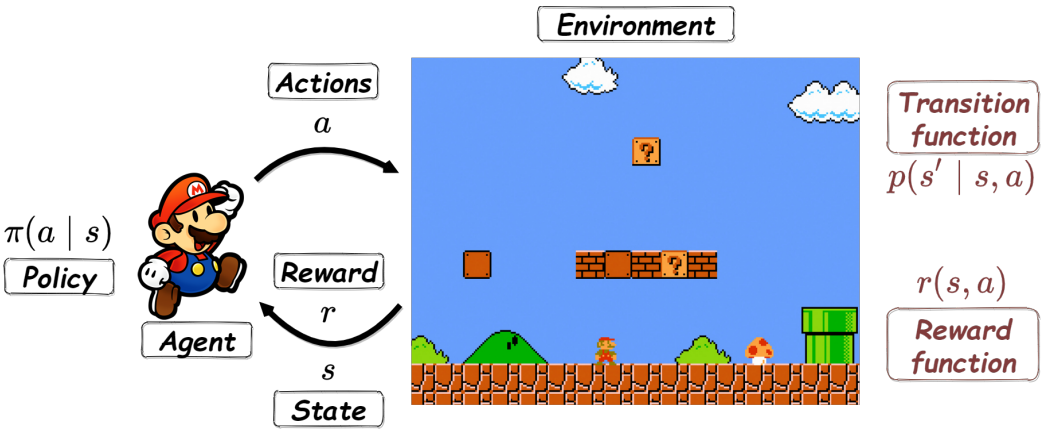
Deep Q-Learning

Reinforcement Learning

D. Sorokin

March 10, 2025

Benchmarks: Video Games



Reminder: Q-learning

Bellman optimality equation for Q^*: $\forall s, a$:

$$Q^*(s,a) := r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \max_{a'} Q^*(s', a')$$

Reminder: Q-learning

Bellman optimality equation for Q^*: $\forall s, a$:

$$Q^*(s,a) \coloneqq r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \max_{a'} Q^*(s',a')$$

Optimal policy:

$$\pi^*(s) \coloneqq \operatorname*{argmax}_a Q^*(s,a)$$

Reminder: Q-learning

Bellman optimality equation for Q^*: $\forall s, a$:

$$Q^*(s,a) := r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \max_{a'} Q^*(s',a')$$

Optimal policy:

$$\pi^*(s) := \operatorname*{argmax}_a Q^*(s,a)$$

Q-learning can learn Q^* from trial and error in *finite* MDPs: $|\mathcal{S}| \ll \infty, |\mathcal{A}| \ll \infty$

Important!

Q-learning is off-policy

Reminder: Q-learning update intuition

For transition (s, a, r, s'):

$$Q^*_{k+1}(s,a) \leftarrow Q^*_k(s,a) + \alpha_k \underbrace{\left(\overbrace{r + \gamma \max_{a'} Q^*_k(s', a')}^{\textbf{Bellman target}} - Q^*_k(s,a) \right)}_{\textbf{temporal difference}}$$

Reminder: Q-learning update intuition

For transition (s, a, r, s'):

$$Q^*_{k+1}(s,a) \leftarrow Q^*_k(s,a) + \alpha_k \underbrace{\left(\overbrace{r + \gamma \max_{a'} Q^*_k(s', a')}^{\textbf{Bellman target}} - Q^*_k(s,a) \right)}_{\textbf{temporal difference}}$$

s, a — considered **fixed**, can be chosen arbitrarily;

(!) *each pair must be updated infinite times;*

Reminder: Q-learning update intuition

For transition (s, a, r, s'):

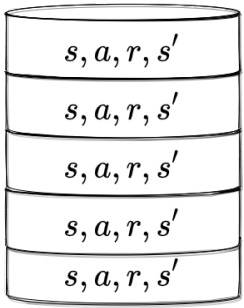
$$Q^*_{k+1}(s,a) \leftarrow Q^*_k(s,a) + \alpha_k \underbrace{\left(\overbrace{r + \gamma \max_{a'} Q^*_k(s', a')}^{\textbf{Bellman target}} - Q^*_k(s,a) \right)}_{\textbf{temporal difference}}$$

s, a — considered **fixed**, can be chosen arbitrarily;

