Reinforcement Learning on OpenAI Gym Taxi

Yuki Kitayama
Department of Statistics
Columbia University
New York, NY 10027
yk2797@columbia.edu

Tianchen Wang
Department of Statistics
Columbia University
New York, NY 10027
tw2665@columbia.edu

Liwei Zhang
Department of Statistics
Columbia University
New York, NY 10027
lz2655@columbia.edu

Yicheng Li
Department of Statistics
Columbia University
New York, NY 10027
yl4104@columbia.edu

Abstract

Our project is Deep Reinforcement Learning with TF-Agents and we tested multiple reinforcement learning algorithms on Taxi environment from OpenAI Gym. In order to achieve a better performance and learn the pros and cons of different algorithms, we have selected regular Q-learning, DQN agent, Double DQN agent and Soft Actor-Critic agent. Our experiments shows that Q-learning could achieve optimal policy within a reasonable timeframe. However other agents using neural networks are hard to converge to the optimal policy without certain tricks. Hence, we write python code from scratch without using tf-agent so as to figure out the reason why they dont work and try to improve their performance on Taxi environment. By trying different tricks in reinforcement learning, we conclude that, to solve Taxi environment with DQN and DDQN, we need to implement tricks like one hot encoding, accelerate learning and epsilon decay to achieve a desirable result.

1 Background

Taxi from OpenAI Gym is a task that a taxi, the agent, goes to the location of a passenger, pick it up, take it to the destination, and drop it off at the destination. The initial location of the taxi, the passenger, and the destination are randomly assigned in each episode. Each episode ends either when successful drop-off is conducted or when time steps reaches 200. State is one dimensional integer array. Each integer represents one state out of 500, which is calculated by 5 * 5 * 4 * 5, because 5 is height of the environment, 5 is width, 4 is destinations, and 5 is passengers location (4 waiting location and 1 inside the taxi). Action of the taxi has 6 dimensions; go east, west, south, north, pick-up, and drop-off. In each time step, the agent collects -1 reward. If the agent has the passenger inside the taxi and conducts drop-off action at the destination, then it collects +20 reward. However, it obtains -10 reward if the agent conducts pick-up action when a passenger is not at the location, or if the agents conducts drop-off action at the wrong location. Maximum reward under the optimal policy is around 9. It varies depending on the initialized location of taxi, passenger, and destination.

With the size of 500 state and 6 action space, we performed Q learning before implementing tf-agents deep Q network in order to get the sense of reinforcement learning. Many empirical results suggest that the Q table in taxi environment is simple enough to obtain optimal policy without using DQN approaches. Our team consider that Taxi setting of finding the optimal path in some geographical

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

area and solve some tasks inside the field is applicable for many other problem settings. Thus, we still moved on to develop DQN in this environment. As the following results show, we found that even though the environment is simple enough to solve with Q learning, it does not necessarily mean that it is also easy to solve with using DQN.

We understand that Project 2 requires us to use TF-Agents. However, the most of the results in this project is obtained from the python codes that we developed, not using TF-Agents. Those python codes are based on the code we discussed in class, and we expanded it with using the current high level language in TensorFlow/keras. First, although we tried many different architectures in neural network in DQN and DDQN, we could not achieve the optimal reward. Second, many small technical tricks which are described in reinforcement learning papers are not implement in the current TF-Agents. For example, there is not epsilon decay, which is discussed in Method section in DQN agent. The paper explains that Soft Actor-Critic is for both discrete and continuous action spaces. However, the current SAC in TF-Agents does not support discrete action space. Thirdly, tf-agents is hard to debug.

The agents that we have implemented are Deep Q Network and Double Deep Q Network. Within online network and target network paradigm in DDQN, the updates of weights of target network was conducted by soft target network update which is introduced in DDPG paper Continuous control with deep reinforcement learning. We added many tricks to main idea of how to update q values of DQN and DDQN. Some are described in papers, but some are we believe only presented in blogs.

