# Dueling DQN & Prioritised Experience Reply

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

*avereshc@buffalo.edu*

October 10, 2019

*Slides are based on paper by Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." (2015)

Schaul, Tom, et al. "Prioritized experience replay." (2015)

# Overview

1. Recap: DQN
2. Recap: Double DQN
3. Dueling DQN
4. Prioritized Experience Replay (PER)

# Table of Contents

# Recap: Deep Q-Networks (DQN)

- Represent value function by deep Q-network with weights $w$

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

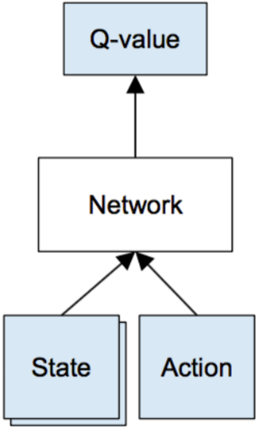
- Leading to the following Q-leaning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

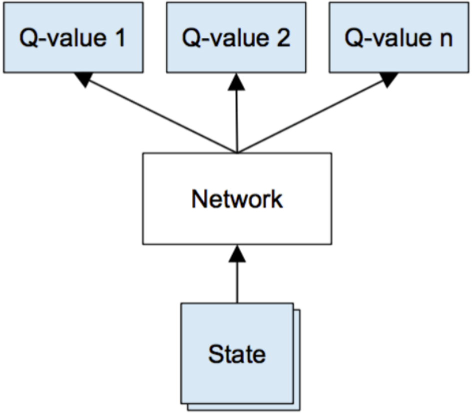
- Optimize objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

# Deep Q-Network (DQN) Architecture

Naive DQN




Optimized DQN used by DeepMind



# DQN Algorithm

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
**For** episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  **For** $t = 1$,T **do**
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ \boxed{r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right)} & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**

# Table of Contents

# Double Q-learning

Two estimators:

- Estimator $Q_1$: Obtain best actions
- Estimator $Q_2$: Evaluate $Q$ for the above action

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha(\text{Target} - Q_1(s,a))$$

**Q Target:** $r(s,a) + \gamma \max_{a'} Q_1(s',a')$

**Double Q Target:** $r(s,a) + \gamma Q_2(s', \arg\max_{a'}(Q_1(s',a')))$

# Double Q-learning

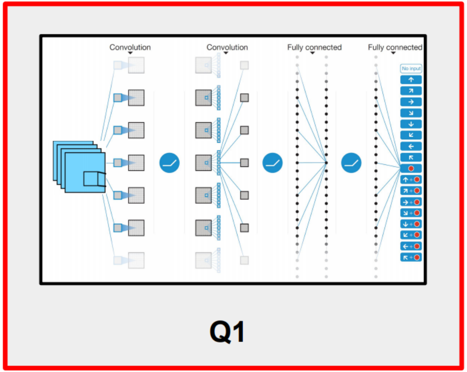
**Algorithm 1** Double Q-learning

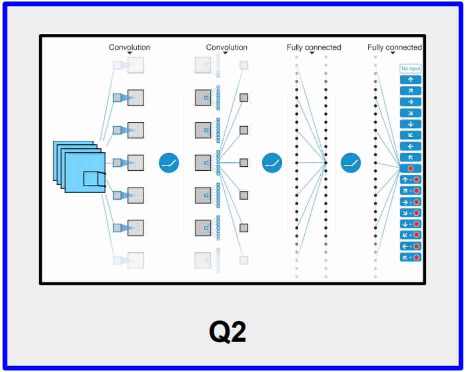
Initialize $Q^A$,$Q^B$,$s$
**repeat**
    Choose $a$, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe $r$, $s'$
    Choose (e.g. random) either UPDATE(A) or UPDATE(B)
    **if** UPDATE(A) **then**
        Define $\underline{a^* = \arg\max_a Q^A(s',a)}$
        $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(\boxed{r + \gamma Q^B(s',a^*)} - Q^A(s,a)\right)$
    **else if** UPDATE(B) **then**
        Define $\underline{b^* = \arg\max_a Q^B(s',a)}$
        $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)\left(\boxed{r + \gamma Q^A(s',b^*)} - Q^B(s,a)\right)$
    **end if**
    $s \leftarrow s'$
**until** end

# Double Deep Q Network

Two estimators:

- Estimator $Q_1$: Obtain best actions
- Estimator $Q_2$: Evaluate $Q$ for the above action



# Double Deep Q Network

**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:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

# Table of Contents

What is Q-values tells us?