(!) *each pair must be updated infinite times;*

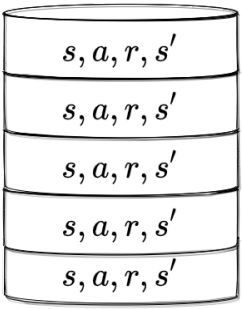
$s' \sim p(s' \mid s, a)$ — **random variable** from interaction experience.

Experience Replay



Collect all experienced transitions in **replay buffer**!

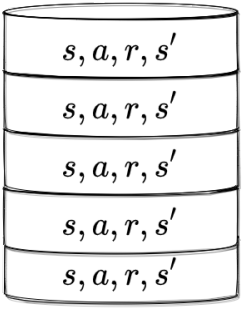
Experience Replay



Collect all experienced transitions in **replay buffer**!

On each training step sample **uniformly** random transition from experience replay and perform update using it.

Experience Replay

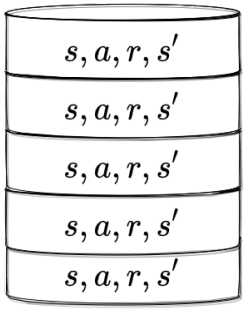


Collect all experienced transitions in **replay buffer**!

On each training step sample **uniformly** random transition from experience replay and perform update using it.

✓ allows reuse of experience;

Experience Replay

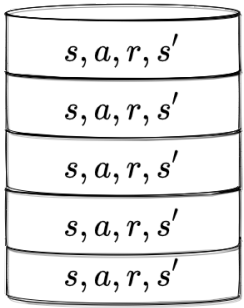


Collect all experienced transitions in **replay buffer**!

On each training step sample **uniformly** random transition from experience replay and perform update using it.

- ✓ allows reuse of experience;
- ✓ allows training on expert data;

Experience Replay



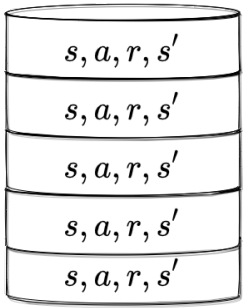
Collect all experienced transitions in **replay buffer**!

On each training step sample **uniformly** random transition from experience replay and perform update using it.

- ✓ allows reuse of experience;
- ✓ allows training on expert data;
- × sample s' comes from empirical approximation:

$$s' \sim \hat{p}(s' \mid s, a) \approx p(s' \mid s, a)$$

Experience Replay



Collect all experienced transitions in **replay buffer**!

On each training step sample **uniformly** random transition from experience replay and perform update using it.

- ✓ allows reuse of experience;
- ✓ allows training on expert data;
- × sample s' comes from empirical approximation:

$$s' \sim \hat{p}(s' \mid s, a) \approx p(s' \mid s, a)$$

- ✓ smoothes out with size of experience replay!

Q-learning with experience replay

Q-learning (online)

Initialize $Q^*(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};

Q-learning with experience replay

`Q-learning (online)`

Initialize $Q^*(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};
- $y := r_k + \gamma \max_{a_{k+1}} Q^*(s_{k+1}, a_{k+1})$;

Q-learning with experience replay

Q-learning (online)

Initialize $Q^*(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};
- $y := r_k + \gamma \max_{a_{k+1}} Q^*(s_{k+1}, a_{k+1})$;
- $Q^*(s_k, a_k) \leftarrow (1 - \alpha_k) Q^*(s_k, a_k) + \alpha_k y$

Q-learning with experience replay

Q-learning (online)

Initialize $Q^*(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};
- $y := r_k + \gamma \max_{a_{k+1}} Q^*(s_{k+1}, a_{k+1})$;
- $Q^*(s_k, a_k) \leftarrow (1 - \alpha_k) Q^*(s_k, a_k) + \alpha_k y$

