Distributional Reinforcement Learning

– C51
– QR-DQN
– IQN
– FQF

Acknowledgement: Most slides were helped by 廖唯辰, 黃柏維, 林九州, 鄭余玄, 郭奎廷

Reference

- C51
  - Bellemare, Marc G., Will Dabney and R. Munos. “A Distributional Perspective on Reinforcement Learning.” ICML (2017).
- QR-DQN
  - Dabney, Will, Mark Rowland, Marc G. Bellemare and R. Munos. “Distributional Reinforcement Learning with Quantile Regression.” AAAI (2018).
- IQN
  - Dabney, Will, Georg Ostrovski, D. Silver and R. Munos. “Implicit Quantile Networks for Distributional Reinforcement Learning.” ICML (2018).
- FQF
  - Yang, Derek, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian and Tie-Yan Liu. “Fully Parameterized Quantile Function for Distributional Reinforcement Learning.” NeurIPS (2019).

Outline

- Introduction
- C51
- QR-DQN
- IQN
- FQF
- Summary

Distributional RL

- Introduction
- C51
- QR-DQN
- IQN
- FQF
- Summary

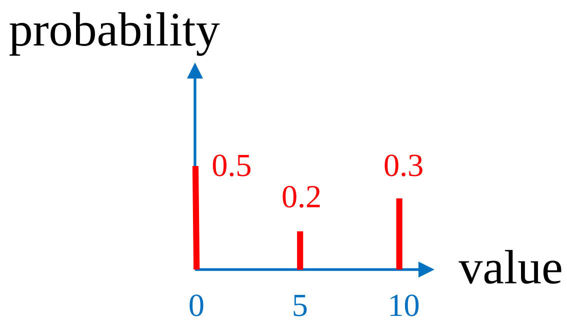
Distributional Reinforcement Learning

- In Q-learning and DQN, we learn the expectation of value
  - Q value: $Q(x,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)]$
  - Bellman operator: $Q(x,a) = \mathbb{E}R(x,a) + \gamma \mathbb{E}Q(X', A')$
- The expectation value is sampled from some value distribution

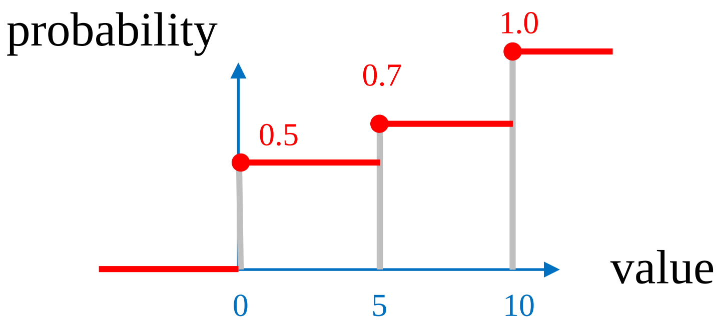
x: state
a: action
γ: discount factor
R: random variable of reward
(X', A'): random variable of next state-action

Value Distribution (Discrete)

probability mass function
(PMF)

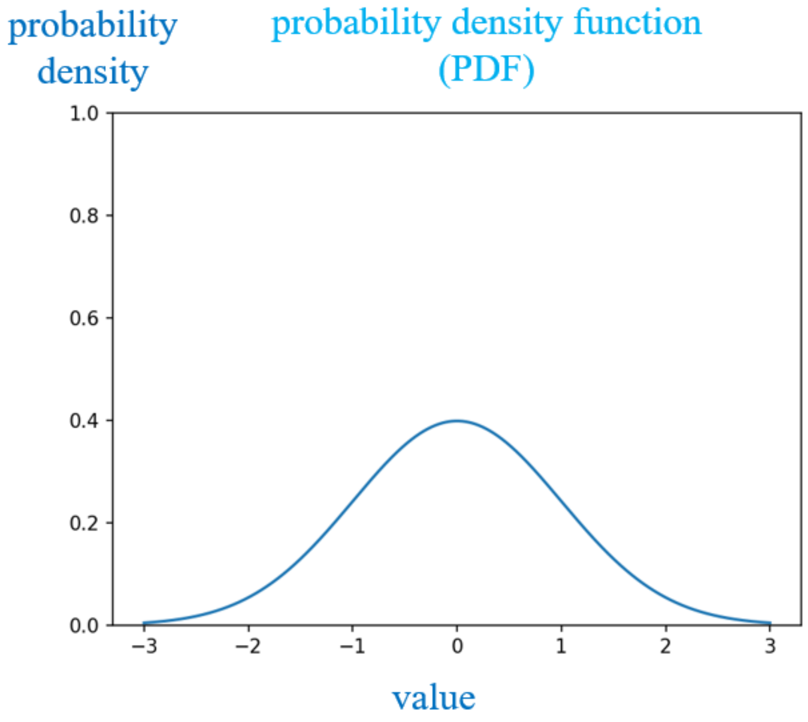


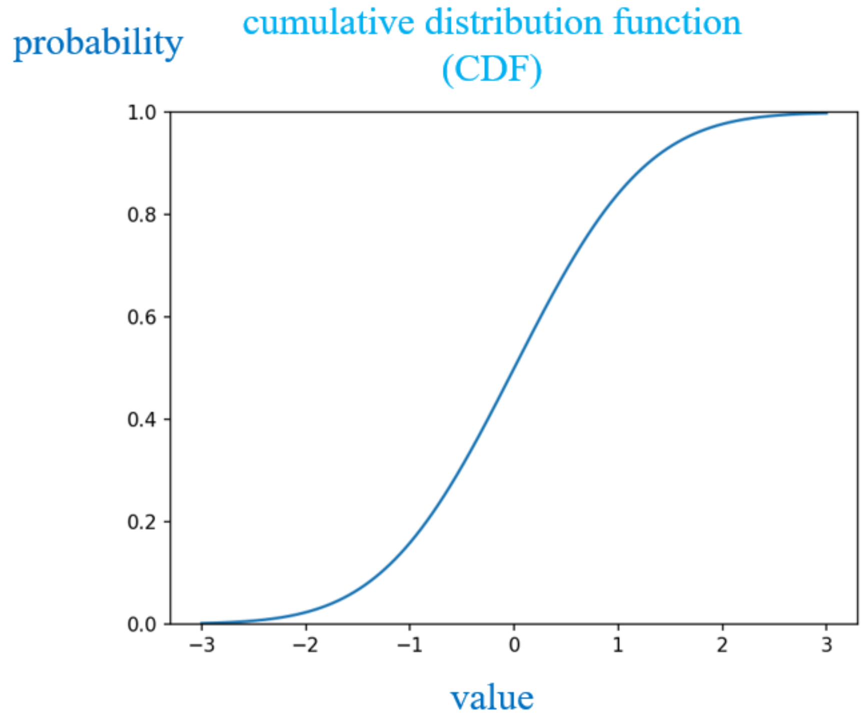
cumulative distribution function
(CDF)



Expectation:
0.5*1 + 0.2*5 + 0.3*10 = 4

Value Distribution (Continuous)





Distributional Reinforcement Learning

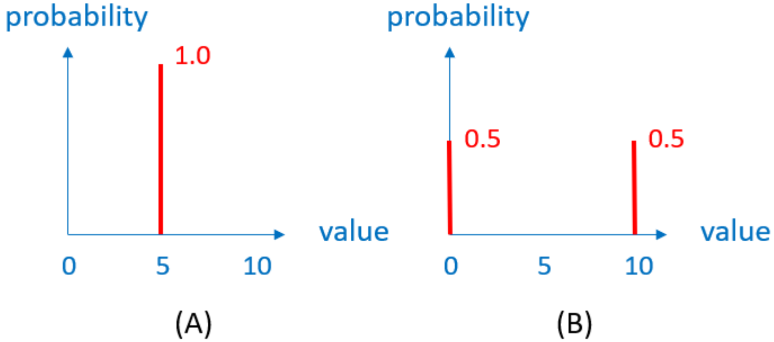
- Distributional RL aims to **learn the value distribution instead of the expectation**
- Significantly improve Atari-57 performance

	Mean	Median	>Human	>DQN
DQN	221%	79%	24	0
PRIOR.	580%	124%	39	48
C51	701%	178%	40	50
RAINBOW	1213%	227%	42	52
QR-DQN	902%	193%	41	54
IQN	1112%	218%	39	54
FQF	**1426%**	**272%**	**44**	**54**

Rainbow is a method that combines extensions of DQN:
Double Q-learning + Prioritized replay + Dueling networks + Multi-step learning (n-step) + Distributional RL (C51) + Noisy Nets

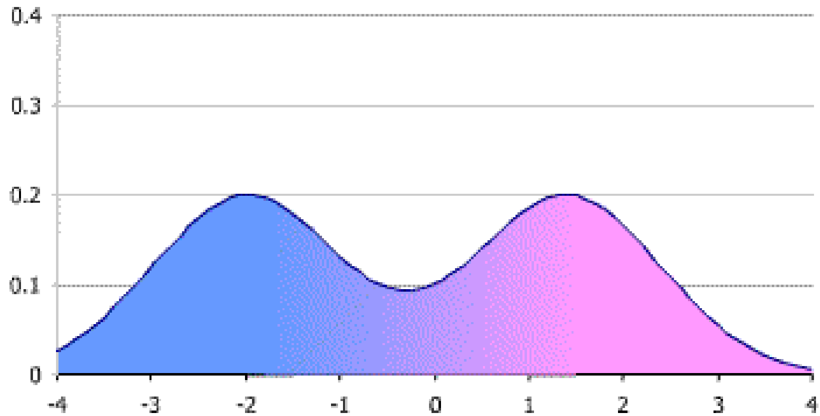
Potential Benefits of Distributional RL

1. Model parametric uncertainty
   - What’s the occasion of get expected return 5?
     - (A) 100% get 5 score
     - (B) 50% get 10 score, 50% get 0 score
2. Design risk-sensitive algorithms
   - From above example, if (A), (B) are two actions
     - We know the risk of getting 0 score in (B)
     - For low risk policy, we may choose (A)



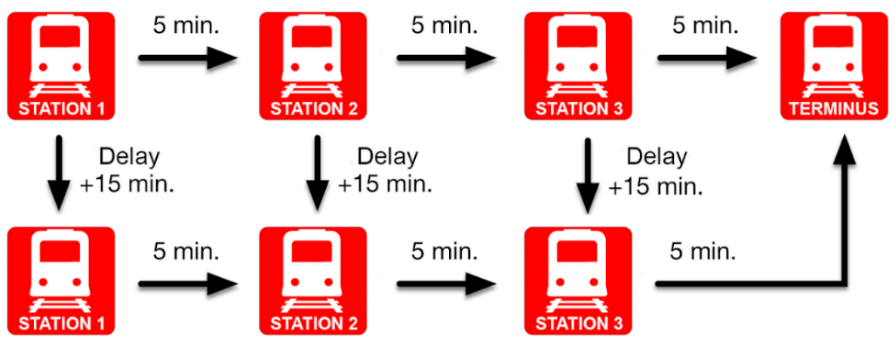
Potential Benefits of Distributional RL

3. Reducing Instability
   - For nonstationary policies, this method can be more stable
     - Chattering in two value does not converge well
4. Better approximations
   - Modeling multimodality (多峰) value function



An Example

- Assumptions:
  - A journey composed of three segments of 5 minutes each.
  - However, the train breaks down once a week (5 working days)
    - Adding another 15 minutes to the trip
- The average commute time is (3 x 5) + 15 / 5 = 18 minutes
  - V(station 1) = 3x5 + (15/5) = 18. → But it is either 15 or 30, never 18,
  - V(station 2) = 2x5 + (15/5) = 13.
  - V(station 3) = 1x5 + (15/5) = 8.
  - For safety, we may favor the one with the least risk, say walking (assuming to have the same average outcome, e.g., 18 for 1).



https://deepmind.com/blog/article/going-beyond-average-reinforcement-learning

Distributional RL

- Introduction
- C51
  - A Distributional Perspective on Reinforcement Learning
- QR-DQN
- IQN
- FQF
- Summary

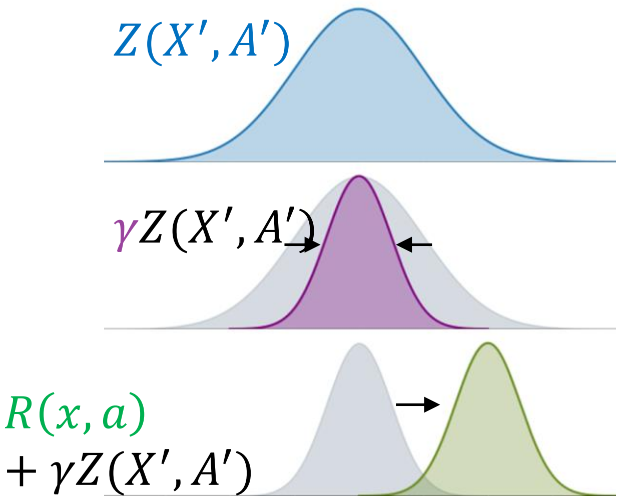
C51

- Categorical DQN
  - All settings are the same as DQN except for modeling Q value
  - Model the value distribution of each action with discrete distributions
- Define and prove value distribution in Q learning
  - Learn the value distribution by distributional Bellman operator
  - Prove that the distributional Bellman operator is a contraction based on Wasserstein metric