2 Method

2.1 Q-learning

It is a way to sequentially update q values in each cell in the q table[14]. The optimal policy is the action which has the highest estimated q values in each state. Usually it sets up the epsilon greedy algorithm in training and the q values are calculated the following type of weighted average formula.

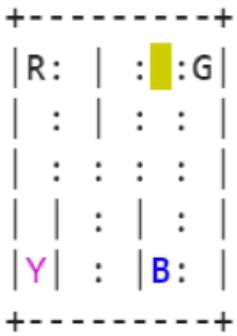
$Q^* = (1-\alpha)Q_{current} + \alpha\left(R + \gamma \max Q_{next}\right)$
$Q^* : The\ estimated\ q\ values$
$\alpha : The\ weight\ to\ balance\ between\ current\ q\ values\ and\ estimated\ q\ values\ from\ the\ current\ action$
$\gamma : Reward$
$\gamma : Discount\ factor$

The advantage of this method is that it does not need models. However, the disadvantage is that if the state is complex, it cannot estimate all the q values. We confirmed that the optimal policy in Taxi environment can be estimated by Q-learning.

2.2 Deep Q Network

This methodology uses neural network model with input state and output q values in each action, which is trained in each time step in episodes[5]. To prepare the training data for the model, we prepare experience replay. It is a buffer containing a tuple of state, action, reward, and next state. We randomly sample from this buffer as batch. This batch is used for training data of neural network. However, for the batch of target variable, DQN uses the model to predict q values, put it into Bellman equation, and this outcome is used for target variable. Hence, training data uses the model prediction so that this is the background of introduction of Double Deep Q Network. In the architecture of neural network, since Taxi of OpenAI Gym only has one dimensional index as state information, we did not implement convolutional layers, as it is popular to solve Atari games with DQN. However, we consider that if we feed in the image (just like the below image) of the Taxi environment as input of the neural network, then it allows us to have convolutional layer, although we did not implement it.

We confirmed that only implementing DQN does not provide the optimal policy in taxi environment. It needs to implement the following tricks.



2.3 Double Deep Q Network

In DDQN, we have two neural networks along reinforcement learning process[3]. One is called online network, and the other is called target network. Online network is used to select action for target network. Target network calculates the q values of the action that online network specifies. Bellman equation is applied for this q values. Q values are predicted by online model, but the q value of the current action is updated the output of Bellman equation. We put the current state as x, and the updated q values as y, and train online model. To updates the weights of the target network, we used soft target network update method which is discussed in the following section. The advantage is that the training becomes more stable than DQN, and it provides a better performance. However, the disadvantage is, we consider, that in each time step, DDQN needs to updates two models so that iterating many episodes is not practical. We confirmed that only applying main frame of DDQN is not sufficient to solve taxi environment. We need additional tricks. The difference of DQN and DDQN appears in Bellman equation, as following.

$DQN : value = reward + discount, factor * max(target, network..predict(next_state))$
$DDQN : value = reward + discount - factor*$
$target, network.predict(nextstate)[argmax(online - network.predict(next_state))]$

Theses values are used for target variable in training neural network. DQN training data is the output own model predicts. However, in DDQN, training of online model is not base on own predictions. It uses the outputs of target network so that it is separate.

2.4 Soft target network update

The idea is introduced in Continuous control with deep reinforcement learning paper[4]. It updates weights of the target network by a weighted average of target network weights and online network weights with hyperparameter tau. So it does not apply the usual forward feeding and backpropagation updates for the weights. The weights of target network are updated by combination of the weights of online network and the weights of target network[9]. In practice, it sets a small value for tau so that learning becomes stable because the target weights are limited to change only slowly. However, it needs to trade-off between the speed of learning and divergence of the critic.

2.5 Epsilon decay

Tf-agents fix epsilon in the epsilon greedy algorithm. However, we implement the method to decay epsilon through the episodes. The learning starts with 1.0 epsilon and decay to 0.1. Once it hits 0.1, fixed at 0.1 for the rest of the episodes. We adopted this method because we learned this from Human-level control through deep reinforcement learning paper[1]. It produces the result that our network trains the weights, but it still explores random states because epsilon is still high at the beginning of the series of the episodes.