Q-learning (with replay buffer)

Initialize $Q^*(s, a)$ arbitrarily, $\mathcal{D} = \varnothing$;

Q-learning with experience replay

Q-learning (online)

Initialize $Q^*(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};
- $y := r_k + \gamma \max_{a_{k+1}} Q^*(s_{k+1}, a_{k+1})$;
- $Q^*(s_k, a_k) \leftarrow (1 - \alpha_k) Q^*(s_k, a_k) + \alpha_k y$

Q-learning (with replay buffer)

Initialize $Q^*(s, a)$ arbitrarily, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};
- store (s_k, a_k, r_k, s_{k+1}) in $\mathcal{D}$;

Q-learning with experience replay

Q-learning (online)

Initialize $Q^*(s, a)$ arbitrarily;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};
- $y := r_k + \gamma \max_{a_{k+1}} Q^*(s_{k+1}, a_{k+1})$;
- $Q^*(s_k, a_k) \leftarrow (1 - \alpha_k) Q^*(s_k, a_k) + \alpha_k y$

Q-learning (with replay buffer)

Initialize $Q^*(s, a)$ arbitrarily, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a))$;
- observe r_k, s_{k+1};
- store (s_k, a_k, r_k, s_{k+1}) in $\mathcal{D}$;
- sample (s, a, r, s') from $\mathcal{D}$;
- $y := r + \gamma \max_{a'} Q^*(s', a')$;
- $Q^*(s, a) \leftarrow (1 - \alpha_k) Q^*(s, a) + \alpha_k y$

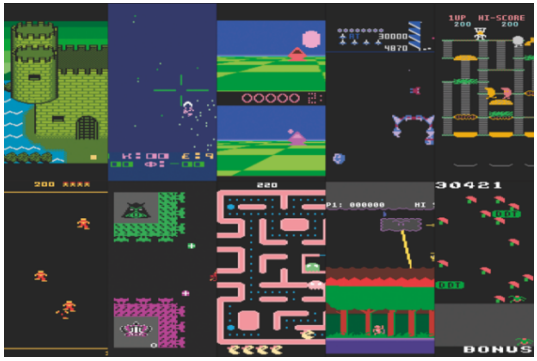
Atari Games

Setup: $|\mathcal{S}| \ll \infty, |\mathcal{A}| \ll \infty$

Atari Games

Setup: ~~$|\mathcal{S}| \ll \infty$~~, $|\mathcal{A}| \ll \infty$

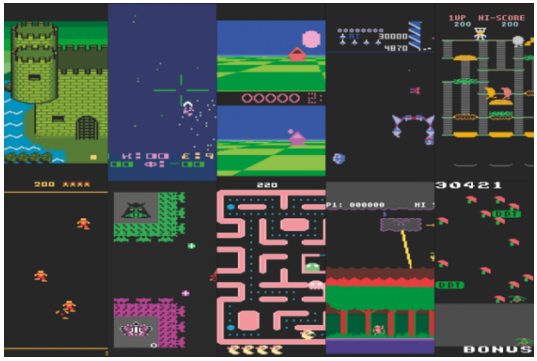
- 57 various games
- Only screen image as input.
- No game-specific features.
- Finite-state case... not quite finite.
- $|\mathcal{A}| \leqslant 18$