Distributional Bellman Operator

- Let $\mathrm{Z}(x, a)$ be a random variable from the value distribution
  - $\mathrm{Z}(x, a) = \sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)$
  - $\mathbb{E}[Z(x, a)] = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)] = Q(x, a)$
- The distributional Bellman operator:
  $\mathrm{Z}(x, a) =^D R(x, a) + \gamma Z(X', A')$
  in distribution

  x: state
  a: action
  γ: discount factor
  R: random variable of reward
  (X', A'): random variable of next state-action

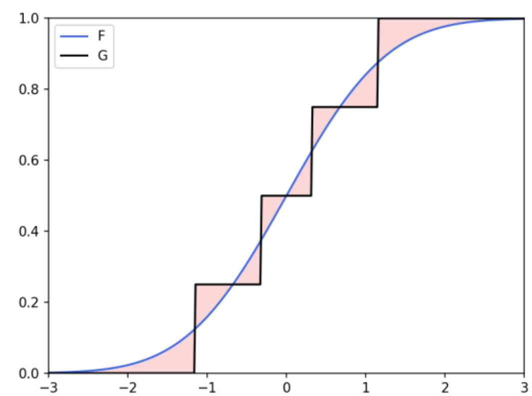


Wasserstein Metric

- Measure the "distance" between 2 cumulative distribution functions (CDF)
- For two CDFs, F and G, with inverse CDFs F^{-1} and G^{-1}, the Wasserstein metric:

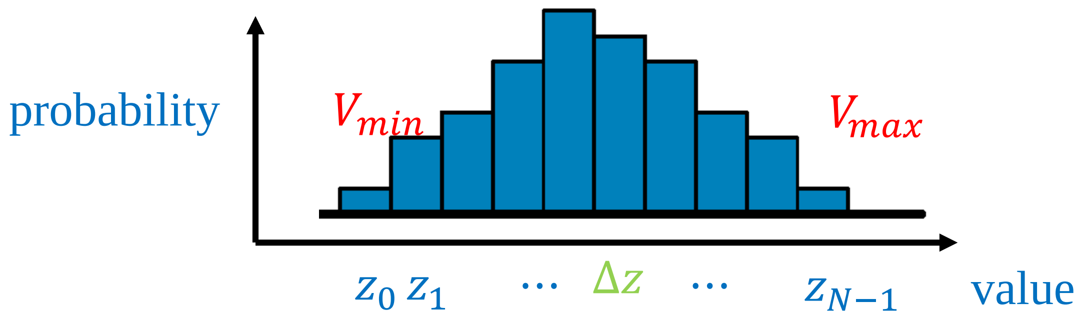
$$W_p(F,G) = \left(\int_0^1 |F^{-1}(u) - G^{-1}(u)|^p du \right)^{\frac{1}{p}}$$

- p=1: Earth Mover Distance
  - The shaded regions



Parametric Distribution

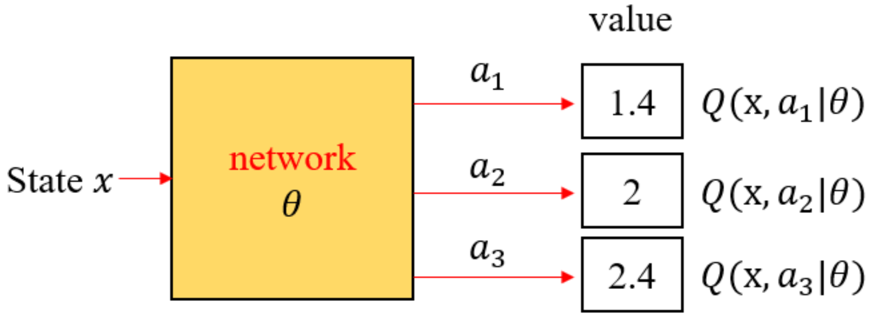
- Atom: The possible values in the discrete distribution
  - Predefine the value bound $[V_{min}, V_{max}]$ and the number of atoms N
  - Equally divide the scope into N atoms: $z_0, z_1, \ldots, z_{N-2}, z_{N-1}$
- C51 predicts the probability of each atom



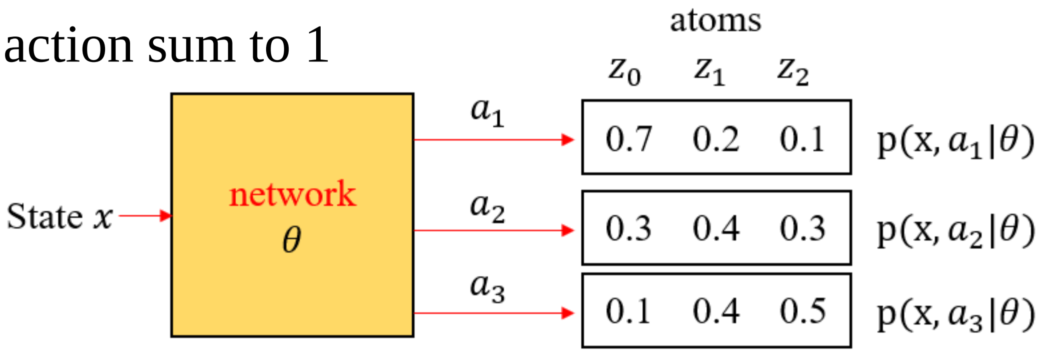
- The distribution can be approximated more accurately with more atoms
  - They tested varying numbers of atoms, and N=51 performs the best ⇒ C51

Network Architecture

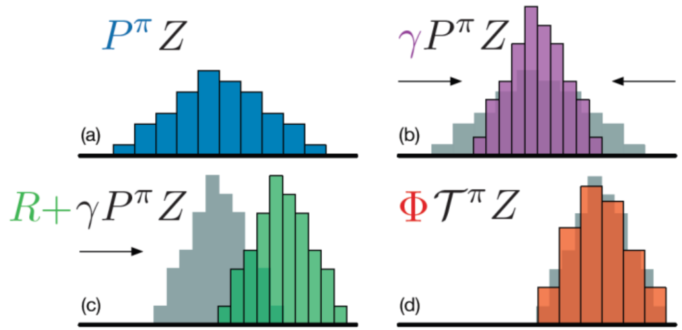
- DQN network
  - Predict the **value** for each action



- C51 network
  - Predict the **probability** of each atom for each action
  - Outputs for each action sum to 1



How to Update



- Using distributional Bellman operator $\mathcal{T}^\pi$
- Assume current distribution Z
  - Atoms: $z_0, \dots, z_{N-1}$
  - Corresponding probabilities: $p_0(x_t, a_t), \dots, p_{N-1}(x_t, a_t)$
- Target distribution $\mathcal{T}^\pi Z = R + \gamma \underline{P^\pi Z}$ Next state distribution

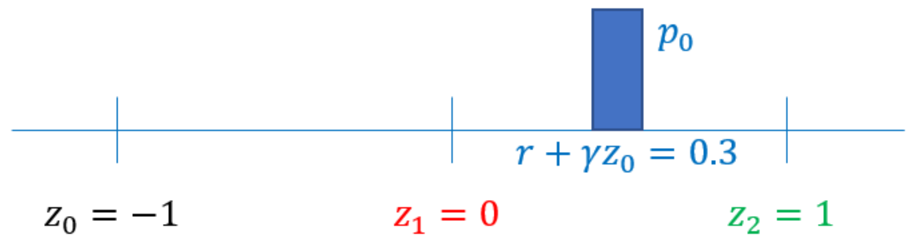
  probability of the atom

  - Select action $a^* = argmax_{a'} Q(x_{t+1}, a') = argmax_{a'} \sum_{i=0}^{N-1} p_i(x_{t+1}, a') z_i$
  - Values: $\mathrm{r_t} + \gamma z_0, \dots, r_t + \gamma z_{N-1}$

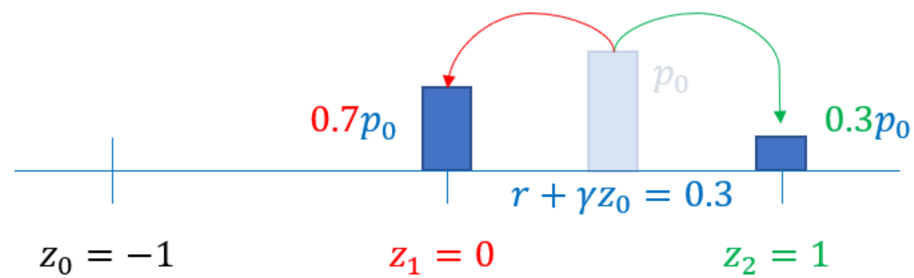
    value of the atom
  - Corresponding probabilities: $p_0(x_{t+1}, a^*), \dots, p_{N-1}(x_{t+1}, a^*)$
- The values of the target distribution may not on the atoms
  - Projection step Φ

Projection Step

- $r + \gamma z_i$ may not on the atoms



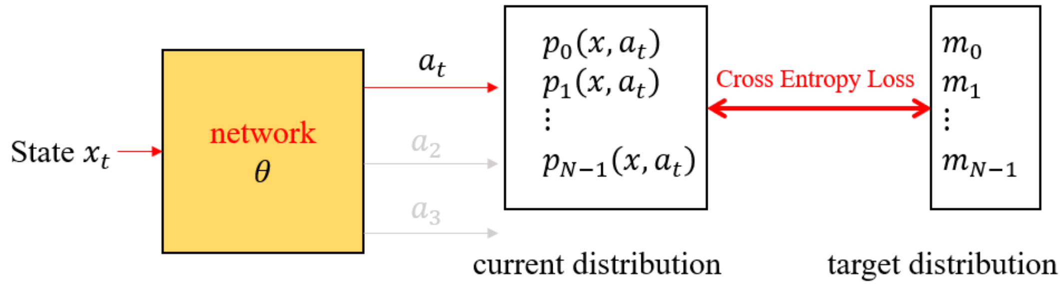
- Distribute probability mass to neighboring atoms
  - $r + \gamma z_0 = 0.7 z_1 + 0.3 z_2$



  - Distribute $0.7p_0$ to atom 1, $0.3p_0$ to atom 2

How to Update

- Current distribution
  - Atoms: $z_0, \dots, z_{N-1}$
  - Corresponding probabilities: $p_0(x_t, a_t), \dots, p_{N-1}(x_t, a_t)$
- Target distribution
  - Using distributional Bellman operator and projection step
  - Atoms: $z_0, \dots, z_{N-1}$
  - Corresponding probabilities: $m_0, \dots, m_{N-1}$
- Cross Entropy Loss
  - $-\sum_{i=0}^{N-1} m_i log\, p_i(x_t, a_t)$
  - Minimizes KL divergence



Algorithm

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

get next action

$m_i = 0, \quad i \in 0, \ldots, N-1$

for $j \in 0, \ldots, N-1$ **do**

Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

Distribute probability of $\hat{\mathcal{T}} z_j$

$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

end for

output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

Algorithm

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

$\boxed{m_i = 0, \quad i \in 0, \ldots, N-1}$

m$_i$ is the probability accumulator for atom i

for $j \in 0, \ldots, N-1$ **do**

Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\mathrm{MIN}}}^{V_{\mathrm{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\mathrm{MIN}})/\Delta z$ # $b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

Distribute probability of $\hat{\mathcal{T}} z_j$

$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

end for

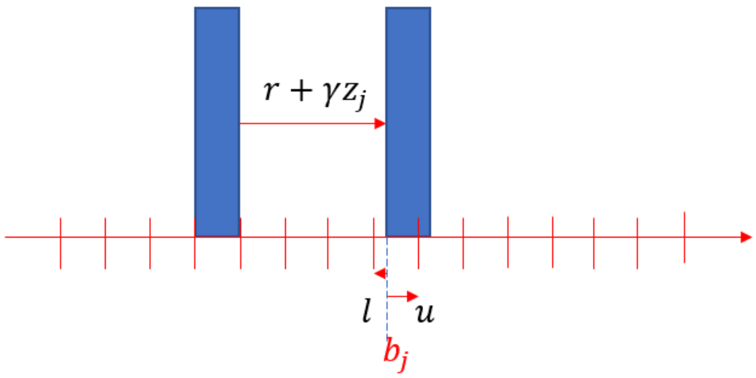
output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

I-Chen Wu

Algorithm

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$
  $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$
  $a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$
  $m_i = 0, \quad i \in 0, \ldots, N-1$
  for $j \in 0, \ldots, N-1$ **do**
    # Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$
    $\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$
    $b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$
    $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$
    # Distribute probability of $\hat{\mathcal{T}} z_j$
    $m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$
    $m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$
  end for
output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss



Algorithm

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$

$Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$

$a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$

$m_i = 0, \quad i \in 0, \dots, N-1$

for $j \in 0, \dots, N-1$ **do**

Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$

$\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$

$b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$

$l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$

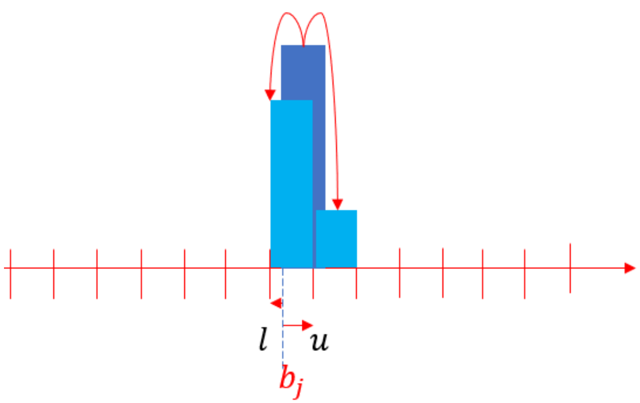
Distribute probability of $\hat{\mathcal{T}} z_j$

$m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$

$m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$

end for

output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss



Algorithm

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$
  $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$
  $a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$
  $m_i = 0, \quad i \in 0, \ldots, N-1$
  for $j \in 0, \ldots, N-1$ **do**
    # Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$
    $\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$
    $b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$
    $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$
    # Distribute probability of $\hat{\mathcal{T}} z_j$
    $m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$
    $m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$
  end for
output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

compute loss

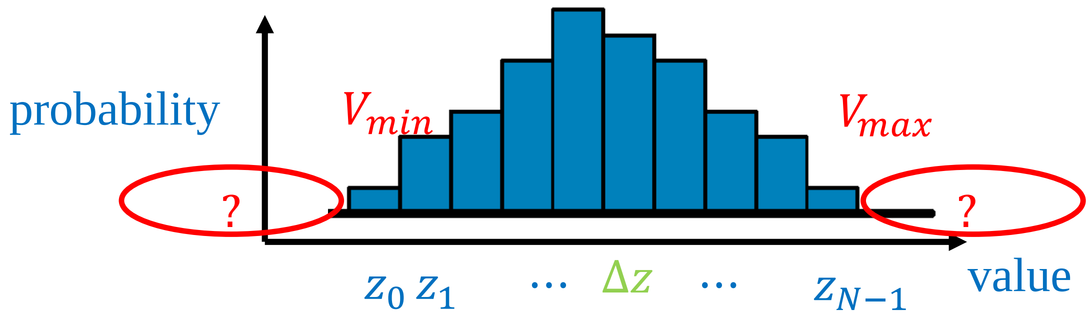
C51 Experiment

- 57 Atari games
- Achieves state-of-the-art in Atari games at that time

	Mean	**Median**	**> H.B.**	**> DQN**
DQN	228%	79%	24	0
DDQN	307%	118%	33	43
DUEL.	373%	151%	37	50
PRIOR.	434%	124%	39	48
PR. DUEL.	592%	172%	39	44
C51	**701%**	**178%**	**40**	**50**

Drawbacks of C51

- Theory-practice gap:
  - Theory: The distributional Bellman operator is a contraction based on Wasserstein metric
  - Practice: Minimizes KL divergence
- The value is bounded



Distributional RL

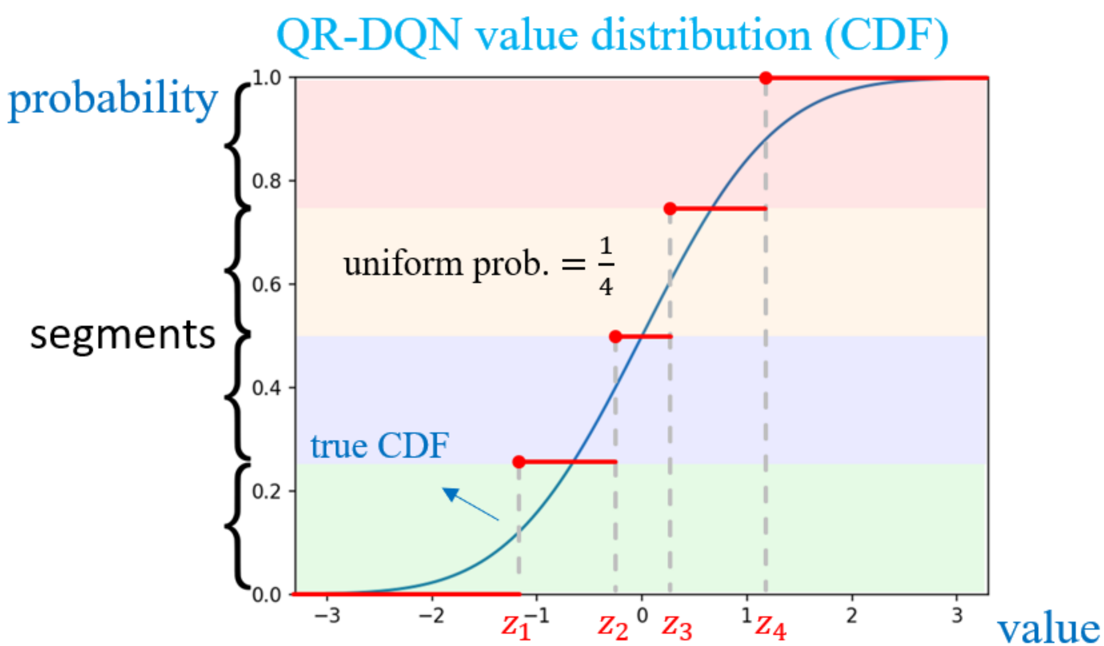
- Introduction
- C51
- QR-DQN
  - Distributional Reinforcement Learning with Quantile Regression
- IQN
- FQF
- Summary

QR-DQN

- Transpose the parametrization from C51
  - C51 uses N fixed values (atoms) for its distribution and predicts their probabilities
  - QR-DQN equally divides CDF into N segments and predict their values
  - e.g. 4 segments with corresponding values z_1, z_2, z_3, z_4

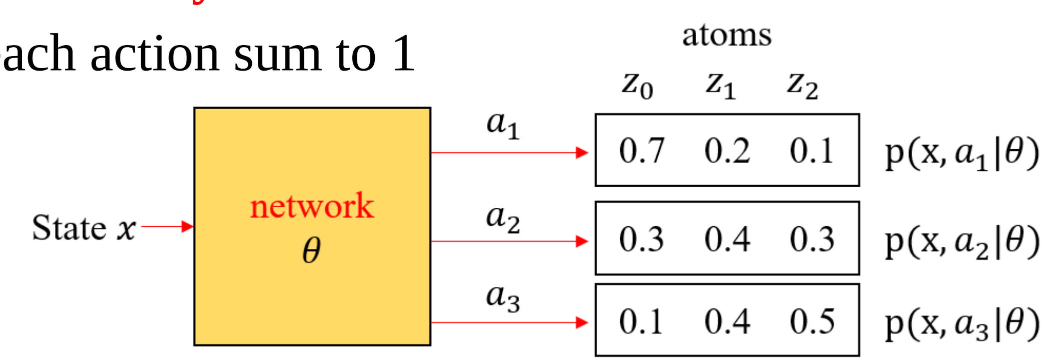
QR-DQN

- Prove contraction mapping results for its algorithm
  - Its method performs distributional RL end-to-end under the Wasserstein metric

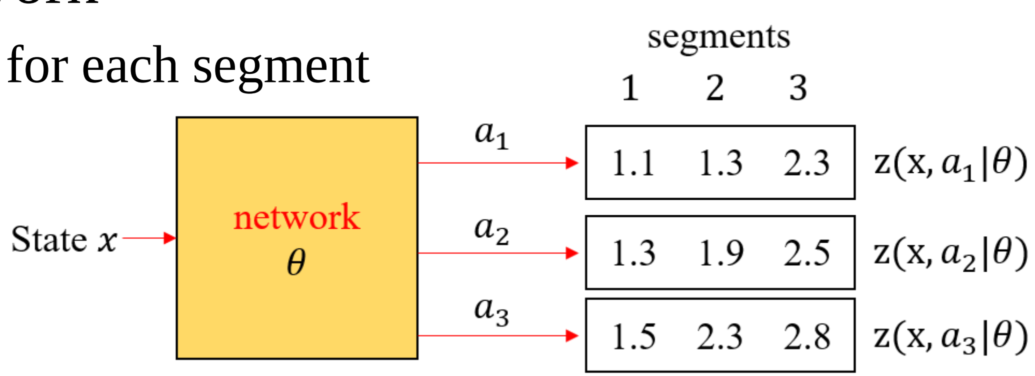


Network Architecture

- C51 network
  - Predict the **probability** of each atom for each action
  - Outputs for each action sum to 1

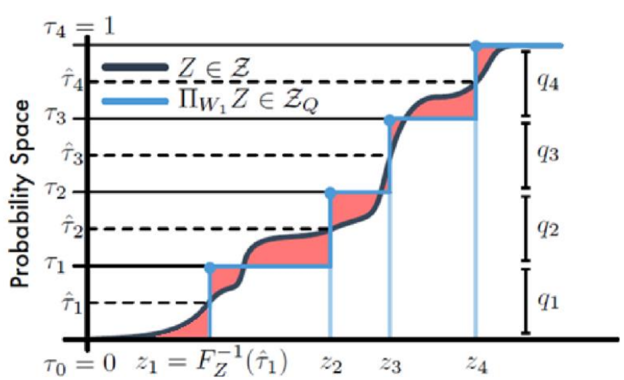


- QR-DQN network
  - Predict **value** for each segment



How to Update

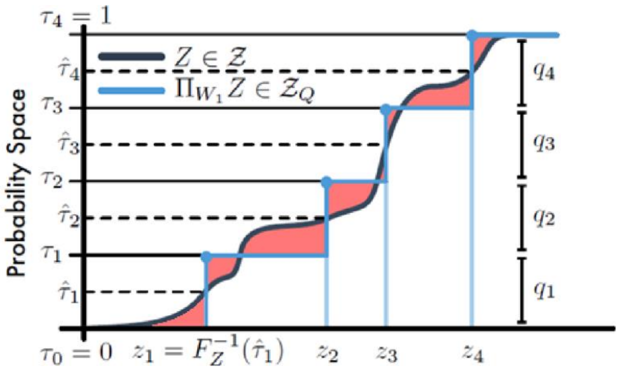
- QR-DQN predicts a value for each segment in CDF
  - What value for each segment can minimize the Wasserstein metric?
- Let
  - An arbitrary distribution Z with CDF F_Z and inverse F_Z^{-1}
  - A set of distribution Z_Q with a fixed value in each segment
- We want to find a distribution in Z_Q that minimizes the 1-Wasserstein metric
  - $\Pi_{W_1} Z := argmin_{Z_\theta \in Z_Q} W_1(Z, Z_\theta)$



Red regions:
1-Wasserstein metric

How to Update

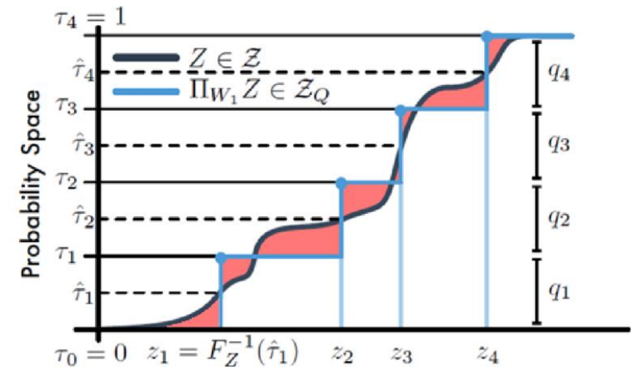
- The paper proves that for each segment between τ and τ'
  - The quantile midpoint $\hat{\tau} = \frac{\tau+\tau'}{2}$
  - $F_Z^{-1}(\hat{\tau})$ minimizes the Wasserstein metric in this segment
- $\Pi_{W_1} Z$ has a fixed value $F_Z^{-1}(\hat{\tau})$ in each segment between τ and τ'



Red regions: 1-Wasserstein metric

How to Update

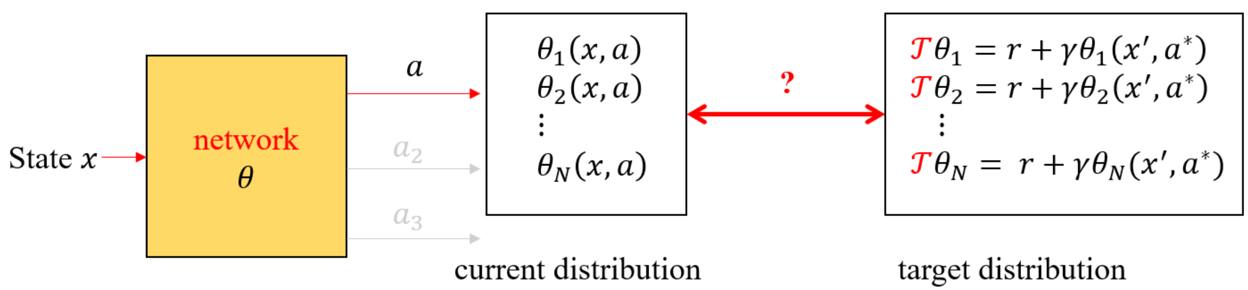
- The values that QR-DQN aims to predict is the value of quantile midpoints
- $z_i,\ i = 1, \ldots, N$: the value of quantile midpoints in Z
- Q-value: $Q(x, a) = \mathbb{E}[Z(x, a)] = \sum_i \frac{1}{N} z_i$
  - e.g. $N = 4$
  - $Q(x, a) = \frac{1}{4} z_1 + \frac{1}{4} z_2 + \frac{1}{4} z_3 + \frac{1}{4} z_4$



