Reinforcement learning

Episode 5

Deep reinforcement learning

What we already know:

- Q-learning

$$L = E_{s \sim S, a \sim A}[(Q(s,a) - (r + \gamma \max_{a'} Q(s', a')))^2]$$

- Approximation of q-values with respect to state $Q(s, a, \Theta)$, where Θ is the vector of weights

- Experience replay

This is not enough!

Autocorrelation

- Target is based on prediction

$$r + \gamma \max_{a'} Q(s', a', \Theta)$$

- Since we use function approximation, when we update $Q(s, a, \Theta)$ we also update $Q(s', a, \Theta)$ towards that direction
- In worst case network may diverge, but usually it becomes unstable.

- **How to stabilize weights?**

Target network

Idea: use network with frozen weights to compute the target

$$L(\Theta) = E_{s \sim S, a \sim A}[(Q(s,a,\Theta) - (r + \gamma \max_{a'} Q(s', a', \Theta^-)))^2]$$

where Θ^- is the frozen weights

Hard target network:

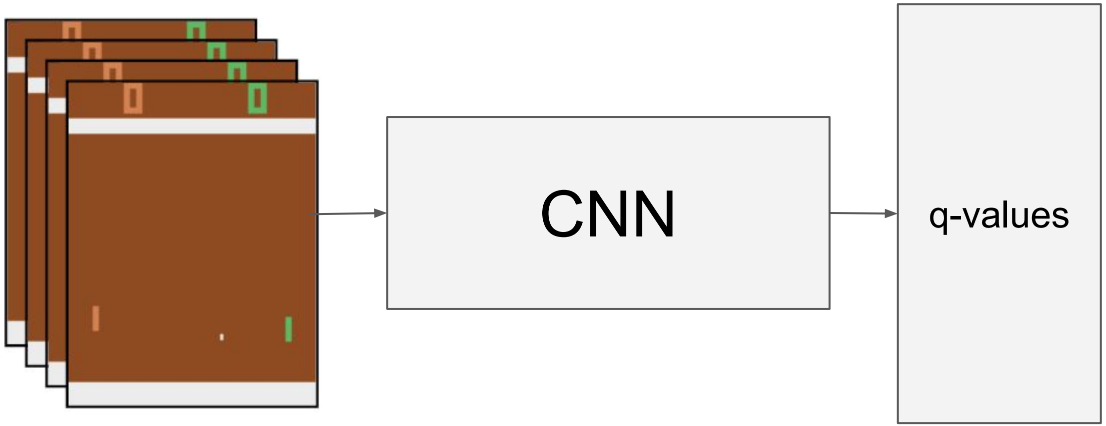
Update Θ^- every **n** steps and set its weights as Θ

Soft target network:

Update Θ^- every step:

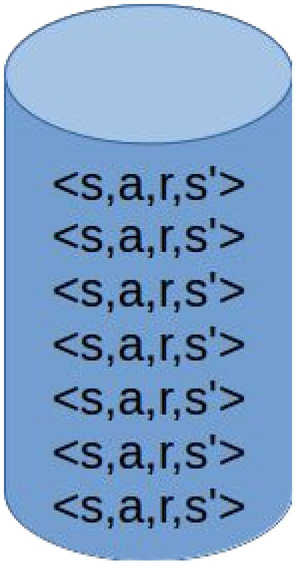
$$\Theta^- = (1 - \alpha)\Theta^- + \alpha\Theta$$

Playing Atari with Deep Reinforcement Learning (2013, Deepmind)



4 last frames as input

Experience replay



10^6 last transitions

Update weights using:

$$L(\Theta) = E_{s\sim S, a\sim A}[(Q(s,a,\Theta) - (r + \gamma \max_{a'} Q(s',a',\Theta^-)))^2]$$