Atari Games

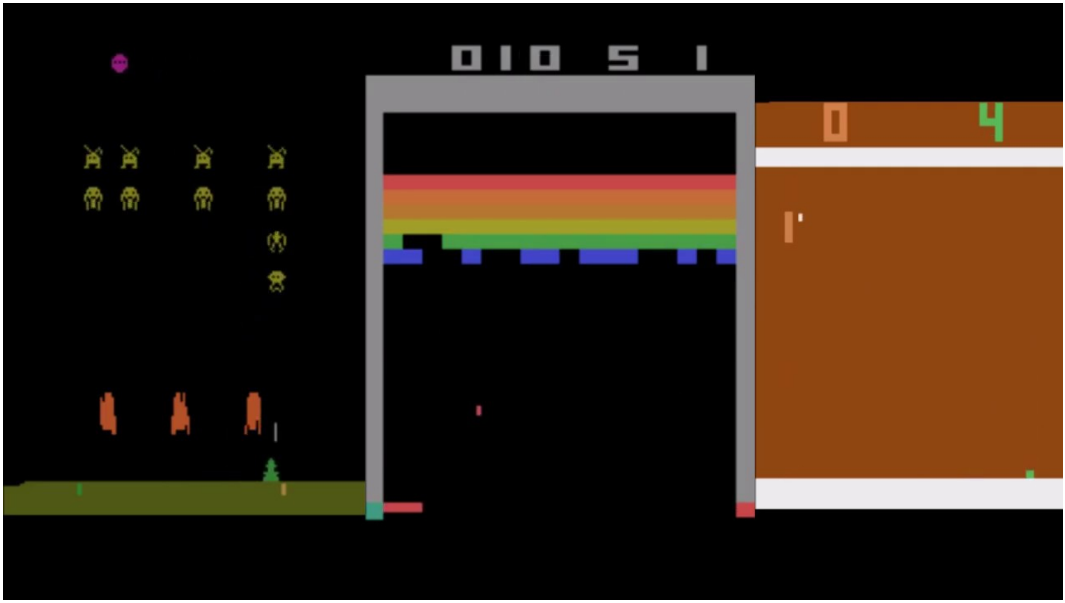
Setup: ~~$|\mathcal{S}| \ll \infty$~~, $|\mathcal{A}| \ll \infty$

- 57 various games
- Only screen image as input.
- No game-specific features.
- Finite-state case... not quite finite.
- $|\mathcal{A}| \leqslant 18$



Approximate $Q^*(s, a)$ with neural network!

Is Atari game a MDP?



Practical notes: preprocessing

Action selection frequency:

- Framestack;
- Frameskip;
- MaxAndSkip;
- (sometimes) Sticky actions;

Atari-specific preprocessing:

- EpisodicLife;
- FireReset;

Standard tricks:

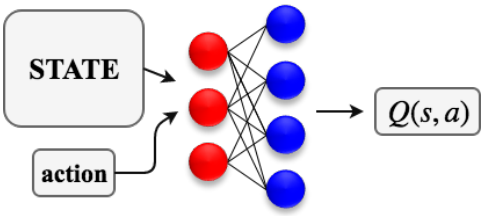
- Crop image;
- Rescale (often to 84x84);
- Grayscale;

Reward preprocessing:

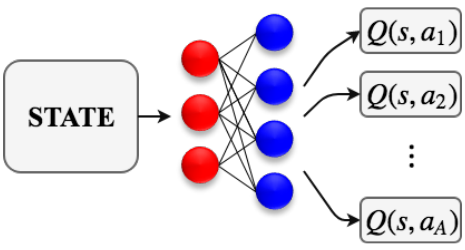
(!) Clip reward to $\{-1, 0, 1\}$;