2.6 One hot encoding of state representation

This index not only represents the location of the taxi, but also represents the states about whether or not the passenger in inside the car, and where the destination is. Without modification of state input, our model takes 1 dimensional input and return 6 dimensional outputs. However, one-hot encoding of state representation allows us to have the network which takes 500 dimensional inputs and returns 6 dimensional outputs. Hence, every time we draw a batch from experience replay, the state index is transformed into one-hot encoded vector. This implementation is observed in Ossenkopps blog[7]. By implementing our DQN and DDQN reinforcement learning code, we confirmed that one hot encoding enables us to obtain positive reward.

2.7 Skipping training

What we mean by skipping training is that in general, neural network models in DQN or DDQN are trained in each time step in an episode, but we set random value with a certain threshold, and skip to train models in some time steps. We could not find supports from paper. However, we contacted Ossenkopp and he mentions, not always training the model stabilizes the training by a bit[7]. We confirmed that his reinforcement learning has a small variability in total rewards over different episodes. However, when we implemented DQN without this skipping training, total rewards often come back to negative total rewards even after the agent started to make successive positive rewards.

3 Results

3.1 Q-learning

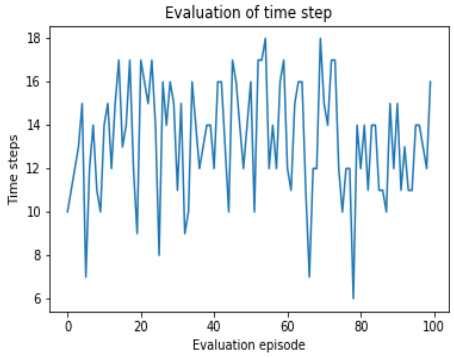
We performed Q-learning with the hyperparameters completed these episodes with around 1 minute with the use of Google Colab GPU. Q-learning provided 500 rows by 6 columns table. As the evaluation plots show, with the use of Q-learning, we can achieve positive results in Taxi environment.

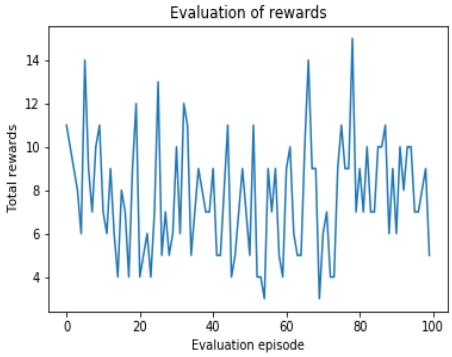
Table 1: Head of Q-table

State index	0	1	2	3	4	5	Action(argmax)
0	0.0	0.0	0.0	0.0	0.0	0.0	0
1	-2.418	-2.364	-2.418	-2.364	-2.273	-11.364	4
2	-1.87	-1.45	-1.87	-1.45	-0.75	-10.45	4
3	-2.364	-2.273	-2.364	-2.273	-2.122	-11.273	4
4	-2.496	-2.497	-2.496	-2.497	-10.426	-8.133	0

Table 2: Tail of Q-table

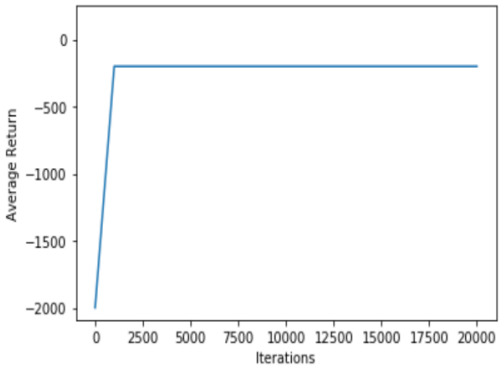
State index	0	1	2	3	4	5	Action (argmax)
495	0.0	0.0	0.0	0.0	0.0	0.0	0
496	-2.189	-2.122	-2.17	-2.122	-7.494	-6.731	1
497	-1.065	0.416	-1.022	-1.234	-4.642	-2.732	1
498	-2.158	-2.122	-2.198	-2.122	-7.595	-6.274	3
499	2.828	0.497	3.695	11.0	-2.369	-3.001	3