What is Q-values tells us?

How good it is to be at state $s$ and taking an action $a$ at that state $Q(s, a)$.

# Advantage Function $A(s, a)$

$$A(s, a) = Q(s, a) - V(s)$$

- If $A(s, a) > 0$: our gradient is pushed in that direction
- If $A(s, a) < 0$ (our action does worse than the average value of that state) our gradient is pushed in the opposite direction

# Dueling DQN

How can we decompose $Q^{\pi}(s, a)$?

$$Q^{\pi}(s, a) =$$

# Dueling DQN

How can we decompose $Q^\pi(s, a)$?

$$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$$
$$V^\pi(s) =$$

# Dueling DQN

How can we decompose $Q^\pi(s, a)$?

$$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$$
$$V^\pi(s) = E_{a \sim \pi(s)}[Q^\pi(s, a)]$$

# Dueling DQN

How can we decompose $Q^\pi(s, a)$?

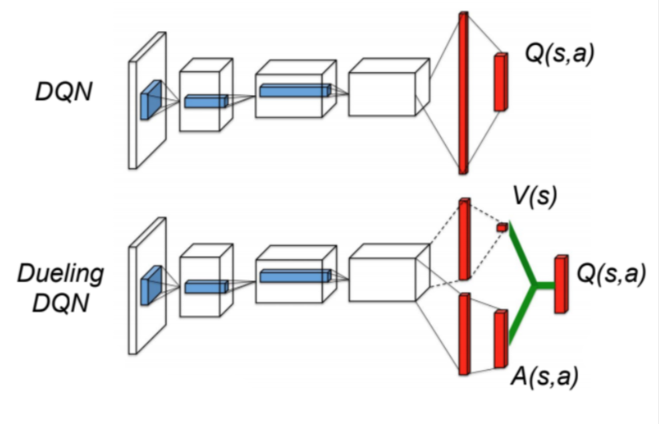
$$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a)$$
$$V^\pi(s) = E_{a \sim \pi(s)}[Q^\pi(s, a)]$$

In Dueling DQN, we separate the estimator of these two elements, using two new streams:

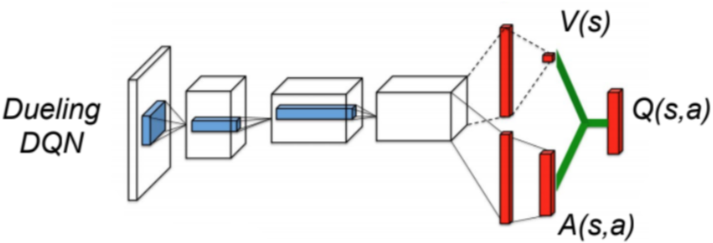
- one estimates the state value $V(s)$
- one estimates the advantage for each action $A(s, a)$

Networks that separately computes the advantage and value functions, and combines back into a single Q-function at the final layer.

# Dueling DQN



# Dueling DQN




- One stream of fully-connected layers output a scalar $V(s; \theta, \beta)$
- Other stream output an $|A|$-dimensional vector $A(s, a; \theta, \alpha)$

Here, $\theta$ denotes the parameters of the convolutional layers, while $\alpha$ and $\beta$ are the parameters of the two streams of fully-connected layers.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

# Dueling DQN

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

**Problem:** Equation is unidentifiable $\rightarrow$ given $Q$ we cannot recover $V$ and $A$ uniquely $\rightarrow$ poor practical performance.

**Solutions:**

1. Force the advantage function estimator to have zero advantage at the chosen action

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left( A(s, a; \theta, \alpha) - \max_{a' \in |A|} A(s, a'; \theta, \alpha) \right)$$

# Dueling DQN

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha)$$

**Problem:** Equation is unidentifiable $\rightarrow$ given $Q$ we cannot recover $V$ and $A$ uniquely $\rightarrow$ poor practical performance.

