# Stability Analysis of $Q$-Learning with Target Network and Truncation

**Pujith Kachana**
Georgia Institute of Technology
pkachana3@gatech.edu

## Abstract

Reinforcement Learning has seen many recent breakthroughs, especially with the rise of Deep Reinforcement Learning (DeepRL) promising to solve tangible problem by bridging the gap between theory and empirical success. The most notable of these breakthroughs can be observed in Game AI, where DeepRL (particularly, $Q$-Learning with function approximation, or QLFA) has been used to solve many classic arcade games (Mnih et al., 2015). Effective as these QLFA algorithms may be, they are far from perfect. Large training times and numerical instability still remain a problem, making QLFA algorithms expensive and unreliable. However, when paired with a target network and truncation, QLFA promises to alleviate this instability and provides convergence guarantees on $Q$-learning with function approximation. In this study, we will evaluate the performance of QLFA with target network and truncation in comparison to existing algorithms.

## 1 Overview

Reinforcement learning is a form of machine learning that uses a system of actions and rewards to learn the optimal behaviour in an environment. It uses an agent to explore the environment, and this agent may be rewarded or punished for certain interactions with the environment. This agent-environment interaction is abstracted to a state-space, in which the state of the agent describes configuration of the agent in the environment. The agent's actions allow it to transition between states, acquiring rewards or punishments. This system of states, actions, rewards, and transitions is referred to as the Markov Decision Process (MDP), which serves as the underlying model for reinforcement learning.

$Q$-learning is a particular policy within reinforcement learning which aims to learn an optimal policy by computing $Q$-values of all state/action pairs in an environment. A $Q$-value is an expected cumulative reward of performing a certain action in a particular state. Once $Q$-values are calculated, an optimal policy can be easily recovered by choosing the action of highest $Q$-value in every state. $Q$-values can be explicitly stored in a $Q$-table mapping a state/action pair to a $Q$-value, or they can be expressed through a $Q$-function which takes a state/action pair as input and returns a $Q$-value as an output. A $Q$-function is often used in practice as it can be used to represent a large and continuous state-space, and can be approximated by function approximators.

The focus of this study will be the stability of QLFA with a target network and truncation. Stability and finite-sample guarantees for this algorithm are presented in (Chen et al., 2022).

### 1.1 Deep $Q$-Learning

$Q$-learning is a model-free, off-policy learning algorithm that finds the optimal action in a given state. The algorithm allows the agent to explore the state space and uses the Bellman equation to compute a temporal difference, or $Q$-value, of a state-action pair with respect to the next state. The Bellman equation for $Q$-values is as follows:

$$Q(s_t, a_t) = E[R(s_{t+1}) + \gamma \max Q'(s', a')|s_t, a_t],$$

where $s$ is state, $a$ is the action, $t$ is the current time-step, $Q(s, a)$ is the corresponding $Q$-value, $R(s, a)$ is the corresponding reward, $Q'(s', a')$ is the $Q$-value of the next state, and $\gamma$ is the discount factor. This value is updated through the following update equation at every iteration of $Q$-learning.

$$Q(s, a) = Q(s, a) + \alpha[R(s, a) + \gamma \max Q'(s', a') - Q(s, a)],$$

where $\alpha$ is the learning rate.

$Q$-learning with function approximation is a more sample-efficient implementation of $Q$-learning that uses some form of internal approximator to predict $Q$-values in a large state-space. The particular model studied in this report will be a Deep $Q$-Network(DQN), a subset of $Q$-learning with function approximation that uses a Neural Network as the approximator (Mnih et al., 2015).

### 1.2 The Deadly Triad

The combination of temporal difference learning (often referred to as "bootstrapping"), off-policy learning, and function approximation results in the deadly triad problem, leading to possible instability in the underlying model (Sutton, 1999). Unfortunately, $Q$-learning with function approximation fits this criteria, making it possible for the model's $Q$-values to diverge. This is a serious problem considering that $Q$-learning with function approximation is a resource-heavy algorithm but may never converge, wasting time and computing power without ever producing a solution.

## 2 $Q$-Learning with Target Network and Truncation

Various optimizations have been proposed for $Q$-learning with function approximation to mitigate the deadly triad issue, each with varying levels of success. We will briefly discuss some relevant optimizations below, along with the QLFA algorithm.