Red regions: 1-Wasserstein metric

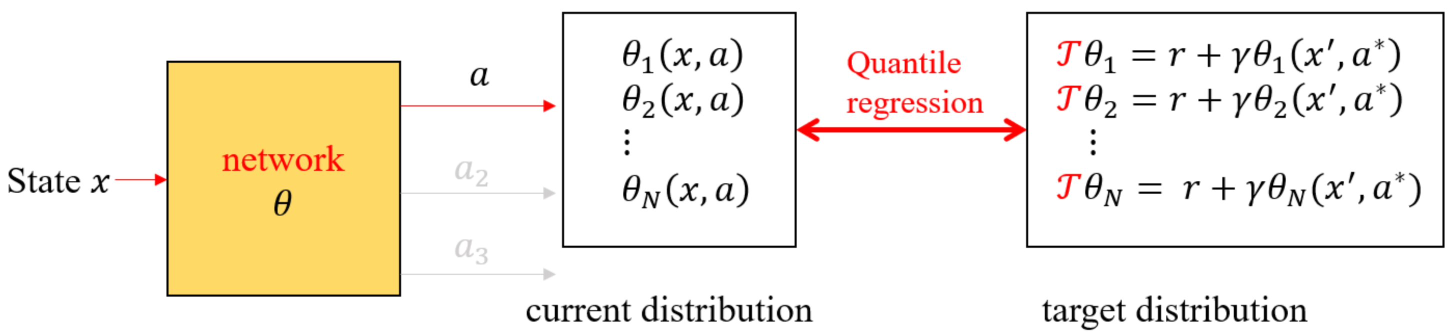
How to Update: Target Distribution

- We don’t know the real distribution
  - Get the target distribution by distributional Bellman operator $\mathcal{T}$
  - N quantile values of current distribution
    - $\theta_1(x,a), \ldots, \theta_N(x,a)$
  - N quantile values of target distribution
    - Select action $a^* = argmax_{a'} Q(x', a')$
    - $\mathcal{T}\theta_i = r + \gamma\theta_i(x', a^*)$
    - $r + \gamma\theta_1(x', a^*), \ldots, r + \gamma\theta_N(x', a^*)$



How to Update: Quantile Regression

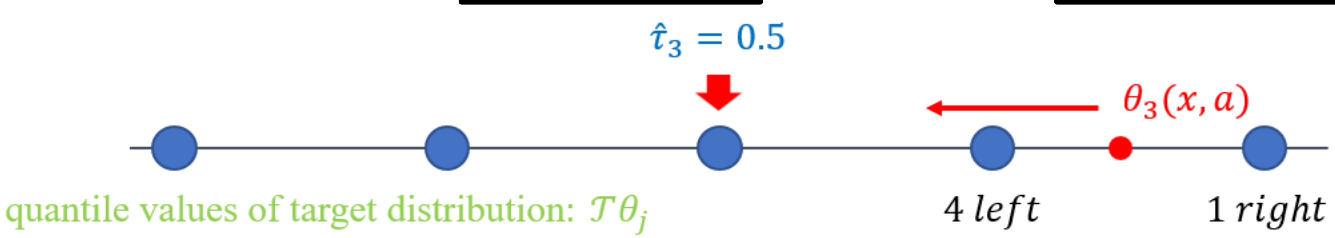
- In DQN, we learn the Q-value
  - Loss: $\left(r + \gamma \cdot Q(x', a^*) - Q(x, a)\right)^2$
- In QR-DQN, we learn the quantile distribution
  - The value of $\theta_i(x, a)$ corresponds to the quantile $\hat{\tau}_i$
  - Quantile regression



How to Update: Quantile Regression

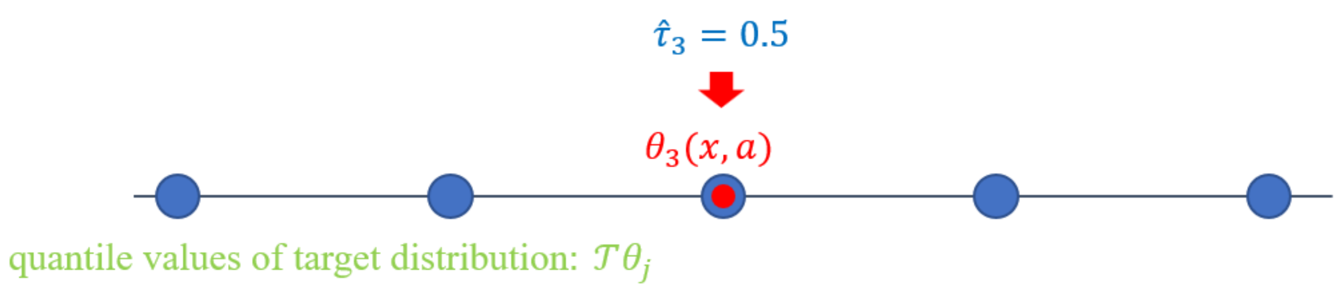
- e.g. $N = 5 \rightarrow [\hat{\tau}_1, \hat{\tau}_2, \hat{\tau}_3, \hat{\tau}_4, \hat{\tau}_5] = \left[\frac{1}{10}, \frac{3}{10}, \frac{5}{10}, \frac{7}{10}, \frac{9}{10}\right]$
  - Note: $\hat{\tau}_i$ is not $\mathcal{T}\theta_j$. (see the figure in the previous 3 slide)
- First, consider determining $\hat{\tau}_3 = 0.5$ (the median)
  - The median has minimum average distance to each points
    $\rightarrow$ $\theta_3(x, a)$ is our guess
  - Loss: $\mathrm{L} = \frac{1}{\mathrm{N}} \sum_{j=1}^{N} \left|\mathcal{T}\theta_j - \theta_3(x, a)\right| = \frac{1}{N}\left(\sum_{left}\left(\theta_3(x, a) - \mathcal{T}\theta_j\right) + \sum_{right}\left(\mathcal{T}\theta_j - \theta_3(x, a)\right)\right)$
  - Gradient: $\frac{\partial \mathrm{L}}{\partial \theta_3(x,a)} = \frac{1}{\mathrm{N}}(\# \, left - \# \, right)$

$$\begin{array}{c} \theta_1(x, a) \\ \theta_2(x, a) \\ \theta_3(x, a) \\ \theta_4(x, a) \\ \theta_5(x, a) \end{array} \longleftrightarrow \begin{array}{l} \mathcal{T}\theta_1 = r + \gamma\theta_1(x', a^*) \\ \mathcal{T}\theta_2 = r + \gamma\theta_2(x', a^*) \\ \mathcal{T}\theta_3 = r + \gamma\theta_3(x', a^*) \\ \mathcal{T}\theta_4 = r + \gamma\theta_4(x', a^*) \\ \mathcal{T}\theta_5 = r + \gamma\theta_5(x', a^*) \end{array}$$



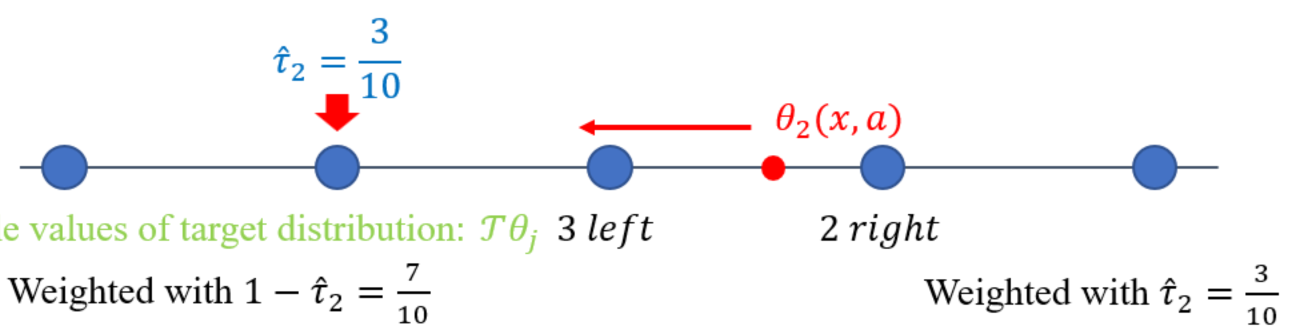
How to Update: Quantile Regression

- Gradient: $\frac{\partial \mathrm{L}}{\partial \theta_3(x,a)} = \frac{1}{\mathrm{N}}(\# \, left - \# \, right)$
- If $\theta_3(x, a)$ is on $\hat{\tau}_3 = 0.5$ (the median)
  - Move left a little $\Rightarrow$ Gradient = $\frac{1}{5}(2-3) < 0 \Rightarrow$ want to move right back
  - Move right a little $\Rightarrow$ Gradient = $\frac{1}{5}(3-2) > 0 \Rightarrow$ want to move left back
  - $\theta_3(x, a)$ on the median minimizes the loss



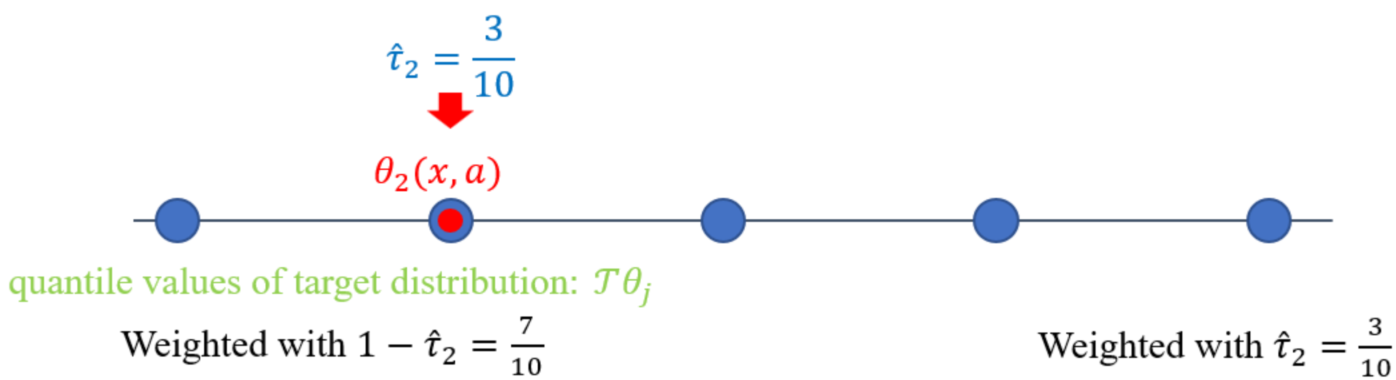
How to Update: Quantile Regression

- Generalize to any quantile $\hat{\tau}_i$
  - $\theta_i(x,a)$ is our guess
  - Give different weight of loss for left and right
    - Weight left with $1-\hat{\tau}_i$, right with $\hat{\tau}_i$
  - Loss: $\mathrm{L} = \frac{1}{N}\left((1-\hat{\tau}_i)\sum_{left}\left(\theta_i(x,a) - \mathcal{T}\theta_j\right) + (\hat{\tau}_i)\sum_{right}\left(\mathcal{T}\theta_j - \theta_i(x,a)\right)\right)$
  - Gradient: $\frac{\partial \mathrm{L}}{\partial \theta_i(x,a)} = \frac{1}{\mathrm{N}}\left((1-\hat{\tau}_i) * \#\, left - (\hat{\tau}_i) * \#\, right\right)$



How to Update: Quantile Regression

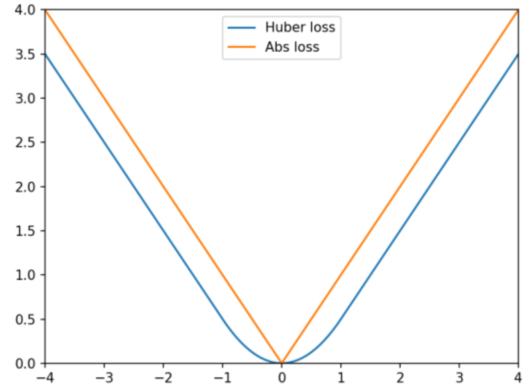
- Gradient: $\frac{\partial \mathrm{L}}{\partial \theta_2(x,a)} = \frac{1}{\mathrm{N}}\left((1-\hat{\tau}_2) * \#left - (\hat{\tau}_2) * \#\, right\right)$
- If $\theta_2(x,a)$ is on $\hat{\tau}_2 = \frac{3}{10}$
  - Move left a little $\Rightarrow$ gradient = $\frac{1}{5}\left(\frac{7}{10} * 1 - \frac{3}{10} * 4\right) < 0 \Rightarrow$ want to move right back
  - Move right a little $\Rightarrow$ gradient = $\frac{1}{5}\left(\frac{7}{10} * 2 - \frac{3}{10} * 3\right) > 0 \Rightarrow$ want to move left back
  - $\theta_2(x,a)$ on $\hat{\tau}_2$ minimizes the loss



