Notes on Distributional Reinforcement Learning

Shijie Huang

January 11, 2021

1 Introduction

With a passion in distributional RL and to better study the algorithms, I implemented the main distributional RL algorithms (Categorical, Quantile and Expectile) [1]. This summary documents my understanding and thoughts, and hopefully it will be helpful to those who intend to understand and implement the algorithms.

While I have tried my best to use plain language and simplified process charts to illustrate the algorithms, reading the original paper is essential to fully understand the rationale behind them (especially the proofs).

When I implement the algorithms, I think the hardest part is to get the shape of arrays/matrix correct, particularly when outcomes are processed in batches. Although implementations may vary from one to another, it is recommended that you draw a diagram with expected outcome shape attached to each process. In the diagrams I draw below, I included the outcome shape at each step. I did not include the batches because it would create unnecessary complications.

This summary does not touch on the issues of state representation/neural network structure. I only pay attention to the RL part because choices of deep learning structure do not really contribute to the understanding of core distributional RL algorithms (but they matter a lot in practice).

2 Environment

I use Cartpole (v0) as the environment. For more information, please visit the github wiki page. Each state (observation) is a (1, 4) vector with continuous values, and each action is a discrete scalar, either 0 or 1.

[1] see `https://github.com/hsjharvey/Reinforcement-Learning`

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Actions

Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

3 Distributional RL

Figure 1 illustrates a typical process of a DQN. The general idea of the distributional reinforcement learning is not sophisticated: given a (state, action) pair, the network does not output action value directly as a single scalar value, but rather a distribution. In the same figure, I highlighted the key steps where distributional RL are different from standard DQN, in a red box.

The main structure of the distributional version remains the same as the DQN, e.g. two networks, TD updates, etc. The main challenges are the following:

1. Which distribution(s) should we choose?
2. How do we apply the concept of value distributions in the general RL framework?

The first question is a more generic question. There are two general ways to approach it: (1) assume some known functional form with unknown meta-parameters (e.g. Gaussian, student-t, etc.) (2) generate samples to approximate a distribution. The pros and cons of each approach are a topic of another field, but it is not too hard to see that in a neural network setting, approximation is the default method. So the question pins down to the following: how do we approximate/characterise a distribution?

A distribution has certain statistics to characterise itself. In plain language, if we know some statistical properties of a distribution, we can roughly know what this distribution roughly "looks like". A typical example is histogram: a probability distribution can be characterised by equal-sized bins with different heights (either in probabilities or frequencies). Thus, if we know those characteristics, we can have an approximated distribution.

The second question is more involved in the context of reinforcement learning. Replacing the scalar action value with a distribution raises more questions. For instance, how do we "choose" a distribution to act upon? In standard DQN, we can follow a specific policy, and compare two scalar values regardless how they are computed. However, in distributional RL, it is not obvious how to

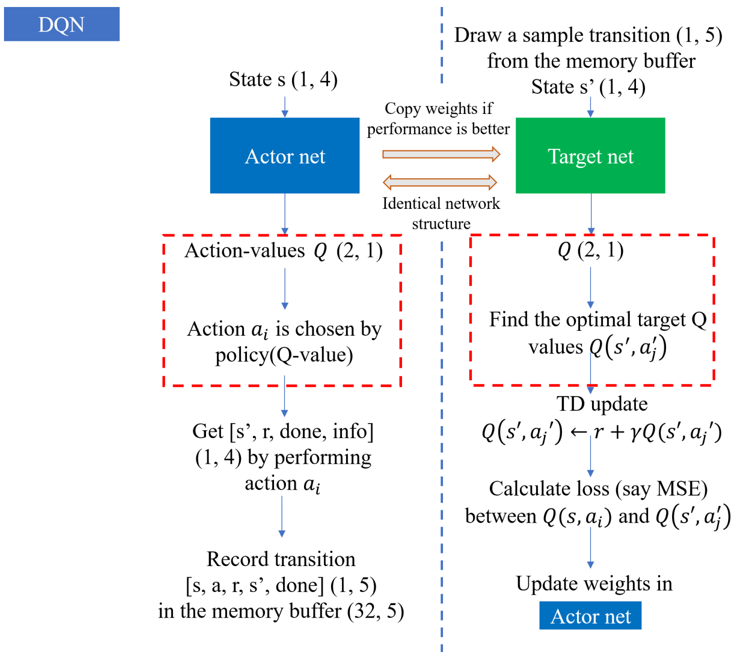

Figure 1: Deep Q Learning.