Update Θ^- every 5000 train steps

Problem of overestimation

We use “max” operator to compute the target

$$L(s,a) = (Q(s,a) - (r + \gamma \max_{a'} Q(s',a')))^2$$

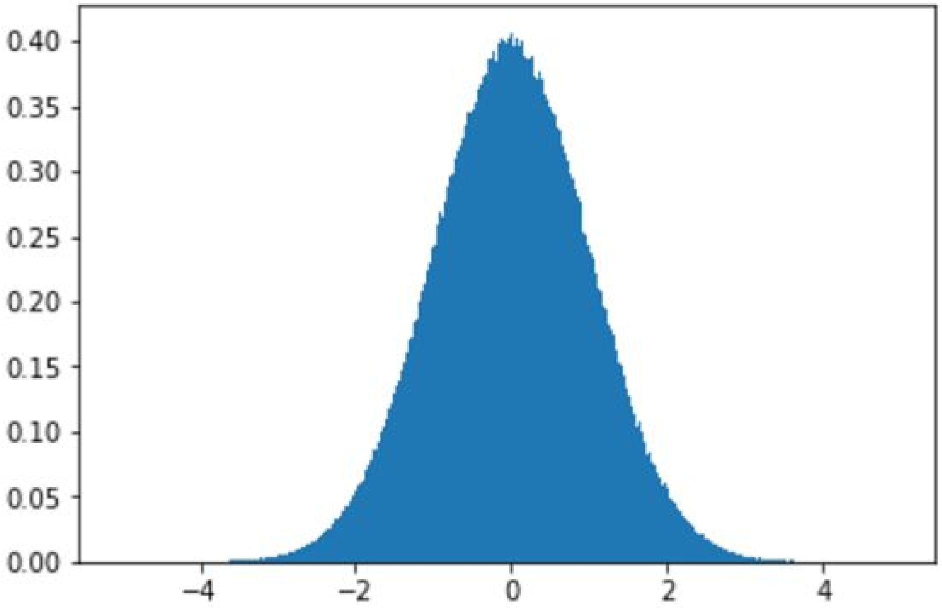
Surprisingly here is a problem

(although we want $\mathrm{E}_{s\sim S,a\sim A}[L(s,a)]$ to be equal zero)