How to Update: Quantile Regression

- Huber loss
  - Absolute value is not differentiable at 0
  - To let it differentiable at 0, replace $\left|\mathcal{T}\theta_j - \theta_i(x,a)\right|$ with the Huber loss:

$$L_\kappa\left(\mathcal{T}\theta_j - \theta_i(x,a)\right) = \begin{cases} \frac{1}{2}\left(\mathcal{T}\theta_j - \theta_i(x,a)\right)^2, if \left|\mathcal{T}\theta_j - \theta_i(x,a)\right| < \kappa \\ \kappa\left(\left|\mathcal{T}\theta_j - \theta_i(x,a)\right| - \frac{1}{2}\kappa\right), otherwise \end{cases}$$

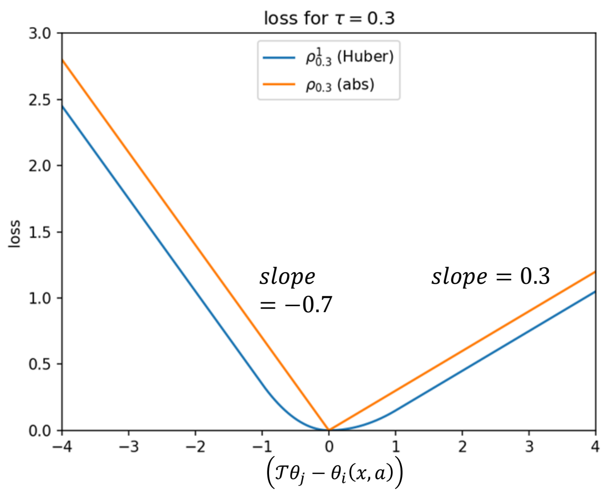


How to Update: Quantile Regression

- Let $\rho_{\tau}^{\kappa}\left(\mathcal{T}\theta_j - \theta_i(x,a)\right) = \begin{cases} (1-\hat{\tau}_i)\mathrm{L}_{\kappa}\left(\mathcal{T}\theta_j - \theta_i(x,a)\right), \text{if } \mathcal{T}\theta_j - \theta_i(x,a) < 0\ (left) \\ (\hat{\tau}_i)\mathrm{L}_{\kappa}\left(\mathcal{T}\theta_j - \theta_i(x,a)\right), \text{if } \mathcal{T}\theta_j - \theta_i(x,a) \geq 0\ (right) \end{cases}$

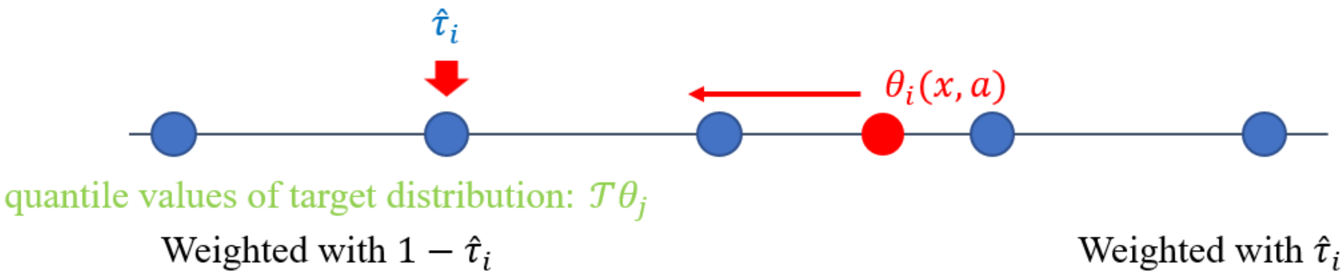
  weight loss

- The quantile Huber loss for $\hat{\tau}_i$: $L_{QR}^{\hat{\tau}_i}(\theta) = \frac{1}{N}\sum_{j=1}^{N} \rho_{\hat{\tau}_i}^{\kappa}\left(\mathcal{T}\theta_j - \theta_i(x,a)\right)$



How to Update: Quantile Regression

- N quantile values of current distribution
  - $\theta_1(x,a), \theta_2(x,a), \ldots, \theta_N(x,a)$
- N quantile values of target distribution
  - $\mathcal{T}\theta_1, \mathcal{T}\theta_2, \ldots, \mathcal{T}\theta_N$
- Total loss: $\underbrace{\sum_{i=1}^{N}}_{\text{N quantiles}} \underbrace{\frac{1}{N}\sum_{j=1}^{N}\left[\rho_{\hat{\tau}_i}^{\kappa}\left(\mathcal{T}\theta_j - \theta_i(x,a)\right)\right]}_{\text{Quantile Huber loss with target distribution for each quantile}}$



Algorithm

Algorithm 1 Quantile Regression Q-Learning

Require: N, κ
input $x, a, r, x', \gamma \in [0, 1)$
  # Compute distributional Bellman target
  $\boxed{\begin{array}{l} Q(x', a') := \sum_j q_j \theta_j(x', a') \\ a^* \leftarrow \arg\max_{a'} Q(x, a') \end{array}}$ get next action, $q_j = \frac{1}{N}$
  $\mathcal{T}\theta_j \leftarrow r + \gamma\theta_j(x', a^*), \quad \forall j$
  # Compute quantile regression loss (Equation 10)
output $\sum_{i=1}^{N} \mathbb{E}_j \left[\rho^{\kappa}_{\hat{\tau}_i}(\mathcal{T}\theta_j - \theta_i(x, a))\right]$

Algorithm

Algorithm 1 Quantile Regression Q-Learning

Require: N, κ

input $x, a, r, x', \gamma \in [0, 1)$

Compute distributional Bellman target

$Q(x', a') := \sum_j q_j \theta_j(x', a')$

$a^* \leftarrow \arg\max_{a'} Q(x, a')$

$\boxed{\mathcal{T}\theta_j \leftarrow r + \gamma\theta_j(x', a^*), \quad \forall j}$ distributional Bellman target

Compute quantile regression loss (Equation 10)

output $\sum_{i=1}^{N} \mathbb{E}_j \left[\rho^{\kappa}_{\hat{\tau}_i}(\mathcal{T}\theta_j - \theta_i(x, a))\right]$

Algorithm

Algorithm 1 Quantile Regression Q-Learning

Require: N, κ

input $x, a, r, x', \gamma \in [0, 1)$

\# Compute distributional Bellman target

$Q(x', a') := \sum_j q_j \theta_j(x', a')$

$a^* \leftarrow \arg\max_{a'} Q(x, a')$

$\mathcal{T}\theta_j \leftarrow r + \gamma\theta_j(x', a^*), \quad \forall j$

\# Compute quantile regression loss (Equation 10)

output $\sum_{i=1}^{N} \mathbb{E}_j \left[\rho^{\kappa}_{\hat{\tau}_i}(\mathcal{T}\theta_j - \theta_i(x, a))\right]$

compute loss

QR-DQN Experiment

- 57 Atari 2600 games
- QR-DQN-0: $\kappa = 0$, strict quantile loss
- QR-DQN-1: $\kappa = 1$, Huber quantile loss
- N: (10, 50, 100, 200)
- Outperforms all previous agents in mean and median human-normalized score

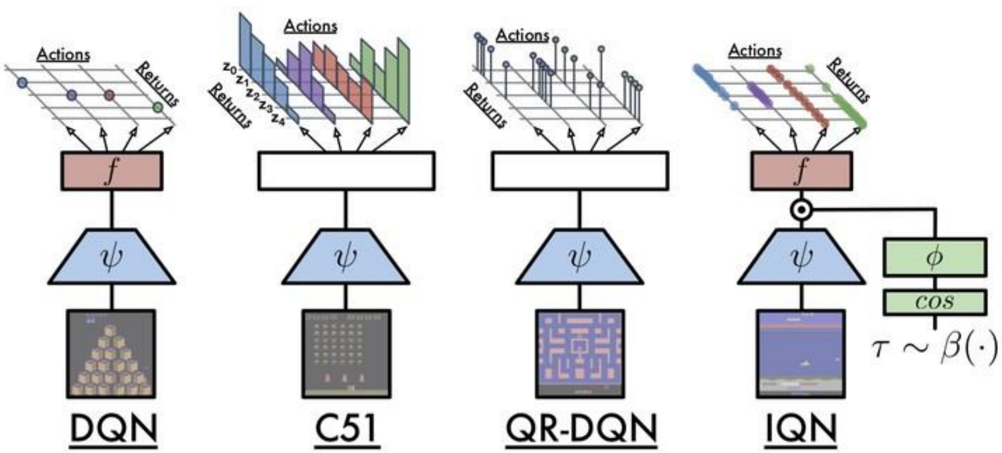
	Mean	**Median**	**>human**	**>DQN**
DQN	228%	79%	24	0
DDQN	307%	118%	33	43
DUEL.	373%	151%	37	50
PRIOR.	434%	124%	39	48
PR. DUEL.	592%	172%	39	44
C51	701%	178%	40	50
QR-DQN-0	881%	199%	38	52
QR-DQN-1	**915%**	**211%**	**41**	**54**

Distributional RL

- Introduction
- C51
- QR-DQN
- IQN
  - Implicit Quantile Networks for Distributional Reinforcement Learning
- FQF
- Summary

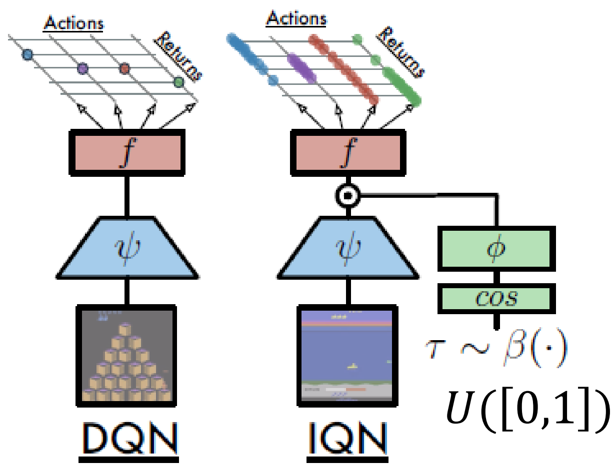
IQN

- IQN extends the approach of QR-DQN
  - QR-DQN predicts the value of some fixed quantiles
  - IQN embeds sampled quantile into NN input, and predicts its value
  - Learn the full quantile function
  - Don't have to decide the number of quantiles
- IQN designs risk-sensitive policies



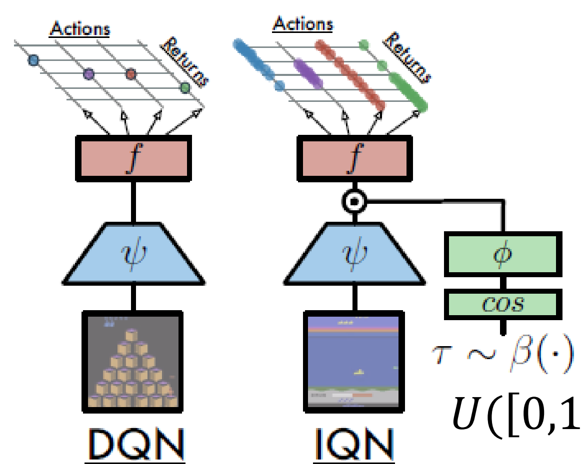
Network Architectures

- Let ψ be the function computed by the convolutional layers
- Let f be the subsequent fully-connected layers mapping $\psi(x)$ to the estimated action-values
  - $Q(x,a) \approx f\big(\psi(x)\big)_a$
- IQN uses an additional function ϕ to compute an embedding for quantile $\tau \sim \beta(\cdot)$ (e.g. uniform distribution $U([0,1])$)
- Combine these to form the approximation
  - $Z_\tau(x,a) \approx f(\psi(x) \odot \phi(\tau))_a$
  - Z_τ: the corresponding value of τ
  - $\odot$ denotes the element-wise product.



