UVA | SCHOOL *of* DATA SCIENCE

Extensions to Q-Learning 1

Reinforcement Learning
School of Data Science
University of Virginia

Last updated: June 23, 2025

Agenda

> Double Q-Learning

> Prioritized Experience Replay

Double Q-Learning

Main Idea

DQN requires target estimates of this form:

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)$$

The max() operation is used to estimate value

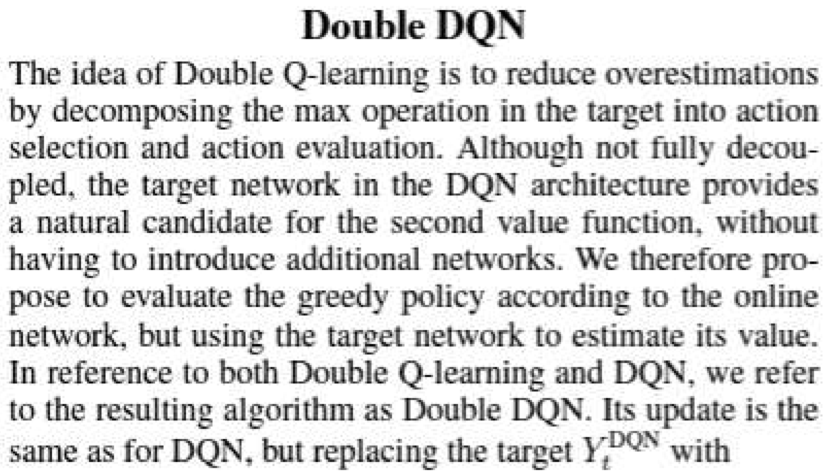
There may be noise in the system

Tends to produce a bias: overestimating value of Q

Paper: Deep Reinforcement Learning with Double Q-learning

Hado van Hasselt, Arthur Guez, David Silver. Google DeepMind

Main Idea, contd.



Double DQN

The idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation. Although not fully decoupled, the target network in the DQN architecture provides a natural candidate for the second value function, without having to introduce additional networks. We therefore propose to evaluate the greedy policy according to the online network, but using the target network to estimate its value. In reference to both Double Q-learning and DQN, we refer to the resulting algorithm as Double DQN. Its update is the same as for DQN, but replacing the target Y_t^{DQN} with

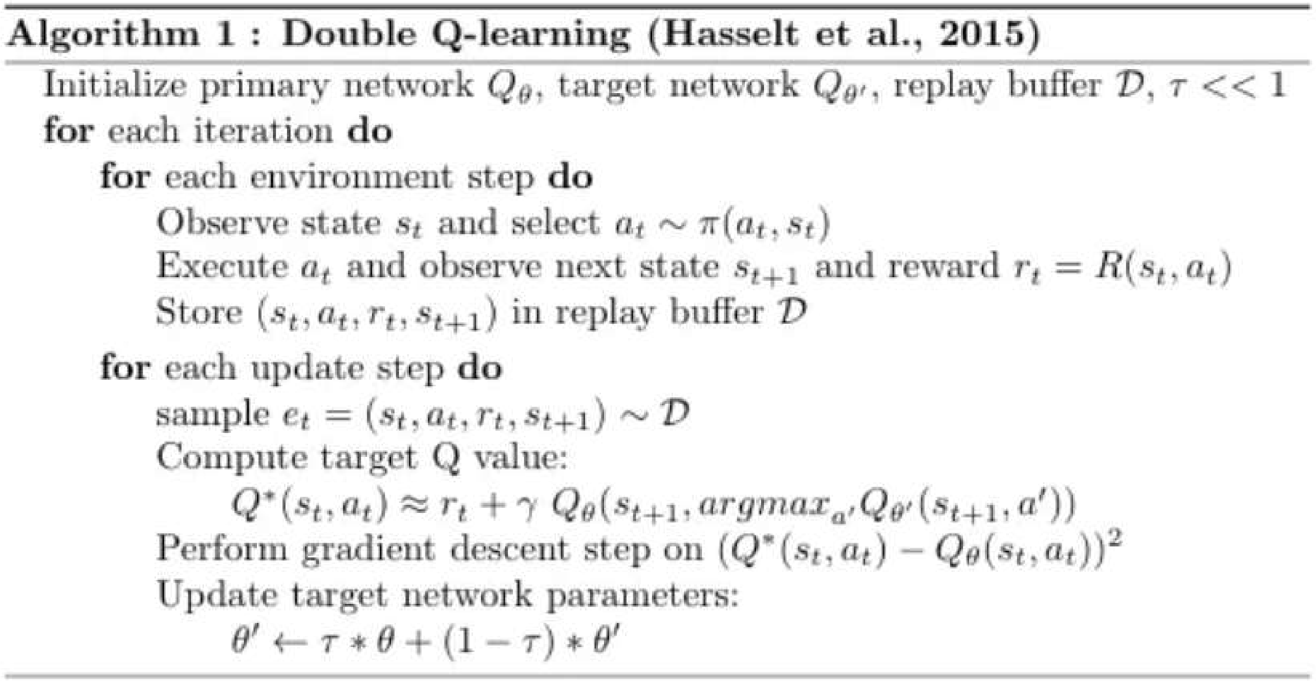
$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