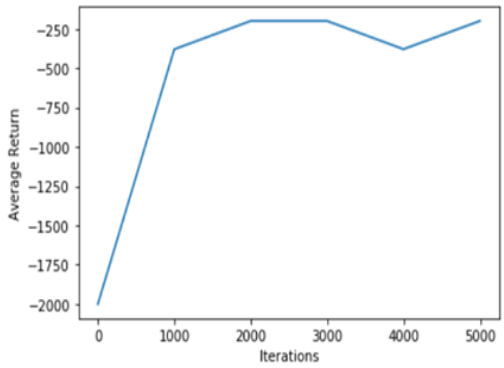




3.2 Tf-agents DQN and DDQN

We selected DQN agent and Double DQN agent from tf-agents with 1 densely connected layer with 100 nodes, 64 batch size, and 0.001 learning rate for Adam optimizer in neural network. The result is the following. In both DQN and DDQN, the average total reward is stuck in -200. It suggests that the agent keeps doing random actions, collect -1 reward per time step, does not complete successful drop-off of the passenger, and ends the episode after 200 time steps. However, it also suggests that the agent at least learned not to make an illegal pick-up or drop-off action because it gains -10 reward if it does so. We wanted to improve this, but tf-agents is hard to debug. Hence, we moved on to develop reinforcement learning code from scratch.





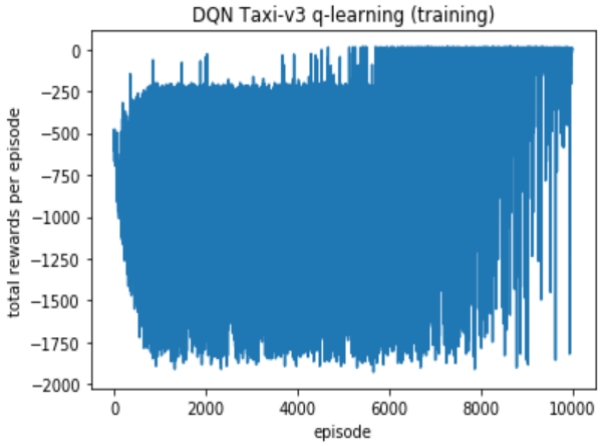
3.3 DQN with our code

We produced the following results with Deep Q Network. We used no hidden layer neural network with one hot encoding state space, using 0.00025 learning rate in Adam optimizer, 32 batch size, 0.95 gamma in Bellman equation, epsilon in epsilon-greedy algorithm starts from 1.0, multiplied by 0.9972 each episode, decay to 0.1 , fix at 0.1 afterwards. With random number, 30% of each time trains neural network.

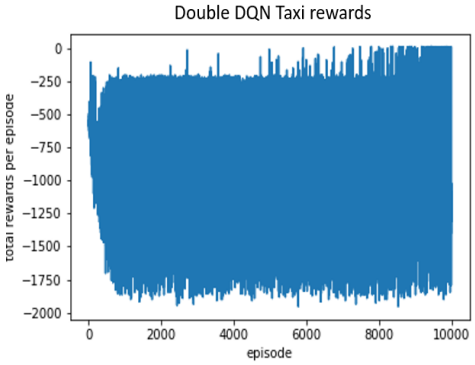
Before 5,000 episodes, most of the maximum rewards is stuck at -200. It means the agent learned not to make illegal action and avoid -10 reward. However, the agent did not succeed in successfully drop off the passenger to gain 20 reward at the end the episode. After 5,000 episodes, the agent improved this behavior, and started to gain positive rewards. However, our problem is, even after hitting positive desirable rewards, the agent still made mistake and produced huge negative rewards.