Distributional Method

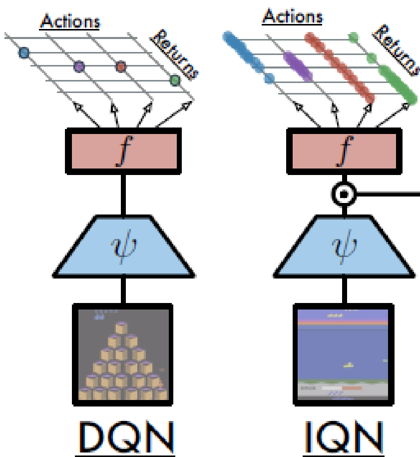
- Embedding τ into NN to construct distribution
  - A simple linear embedding was insufficient for good performance
  - They test some variants, cosine performs the best
  - Use cosine function, embedding dimension n = 64
  - $\phi_j(\tau) := ReLU(\sum_{i=0}^{n-1} \cos(\pi i \tau) w_{ij} + b_j)$



$U([0,1])$

Distributional Method

- IQN loss function: $\underbrace{\sum_{i=1}^{N}}_{\text{N quantiles}} \underbrace{\frac{1}{N'}\sum_{j=1}^{N'} \rho_{\tau_i}^{\kappa}(r+\gamma Z_{\tau_j'}(x',a') - Z_{\tau_i}(x,a\,))}_{\text{loss for each quantile}}$
  - Z_τ: the corresponding value of τ
  - τ_i: quantiles to be updated, τ_j': targets sampled
  - N, N': the number of samples of $\tau_i, \tau_j' \sim U([0,1])$



$U([0,1])$

τ_i: the corresponding quantile

sampled quantile value of current distribution: Z_{τ_i}(x,a)

sampled quantile values of target distribution: $r+\gamma Z_{\tau_j'}(x',a')$

Algorithm

Algorithm 1 Implicit Quantile Network Loss

Require: N, N', K, κ and functions β, Z

β: U([0,1])

input $x, a, r, x', \gamma \in [0, 1)$

\# Compute greedy next action

$a^* \leftarrow \arg\max_{a'} \frac{1}{K} \sum_k^K Z_{\tilde{\tau}_k}(x', a'), \quad \tilde{\tau}_k \sim \beta(\cdot)$

Choose an action that maximizes the expectation

\# Sample quantile thresholds

$\tau_i, \tau'_j \sim U([0, 1]), \quad 1 \leq i \leq N, 1 \leq j \leq N'$

\# Compute distributional temporal differences

$\delta_{ij} \leftarrow r + \gamma Z_{\tau'_j}(x', a^*) - Z_{\tau_i}(x, a), \quad \forall i, j$

\# Compute Huber quantile loss

output $\sum_{i=1}^N \mathbb{E}_{\tau'} \left[\rho^\kappa_{\tau_i}(\delta_{ij})\right]$

Algorithm

Algorithm 1 Implicit Quantile Network Loss

Require: N, N', K, κ and functions β, Z

β: U([0,1])

input $x, a, r, x', \gamma \in [0, 1)$

Compute greedy next action

$a^* \leftarrow \arg\max_{a'} \frac{1}{K} \sum_k^K Z_{\tilde{\tau}_k}(x', a'), \quad \tilde{\tau}_k \sim \beta(\cdot)$

$\boxed{\text{\# Sample quantile thresholds} \quad \tau_i, \tau'_j \sim U([0,1]), \quad 1 \le i \le N, 1 \le j \le N'}$

sample quantiles

Compute distributional temporal differences

$\delta_{ij} \leftarrow r + \gamma Z_{\tau'_j}(x', a^*) - Z_{\tau_i}(x, a), \quad \forall i, j$

Compute Huber quantile loss

output $\sum_{i=1}^N \mathbb{E}_{\tau'} \left[\rho^\kappa_{\tau_i}(\delta_{ij})\right]$

Algorithm

Algorithm 1 Implicit Quantile Network Loss

Require: N, N', K, κ and functions β, Z

input $x, a, r, x', \gamma \in [0, 1)$

Compute greedy next action

$a^* \leftarrow \arg\max_{a'} \frac{1}{K} \sum_k^K Z_{\tilde{\tau}_k}(x', a'), \quad \tilde{\tau}_k \sim \beta(\cdot)$

Sample quantile thresholds

$\tau_i, \tau'_j \sim U([0, 1]), \quad 1 \le i \le N, 1 \le j \le N'$

Compute distributional temporal differences

$\delta_{ij} \leftarrow r + \gamma Z_{\tau'_j}(x', a^*) - Z_{\tau_i}(x, a), \quad \forall i, j$

Compute Huber quantile loss

output $\sum_{i=1}^{N} \mathbb{E}_{\tau'} \left[\rho^{\kappa}_{\tau_i}(\delta_{ij})\right]$

β: U([0,1])

compute distributional temporal difference

Algorithm

Algorithm 1 Implicit Quantile Network Loss

Require: N, N', K, κ and functions β, Z

input $x, a, r, x', \gamma \in [0, 1)$

Compute greedy next action

$a^* \leftarrow \arg\max_{a'} \frac{1}{K} \sum_k^K Z_{\tilde{\tau}_k}(x', a'), \quad \tilde{\tau}_k \sim \beta(\cdot)$

Sample quantile thresholds

$\tau_i, \tau'_j \sim U([0, 1]), \quad 1 \leq i \leq N, 1 \leq j \leq N'$

Compute distributional temporal differences

$\delta_{ij} \leftarrow r + \gamma Z_{\tau'_j}(x', a^*) - Z_{\tau_i}(x, a), \quad \forall i, j$

Compute Huber quantile loss

output $\sum_{i=1}^N \mathbb{E}_{\tau'} \left[\rho^\kappa_{\tau_i}(\delta_{ij})\right]$

β: U([0,1])

compute loss

IQN Experiment

- 57 Atari games
- Outperforms QR-DQN

	Mean	Median
DQN	228%	79%
PRIOR.	434%	124%
C51	701%	178%
RAINBOW	**1189%**	**230%**
QR-DQN	864%	193%
IQN	1019%	218%

Rainbow is a method that combines extensions of DQN:
Double Q-learning + Prioritized replay + Dueling networks + Multi-step learning (n-step) + Distributional RL (C51) + Noisy Nets

Effect of The Number of Samples

- $N = N' = 8$ appears to be sufficient to achieve the majority of improvements for long-term performance
- Higher N, N' are better, but not sensitive
  - So, fixed it at 32 for all experiments.
  - Better than DQN even for $N = N' = 1$ (surprisingly)

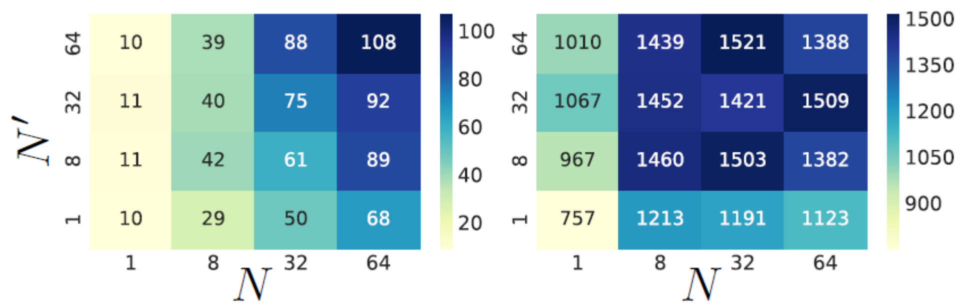


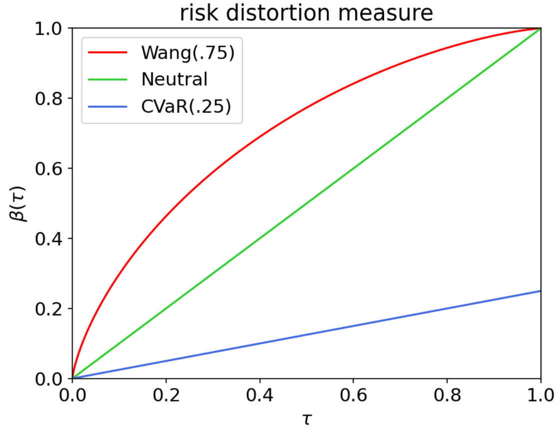
Figure 2. Effect of varying N and N', the number of samples used in the loss function in Equation 3. Figures show human-normalized agent performance, averaged over six Atari games, averaged over first 10M frames of training (left) and last 10M frames of training (right). Corresponding values for baselines: DQN (32, 253) and QR-DQN (144, 1243).

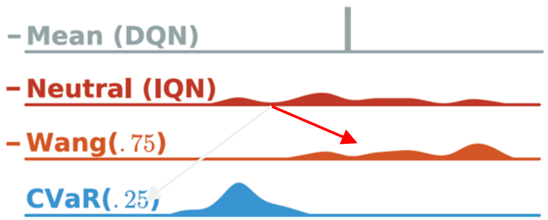
Risk

- The policy used was based entirely on the mean of the return distribution
- Risk-sensitive policies
  - Using information provided by the distribution over returns
  - Risk distortion measure

Risk Distortion Measure

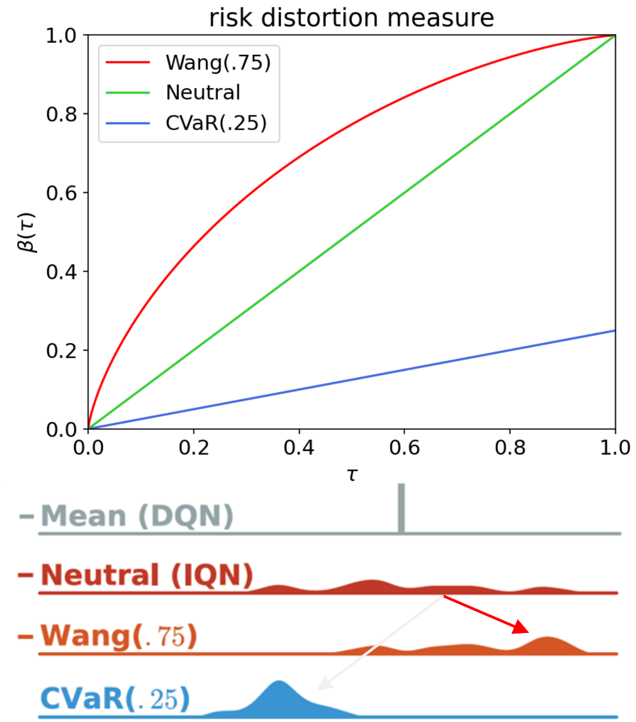
- Change the sampling distribution for $\tau \sim U([0,1])$ and compute its expectation
- Let $\beta: [0,1] \rightarrow [0,1]$ be a risk distortion measure
- The expectation: $Q_\beta(x,a) \coloneqq \mathbb{E}_{\tau \sim U([0,1])}[Z_{\beta(\tau)}(x,a)]$
- Risk-neutral
  - e.g. Neutral: $\beta(\tau) = \tau$
  - Original distribution
- Risk-seeking (optimistic)
  - e.g. Wang(.75): $\beta(\tau) \geq \tau$
  - The values of higher quantiles dominate the expectation
- Risk-averse (pessimistic)
  - e.g. CVaR(.25): $\beta(\tau) \leq \tau$
  - The values of lower quantiles dominate the expectation





Risk Distortion Measure

- e.g. Cumulative distribution of two actions
  (A) get 5 score in $[0, 1.0]$
  (B) get 0 score in $[0, 0.5)$, get 10 score in $[0.5, 1.0]$
  If we sample two quantiles: $\tau_1 = 0.4$ and $\tau_2 = 0.8$
  - Risk-neutral: Neutral
    - $\beta(\tau_1) = 0.4, \beta(\tau_2) = 0.8$
      (A) two 5 score
      (B) one 10 score and one 0 score
  - Risk-seeking (optimistic): Wang(.75)
    - $\beta(\tau_1) \approx 0.69, \beta(\tau_2) \approx 0.94$
      (A) two 5 score
      (B) two 10 score
  - Risk-averse (pessimistic): CVaR(.25)
    - $\beta(\tau_1) = 0.1, \beta(\tau_2) = 0.2$
      (A) two 5 score
      (B) two 0 score



Algorithm

Algorithm 1 Implicit Quantile Network Loss

Require: N, N', K, κ and functions β, Z

input $x, a, r, x', \gamma \in [0, 1)$

Compute greedy next action

$\boxed{a^* \leftarrow \arg\max_{a'} \frac{1}{K} \sum_k^K Z_{\tilde{\tau}_k}(x', a'), \quad \tilde{\tau}_k \sim \beta(\cdot)}$

Sample quantile thresholds

$\tau_i, \tau'_j \sim U([0, 1]), \quad 1 \leq i \leq N, 1 \leq j \leq N'$

Compute distributional temporal differences

$\delta_{ij} \leftarrow r + \gamma Z_{\tau'_j}(x', a^*) - Z_{\tau_i}(x, a), \quad \forall i, j$

Compute Huber quantile loss

output $\sum_{i=1}^N \mathbb{E}_{\tau'} \left[\rho^\kappa_{\tau_i}(\delta_{ij})\right]$

β: a risk distortion measure

Choose an action that maximizes the distorted expectation

risk-seeking: higher quantiles dominate ⇒ optimistic

risk-averse: lower quantiles dominate ⇒ pessimistic

Effects of Sampling Distribution

- Risk-seeking policy significantly underperforms the risk-neutral policy on three of the six games.