Problem of overestimation

Normal distribution
$3*10^6$ samples

mean: ~0.0004



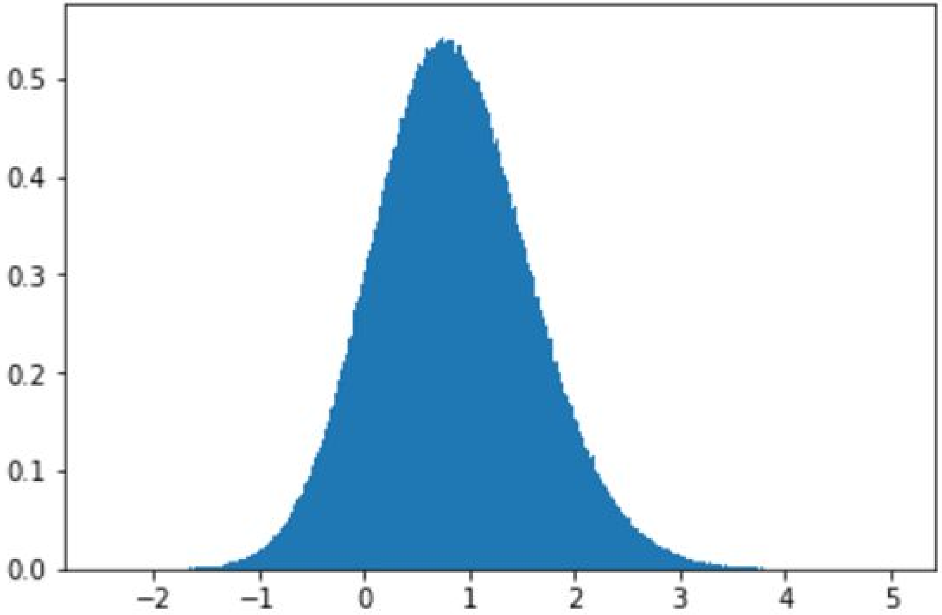
Problem of overestimation

Normal distribution

$3*10^6$ x 3 samples

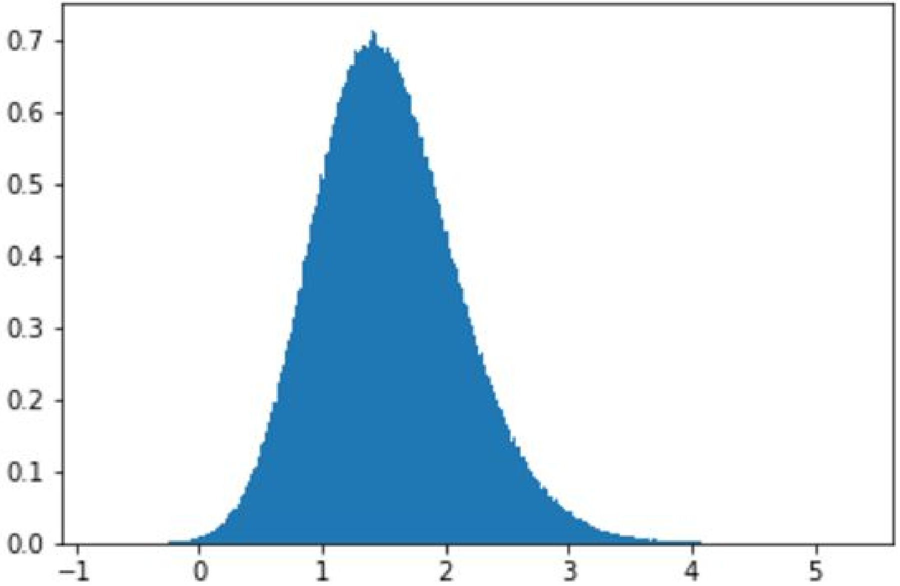
Then take maximum of every tuple

mean: ~0.8467

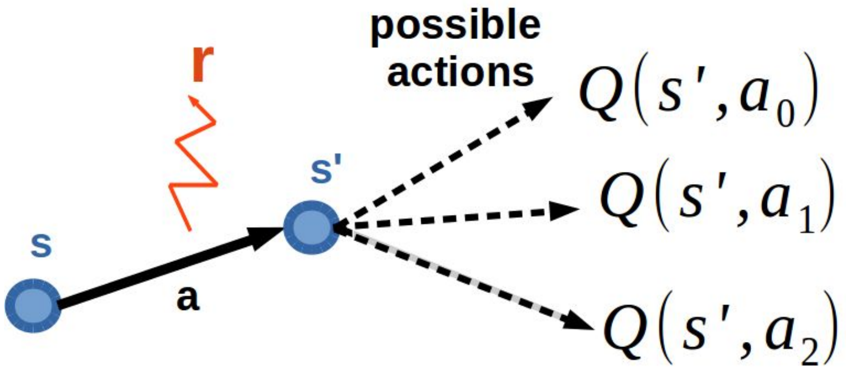


Problem of overestimation

Normal distribution
$3*10^6$ x 10 samples
Then take maximum of every tuple
mean: ~1.538



Problem of overestimation



Suppose true $Q(s', a')$ are equal **0** for all a'

But we have an approximation (or other kind) error $\sim N(0, \sigma^2)$

So $Q(s, a)$ should be equal r

But if we update $Q(s, a)$ towards $r + \gamma \max_{a'} Q(s', a')$

we will have overestimated $Q(s, a)$ > r because

$$\mathrm{E}[\max_{a'} Q(s', a')] >= \max_{a'} E[Q(s', a')]$$

Double Q-learning (NIPS 2010)

Idea: use two estimators of q-values: Q^A, Q^B
They should compensate mistakes of each other because they will be independent
Let’s get argmax from another estimator!