**Solutions:**

1. Force the advantage function estimator to have zero advantage at the chosen action

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |A|} A(s,a';\theta,\alpha)\right)$$

$$a^* = \arg\max_{a' \in A} Q(s,a';\theta,\alpha,\beta)$$

# Dueling DQN

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

**Problem:** Equation is unidentifiable $\rightarrow$ given $Q$ we cannot recover $V$ and $A$ uniquely $\rightarrow$ poor practical performance.

**Solutions:**

1. Force the advantage function estimator to have zero advantage at the chosen action

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |A|} A(s, a'; \theta, \alpha)\right)$$

$$\begin{aligned} a^* &= \arg\max_{a' \in A} Q(s, a'; \theta, \alpha, \beta) \\ &= \arg\max_{a' \in A} A(s, a'; \theta, \alpha) \end{aligned}$$

# Dueling DQN

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

**Problem:** Equation is unidentifiable $\rightarrow$ given $Q$ we cannot recover $V$ and $A$ uniquely $\rightarrow$ poor practical performance.

**Solutions:**

1. Force the advantage function estimator to have zero advantage at the chosen action

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |A|} A(s, a'; \theta, \alpha)\right)$$

$$\begin{aligned} a^* &= \arg\max_{a' \in A} Q(s, a'; \theta, \alpha, \beta) \\ &= \arg\max_{a' \in A} A(s, a'; \theta, \alpha) \end{aligned}$$

$$Q(s, a^*; \theta, \alpha, \beta) = V(s; \theta, \beta)$$

# Dueling DQN

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

**Problem:** Equation is unidentifiable $\rightarrow$ given $Q$ we cannot recover $V$ and $A$ uniquely $\rightarrow$ poor practical performance.

**Solutions:**

2. Replaces the max operator with an average

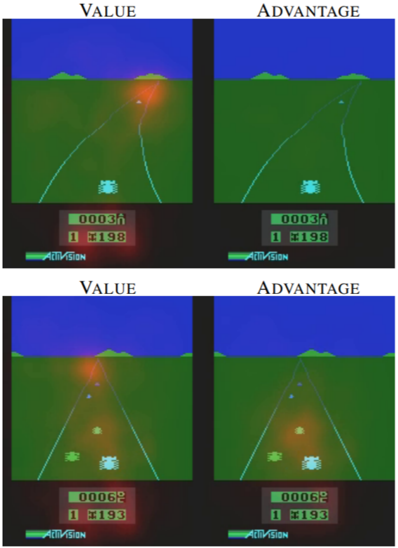
$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha)\right)$$

It increases the stability of the optimization: the advantages only need to change as fast as the mean, instead of having to compensate any change.

# Dueling DQN: Example

Value and advantage saliency maps for two different time steps

- **Leftmost pair** - the value network stream pays attention to the road and the score.
- The advantage stream does not pay much attention to the visual input because its action choice is practically irrelevant when there are no cars in front.
- **Rightmost pair** - the advantage stream pays attention as there is a car immediately in front, making its choice of action very relevant.



# Dueling DQN: Summary

- Intuitively, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state.
- The dueling architecture represents both the value $V(s)$ and advantage $A(s, a)$ functions with a single deep model whose output combines the two to produce a state-action value $Q(s, a)$.

# Table of Contents

# Recap: Experience replay

**Problem:** Online RL agents incrementally update their parameters while they observe a stream of experience. In their simplest form, they discard incoming data immediately, after a single update. Two issues are

1. Strongly correlated updates that break the i.i.d. assumption
2. Rapid forgetting of possibly rare experiences that would be useful later on.