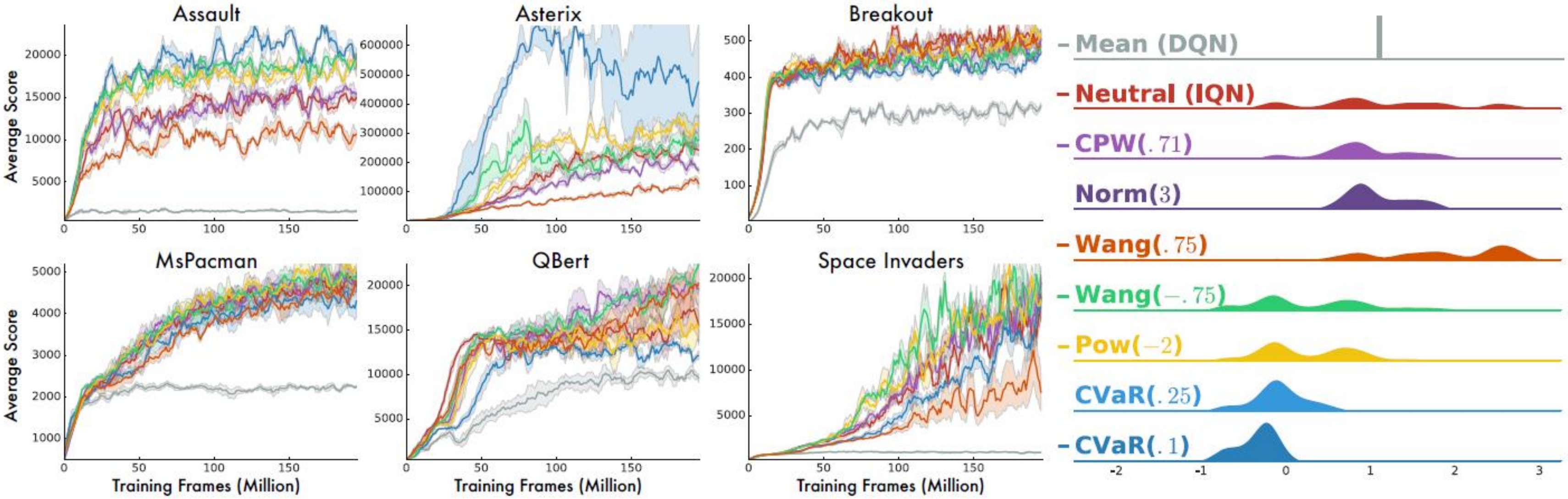


Figure 3. Effects of various changes to the sampling distribution, that is various cumulative probability weightings.

Distributional RL

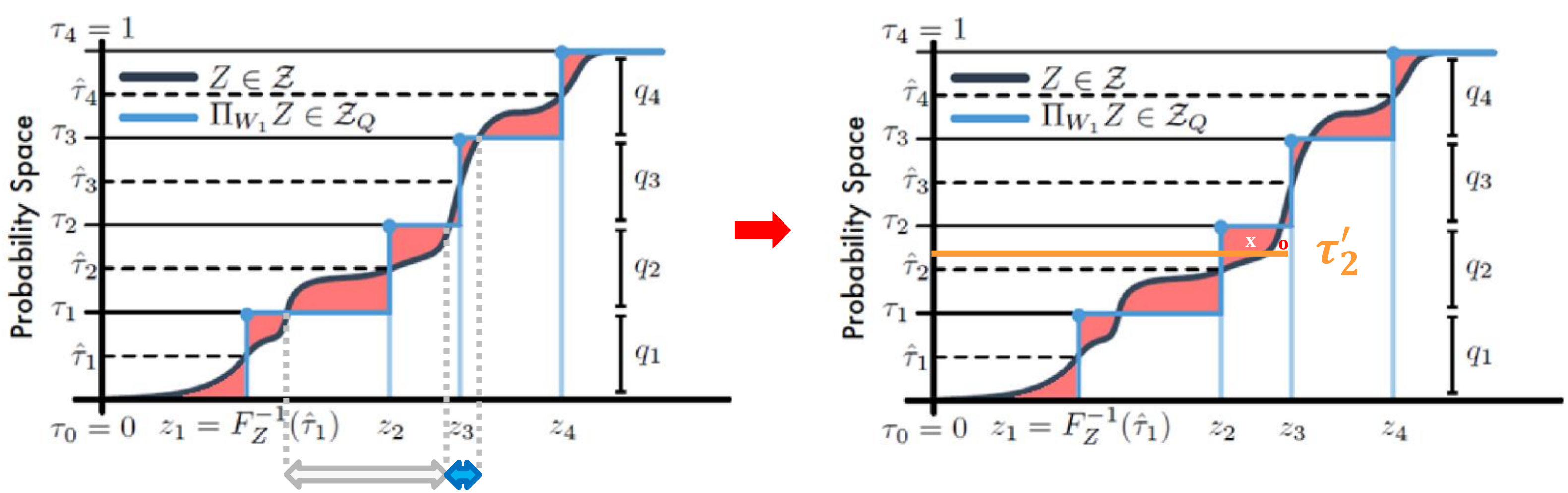
- Introduction
- C51
- QR-DQN
- IQN
- FQF
  - Fully Parameterized Quantile Function for Distributional Reinforcement Learning
- Summary

Fully Parameterized Quantile Function for Distributional Reinforcement Learning

FQF

Drawback of QR-DQN

- For fixed number of quantiles
  - The range of each segment is fixed, can't further minimize the Wasserstein metric
  - e.g. The value of Z increases a lot in $[\tau_1, \tau_2]$, but increases a little in $[\tau_2, \tau_3]$
  - Adjust the position of τ_2 can further minimize the Wasserstein metric

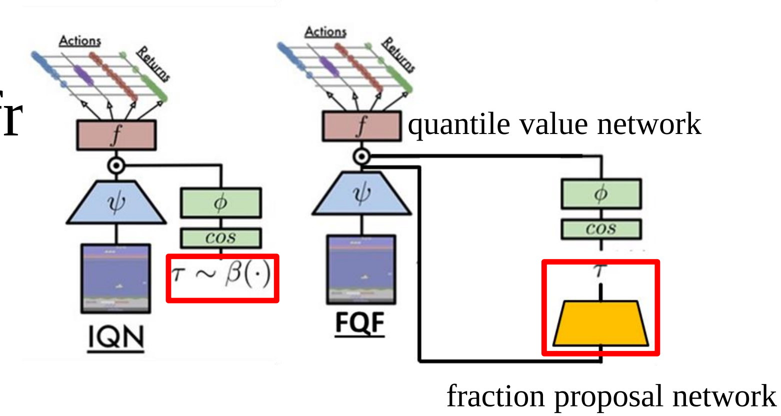


$F_Z^{-1}(\hat{\tau}_2)$ $F_Z^{-1}(\hat{\tau}_3)$ Red regions: Wasserstein metric

FQF

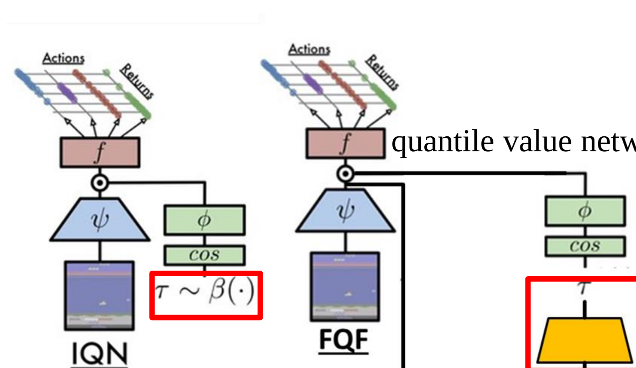
- IQN is able to approximate the full quantile function
  - In practice, the number of sampled quantile fractions is limited
  - No guarantee in providing better quantile function approximation
- FQF parameterizes both quantile fr corresponding quantile values

  1. Fraction proposal network
     - Generates a discrete set of quantile fractions
     - Aims to better utilize quantile fractions
  2. Quantile value network
     - Gives corresponding quantile values
     - Similar to IQN



Fraction Proposal Network

- Take the state embedding of original IQN as input and generate quantile fractions
- One fully-connected MLP layer
  - Use cumulated softmax to ensure the output is sorted
  - e.g. Softmax layer output: [0.1, 0.3, 0.2, 0.3, 0.1]
  - Quantile fractions: [0, 0.1, 0.4, 0.6, 0.9, 1.0]

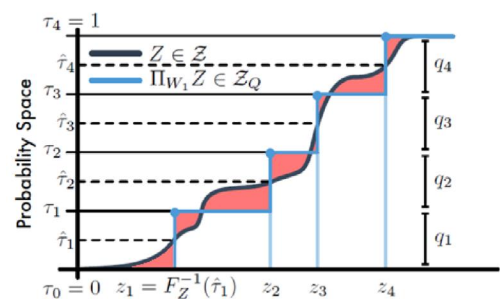


quantile value network

fraction proposal network

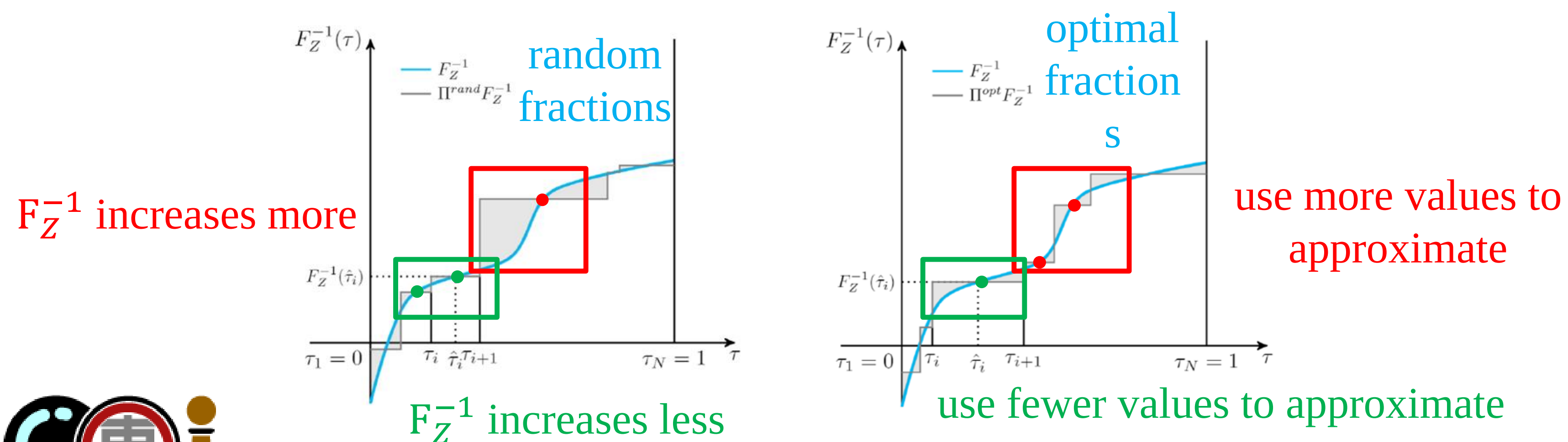
Fraction Proposal Network

- Consider a set of quantile fractions $\tau_0, \dots \tau_N \in [0,1]$
  - Predict a value for each segment between τ_i and $\tau_{i+1}, 0 \le i < N$
  - QR-DQN paper proves that
    - The quantile midpoint $\hat{\tau}_i = \frac{\tau_i + \tau_{i+1}}{2}$
    - $F_Z^{-1}(\hat{\tau}_i)$ minimizes the Wasserstein metric in this segment
- Use quantile value network to approximate $F_Z^{-1}(\hat{\tau})$
- w_1: fraction proposal network, w_2: quantile value network
  - Q-value: $Q(x,a) = \sum_{i=0}^{N-1} \underbrace{(\tau_{i+1} - \tau_i)}_{\text{segment between fractions}} \underbrace{F_{Z,w_2}^{-1}(\hat{\tau}_i)}_{\text{value of } \hat{\tau}_i}$



Fraction Proposal Network

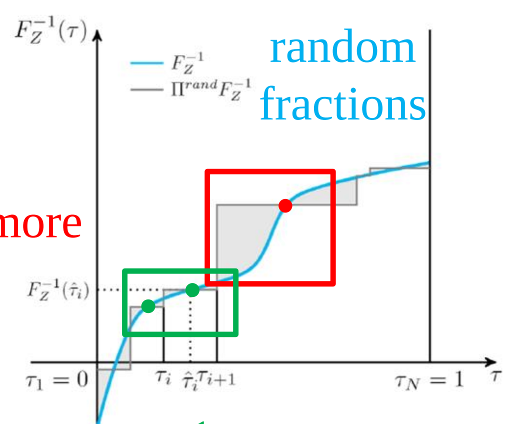
- For a set of quantile fractions, the Wasserstein metric is minimized by:
  - Assign the value of quantile midpoints to each segment
- Different sets of quantile fractions may have different Wasserstein metric
- Find a set of quantile fractions that minimizes the Wasserstein metric



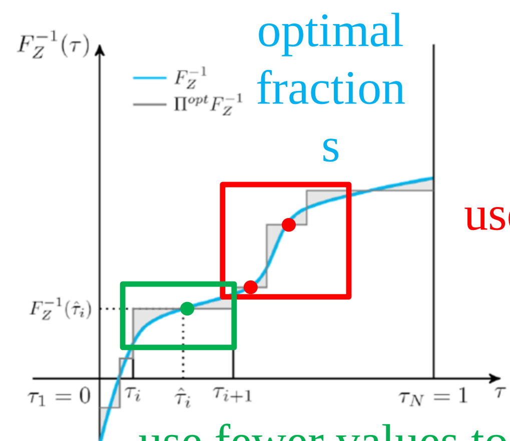
Fraction Proposal Network

- Find a set of quantile fractions that minimizes the Wasserstein metric
  - Loss: computing the Wasserstein metric without bias is usually impractical (needs integral)
  - Minimize the Wasserstein metric by iteratively applying gradient descent without computing it
  - Gradient descent: $\frac{\partial W_1}{\partial \tau_i} = 2F_Z^{-1}(\tau_i) - F_Z^{-1}(\hat{\tau}_i) - F_Z^{-1}(\hat{\tau}_{i-1})$