Decompose max() operation into action selection, action evaluation

Notice there are two Qs

- One determines greedy policy using online network $\boldsymbol{\theta}_t$
- Another fairly evaluates the policy using the target network $\boldsymbol{\theta}'_t$

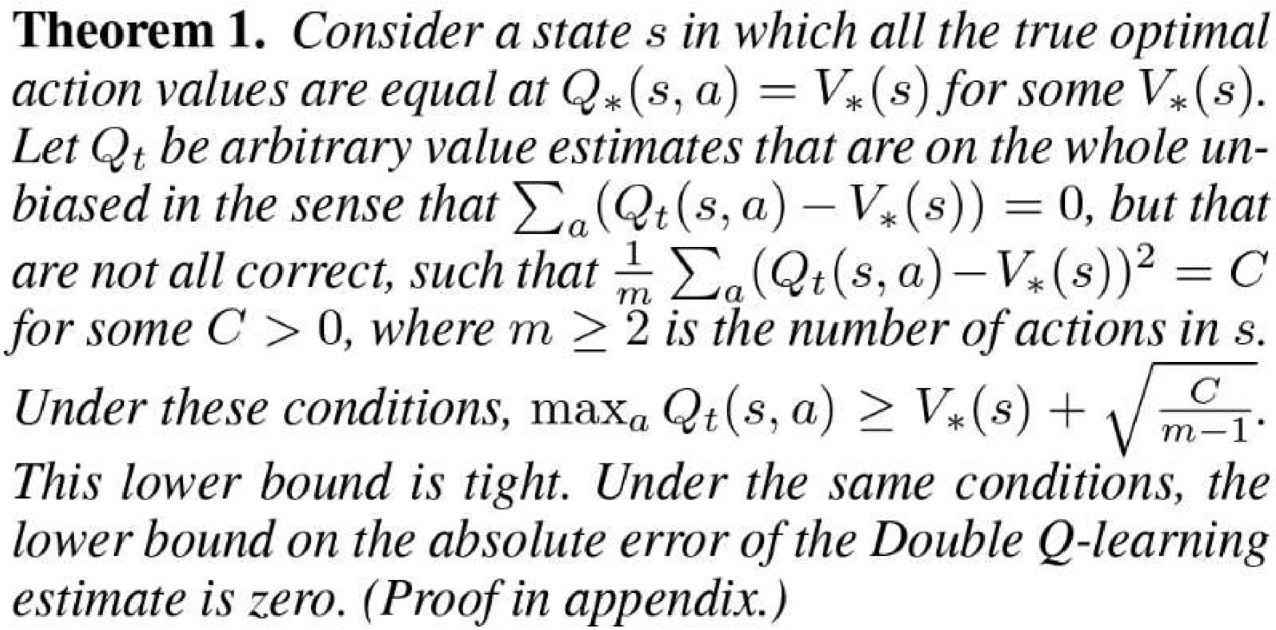
Double Q-Learning Algorithm



Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

```
Initialize primary network $Q_\theta$, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration do
    for each environment step do
        Observe state $s_t$ and select $a_t \sim \pi(a_t, s_t)$
        Execute $a_t$ and observe next state $s_{t+1}$ and reward $r_t = R(s_t, a_t)$
        Store $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
    for each update step do
        sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
        Compute target Q value:
            $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
        Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
