Rainbow DQN

온성권

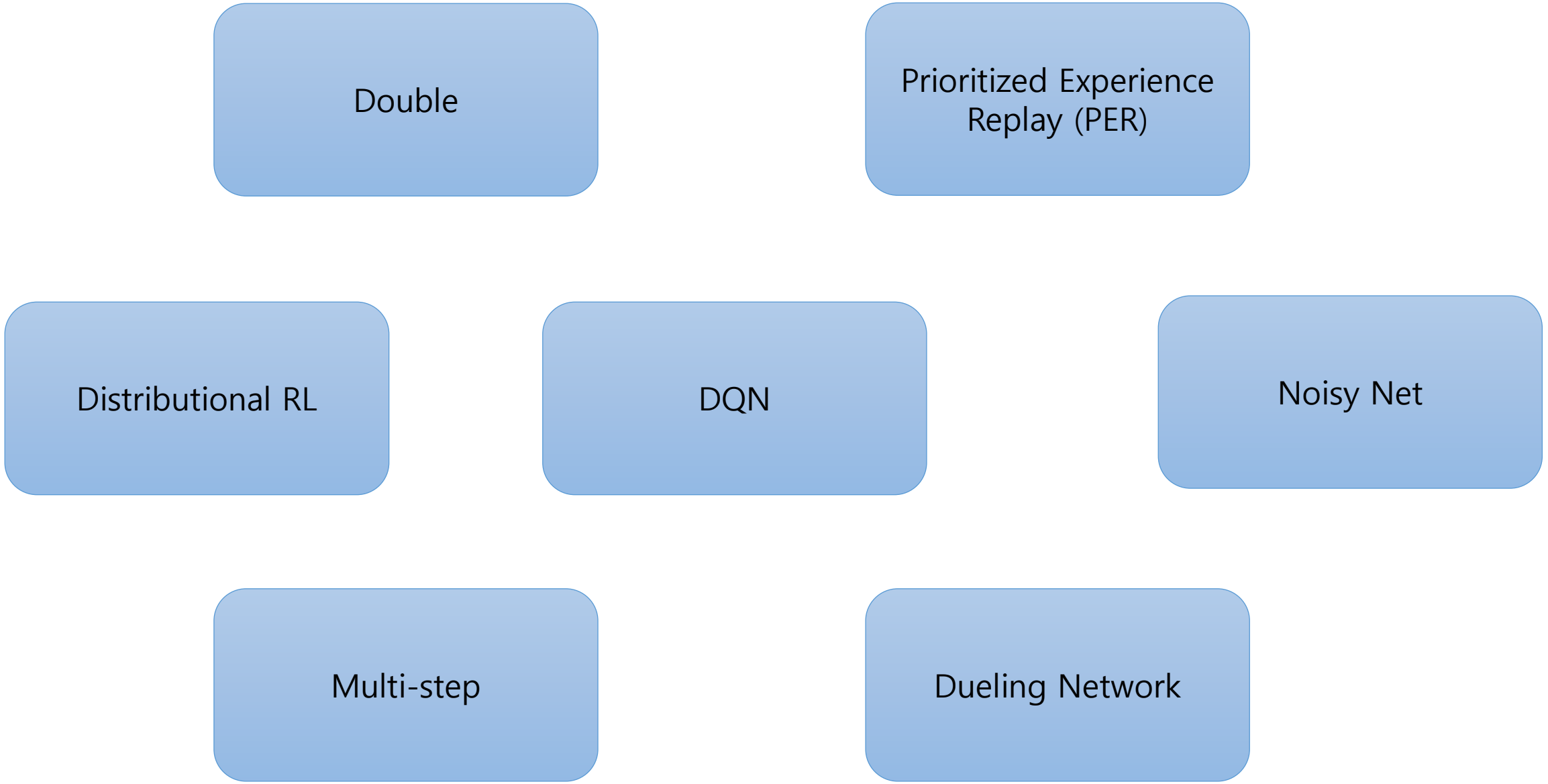
2020 November

First impression

- 짬뽕같은 논문이다……
- 그래프 색감이 좋군!
- Appendix에 그래프 양 문제있어?