"compare distributions". Moreover, it is also not obvious that we can simply "update a distribution" so that it converges to the target distribution, just like what we do in standard DQN. Although the actual implementation varies from algorithms to algorithms, it is important to have these top layer questions in mind when we study the algorithms.

4 Categorical [1]

In categorical RL, the choice of distribution to be approximated is histogram. To characterise a histogram, we need two arrays: (1) bins (2) corresponding height (frequency). In categorical RL, we assume that the true Q-values lie within a boundary, $[V_{min}, V_{max}]$; we also assume that there are $n = 51$ equal sized (ΔZ) bins. There are in total 51 points (called atoms) on the x-axis (hence the name C51). The corresponding frequencies are the output of neural networks. The limitation of this approach is obvious: (1) how do we know that the true Q-value lie inside the boundary? (2) how do we determine the appropriate boundary in the first place?

In categorical RL, we compare actions by comparing the expected value of its PDF. Notice that this expected value is the mean action-value (Q-value).

To perform a TD update, we first shrink (γ) and move $(reward)$ the atoms (x-axis), then we re-distribute the frequencies back to where the original bound-

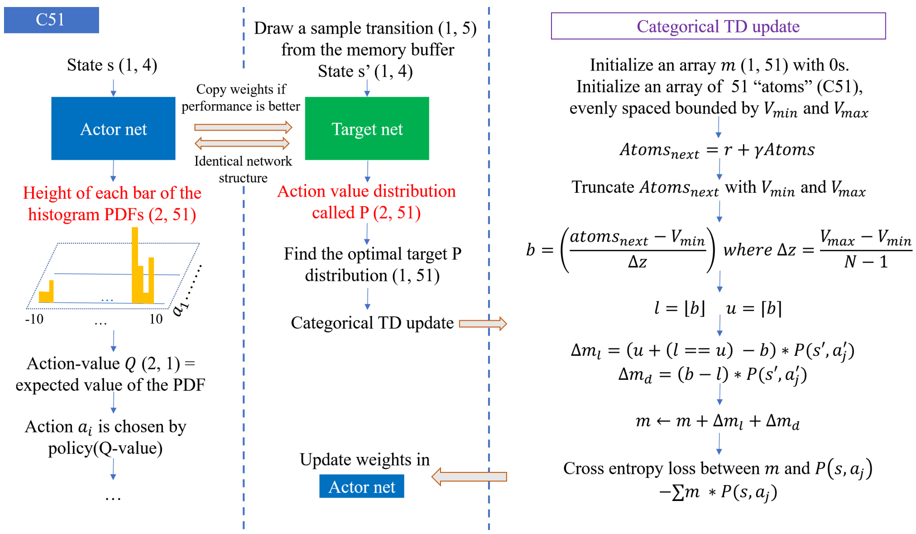

Figure 2: Categorical DQN (C51)

aries. This process is illustrated clearly in the original paper (see figure 3).

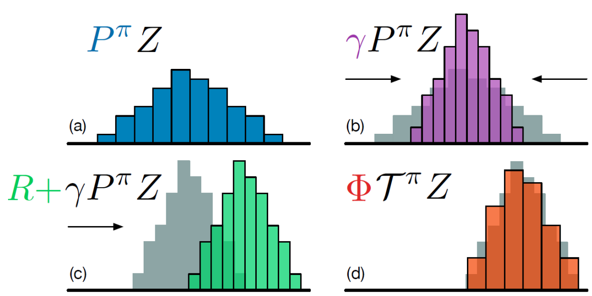

Figure 3: Distribution Bellman Operator [1]

To see how it works in an atari game, one can visit https://www.youtube.com/watch?v=vIz5P6s80qA.