        Update target network parameters:
            $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
```

Bias Estimates



Theorem 1. *Consider a state s in which all the true optimal action values are equal at $Q_*(s,a) = V_*(s)$ for some $V_*(s)$. Let Q_t be arbitrary value estimates that are on the whole unbiased in the sense that $\sum_a (Q_t(s,a) - V_*(s)) = 0$, but that are not all correct, such that $\frac{1}{m}\sum_a (Q_t(s,a) - V_*(s))^2 = C$ for some $C > 0$, where $m \geq 2$ is the number of actions in s. Under these conditions, $\max_a Q_t(s,a) \geq V_*(s) + \sqrt{\frac{C}{m-1}}$. This lower bound is tight. Under the same conditions, the lower bound on the absolute error of the Double Q-learning estimate is zero. (Proof in appendix.)*

=> For larger #actions *m*, the lower bound decreases

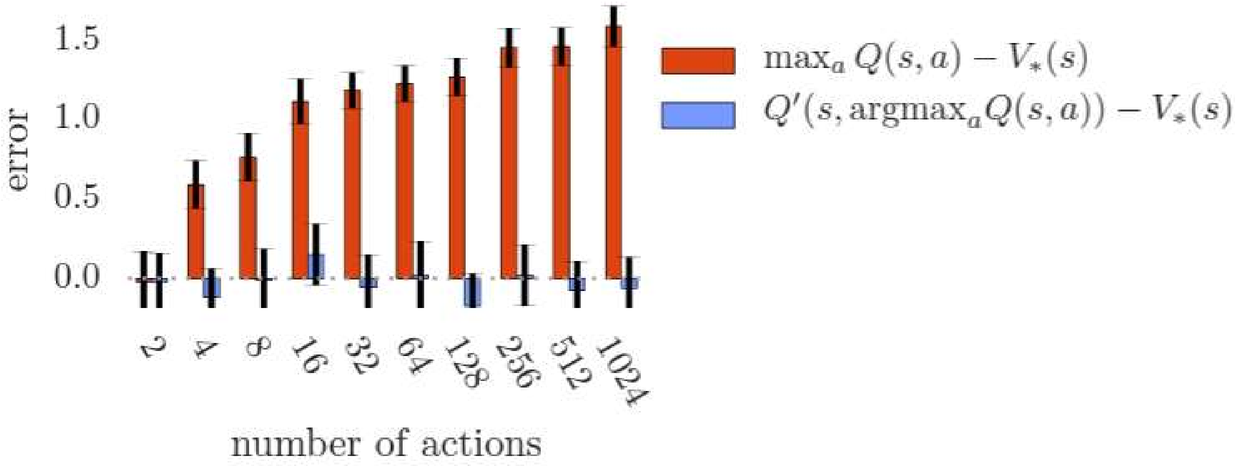
Hence, overoptimism increases for larger *m*

Bias Estimates, contd.

