CS 461: Artificial Intelligence

Progress Report

Exploring a New Deep RL Approach: QR-DQN

Ahmet Salman, 21901004

Ahmet Cemal Alicioglu, 21801700

Alp Uneri, 21802481

Beril Canbulan, 21602648

Mustafa Hakan Kara, 21703317

03/05/2022

Course Instructor: Özgür Salih Öğüz

Abstract

Objective of this paper is to implement, test, and compare the performance of state-of-the-art Deep RL algorithm with vanilla Q-learning. We will develop a Deep RL algorithm that falls under the category of Distributional RL, it is called QR-DQN. Normal Q-learning has been shown to have some instabilities due to the score meaning aspect of it, QR-DQN hopes to solve that issue, and promises better performance, by using the probability distribution rather than mean scores. If this algorithm proved to be robust and efficient, it could open new paths in the field of self-leanring and designing more "rational" agents

1.0 Introduction

At this point, it is fair to claim that our project has crossed several considerable milestones. Most of the major parts have been completed, only minor changes and modifications are required to deliver the, originally, promised product, which is QR-DQN applied to the Pacman game, whose implementation is provided by UC Berkeley[1]

1.1 Problem Definition

The objective of this project is to implement and test a state-of-the-art Deep RL algorithm and note the differences between it and vanilla RL. More concretely, this project wishes to compare the performance of Q-learning (implemented in Project 3) and Distributional RL with Deep Q-Network (QR-DQN).

Classical DQN aims to approximate the expected discounted rewards (Q-values). In other words, it tries to find the average cumulative reward of each action in a particular state. However, Bellemare et. al [1] have shown that estimating only the mean of the distribution does not always yield a desirable result. For example, some problems may require specifying a threshold to make more risk-averse (or risk-taking) decisions. In those cases, approximating the whole probability distribution rather than only its mean can come in handy.

[1] http://ai.berkeley.edu/project_overview.html

1.2 Problem Significance

Briefly, we aim to adopt a new and more robust approach to deep RL, particularly in the domain of the Pacman game, we hope to show that with QR-DQN, we get more consistent and, in the longer run, better performance, both for Pacman and other domains which QR-DQN could be generalized to.

If this project came to fruition, it could show that the novel approach to Q-learning, which is learning probability distributions rather than value averages, is better and more applicable to modeling real-life situations. This result will allow us to build more “rational” agents.

1.3 Possible Solutions

As previously mentioned, Bellemare et. al showed that estimating the mean of the rewards does not necessarily yield the best course of action. This is why they proposed a novel method by which they find the probability distribution of the discounted rewards, rather than only the mean.

This probability distribution is estimated by minimizing the loss function of Quantile Regression. This regression differs from Ordinary Least Squares in that it estimates the best weights for the specified quantiles (a.k.a. percentiles). Thus, we can successfully approximate the true probability distribution by computing the weights for each quantile [2].

1.4 Technical Challenges

The initial challenge was more about understanding the theory behind this new approach to deep RL. Our current exposure to research papers, especially in the field of AI, was very minimal up to this point. So it took some time for us to get comfortable with the formatting, approach, and writing style of research papers. When that was done, I believe we had overcome the first obstacle.

In terms of technical challenges, the first one was that it was close to impossible for us to build the domain, game, agents, and the mechanics from scratch during the given time period. Luckily, we used the Pacman game built by UC Berkley as a base, we have imported the game logic, some maps, the graphics, and some utility functions. The remaining part was what we hope to experiment on, which is the QR-DQN agent. For that goal, we modified the already existing pacman.py and created a new file called pacmanDQN_Agents which contains the “rational” agent. Thanks to a GitHub repository written by Bellemare[2], we were able to understand the field and domain, relatively, quickly. Although a framework was provided, we did not use it as we had already chosen the UC Berkley Pacman as our domain and we planned to test everything there as we are familiar with the codebase in the Pacman project.