### 2.1 Target Network

$Q$-learning uses the Bellman Update equation to compute $Q$-values using $maxQ'(s', a')$, the maximum $Q$-value from the next state. However, unlike in traditional $Q$-learning where only one $Q$-value is updated at a time, a DQN updates its neural network for every time-step. This may cause multiple $Q$-values to be changed at once, possibly updating both $Q(s, a)$ and $Q(s', a')$ in the same time-step, producing instability. A target network combats this issue by periodically storing a copy of the DQN's neural network and using it for the $Q(s', a')$ values, thus generating stable updates to $Q(s, a)$ (Mnih et al., 2015).

### 2.2 Truncation of Temporal Difference

A clear symptom of instability within a DQN is unfeasibly large or small $Q$-values. Taking $R^*$ to be the largest possible reward in any state of an environment, we can consider the maximum $Q$-value to be a sum of an infinite geometric series with $\gamma$ as the common ratio, yielding the expression $\frac{R^*}{1-\gamma}$ as the maximum $Q$-value (Chen et al., 2022). The original DQN implementation (Mnih et al., 2015) uses this maximum $Q$-value to threshold $R(s, a) + \gamma maxQ'(s', a') - Q(s, a)$, the temporal difference, to suppress instability of $Q$-values.

### 2.3 Truncation of Next $Q$-Value

The truncation modification employed in (Chen et al., 2022) truncates a different value from the Bellman equation. Whereas the original DQN implementation truncates the entire temporal difference, an alternative is to only truncate $Q'(s', a')$, the next $Q$-value. This is a more intuitive truncation considering that the truncation thresholds are set to the maximum and minimum possible $Q$-values. This truncation variant provides some stability and finite-sample guarantees further discussed in (Chen et al., 2022).

## 3 $Q$-Learning Stability Analysis

### 3.1 Environment

The environment used in this study is CartPole-v0 from OpenAI Gym. The objective of this environment is to balance a pole mounted on a cart, with the action space consisting of moving the cart left, moving the cart right, or remaining in place. The observation space consists of the cart's position, the cart's velocity, the pole's angle, and the pole's velocity at the tip. The CartPole episode terminates if the pole is more than 15 degrees from its vertical position, the cart moves more than 2.4 units from the environment's center, or if 200 time-steps have passed. A +1 reward is awarded for every time-step, making the maximum reward 200.

### 3.2 Timeline and Procedure

This study of the QLFA algorithm and its stability lasted roughly 10 weeks. The first two weeks were spent conducting a literature review, in which (Chen et al., 2022) and the original DQN paper (Mnih et al., 2015) were closely analyzed to gain an understanding of the QLFA algorithm, the deadly triad problem, target networks, truncation, and DQN's implementation. The next week was spent researching suitable environments for DQN testing to generate meaningful benchmarks for the numerous variants of the $Q$-learning algorithms. During this week, it was decided that CartPole would be an appropriate environment for this task given its relative simplicity and available resources. Various open-source CartPole DQN models were referenced to understand the code implementation behind this task. These first few weeks laid the foundation for the following experiments.

After the environment was decided, a suitable implementation of CartPole DQN was chosen from the collection of open-source implementations on the internet. The code can be accessed from the Appendix below. This particular implementation also includes a replay buffer to combat an issue known as catastrophic forgetting, where the models tends to forget previously known policies over time. This initially chosen DQN implementation was occasionally successful, but frequently suffered from long processing times and inconclusive trajectories. Two additional weeks were spent tuning the hyperparameters for the model for better performance. The tuning procedure can be found in the Hyperparameter Tuning section.

After the hyperparameter tuning period, it was determined that the chosen DQN implementation was not entirely suitable for the purpose of this study given its inconsistent results. It appeared that the model still suffered from catastrophic forgetting despite the use of a replay buffer, implying that the buffer was incorrectly implemented or that the buffer size or batch size were not properly tuned. Some time was spent researching other, possibly simpler environments for this study. The simple environments found were Acrobot and Gridworld. However, these environments were not much different from CartPole in terms of complexity and had far lesser open-source resources available. After further inspection, another promising implementation of CartPole was found, this time with pre-tuned hyperparameters and a consistent model. This model was used to conduct this study comparing the stability of $Q$-learning with no truncation, $Q$-learning with the truncation of the entire temporal difference (as used in the original DQN implementation), and $Q$-learning with the truncation of only the next $Q$-value (as proposed in the QLFA algorithm). The benchmarks for the comparison between each algorithm were generated by averaging the rewards over 3 trajectories of 600 episodes for each algorithm.