Red bars quantify this overoptimism

The gap increases for increasing #actions

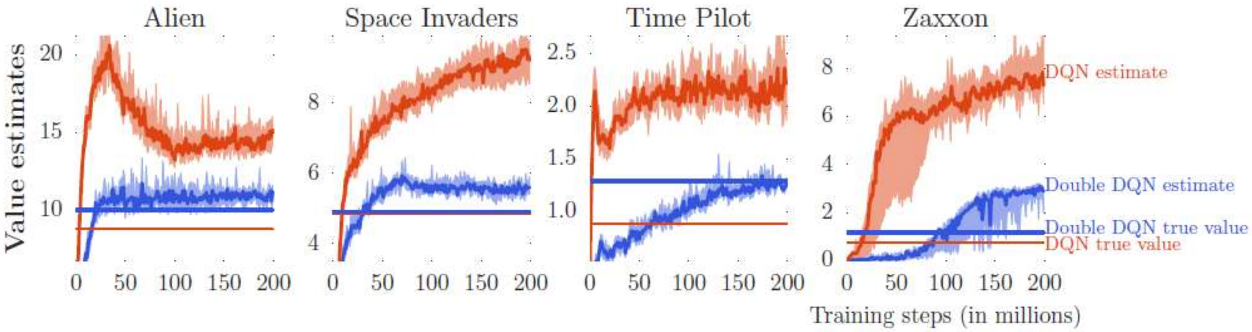
However, for Double Q-Learning, bias remains small



Bias Estimates - Atari

Red = DQN   Blue = Double DQN

DQN is overestimating values more than Double DQN (compare the curves)



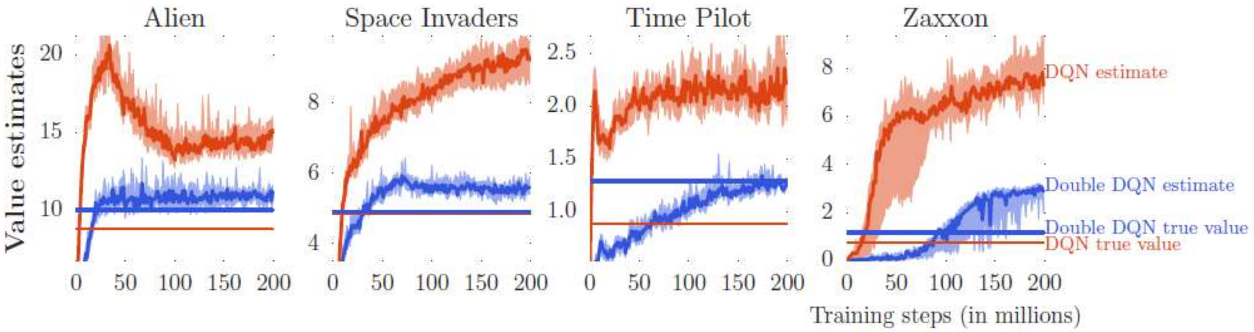
Bias Estimates - Atari

Red = DQN   Blue = Double DQN

DQN is overestimating values more than Double DQN (compare the curves)

Horizontal lines are **actual discounted value** of best learned policy (unbiased)
Computed by running learned policy and averaging actual discounted returns

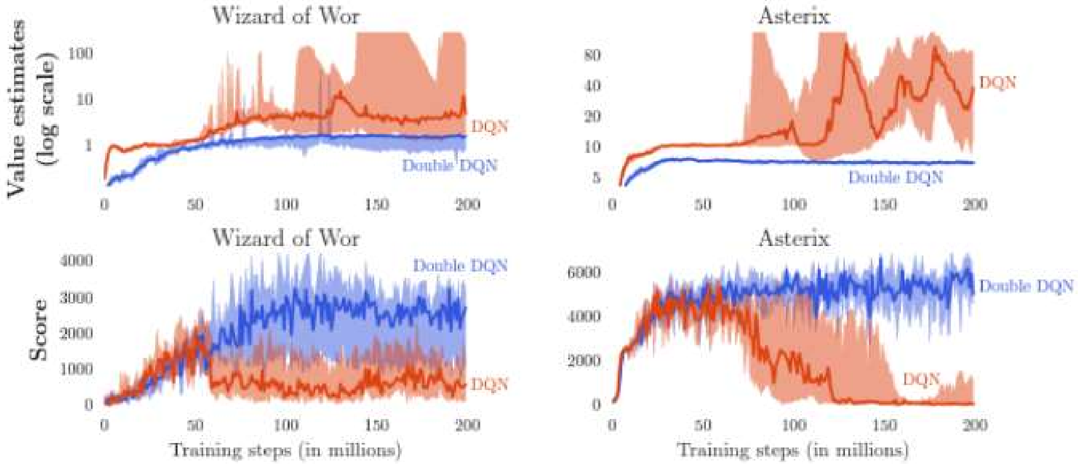
Double DQN best policy (blue horiz. line) outperforms DQN best policy (red horiz. line)