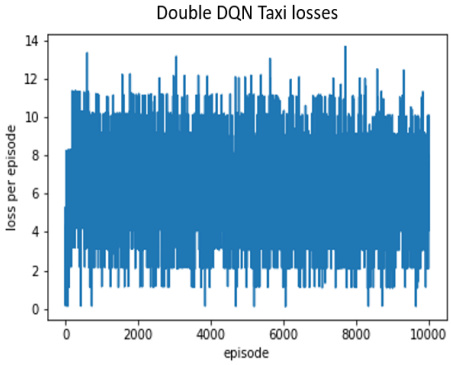
3.4 Double DQN with our code

We generated the following results of Double DQN. Online network and target network share the same architecture with no hidden layer and one hot encoding state input. Online network was trained with batch data from experience replay while the weights of target network was updated with soft target network update with tau 0.001. The other hyperparameters are shown in the appendix. The learning started to make positive rewards after 8,000 episodes. It is peculiar for us because getting



positive rewards is slower than DQN, and the variability of rewards over the episodes is much bigger than DQN. However, in paper Deep Reinforcement Learning with Double Q-learning, it was mentioned that Double DQN would result in better performance and stability. We assume that possibly our coding implementation is not accurate, or that hyperparameter tau in soft target network update is not appropriate to be chosen. Possibly, we could iterate more episodes to see the results. Additionally, the training suffered from the long time of training because it needs to update two neural networks. The trainings were conducted in Colab GPU. However, the training spent more than 10 hours, disconnected frequently, and difficult to produce desirable results this time.





4 Discussion

In this project, major approaches of reinforcement learning were performed towards Taxi environment from OpenAI Gym. We tested Q-learning, Deep Q Network, and Double Deep Q Network. We observed that Taxi environment is simple enough to obtain the optimal policy with Q-learning. We applied DQN and DDQN agents from tf-agents. However, we did not obtain the desirable results. It was not intuitive for us because we assumed that DQN and DDQN would be easier to get the optimal policy because model-free Q-learning got quite easily. We finally successfully achieved positive rewards after implementing several minor tricks in reinforcement learning, such as one hot encoding representation of state space, epsilon decay, and skipping training. However, the learning process needs to be improved to provide better performance and stability as we have seen in DQN and Double DQN results. Thus, we conclude that further attempts of deep reinforcement learning require not only building powerful neural network in the learning, but also implementing practical ideas to stabilize and improve the learning performance.

References

[1] Brockman, G. Cheung, V. Pettersson, L. Schneider, J Schulman, J Tang, J Zaremba, W (2016) *OpenAI Gym*.

[2] Greavs, J. (2017) *Understanding RL: The Bellman Equations*. Retrieved from https://joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations.

[3] Hasselt, V.H. & Guez, A. &Silver, D. (1995) *Deep Reinforcement Learning with Double Q-Learning*.

[4] Kansai, S. Martin, B. (2019) *Reinforcement Q-Learning from Scratch in Python with OpenAI Gym*. Retrieved from https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

[5] Lillicrap, T.P. & Hunt, J.J. (2016) *Continuous Control with Deep Reinforcement Learning*.

[6] Mnih, V. & Kavukcuoglu1, K. & Silver, D. Rusu, A.A. & Veness, J. & Bellemare, M. G. & Graves, A. 2015)$Human-levelcontrolthroughdeepreinforcementlearning$.

[7] Nguyen, T. T. & Nguyen, N. D Nahavandi, S (2019) *Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications*.

[8] Ossenkopp, P. (2018) *Reinforcement learning Part 2: Getting started with Deep Q-Networks*. Retrieved from https://www.novatec-gmbh.de/en/blog/deep-q-networks.

[9] Ouyang, J. (2018) *Reinforcement Learning Lane Change Algorithm (DDQN)*. Retrieved from https://blog.goodaudience.com/reinforcement-learning-lane-change-algorithm-ddqn-3d38dabfa087.