Introduction

- Ever since DQN first stroke this world, many extensions have been proposed to enhance its learning speed, converging score or/and stability
- The selection of **6** extension algorithms (which has the most promising results) of DQN is combined with the DQN itself and thus it is called the "***Rainbow***" DQN


Double
Prioritized Experience Replay (PER)
Distributional RL
DQN
Noisy Net
Multi-step
Dueling Network

DQN

Ordinary Q-learning algorithm combined with some spectacular features:

- **CNN** for efficient feature extraction of states
- **Experience replay** to help off-line learning
- **Separate target network** for stable training
- Deep learning suited **optimization methods**, RMSprop(2013) & Adam(2016~)

Loss:

$$(R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t))^2$$

Double

- Due to maximization step in DQNs loss function, overestimation bias may arise, which harms learning.
- Double DQN selects action that has the maximum Q value, and the Q-value of the action is used in the loss, hance the name, double! This reduces the bias!

$$(R_{t+1} + \gamma_{t+1} q_{\overline{\theta}}(S_{t+1}, \underset{a'}{\operatorname{argmax}}\, q_\theta(S_{t+1}, a')) - q_\theta(S_t, A_t))^2$$

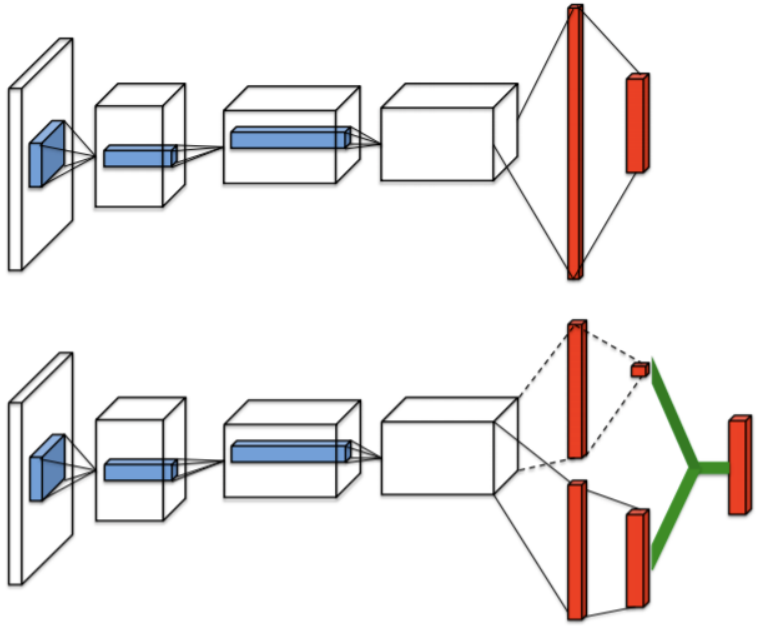
Prioritized Experience Replay (PER)

- All past transitions carrying same weight might be waste of time for the agents point of view.
- By giving weights to transitions that are more significant, the agent can focus more on the “important” past transitions and reuse them to learn more frequently.
- The weights are given proportional to their TD error.

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\overline{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t) \right|^\omega$$

Dueling Network

- Splits the final layer of the network into two sections.
- One estimating value function
- One estimating Advantage for each actions
- This structure helps in generalizing across actions



$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha) \right)$$

Multi-step

- Q-learning only accumulates a single reward, then bootstrap.
- Here, we introduce a forward-view multi-step (n-step) target return;