Bias Estimates – Atari, Extreme Cases

In these cases, overoptimism in DQN is extreme (red curve)

Where red separates from blue, bias increases



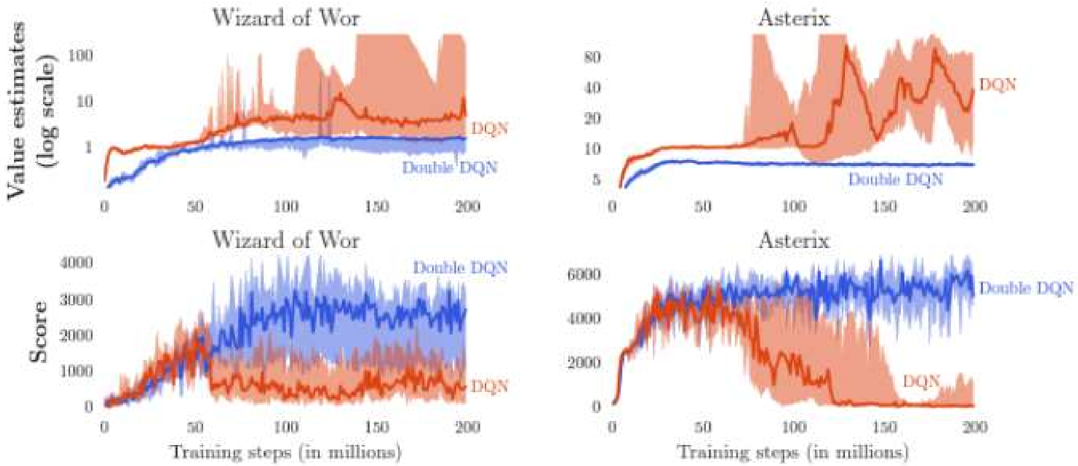
Bias Estimates – Atari, Extreme Cases

In these cases, overoptimism in DQN is extreme (red curve)

Where red separates from blue, bias increases

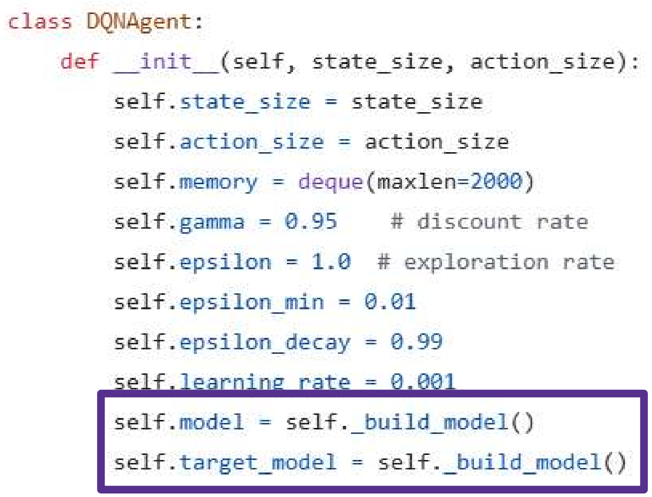
This has detrimental effect on score (Double DQN outperforms)