[10] Simmons, L. (2018) *Double DQN Implementation to Solve OpenAI Gyms CartPole v-0*. Retrieved from https://medium.com/@leosimmons/double-dqn-implementation-to-solve-openai-gyms-cartpole-v-0-df554cd0614d.

[11] Surma, G. (2018) *Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning)*. Retrieved from https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288.

[12] Wang, Z. Schaul, T. Hessel, M (2016) *Dueling Network Architectures for Deep Reinforcement Learning*.

[13] Weng, L. (2018) *Implementing Deep Reinforcement Learning Models with Tensorflow + OpenAI Gym*. Retrieved from https://lilianweng.github.io/lil-log/2018/05/05/implementing-deep-reinforcement-learning-models.html.

[14] Yoon, C. (2017) *Double Deep Q Networks*. Retrieved from https://towardsdatascience.com/double-deep-q-networks-905dd8325412.

Appendix

Table 3: Hyperparameters and Values of Q-learning

Hyperparameter	Value	Description
Alpha	0.1	The learning rate in Q-learning. The smaller it is, the more the current q value is influential
Gamma	0.6	The discount factor. The larger it is, the more the agent considers the future reward
Epsilon	0.1	The parameter in the epsilon greedy algorithm. It is it 0.1, 1 out of 10 episodes does not follow the estimated q value, and conducts random actions
Episode	100,000	The number of iterations

Table 4: Hyperparameters and Values of DQN

Model	Score	Agent
Learning rate	0.00025	The learning rate in Adam optimizer of neural network
Batch size	32	Training data size of neural network
Gamma	0.95	The discount factor in Bellman equation
Starting epsilon	1.0	The parameter in epsilon greedy algorithm
Epsilon decay multiplier	0.9972	In each episode, this number is multiplied to the current epsilon, and epsilon gradually becomes smaller over the episodes
Minimum epsilon	0.1	By epsilon decay multiplier, after epsilon hits this value, it fixes at this value for the following episodes
Time step range	200	Times steps within each episode
Input size	500	Input shape of neural network. 1 dimensional state index is converted to 500 dimensions by one hot encoding
Output size	6	Action size; east, west, south, north, dropoff, pickup
Episodes	5000	The number of iterations

Table 5: Hyperparameters and Values of DDQN

Hyperparameter	Value	Description
Tau	0.001	Parameter in soft target network update. The smaller the value is, the slower the update of target network is, because tau is multiplied with the weights of the online network
Learning rate	0.00025	The learning rate in Adam optimizer of neural network
Batch size	16	Training data size of neural network
Gamma	0.99	The discount factor in Bellman equation
Starting epsilon	1.0	The parameter in epsilon greedy algorithm
Epsilon decay multiplier	0.9972	In each episode, this number is multiplied to the current epsilon, and epsilon gradually becomes smaller over the episodes
Minimum epsilon	0.1	By epsilon decay multiplier, after epsilon hits this value, it fixes at this value for the following episodes
Time step range	200	Times steps within each episode
Input size	500	Input shape of neural network. 1 dimensional state index is converted to 500 dimensions by one hot encoding
Output size	6	Action size; east, west, south, north, dropoff, pickup
Episodes	10000	The number of iterations

Table 6: Submission of notebooks

Method	Notebook
Q-learning	https://github.com/tw2665/AML-Project-/blob/master/Q$_{l}earning_{t}axi_{g}ym_{v}2.ipynb$
DQN tf-agents	https://github.com/tw2665/AML-Project-/blob/master/Deep$_{QN}etwork_{T}axi.ipynb$
Double DQN tf-agents	https://github.com/tw2665/AML-Project-/blob/master/DDQL.ipynb
DQN with one hot encoding skip	https://github.com/tw2665/AML-Project-/blob/master/Deep$_{QN}etwork_{T}axi_{g}ym_{t}f_{v}8.ipynb$
Double DQN with one hot encoding skip	https://github.com/tw2665/AML-Project-/blob/master/Double$_{D}QN_{T}axi_{g}ym_{T}ensorFlow_{v}3.ipynb$