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1}$$

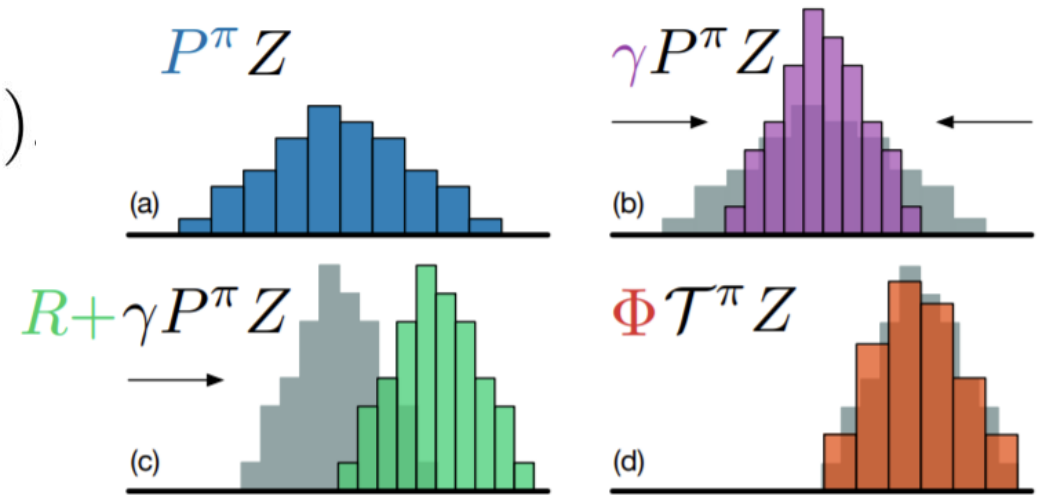
$$(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_{\overline{\theta}}(S_{t+n}, a') - q_\theta(S_t, A_t))^2$$

Distributional RL

- Network learns distribution of returns (treating as a random variable) instead of fixed Q-values.
- Loss is computed as KL-divergence of the target and current distribution of returns.
- Hence, loss is in a form of cross-entropy.

$$d'_t \equiv (R_{t+1} + \gamma_{t+1}\boldsymbol{z},\ \ \boldsymbol{p}_{\overline{\theta}}(S_{t+1}, \overline{a}^*_{t+1}))$$

$$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d'_t || d_t)$$



Noisy Net

- To overcome limitations of exploring in some ALE environments, add stochastic layers in the network.

$$\boldsymbol{y} = (\boldsymbol{b} + \mathbf{W}\boldsymbol{x}) + (\boldsymbol{b}_{noisy} \odot \epsilon^{b} + (\mathbf{W}_{noisy} \odot \epsilon^{w})\boldsymbol{x})$$

The Integrated Agent (Dueling & Noisy net)

- Simply add DEULING & NOISY NET by changing the form of the network.

- Others are slightly trickier~~
- Since others need to deal with loss functions

The Integrated Agent (Multi-step & Distributional & Double)

- Add multi-step to distributional RL by defining target distribution as a n-step look-ahead returns;

$$d_t^{(n)} = (R_t^{(n)} + \gamma_t^{(n)} \boldsymbol{z}, \quad \boldsymbol{p}_{\bar{\theta}}(S_{t+n}, a^*_{t+n}))$$

- Hence, the loss becomes;

$$D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t)$$

- Double is incorporated in a process of selecting action with max Q within Distributional RL