1.5 Summary of Contribution

- Deep RL
- QR-DQN
- Research Paper Analysis
- ML Agents
- Distributional RL
- Wasserstein Metric

2.0 Methodology

2.1 Previous Methods

Many of the RL algorithms, both old and new, try to estimate the action-value function, this is generally the function used to estimate the values:

$$Q^\pi(x,a) := \mathbb{E}\left[Z^\pi(x,a)\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)\right]$$

Although this can be shown to converge eventually to true values that yield an optimal policy, it is not particularly computationally efficient. An improvement over this approach is temporal Difference (TD). TD manages to speed up the learning process by improving

[2] https://github.com/mgbellemare/Arcade-Learning-Environment

the estimate of Q^π incrementally by utilizing Dynamic Programming approaches, such as the *Bellman operator*

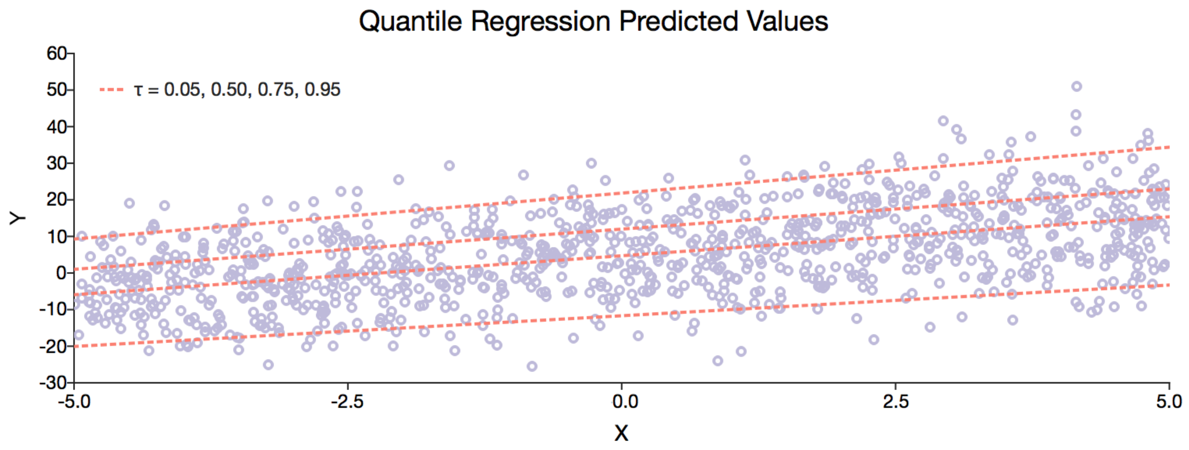
$$\mathcal{T}^\pi Q(x,a) = \mathbb{E}\left[R(x,a)\right] + \gamma \mathbb{E}_{P,\pi}\left[Q(x',a')\right].$$

These approaches have been used widely in a multitude of fields, and they showed a lot of promise, however, due to the fact that function approximations are being used, it was shown that this approach may result in instabilities in the learning process (Tsitsiklis and Van Roy 1997). This leads us to the introduction of a new and stable deep RL algorithm.

2.2 New Approach

The idea we implement is mainly based on using Quantile Regression as our loss function. This regression method is essentially employed to find a function that maps its input to an output given a certain quantile. A quantile can be simply defined as an interval containing the values the random variable can take on with a certain probability. An example of quantiles is percentiles, which denotes slicing (or discretizing) the distribution into hundred ranges and is commonly used in daily life. For instance, being in the 90th percentile in a competition means receiving a higher score than 90% of all the participants. In other words, if the lowest score in this competition is 0 and our score is 70, it means that the probability of receiving a score between 0 and 70 is 90%.

The idea behind quantile regression is to fit the data points to a curve that most successfully explains a specified percentile of them, as shown in the figure below.