F_Z^{-1} increases more



F_Z^{-1} increases less

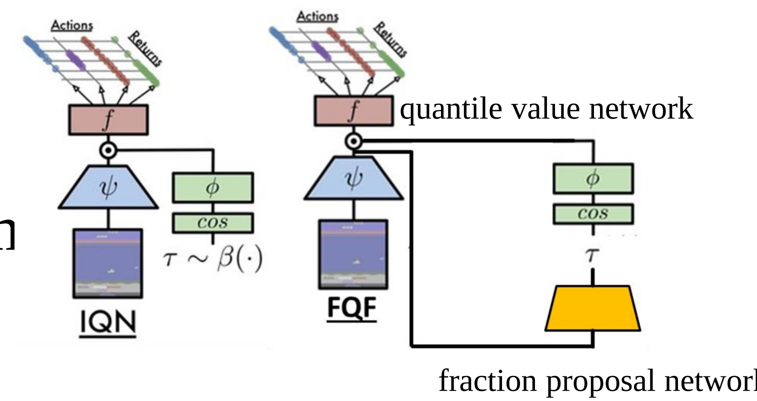


use more values to approximate

use fewer values to approximate

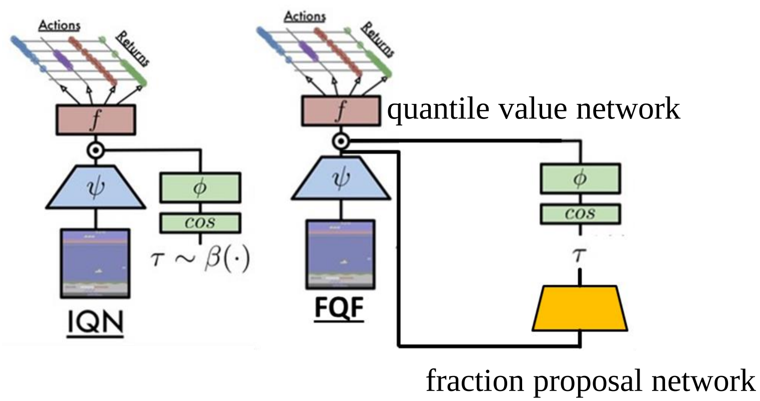
Quantile Value Network

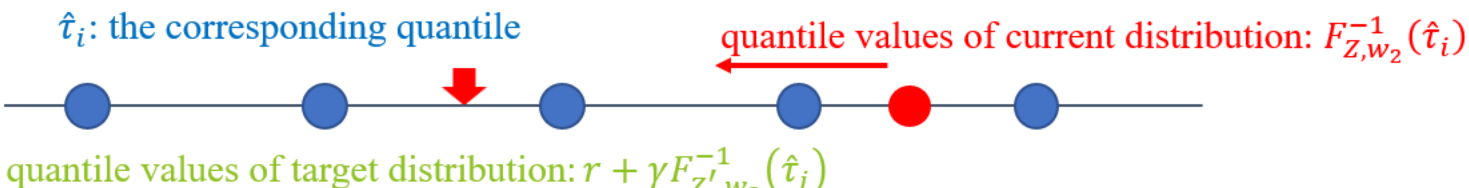
- Similar to IQN
- ψ: The function computed by the convolutional layers
- f: quantile value network
- ϕ: The embedding
- Combine these to form the approxim
  - $F_Z^{-1}(\tau) \approx F_{Z,w_2}^{-1}(\psi(x) \odot \phi(\tau))$
  - $\odot$ denotes the element-wise product



Quantile Value Network

- Loss function
  - $\mathrm{L}(x,a,r,x') = \underbrace{\sum_{i=0}^{N-1}\frac{1}{N}}_{\text{N quantiles}} \underbrace{\sum_{j=0}^{N-1}\rho^{\kappa}_{\hat{\tau}_i}\left(r+\gamma F^{-1}_{Z',w_2}(\hat{\tau}_j) - F^{-1}_{Z,w_2}(\hat{\tau}_i)\right)}_{\text{loss for each quantile}}$
  - $\hat{\tau}$: quantile midpoints from fraction proposal network
  - N: the number of quantile midpoints
  - F^{-1}_{Z,w_2} and its Bellman target shares the same quantiles to reduce computation





Algorithm

Algorithm 1: FQF update

Parameter : N, κ
Input: $x, a, r, x', \gamma \in [0, 1)$
// Compute proposed fractions for x, a
$\tau \leftarrow P_{w_1}(x, a)$;
// Compute proposed fractions for x', a'
for $a' \in \mathcal{A}$ **do**
  $\tau^{a'} \leftarrow P_{w_1}(x', a')$;
end
// Compute greedy action
$Q(s', a') \leftarrow \sum_{i=0}^{N-1}(\tau_{i+1}^{a'} - \tau_i^{a'})F_{Z',w_2}^{-1}(\hat{\tau}_i^{a'})$;
$a^* \leftarrow \underset{a'}{\text{argmax}} Q(s', a')$;
// Compute L
for $0 \le i \le N-1$ **do**
  for $0 \le j \le N-1$ **do**
    $\delta_{ij} \leftarrow r + \gamma F_{Z',w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_j)$
  end
end
$\mathcal{L} = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\rho_{\hat{\tau}_j}^{\kappa}(\delta_{ij})$;
// Compute $\frac{\partial W_1}{\partial \tau_i}$ for $i \in [1, N-1]$
$\frac{\partial W_1}{\partial \tau_i} = 2F_{Z,w_2}^{-1}(\tau_i) - F_{Z,w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_{i-1})$;
Update w_1 with $\frac{\partial W_1}{\partial \tau_i}$; Update w_2 with $\nabla\mathcal{L}$;
Output: Q

fraction proposal network

Algorithm

Algorithm 1: FQF update

Parameter : N, κ
Input: $x, a, r, x', \gamma \in [0, 1)$
// Compute proposed fractions for x, a
$\tau \leftarrow P_{w_1}(x, a)$;
// Compute proposed fractions for x', a'
for $a' \in \mathcal{A}$ **do**
  $\tau^{a'} \leftarrow P_{w_1}(x', a')$;
end
// Compute greedy action
$Q(s', a') \leftarrow \sum_{i=0}^{N-1}(\tau_{i+1}^{a'} - \tau_i^{a'})F_{Z',w_2}^{-1}(\hat{\tau}_i^{a'})$;
$a^* \leftarrow \underset{a'}{\operatorname{argmax}} Q(s', a')$;

get next action

// Compute $\mathcal{L}$
for $0 \leq i \leq N-1$ **do**
  for $0 \leq j \leq N-1$ **do**
    $\delta_{ij} \leftarrow r + \gamma F_{Z',w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_j)$
  end
end
$\mathcal{L} = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\rho_{\hat{\tau}_j}^{\kappa}(\delta_{ij})$;
// Compute $\frac{\partial W_1}{\partial \tau_i}$ for $i \in [1, N-1]$
$\frac{\partial W_1}{\partial \tau_i} = 2F_{Z,w_2}^{-1}(\tau_i) - F_{Z,w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_{i-1})$;
Update w_1 with $\frac{\partial W_1}{\partial \tau_i}$; Update w_2 with $\nabla \mathcal{L}$;
Output: Q

Algorithm

Algorithm 1: FQF update

Parameter : N, κ
Input: $x, a, r, x', \gamma \in [0, 1)$
`// Compute proposed fractions for` x, a
$\tau \leftarrow P_{w_1}(x, a)$;
`// Compute proposed fractions for` x', a'
for $a' \in \mathcal{A}$ **do**
  $\tau^{a'} \leftarrow P_{w_1}(x', a')$;
end
`// Compute greedy action`
$Q(s', a') \leftarrow \sum_{i=0}^{N-1}(\tau_{i+1}^{a'} - \tau_i^{a'})F_{Z',w_2}^{-1}(\hat{\tau}_i^{a'})$;
$a^* \leftarrow \underset{a'}{\text{argmax}} Q(s', a')$;
`// Compute` $\mathcal{L}$
for $0 \le i \le N-1$ **do**
  for $0 \le j \le N-1$ **do**
    $\delta_{ij} \leftarrow r + \gamma F_{Z',w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_j)$
  end
end
$\mathcal{L} = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\rho_{\hat{\tau}_j}^{\kappa}(\delta_{ij})$;
`// Compute` $\frac{\partial W_1}{\partial \tau_i}$ `for` $i \in [1, N-1]$
$\frac{\partial W_1}{\partial \tau_i} = 2F_{Z,w_2}^{-1}(\tau_i) - F_{Z,w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_{i-1})$;
Update w_1 with $\frac{\partial W_1}{\partial \tau_i}$; Update w_2 with $\nabla\mathcal{L}$;
Output: Q

loss for quantile value network

Algorithm

Algorithm 1: FQF update

Parameter : N, κ
Input: $x, a, r, x', \gamma \in [0, 1)$
// Compute proposed fractions for x, a
$\tau \leftarrow P_{w_1}(x, a)$;
// Compute proposed fractions for x', a'
for $a' \in \mathcal{A}$ **do**
  $\tau^{a'} \leftarrow P_{w_1}(x', a')$;
end
// Compute greedy action
$Q(s', a') \leftarrow \sum_{i=0}^{N-1}(\tau_{i+1}^{a'} - \tau_i^{a'})F_{Z',w_2}^{-1}(\hat{\tau}_i^{a'})$;
$a^* \leftarrow \underset{a'}{\mathrm{argmax}} Q(s', a')$;
// Compute L
for $0 \leq i \leq N-1$ **do**
  for $0 \leq j \leq N-1$ **do**
    $\delta_{ij} \leftarrow r + \gamma F_{Z',w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_j)$
  end
end
$\mathcal{L} = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\rho_{\hat{\tau}_j}^{\kappa}(\delta_{ij})$;
// Compute $\frac{\partial W_1}{\partial \tau_i}$ for $i \in [1, N-1]$
$\frac{\partial W_1}{\partial \tau_i} = 2F_{Z,w_2}^{-1}(\tau_i) - F_{Z,w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_{i-1})$;
Update w_1 with $\frac{\partial W_1}{\partial \tau_i}$; Update w_2 with $\nabla \mathcal{L}$;
Output: Q

gradient for fraction proposal network

Algorithm

Algorithm 1: FQF update

Parameter : N, κ

Input: $x, a, r, x', \gamma \in [0, 1)$

```
// Compute proposed fractions for x, a
```
$\tau \leftarrow P_{w_1}(x, a)$;

```
// Compute proposed fractions for x', a'
```
for $a' \in \mathcal{A}$ **do**

  $\tau^{a'} \leftarrow P_{w_1}(x', a')$;

end

```
// Compute greedy action
```
$Q(s', a') \leftarrow \sum_{i=0}^{N-1}(\tau_{i+1}^{a'} - \tau_i^{a'})F_{Z',w_2}^{-1}(\hat{\tau}_i^{a'})$;

$a^* \leftarrow \underset{a'}{\mathrm{argmax}} Q(s', a')$;

```
// Compute L
```
for $0 \leq i \leq N - 1$ **do**

  for $0 \leq j \leq N - 1$ **do**

    $\delta_{ij} \leftarrow r + \gamma F_{Z',w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_j)$

  end

end

$\mathcal{L} = \frac{1}{N} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \rho_{\hat{\tau}_j}^{\kappa}(\delta_{ij})$;

```
// Compute ∂W1/∂τi for i ∈ [1, N − 1]
```
$\frac{\partial W_1}{\partial \tau_i} = 2F_{Z,w_2}^{-1}(\tau_i) - F_{Z,w_2}^{-1}(\hat{\tau}_i) - F_{Z,w_2}^{-1}(\hat{\tau}_{i-1})$;

Update w_1 with $\frac{\partial W_1}{\partial \tau_i}$; Update w_2 with $\nabla \mathcal{L}$;

Output: Q

update network

FQF Experiment

- 55 Atari games (except Surround and Defender)
- Roughly 20% slower than IQN due to the additional fraction proposal network

	Mean	Median	>Human	>DQN
DQN	221%	79%	24	0
PRIOR.	580%	124%	39	48
C51	701%	178%	40	50
RAINBOW	1213%	227%	42	52
QR-DQN	902%	193%	41	54
IQN	1112%	218%	39	54
FQF	**1426%**	**272%**	**44**	**54**

Rainbow is a method that combines extensions of DQN:
Double Q-learning + Prioritized replay + Dueling networks + Multi-step learning (n-step) + Distributional RL (C51) + Noisy Nets

SUMMARY

Distributional Reinforcement Learning

- Distributional RL aims to **learn the value distribution instead of the expectation**
- Significantly improve Atari-57 performance

	Mean	Median	>Human	>DQN
DQN	221%	79%	24	0
PRIOR.	580%	124%	39	48
C51	701%	178%	40	50
RAINBOW	1213%	227%	42	52
QR-DQN	902%	193%	41	54
IQN	1112%	218%	39	54
FQF	**1426%**	**272%**	**44**	**54**

Rainbow is a method that combines extensions of DQN:
Double Q-learning + Prioritized replay + Dueling networks + Multi-step learning (n-step) + Distributional RL (C51) + Noisy Nets

Illustration