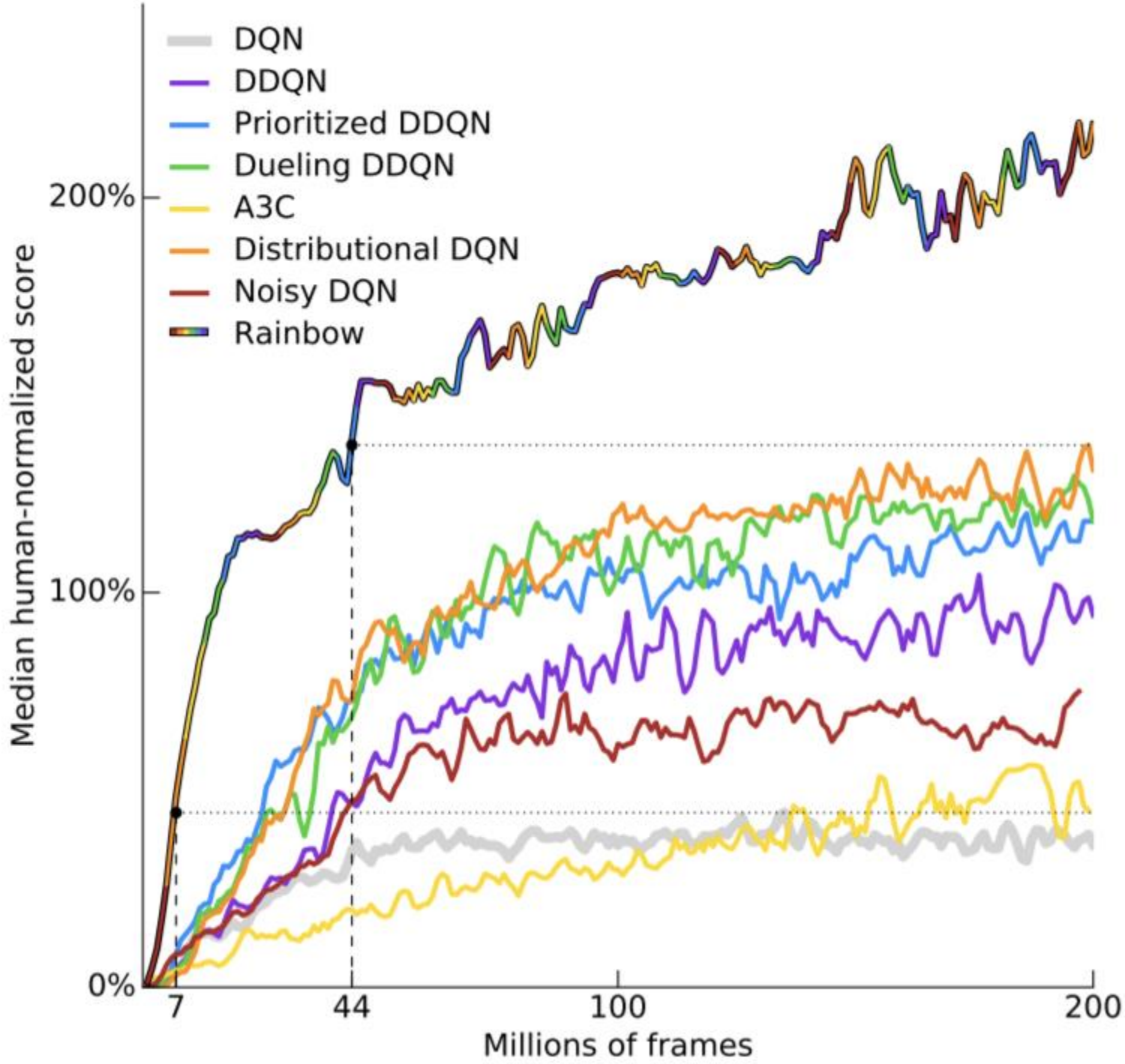
The Integrated Agent (PER)