$y = r + \gamma \max_{a'} Q(s', a')$ - Q-learning target

$y = r + \gamma Q(s', \operatorname{argmax}_{a'} Q(s', a'))$ - Rewrited Q-learning target

$y = r + \gamma Q^A(s', \operatorname{argmax}_{a} Q^B(s', a'))$ - Double Q-learning target

Double Q-learning (NIPS 2010)

Algorithm 1 Double Q-learning

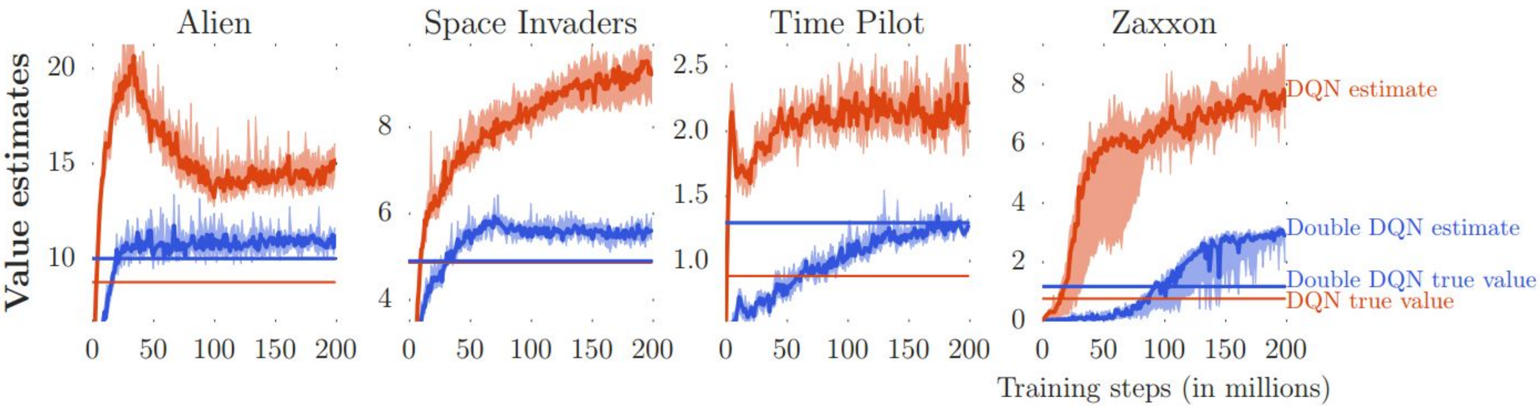
Initialize Q^A,Q^B,s
repeat
    Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
    Choose (e.g. random) either UPDATE(A) or UPDATE(B)
    if UPDATE(A) **then**
        Define $a^* = \arg\max_a Q^A(s',a)$
        $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*)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

How to apply this algorithm in deep reinforcement learning?

Deep Reinforcement Learning with Double Q-learning (Deepmind, 2015)

Idea: use main network to choose action!

$$y_{dqn} = r + \gamma \max_{a'} Q(s', a', \Theta^-)$$
$$y_{ddqn} = r + \gamma Q(s', \text{argmax}_{a'} Q(s', a', \Theta), \Theta^-)$$



	DQN	Double DQN	Double DQN (tuned)
Median	47.5%	88.4%	116.7%
Mean	122.0%	273.1%	475.2%

Prioritized Experience Replay (2016, Deepmind)

Idea: sample transitions from xp-replay more clever

We want to set probability for every transition. Let's use the absolute value of TD-error of transition as a probability!

$$\text{TD-error } \delta = Q(s, a) - (r + \gamma Q(s', \operatorname{argmax}_{a'} Q(s', a', \Theta), \Theta^{-}))$$

$$p = |\delta|$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$ where α is the priority parameter (when α is 0 it's the uniform case)

Do you see the problem?

Prioritized Experience Replay (2016, Deepmind)

Idea: sample transitions from xp-replay more clever

We want to set probability for every transition. Let's use the absolute value of TD-error of transition as a probability!

$$\text{TD-error } \delta = Q(s,a) - (r + \gamma Q(s', \text{argmax}_{a'} Q(s', a', \Theta), \Theta^-))$$

$$p = |\delta|$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$ where α is the priority parameter (when α is 0 it's the uniform case)

Do you see the problem?