5 Quantile [4]

Another way of approximating a distribution is via quantiles. Instead of assuming equal-spaced bins on the x-axis like we do in a history, we assume equal-spaced values on the y-axis. The neural network will output the position of the ith quantile value on the x-axis. By further assuming that each position has the same height (Dirac distribution), we will have an approximated value distribution. Notice that the density of a distribution is not represented by

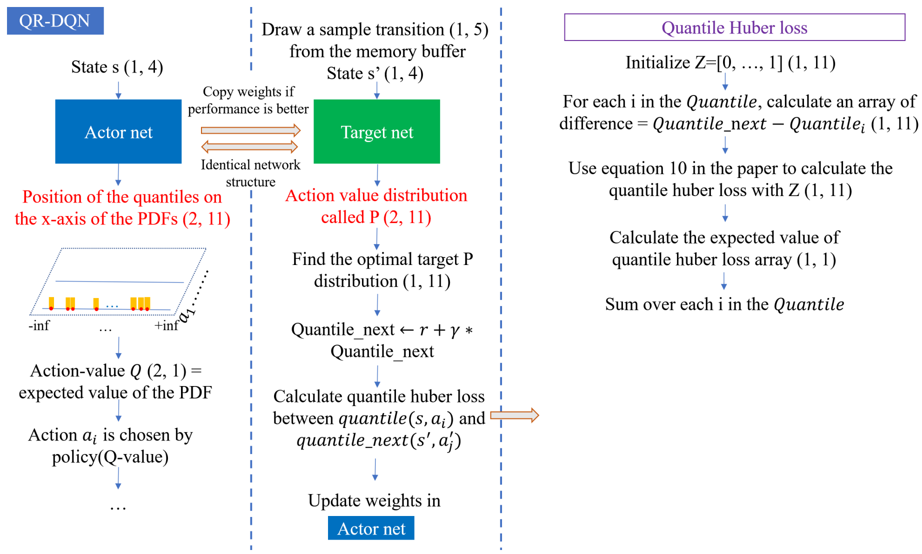

Figure 4: QR-DQN

"how tall" a bin is, but rather how "clustered" those equal height bins are. So arguably it makes more sense to visualize the values in CDF rather than PDF.

In the quantile regression DQN (QR-DQN), we also compare actions by comparing the expected value of its PDF. By assuming equal height, the mean action value is simply the mean of all quantile values.

Unlike the seemly sophisticated TD update process in Categorical DQN, the TD update of QR-DQN is relatively straightforward. Because no boundary assumptions are imposed on the quantile values, to perform distribution update, we can simply perform TD update on distributions/samples on the x-axis (i.e. shrink and move the samples). What's left is to find an appropriate loss function to calculate the loss between quantile values from the actor network and the target quantile values from the target network.

If you look closely at the figure 4, it may be a bit confusing since the TD updates are performed directly on the quantile values, rather than the distribution. There are important assumptions for that to work. I will discuss these assumptions in section 7. The same issue is highlighted by the authors later in the expectile paper [10].

To see how the quantile method works in an atari game, one can visit `https://www.youtube.com/watch?v=zdh_BTOcVYs`. Be aware that in the video, the implemented algorithm is Implicit Quantile Network (IQN), which is an improved version, see [3] for more details.

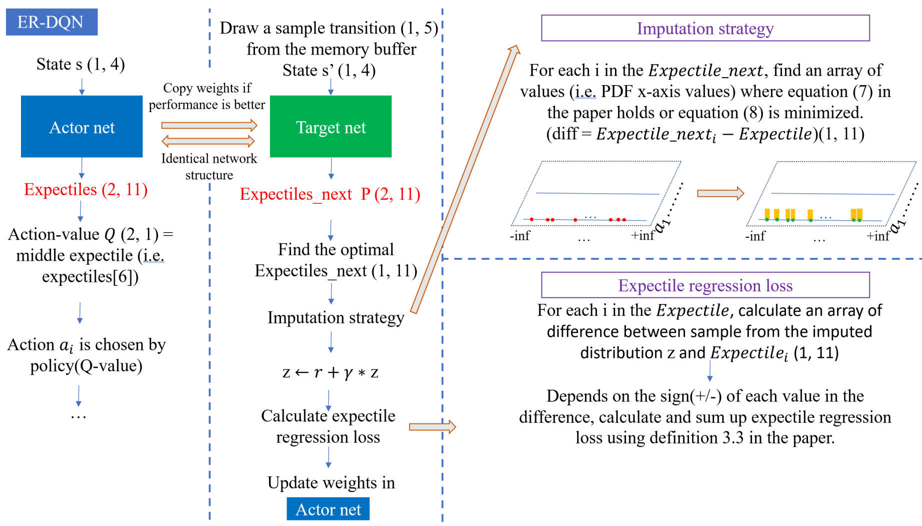

Figure 5: ER-DQN

6 Expectile [10]

The expectile method is very similar to the quantile method. We also assume equal-spaced bins on the y-axis. However, in the expectile method, we say that each value on the x-axis that corresponds to the ith value on the y-axis is the ith expectile value of the distribution. The neural network will output the position of the ith expectile on the x-axis.