### 3.3 Hyperparameter Tuning

The hyperparameters tuned for the model were the neural network structure, learning rate, replay buffer size, replay batch sample size, target network synchronization frequency, and the epsilon update policy. The tuning process consisted of selecting reasonable values for each hyperparameter and testing the model in a grid search manner. Each hyperparameter was tuned one at a time with the other kept constant at their initial values. The value resulting in the best performance was chosen for each hyperparameter. The individual trajectories and chosen values are shown in the Appendix.

### 3.4 Results

The average rewards for three trajectories of regular DQN, DQN with truncation of entire temporal difference, and DQN with truncation of next $Q$-values over 600 episodes are shown below. A rolling average over 10 episodes was performed for visualization purposes (the sharp drop-off at the end of the graph is an artifact of this rolling average).

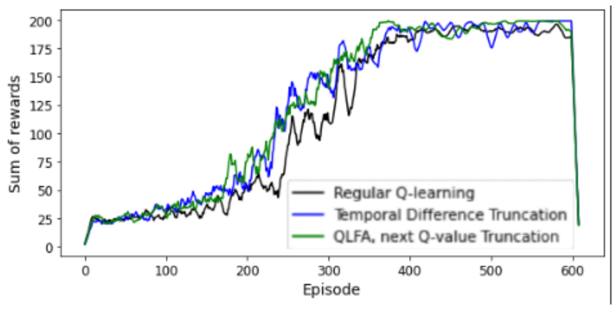

Figure 1: DQN Rewards Distribution

The results show that all three $Q$-learning variants converge or come very close to convergence in 600 episodes. However, there are noticeable differences in speed of learning and stability. Regular DQN is almost consistently lower in average reward value during almost all episodes compared to its truncated counterparts. Regular DQN also sees more fluctuation in its reward values, whereas the truncated DQN trajectories have a smoother approach to convergence, implying greater stability with truncation. DQN with next $Q$-value truncation, as suggested in (Chen et al., 2022), is the first trajectory to reach absolute convergence at around 525 episodes. This provides empirical evidence that the QLFA algorithm is more stable and sample efficient with a target network and truncation of the next $Q$-value than regular $Q$-learning with function approximation and the original DQN implementation, as guaranteed in (Chen et al., 2022).

## 4 Conclusion

$Q$-learning relies heavily on exploration of the state space, making it very sample-inefficient in nature. This inefficiency can be reduced through the use of a function approximator to help generalize sample observations to the broader sample space. However, $Q$-learning with function approximation carries the risk of instability rooted in the deadly triad problem, leading to a divergent learning process. We have shown that $Q$-learning with function approximation using a target network and truncation can alleviate this instability and drive the model to converge quicker. Future work directions include generating performance metrics on more complex environments and using different function approximators for $Q$-values.

## References

Chen, Z., Clarke, J. P., and Maguluri, S. T. (2022). Target Network and Truncation Overcome The Deadly triad in $Q$-Learning. *Preprint arXiv:2203.02628*.

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

Sutton, R. S. (1999). Open theoretical questions in reinforcement learning. In *European Conference on Computational Learning Theory*, pages 11–17. Springer.

# A Appendix

## A.1 Notes of Study

Notes:
https://docs.google.com/document/d/1qy7iyuZFVoLpC30MKOd3a6sEGD0bAF9Ip1Oniz-juP0/edit?usp=sharing

## A.2 CartPole DQN Code

Initial Code:
https://colab.research.google.com/drive/1sUtAeUxfRMEEeZ0ojl7FWR3nKKIh0q_a?usp=sharing
Final Code:
https://colab.research.google.com/github/ageron/handson-ml2/blob/master/18_reinforcement_learning.ipynb
#scrollTo=xVdxh1Rl2r_W

## A.3 Hyperparameter Values

Optimal Hyper-parameters, chosen independently:

- Gamma: 0.95
- NN structure: 64 neuron hidden layer
- LR:0.00025
- Exp Replay Size: 256
- Batch Sample Size:32
- Synchronize Frequency:10
- Epsilon Update: Multiplicative