Transitions become non i.i.d. and therefore we introduce the bias.

Prioritized Experience Replay (2016, Deepmind)

Solution: we can correct the bias by using importance-sampling weights

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}$$ where β is the parameter

We also normalize weights by $1/\max_i w_i$ (here is not mathematical reason)

When we put transition into experience replay, we set maximal priority $p_t = \max_{i<t} p_i$

Prioritized Experience Replay (2016, Deepmind)

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, 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, γ_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

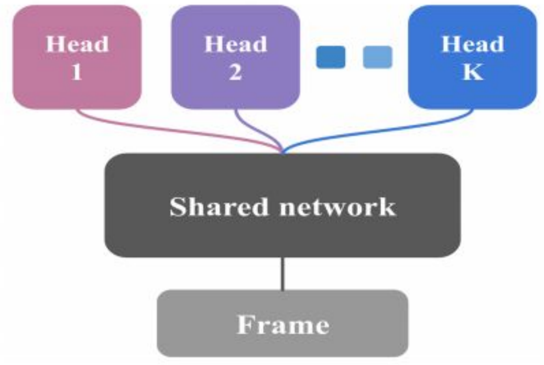
Prioritized Experience Replay (2016, Deepmind)

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, 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, γ_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 $\delta_j = R_j + \gamma_j Q_{\text{target}}\left(S_j, \arg\max_a Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1})$
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

It is the bonus homework!

Bootstrapped DQN (2016, Deepmind & Stanford)



Idea: make exploration ~~great again~~ more clever.

Usually we use e-greedy strategy to pick action.

Now we maintain K separate heads (for example, FC-layers) and shared body (for example, convolutional) weights.

Every episode we pick a head randomly and train both this head and shared network on transitions from that episode.

It allow us to make **deep** exploration.

Often to explore agent should make several non-greedy actions in a row, e-greedy strategy doesn't allow to do it.

Let’s watch a video…

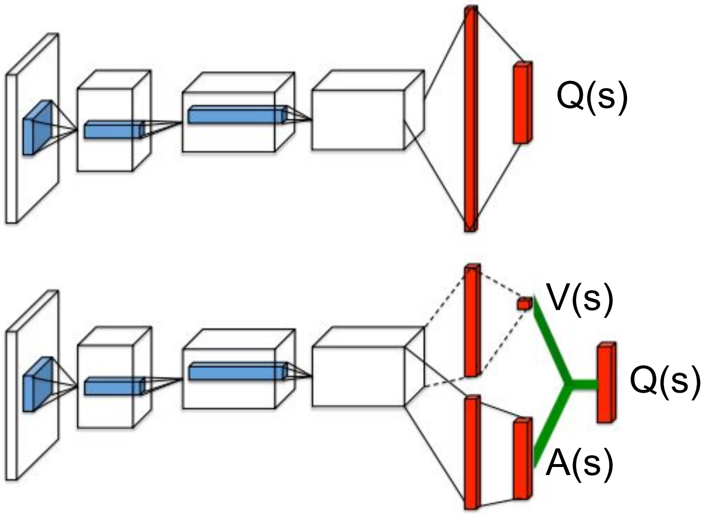
https://www.youtube.com/watch?v=UXurvvDY93o

Dueling Network Architectures for Deep Reinforcement Learning (2016, Deepmind)

Idea: change the network's architecture.