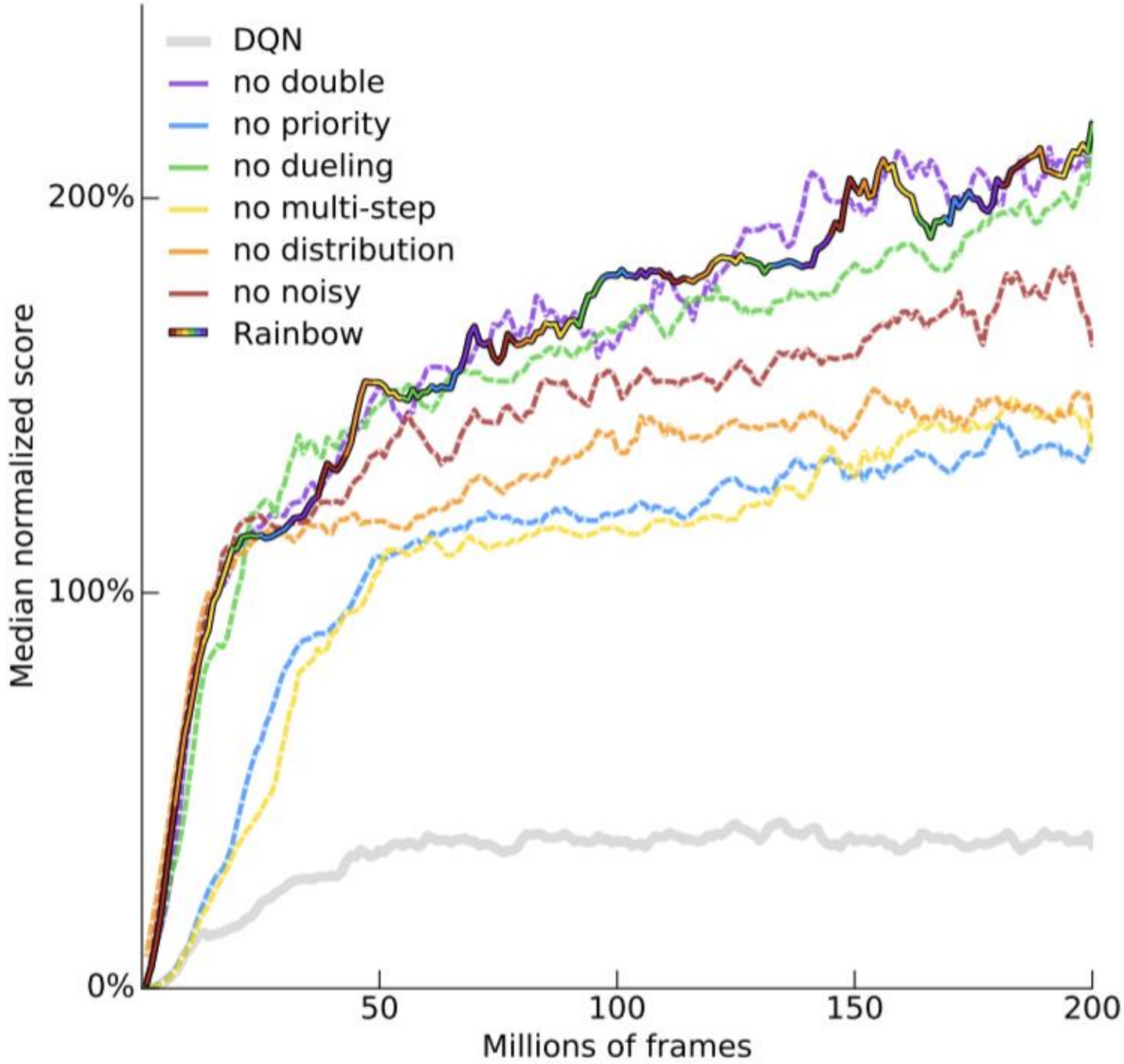
- Before we used TD error as weight to transitions
- Because we already compute TD error in calculating the loss.
- But in Distributional RL, we no longer do that (because we deal with probability distributions, ideally square term may force values fall into local optimum)
- Hence, we change the form of the weights to the appropriate loss;