# Recap: Experience replay

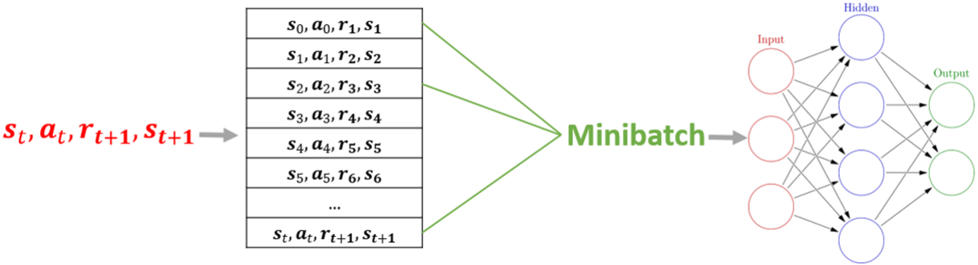
**Problem:** Online RL agents incrementally update their parameters while they observe a stream of experience. In their simplest form, they discard incoming data immediately, after a single update. Two issues are

1. Strongly correlated updates that break the i.i.d. assumption
2. Rapid forgetting of possibly rare experiences that would be useful later on.

**Solution: Experience replay**

- Break the temporal correlations by mixing more and less recent experience for the updates
- Rare experience will be used for more than just a single update

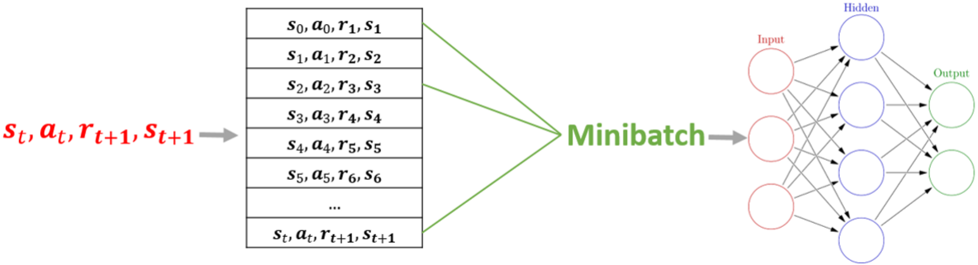
# Prioritized Experience Replay (PER)




Two design choices:

1. Which experiences to store?
2. Which experiences to replay?

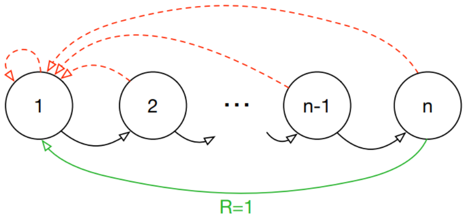
# Prioritized Experience Replay (PER)

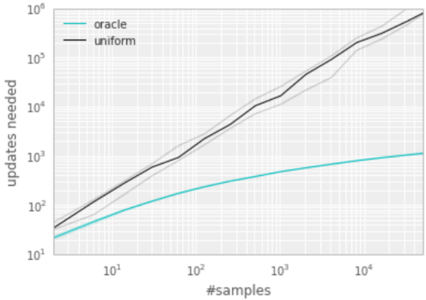



Two design choices:

1. Which experiences to store?
2. Which experiences to replay? PER tries to solve this

# PER: Example ‘Blind Cliffwalk’




- Two actions: ‘right’ and ‘wrong’
- The episode is terminated when ‘wrong’ action is chosen.
- Taking the ‘right’ action progresses through a sequence of $n$ states, at the end of which lies a final reward of 1; reward is 0 elsewhere

# Prioritized Experience Replay (PER): TD error

TD error for vanilla **DQN**:

$$\delta_i = r_t + \gamma \max_{a \in \mathcal{A}} Q_{\theta^-}(s_{t+1}, a) - Q_\theta(s_t, a_t)$$

TD error for **Double DQN**:

$$\delta_i = r_t + \gamma Q_{\theta^-}(s_{t+1}, \mathrm{argmax}_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a)) - Q_\theta(s_t, a_t)$$

we use $|\delta_i|$ as the magnitude of the TD error.

What $|\delta_i|$ shows us?

# Prioritized Experience Replay (PER): TD error

TD error for vanilla **DQN**:

$$\delta_i = r_t + \gamma \max_{a \in \mathcal{A}} Q_{\theta^-}(s_{t+1}, a) - Q_\theta(s_t, a_t)$$

TD error for **Double DQN**:

$$\delta_i = r_t + \gamma Q_{\theta^-}(s_{t+1}, \mathrm{argmax}_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a)) - Q_\theta(s_t, a_t)$$

we use $|\delta_i|$ as the magnitude of the TD error.