Double DQN is also more stable



Double DQN - Implementation Example

RL Applied to Sepsis Case – DQN Agent

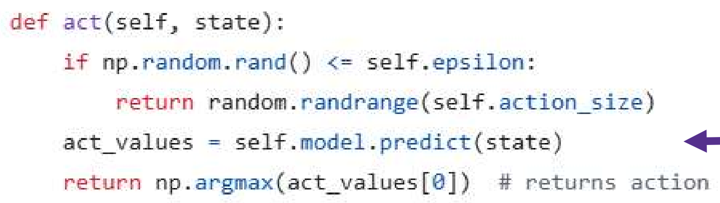


```
class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = deque(maxlen=2000)
        self.gamma = 0.95    # discount rate
        self.epsilon = 1.0  # exploration rate
        self.epsilon_min = 0.01
        self.epsilon_decay = 0.99
        self.learning_rate = 0.001
        self.model = self._build_model()
        self.target_model = self._build_model()
```

Instantiate online and target models

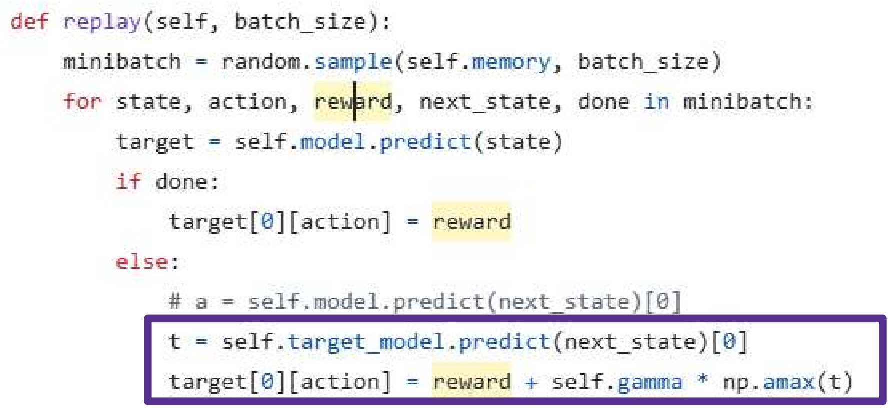
Source: https://github.com/keon/deep-q-learning

RL Applied to Sepsis Case – Act & Evaluate



```
def act(self, state):
    if np.random.rand() <= self.epsilon:
        return random.randrange(self.action_size)
    act_values = self.model.predict(state)
    return np.argmax(act_values[0])  # returns action
```

Action selection using online network



```
def replay(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)
    for state, action, reward, next_state, done in minibatch:
        target = self.model.predict(state)
        if done:
            target[0][action] = reward
        else:
            # a = self.model.predict(next_state)[0]
            t = self.target_model.predict(next_state)[0]
            target[0][action] = reward + self.gamma * np.amax(t)
```

Evaluation using target network

Source: https://github.com/keon/deep-q-learning

Findings

The bias resulting from max() in DQN can be large, **and it matters**

Double DQN can be more stable and reliable than DQN

Double DQN can find better policies than DQN

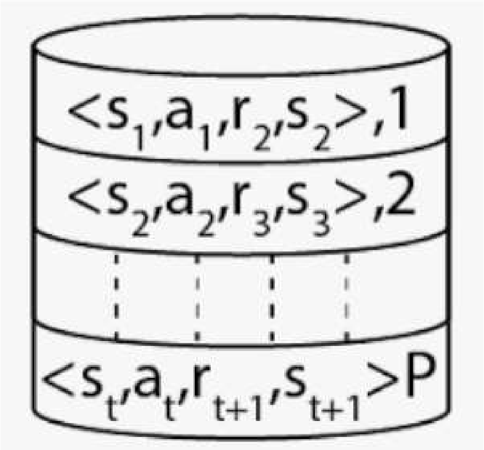
The code change to implement Double DQN is small

Prioritized Experience Replay

Buffer

DQN uses a buffer to store, retrieve, and replay past transitions (s_t, a_t, r_{t+1}, s_{t+1})

This makes DQN more efficient than Q-Learning



The transitions are sampled uniformly at random

Prioritized Experience Replay (PER)
tries to be more strategic about sampling

Paper: Prioritized Experience Replay. Schaul et. al.