In the expectile regression DQN (ER-DQN), we compare actions by comparing the middle expectile because the middle expectile value is the mean.

Similar to the QR-DQN, we can perform TD update directly on expectile values so that the expectile values are closer to the target expectile (i.e. shifting the CDF towards the target CDF by moving the x-axis values). However, this naive method turns out to be problematic in the expectile setting due to the difference between samples and their statistics (see section 3.1 in the paper). I will discuss the issue and the assumptions in detail in section 7.

A better approach is to first re-generate the entire distribution from the statistics (i.e. quantile or expectile), and then perform TD update on the re-generated distribution. This process of re-generating a distribution from its statistics is called an "imputation strategy".

After the target network outputs target expectile values, we want to find a distribution in which its expectile values match the target expectile values (root-finding problem, equation 7 of [10]) or one that minimises the expectile-regression loss, given the target expectile values (minimization problem, equation 8 of [10]).

We will then perform a TD update on the imputed distribution (samples). The final step is to update weights in the actor network succh that its output, expectile values, minimises the expectile regression loss, given the updated target distribution.

7 Samples and Statistics

In the expectile paper [10], the authors highlighted the difference between "samples" and "statistics", and proposed the "imputation" method in addition to the "naive" method. I will expand the discussion further with examples in this section.

Suppose that we have the following three samples/distributions y_1, y_2 and y_3. We want to find five quantile values correspond to τ. In this case, one can easily verify that all three samples have exactly the same quantile values.

$$
\begin{aligned}
y_1 &= [1, 1, 2, 2, 4, 4, 6, 8, 8, 8, 8, 10, 10] \\
y_2 &= [1, 2, 6, 8, 10] \\
y_2 &= [1, 1, 1.5, 2, 6, 6, 6, 8, 8, 8, 9, 10, 10] \\
\tau &= [0, 0.25, 0.5, 0.75, 1]
\end{aligned}
$$

I believe this example can illustrate the key message that [10] intended to deliver: two distributions are not necessarily the same even if they have the same statistics.

We can now discuss the implication of this message in the context of distributional reinforcement learning. Broadly speaking, the TD update (i.e. $Z \leftarrow r + \gamma * Z$) requires a distribution Z, not its statistics. However, in practice for QR-DQN (originally proposed in [4], later in [10] it is called the "naive" version), we perform TD update directly on quantile values. For that to work, two important assumptions are required. First, the distributional form is Dirac $\sum \delta_z$. Second, the number of samples equals the number of quantiles. In the above example, y_2 happens to be the five τ quantile values of itself.

Nonetheless, we need to appreciate that the above case is really a special example, and this becomes problematic in the expectile method. The τ_3th expectile of y_2 (mean) = 5.4. Consequently, even if we have the exact same assumptions in place, the sample values in the distribution are different from their expectile values.

In [10], the authors highlighted this issue and introduced the "imputation" strategy. An imputation strategy effectively takes in statistics (e.g. expectiles) and outputs the samples/distribution [10].

In practice, to use the Scipy *root* method to implement the imputation strategy (equation 7 in [10]), one has to assume that the number of samples equals the number of expectiles due to the technical limitation of the *root* function[2] (this is

[2] By default, Scipy uses the "hybr" method, which requires the input shape and the output shape to be the same, see `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.root.html` for more information.

also implied in the Algorithm 2 of the paper). If one pursues the minimization method (equation 8 in [10]), this assumption is not necessary.

8 Limitations, Assumptions and Recent Work

In this section, I will discuss some key general limitations across the three approaches, and briefly introduce some recent researches which intend to address those limitations[3].

8.1 Pre-assumed statistics locations

All three approaches are required to have pre-determined statistics, typically equal-spaced values within a certain range (e.g. equal-spaced values $\in [0, 1]$). Those pre-determined statistics, both in terms of the number and in terms of their positions, obviously place restrictions on how well the distributions can be approximated. In principle, we want as many statistics as possible, but in reality it is not possible as we have limited model capacity.

To address this issue, a recent paper proposed to use another neural network that outputs quantile positions, instead of pre-determined location like $[25\%, 50\%, 75\%]$ [11]. Although the number of quantiles is still required, at least the locations are parameterized.