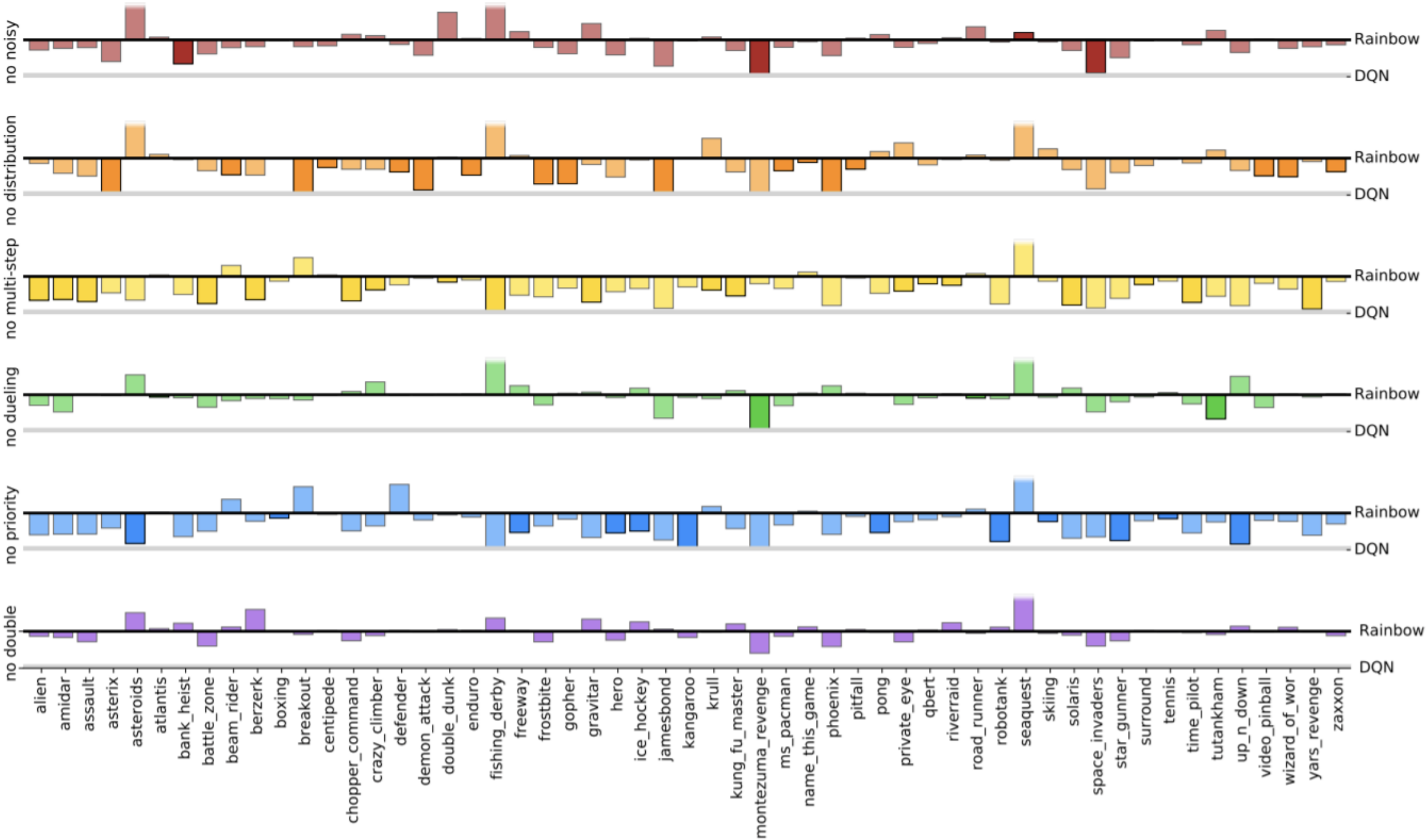
$$p_t \propto \left(D_{\mathrm{KL}}(\Phi_{\boldsymbol{z}} d_t^{(n)} || d_t) \right)^{\omega}$$

Hyperparameters

Parameter	Value
Min history to start learning	80K frames
Adam learning rate	0.0000625
Exploration ϵ	0.0
Noisy Nets σ_0	0.5
Target Network Period	32K frames
Adam ϵ	1.5×10^{-4}
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Multi-step returns n	3
Distributional atoms	51
Distributional min/max values	$[-10, 10]$


DQN
DDQN
Prioritized DDQN
Dueling DDQN
A3C
Distributional DQN
Noisy DQN
Rainbow
Median human-normalized score
200%
100%
0%
7
44
100
200
Millions of frames


DQN
no double
no priority
no dueling
no multi-step
no distribution
no noisy
Rainbow
200%
100%
0%
Median normalized score
50
100
150
200
Millions of frames


no noisy
no distribution
no multi-step
no dueling
no priority
no double
Rainbow
DQN
alien
amidar
assault
asterix
asteroids
atlantis
bank_heist
battle_zone
beam_rider
berzerk
boxing
breakout
centipede
chopper_command
crazy_climber
defender
demon_attack
double_dunk
enduro
fishing_derby
freeway
frostbite
gopher
gravitar
hero
ice_hockey
jamesbond
kangaroo
krull
kung_fu_master
montezuma_revenge
ms_pacman
name_this_game
phoenix
pitfall
pong
private_eye
qbert
riverraid
road_runner
robotank
seaquest
skiing
solaris
space_invaders
star_gunner
surround
tennis
time_pilot
tutankham
up_n_down
video_pinball
wizard_of_wor
yars_revenge
zaxxon

마무리

- DQN의 끝을 보여주는 논문
- DQN을 이렇게까지 쓸 수 있구나 라고 생각이 드는 논문
- 하지만 너무 구현하기에는 너무 복잡한 알고리즘!