, bar chart

Description automatically generatedShape

Description automatically generated with low confidence

**CONCLUSIONS:**

Performance Comparison : while all agent trying to solve the same problem with the same environment the number of episodes and time used to compare the performance

|  |  |  |
| --- | --- | --- |
| Agent | Time | Number of episodes |
| DQN | 32 min | 103 |
| Double DQN | 40 min | 124 |
| Dueling DQN | 50 min | 1910 |
| actor critic methods | 4.5 min | 1197 |

The best performance according to number of episodes is DQN but comparing via time the actor critic methods is the best with 4.5 min the big difference is not related to the agent and mathods when the actor critic methods should take a lot of time

In this report we provide implementation details of a few reinforcement learning algorithms used for solving the Cart-Pole problem. The implementation code is written in Python and makes use of OpenAI/Gym simulation framework and Keras deep learning tools. DDQN and Dueling architectures do not provide any significant improvement over DQN as the problem is too simple to warrant such complex architectures. The codes provided could be executed on Google Colab which provides free access to a GPU cloud. We believe that these details will be of interest to students and novice practitioners and will motivate them to explore further and make novel contributions to this field.

Overall, there is a slightly higher performance of the Double Q network concerning the Dueling Q network. However, the performance difference is relatively small and may vary due to random network initialization. As such, in a reasonably simple environment like the CartPole environment, the benefits of the Dueling Q network over the Double Q network may not be realized. However, in more complex environments like Atari, the Dueling Q architecture will likely be superior to Double Q (the original Dueling Q paper has shown). So, in future tutorials, we will demonstrate the Dueling DQN architecture in Atari or other more complex environments.

**REFERENCES:**

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6):26–38, 2017.

[2] E. Bisong. Google colaboratory. In Building Machine Learning and Deep Learning Models on Google Cloud Platform, pages 59–64. Springer, 2019.

[3] S. Borowiec. Alphago seals 4-1 victory over go grandmaster lee sedol. The Guardian, 15, 2016.

[4] J. Fang, H. Su, and Y. Xiao. Will artificial intelligence surpass human intelligence? Available at SSRN 3173876, 2018.

[5] D. A. Ferrucci. Introduction to this is watson. IBM Journal of Research and Development, 56(3.4):1–1, 2012.

[6] Google Colaboratory. Online gpu cloud by google. https:// colab.research.google.com/.

[7] A. Gulli and S. Pal. Deep learning with Keras. Packt Publishing Ltd, 2017.

[8] H. V. Hasselt. Double q-learning. In Advances in neural information processing systems, pages 2613–2621, 2010.

[9] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, and J. Wang. Overview on deepmind and its alphago zero ai. In Proceedings of the 2018 international conference on big data and education, pages 67–71, 2018.

[10] Kaggle. Online gpu cloud with datasets. https://www.kaggle. com/.

[11] P. Kraikivski. Seeding the singularity for ai. arXiv preprint arXiv:1908.01766, 2019.

[12] S. Kumar. Reinforcement learning code for cartpole system. https: //github.com/swagatk/RL-Projects-SK.git, 2020.

[13] Y. Li. Reinforcement learning applications. arXiv preprint arXiv:1908.06973, 2019.

[14] J. Markoff. Computer wins on jeopardy!: trivial, its not. New York Times, 16, 2011.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[16] OpenAI Gym. Toolkit for developing and comparing reinforcement learning algorithms. <https://gym.openai.com/>.

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

[18] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In Thirtieth AAAI conference on artificial intelligence, 2016.

[19] C. J. Watkins and P. Dayan. Q-learning. Machine learning, 8(3- 4):279–292, 1992.