Priority

PER uses absolute TD error $|\delta_i|$ as the priority

Could select transition with max abs TD error, but greedy approach may backfire

Instead, paper uses stochastic prioritization with two methods:

1) Proportional

2) Rank-based

Proportional Priority

Define proportional priority where $p_i = |\delta_i| + \epsilon$

Epsilon prevents transitions from not being visited when error is zero

Define probability of sampling transition

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Alpha controls amount of prioritization; alpha = 0 is uniform case.

Rank-Based Priority

Rank is used for robustness

Define probability of sampling transition

$$p_i = \frac{1}{\text{rank}(i)}$$

Ranks are based on absolute TD error

Importance Sampling

PER induces bias by changing probability of transition selection

Can adjust for bias with importance sampling

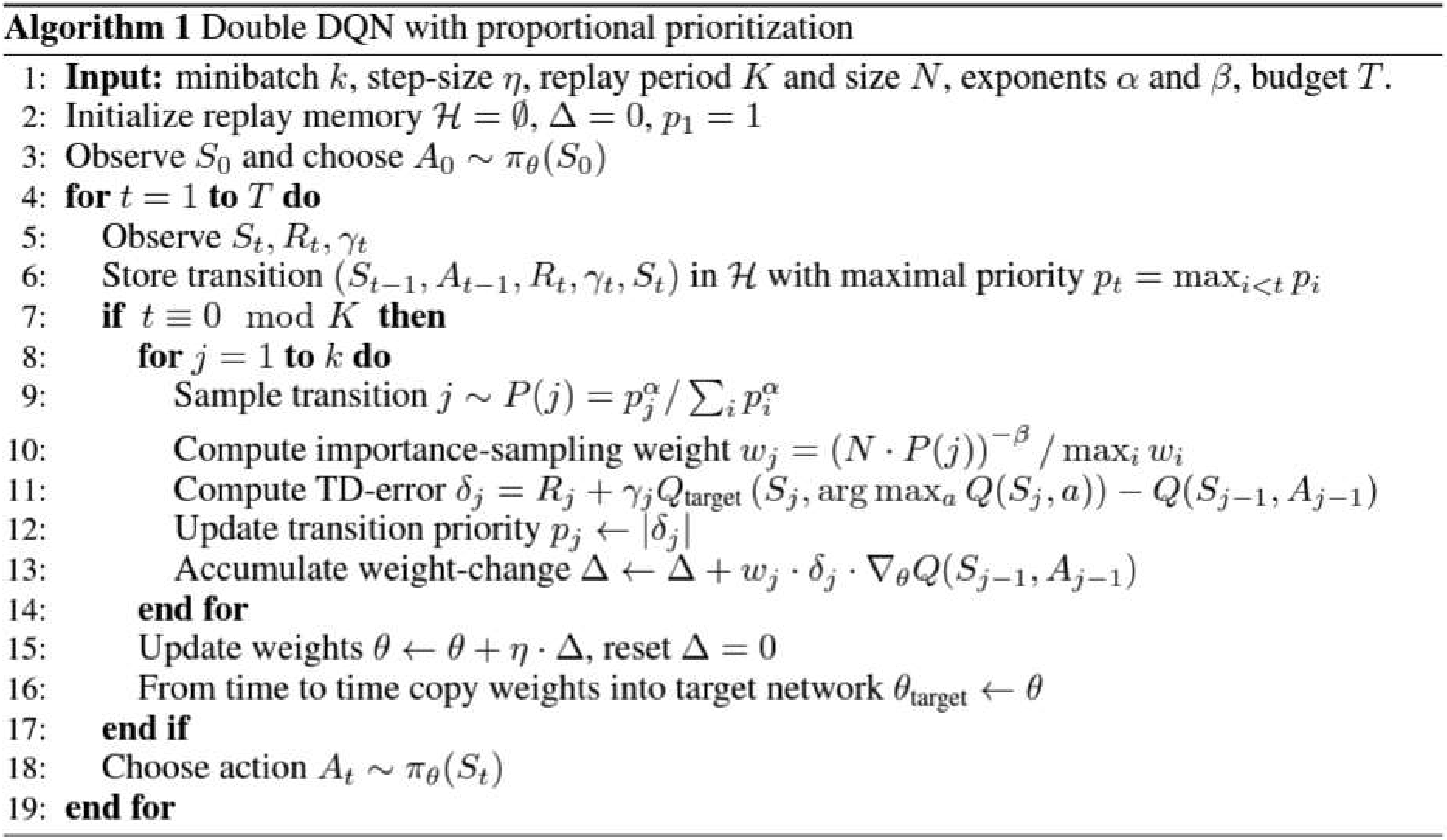
$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta}$$