8.2 Non-monotonic order statistic values from the neural network

The other issue, or rather a puzzle that I had during implementation was that there was no guarantee that the output quantile values from the network would be monotonically increasing. This is a known as the "crossing" issue in statistics. There are many discussions on this topic. A recent paper addressed this issue in the context of distributional reinforcement learning by proposing a new neural network structure to ensure that the output are monotonically increasing [6].

8.3 Use of "stastistics" to obtain action value distributions

Notice that in all three approaches, statistical properties are required as an "interim" process to get the action value distributions. A recent work proposed to use deterministic sampling to directly approximate the distribution via "particles" [8].

On a related matter, the other key component in distributional-RL is the measure of distance/difference between two action value distributions. In distributional RL, we try to optimize the actor network so that it will converge to the target network. This component is crucial in loss function design, yet very

[3] Acknowledgements to a blog post by Microsoft Research Lab (Asia) (In Chinese) https://www.zhihu.com/question/312164724/answer/1667432328

difficult, and often it creates discrepancies between theories and practice (e.g. Wasserstein metrics in theory vs. KL divergence in practice in [1]).

In the same recent work, the authors proposed to use Maximum Mean Discrepancy to measure the distance between two sets of "particles" in the loss function. According to the authors, this approach generalized the old metrics. However, in practice, one will still have to make assumptions of the number of particles, and a kernel function to be able to carry out this approach.

9 General Discussion and Others

After studying the three algorithms carefully, I realize that, in a nutshell in distributional RL, we intend to learn action value better via distributions. The system can be complicated and fragile in practice: one has to impose many assumptions. However, if designed and implemented properly, distributional RL can learn much richer representations of the action values, and thus lead to better actions.

In my opinion, the categorical RL is the easiest to understand. However, the obvious disadvantage is the complicated process to perform the Bellman update. Additionally, choices of V_{min}, V_{max} and the number of atoms really matter. One has to roughly know where the true action value is to be able to carry out a successful training.

Although in QR-DQN we still have to assume the number of quantiles, we do not necessarily have to know where the true action value is. However, we may have to be a bit careful when designing the loss function, as stated in the original paper.

The key to a successful ER-DQN algorithm is the imputation strategy. In practice, training could be problematic if the root finding (or the minimization) is not successful. Plus, training time can be significantly longer than the categorical method and QR-DQN due to the time spent on imputing distributions.

Finally, as I mentioned at the beginning, this notes focus on the RL part. An appropriate state representation, or network structure, is also crucial to a successful training, particularly when it comes to complicated environments.

References

[1] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR.org, 2017.

[2] Marc G Bellemare, Nicolas Le Roux, Pablo Samuel Castro, and Subhodeep Moitra. Distributional reinforcement learning with linear function approximation. *arXiv preprint arXiv:1902.03149*, 2019.

[3] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

[4] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] Will Dabney, Zeb Kurth-Nelson, Naoshige Uchida, Clara Kwon Starkweather, Demis Hassabis, Rémi Munos, and Matthew Botvinick. A distributional code for value in dopamine-based reinforcement learning. *Nature*, pages 1–5, 2020.

[6] Xingdong Feng Fan Zhou, Jianing Wang. Non-crossing quantile regression for deep reinforcement learning. In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.

[7] Clare Lyle, Pablo Samuel Castro, and Marc G Bellemare. A comparative analysis of expected and distributional reinforcement learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2019.

[8] Thanh Tang Nguyen, Sunil Gupta, and Svetha Venkatesh. Distributional reinforcement learning with maximum mean discrepancy. *arXiv preprint arXiv:2007.12354*, 2020.

[9] Mark Rowland, Marc G Bellemare, Will Dabney, Rémi Munos, and Yee Whye Teh. An analysis of categorical distributional reinforcement learning. *arXiv preprint arXiv:1802.08163*, 2018.

[10] Mark Rowland, Robert Dadashi, Saurabh Kumar, Rémi Munos, Marc G Bellemare, and Will Dabney. Statistics and samples in distributional reinforcement learning. *arXiv preprint arXiv:1902.08102v1*, 2019.

[11] Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tie-Yan Liu. Fully parameterized quantile function for distributional reinforcement learning. In *Advances in neural information processing systems*, pages 6193–6202, 2019.