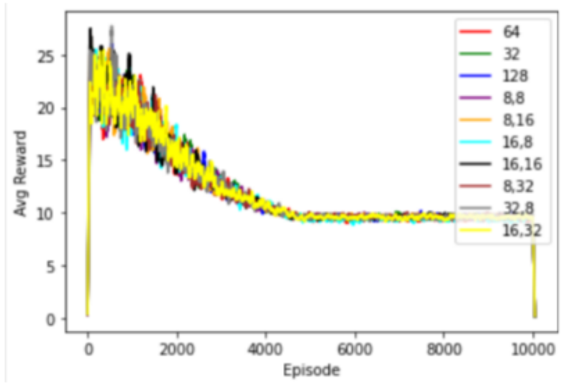

Figure 2: Tuning Structure of Neural Network

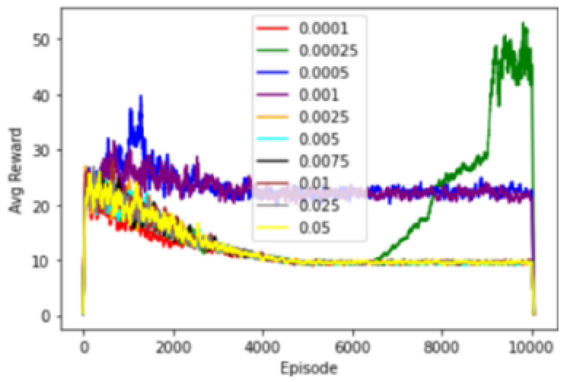

Figure 3: Tuning Learning Rate

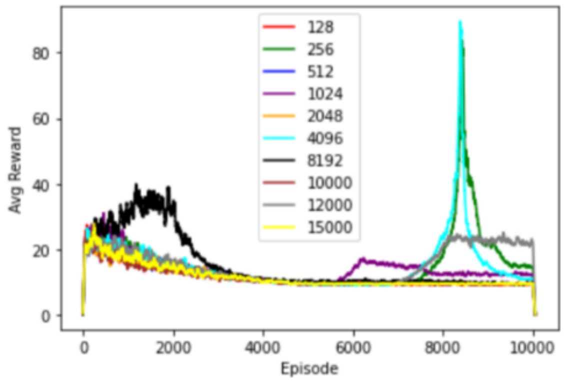

Figure 4: Tuning Replay Buffer Size

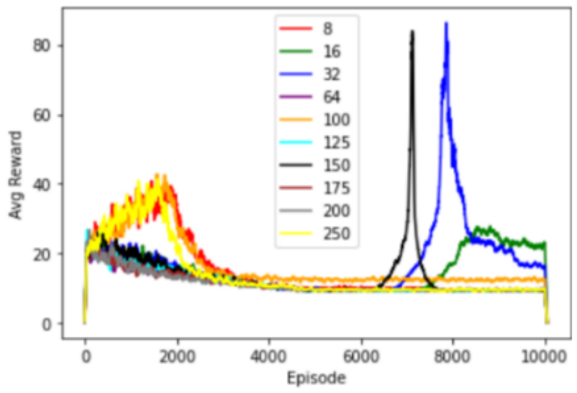

Figure 5: Tuning Replay Batch Sample Size

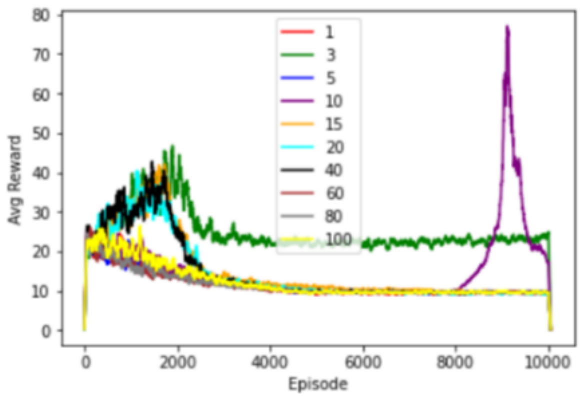

Figure 6: Tuning Target Network Synchronization Frequency

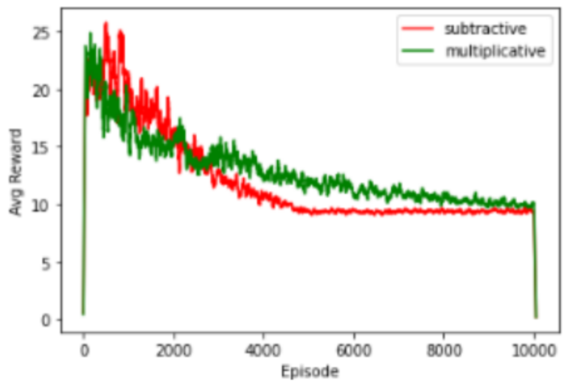

Figure 7: Tuning Epsilon Update Policy