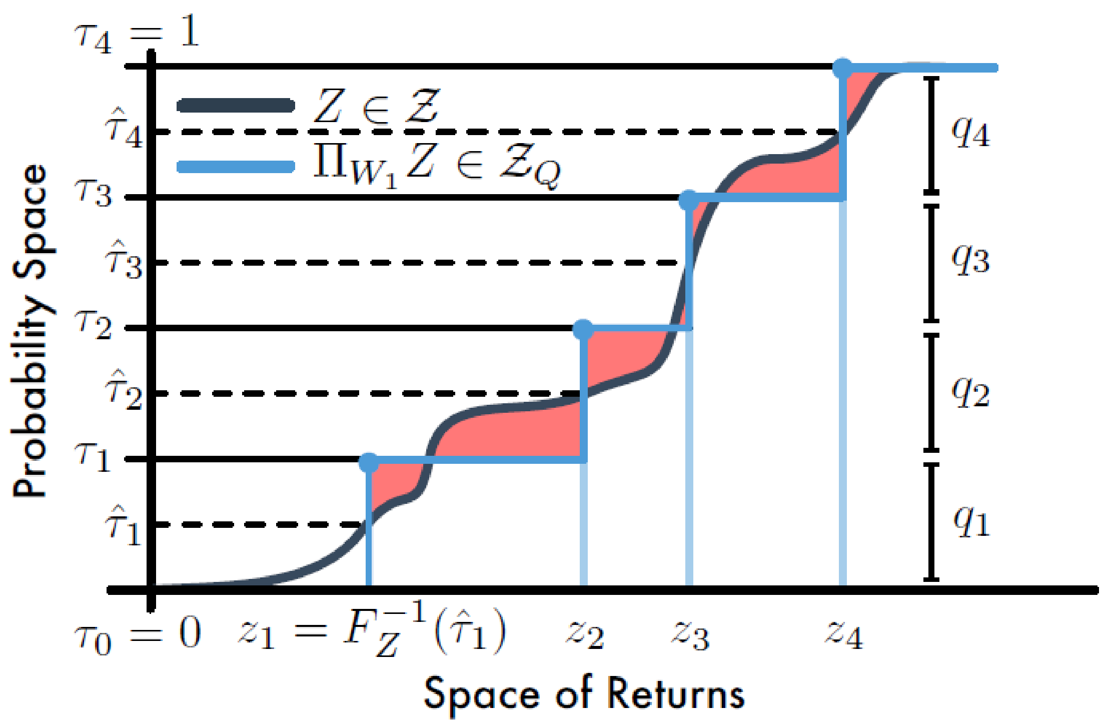


In this figure, four trendlines are plotted for four different percentiles (τ values). The uppermost line fits best to the points with a Y value higher than 95% of all the data points. Note that if we have a sufficient number of quantiles and find functions that explain them best according to a certain metric, we can approximate the true distribution.

We use this idea to find the probability distribution of Q-values given a state and action pair. This differs from classical Q-learning in that instead of estimating merely the expected Q-value, we estimate the whole probability distribution of the Q-values for a certain state and action. Once we have an approximation for the true distribution, we can use the insights provided by that in many ways. For instance, we might be quite risk-averse and want to pick the Q-value higher than only 10% of the distribution. This strategy almost guarantees that we never overestimate the true Q-value of a state-action pair, even though we most likely underestimate it.

However, in each Bellman Update, the algorithm we implement finds the expected value of the approximate distribution. In other words, instead of directly estimating the mean, we estimate the distribution and find the mean of that on the fly (i.e., during the estimation). Intuitively, this may appear to be an absurd idea as both estimations seemingly produce the same result. However, Bellemare et. al [1] have mathematically shown that this method generates more accurate predictions when the range of Q-values varies greatly among states, which is also supported by the empirical evidence they provided. The proof is essentially based on the fact that the classical method cannot always minimize the Wasserstein metric, which is stated in the following inequality.

$$\arg\min_{\mu} \mathbb{E}_{Y_{1:m}} [W_p(\hat{Y}_m, B_\mu)] \neq \arg\min_{\mu} W_p(B, B_\mu).$$



3.0 Results

Given that the implementation has not been fully completed yet. The results and comparisons using this new approach will be left until implementation is completed, tested, and trained.

4.0 What is Left To Be Done

The first step of building the QR-DQN agent is building a network capable of Vanilla DQN, similar to C51. That segment of the project has been successfully developed, tested, and trained. The results seem promising when Pacman was training on the small layout map. The next step is to make a small modification to the DQN network so it takes into account the probabilities as well. This part is the remaining segment of the project, aside from the comparisons.

5.0 References

[1] Bellemare, M. G., Dabney, W., & Munos, R. (2017, July). A distributional perspective on reinforcement learning. In International Conference on Machine Learning (pp. 449-458). PMLR.

[2] Dabney, W., Rowland, M., Bellemare, M., & Munos, R. (2018, April). Distributional reinforcement learning with quantile regression. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

[3] https://github.com/mgbellemare/Arcade-Learning-Environment