## What $|\delta_i|$ shows us?

A big difference between our prediction and the TD target $\longrightarrow$ we have to learn a lot

# Prioritized Experience Replay (PER)

Two ways of getting priorities, denoted as $p_i$:

1. Direct, proportional prioritization:

$$p_i = |\delta_i| + \epsilon$$

   where $\epsilon$ is a small constant ensuring that the sample has some non-zero probability of being drawn

# Prioritized Experience Replay (PER)

Two ways of getting priorities, denoted as $p_i$:

1. Direct, proportional prioritization:

$$p_i = |\delta_i| + \epsilon$$

   where $\epsilon$ is a small constant ensuring that the sample has some non-zero probability of being drawn

2. A rank based method:

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

   where $rank(i)$ is the rank of transition $i$ when the replay memory is sorted according to $|\delta_i|$

# PER: Stochastic sampling method

**Problem:** During exploration, $p_i$ terms are not known for brand-new samples.
**Solution:** interpolate between pure greedy prioritization and uniform random sampling.
Probability of sampling transition $i$

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

where $p_i > 0$ is the priority of transition $i$; $\alpha$ is the level of prioritization.

# PER: Stochastic sampling method

**Problem:** During exploration, $p_i$ terms are not known for brand-new samples.
**Solution:** interpolate between pure greedy prioritization and uniform random sampling.
Probability of sampling transition $i$

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

where $p_i > 0$ is the priority of transition $i$; $\alpha$ is the level of prioritization.

- If $\alpha \to 0$, there is no prioritization, because all $p(i)^\alpha = 1$ (uniform case)

# PER: Stochastic sampling method

**Problem:** During exploration, $p_i$ terms are not known for brand-new samples.
**Solution:** interpolate between pure greedy prioritization and uniform random sampling.
Probability of sampling transition $i$

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

where $p_i > 0$ is the priority of transition $i$; $\alpha$ is the level of prioritization.

- If $\alpha \to 0$, there is no prioritization, because all $p(i)^\alpha = 1$ (uniform case)
- If $\alpha \to 1$, then we get to full prioritization, where sampling data points is more heavily dependent on the actual $|\delta_i|$ values.

# PER: Stochastic sampling method

**Problem:** During exploration, $p_i$ terms are not known for brand-new samples.
**Solution:** interpolate between pure greedy prioritization and uniform random sampling.
Probability of sampling transition $i$

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

where $p_i > 0$ is the priority of transition $i$; $\alpha$ is the level of prioritization.

- If $\alpha \to 0$, there is no prioritization, because all $p(i)^\alpha = 1$ (uniform case)
- If $\alpha \to 1$, then we get to full prioritization, where sampling data points is more heavily dependent on the actual $|\delta_i|$ values.

This will ensure that the probability of being sampled is monotonic in a transition's priority, while guaranteeing a non-zero probability even for the lowest-priority transition.

# PER: Importance-sampling (IS) weights

Use importance sampling weights to adjust the updating by reducing the weights of the often seen samples.

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

$\beta$ is the exponent, which controls how much prioritization to apply.

For stability reasons, we always normalize weights by $1/\max_i w_i$ so that they only scale the update downwards.

**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**

# Prioritized Experience Replay (PER): Summary

- Built on top of experience replay buffers

# Prioritized Experience Replay (PER): Summary

- Built on top of experience replay buffers
- Uniform sampling from a replay buffer is a good default strategy, but it can be improved by prioritized sampling, that will weigh the samples so that "important" ones are drawn more frequently for training.

# Prioritized Experience Replay (PER): Summary

- Built on top of experience replay buffers
- Uniform sampling from a replay buffer is a good default strategy, but it can be improved by prioritized sampling, that will weigh the samples so that "important" ones are drawn more frequently for training.
- Key idea is to increase the replay probability of experience tuples that have a high expected learning progress (measured by $|\delta|$. This lead to both faster learning and to better final policy quality, as compared to uniform experience replay.