Recall:
Advantage Function A(s,a) = Q(s,a) - V(s)

So, Q(s,a) = A(s,a) + V(s)

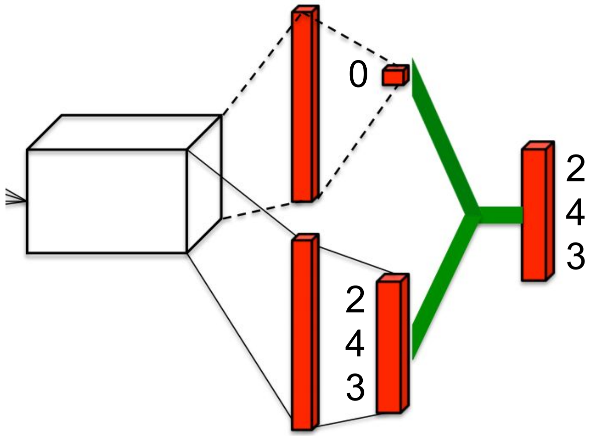


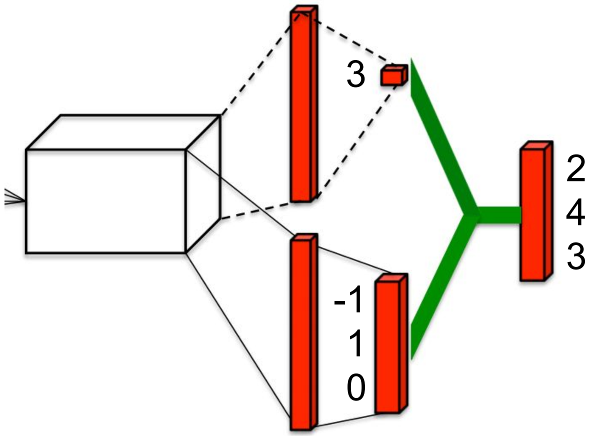
Do you see the problem?

Dueling Network Architectures for Deep Reinforcement Learning (2016, Deepmind)

Here is one extra freedom degree!

Example:





Dueling Network Architectures for Deep Reinforcement Learning (2016, Deepmind)

Solution: require $\max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha)$ to be equal to zero!

So the **Q-function** computes as:

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

Authors of this papers also introduced this way to compute Q-values:

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

They wrote that this variant increases stability of the optimization
(The fact that this loses the original semantics of Q doesn't matter)

Dueling Network Architectures for Deep Reinforcement Learning (2016, Deepmind)

Solution: require $\max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha)$ to be equal to zero!

So the **Q-function** computes as:

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

It's the homework!

Authors of this papers also introduced this way to compute Q-values:

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

They wrote that this variant increases stability of the optimization
(The fact that this loses the original semantics of Q doesn't matter)

LEARNING TO PLAY IN A DAY: FASTER DEEP REINFORCEMENT LEARNING BY OPTIMALITY TIGHTENING(2016)

Bellman equation:

$Q(s,a) = r + \gamma \max_{a'} Q(s', a')$

We can estimate it this way:

$$Q(s_t, a_t) = E[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})] >= E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^k \max_{a_{t+k}} Q(s_{t+k}, a_{t+k})]$$

Also we can rewrite it this way (divide all parts by γ^k and subtract k from indices)

$$Q(s_t, a_t) <= \max_{a_t} Q(s_t, a_t) <= E[\gamma^{-k} Q(s_{t-k}, a_{t-k}) - \gamma^{-k} r_{t-k} - \gamma^{-(k-1)} r_{t-k+1} - ... - \gamma^{-1} r_{t-1}]$$

We can store these two estimations together with the transition:

$$U^{min} = \gamma^{-k} Q(s_{t-k}, a_{t-k}) - \gamma^{-k} r_{t-k} - \gamma^{-(k-1)} r_{t-k+1} - ... - \gamma^{-1} r_{t-1}$$

$$L^{max} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^k \max_{a_{t+k}} Q(s_{t+k}, a_{t+k}))$$

LEARNING TO PLAY IN A DAY: FASTER DEEP REINFORCEMENT LEARNING BY OPTIMALITY TIGHTENING(2016)