Case where beta = 1 fully compensates for non-uniform probabilities

For Q-learning updates, use $|\delta_i|$ weighted by w_i

For stability, scale weights by 1 / max w_i

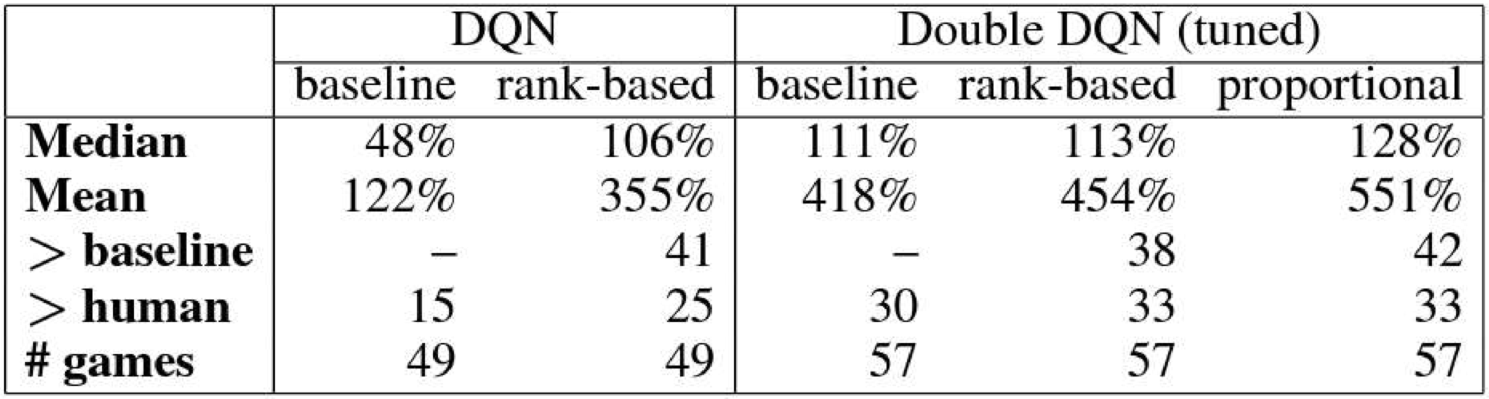
Algorithm



Algorithm 1 Double DQN with proportional prioritization

```
Input: minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and $\beta$, budget $T$.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ to $T$ do
    Observe $S_t, R_t, \gamma_t$
    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ then
        for $j = 1$ to $k$ do
            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
            Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
            Update transition priority $p_j \leftarrow |\delta_j|$
            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
        end for
        Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
        From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
    end if
    Choose action $A_t \sim \pi_\theta(S_t)$
end for
```

Results: Summary of Normalized Scores



	DQN		Double DQN (tuned)		
	baseline	rank-based	baseline	rank-based	proportional
Median	48%	106%	111%	113%	128%
Mean	122%	355%	418%	454%	551%
> baseline	–	41	–	38	42
> human	15	25	30	33	33
# games	49	49	57	57	57

DQN with rank-based priority results in higher scores for 41 out of 49 games

Double DQN is similarly helped by PER