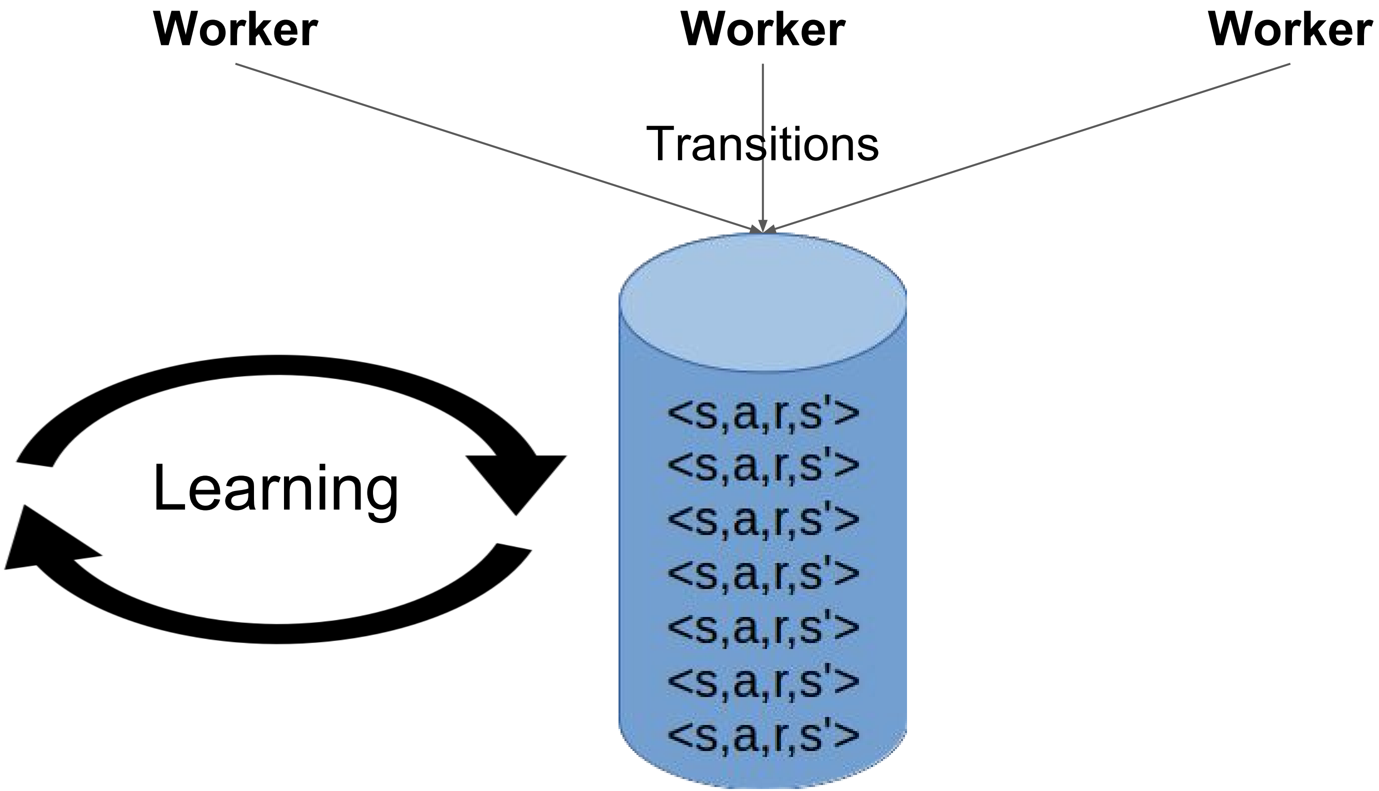
And now we can modify our loss function using these estimations:

$$y = r + \gamma Q(s', \mathrm{argmax}_{a'} Q(s', a', \Theta), \Theta^-)$$

$$L(\Theta) = E[(Q(s,a,\Theta) - y)^2 + \lambda(L^{max} - Q(s,a,\Theta))_+^2 + \lambda(Q(s,a,\Theta) - U_{min})_+^2]$$

Ours (10M)	**less than 1 day (1 GPU)**	**345.70%**	**105.74%**
DQN (200M)	more than 10 days (1 GPU)	241.06%	93.52%

Asynchronous Methods for Deep Reinforcement Learning (2016, Deepmind)



Questions?