Deep Q-network

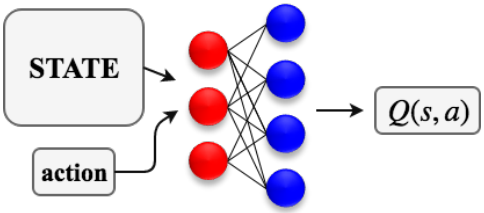
Option 1



Option 2

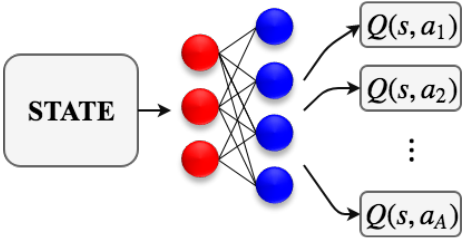


Option 1



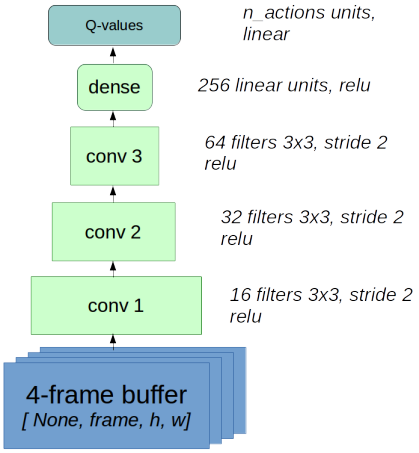
× $\underset{a}{\operatorname{argmax}}\, Q^*(s, a, \theta)$ — expensive!

Option 2



✓ $\underset{a}{\operatorname{argmax}}\, Q^*(s, a, \theta)$ — one forward pass.

Deep Q-network: architecture



- 3-4 convolutional layers followed by 1-2 dense layers;
- stride > 1 for size reduction; linear layers still wide;

Think twice before using:

- × max pooling
- × batch normalization
- × dropout

How to train your network?

Given:

- dataset (x, y), where x — input, y — output;
- loss function $\text{Loss}(y, \hat{y})$;
- parametric family $\hat{y}(x, \theta)$;

How to train your network?

Given:

- dataset (x, y), where x — input, y — output;
- loss function $\text{Loss}(y, \hat{y})$;
- parametric family $\hat{y}(x, \theta)$;

in **supervised learning** we optimized

$$\sum_{(x,y)} \text{Loss}(y, \hat{y}(x, \theta)) \to \min_{\theta}$$

using techniques based on stochastic gradient descent.

How to train your network?

Given:

- dataset (x, y), where x — input, y — output;
- loss function $\text{Loss}(y, \hat{y})$;
- parametric family $\hat{y}(x, \theta)$;

in **supervised learning** we optimized

$$\sum_{(x,y)} \text{Loss}(y, \hat{y}(x, \theta)) \to \min_{\theta}$$

using techniques based on stochastic gradient descent.

In **reinforcement learning**:

- we have to collect all data ourself
  - some exploration-exploitation technique is required, e. g. ε-greedy;
  - the data is not i.i.d.

How to train your network?

Given:

- dataset (x, y), where x — input, y — output;
- loss function $\text{Loss}(y, \hat{y})$;
- parametric family $\hat{y}(x, \theta)$;

in **supervised learning** we optimized

$$\sum_{(x,y)} \text{Loss}(y, \hat{y}(x, \theta)) \to \min_{\theta}$$

using techniques based on stochastic gradient descent.

In **reinforcement learning**:

- we have to collect all data ourself
  - some exploration-exploitation technique is required, e. g. ε-greedy;
  - the data is not i.i.d.
- what is the **target** y?
- what is the loss function?

How to train your network?

Given:

- dataset (x, y), where x — input, y — output;
- loss function $\text{Loss}(y, \hat{y})$;
- parametric family $\hat{y}(x, \theta)$;

in **supervised learning** we optimized

$$\sum_{(x,y)} \text{Loss}(y, \hat{y}(x, \theta)) \to \min_{\theta}$$

using techniques based on stochastic gradient descent.

In **reinforcement learning**:

- we have to collect all data ourself
  - some exploration-exploitation technique is required, e. g. ε-greedy;
  - the data is not i.i.d.
- what is the **target** y?
- what is the loss function?

Important

Q-learning is SGD.

Intuition: Q-learning as gradient descent

Consider the following setting:

- *tabular* parametric family $Q^*(s, a, \theta) := \theta_{s,a}$;

Intuition: Q-learning as gradient descent

Consider the following setting:

- *tabular* parametric family $Q^*(s, a, \theta) := \theta_{s,a}$;
- define Bellman target as $y := r(s, a) + \gamma \max_{a'} Q^*(s', a', \theta)$;

Intuition: Q-learning as gradient descent

Consider the following setting:

- *tabular* parametric family $Q^*(s, a, \theta) := \theta_{s,a}$;
- define Bellman target as $y := r(s, a) + \gamma \max_{a'} Q^*(s', a', \theta)$;

$$\theta_{s,a} \leftarrow \theta_{s,a} + \frac{\alpha}{2}\left(y - \overbrace{Q^*(s, a, \theta)}^{\hat{y}}\right)$$

Intuition: Q-learning as gradient descent

Consider the following setting:

- *tabular* parametric family $Q^*(s, a, \theta) := \theta_{s,a}$;
- define Bellman target as $y := r(s, a) + \gamma \max_{a'} Q^*(s', a', \theta)$;

$$\theta_{s,a} \leftarrow \theta_{s,a} + \frac{\alpha}{2}\left(y - \overbrace{Q^*(s, a, \theta)}^{\hat{y}}\right)$$

$$\theta_{s,a} \leftarrow \theta_{s,a} - \alpha \nabla_{\hat{y}} \left(y - \hat{y}\right)^2$$

Intuition: Q-learning as gradient descent

Consider the following setting:

- *tabular* parametric family $Q^*(s, a, \theta) := \theta_{s,a}$;
- define Bellman target as $y := r(s, a) + \gamma \max_{a'} Q^*(s', a', \theta)$;

$$\theta_{s,a} \leftarrow \theta_{s,a} + \frac{\alpha}{2}\left(y - \overbrace{Q^*(s, a, \theta)}^{\hat{y}}\right)$$

$$\theta_{s,a} \leftarrow \theta_{s,a} - \alpha \nabla_{\hat{y}} \left(y - \hat{y}\right)^2$$

$$\theta \leftarrow \theta - \alpha \nabla_{\hat{y}} \left(y - \hat{y}\right)^2 \mathrm{OHE}\left(s, a\right)$$

Intuition: Q-learning as gradient descent

Consider the following setting:

- *tabular* parametric family $Q^*(s, a, \theta) := \theta_{s,a}$;
- define Bellman target as $y := r(s, a) + \gamma \max_{a'} Q^*(s', a', \theta)$;

$$\theta_{s,a} \leftarrow \theta_{s,a} + \frac{\alpha}{2}\left(y - \overbrace{Q^*(s, a, \theta)}^{\hat{y}}\right)$$

$$\theta_{s,a} \leftarrow \theta_{s,a} - \alpha \nabla_{\hat{y}} \left(y - \hat{y}\right)^2$$

$$\theta \leftarrow \theta - \alpha \nabla_{\hat{y}} \left(y - \hat{y}\right)^2 \mathrm{OHE}\left(s, a\right)$$

$$\theta \leftarrow \theta - \alpha \underbrace{\nabla_{\hat{y}} \left(y - \hat{y}\right)^2 \nabla_\theta \hat{y}}_{\nabla_\theta \mathrm{MSE}(s,a,y,\theta)}$$

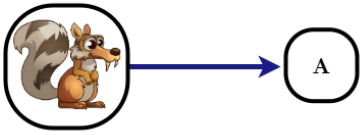
Why stop gradients from target?

$$\mathbb{E}_{s,a,r,s'}\Big(\underbrace{r + \gamma \max_{a} Q^*(s', a')}_{\text{fixed; no gradient flow!}} - Q^*(s, a)\Big)^2 \to \min_{Q^*}$$

Why stop gradients from target?

$$\mathbb{E}_{s,a,r,s'}\Big(\underbrace{r + \gamma \max_{a} Q^*(s', a')}_{\text{fixed; no gradient flow!}} - Q^*(s, a)\Big)^2 \to \min_{Q^*}$$

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



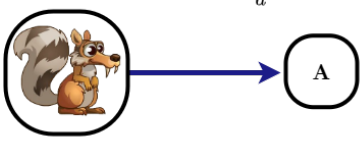
Why stop gradients from target?

$$\mathbb{E}_{s,a,r,s'}\Big(\underbrace{r + \gamma \max_a Q^*(s', a')}_{\text{fixed; no gradient flow!}} - Q^*(s, a)\Big)^2 \to \min_{Q^*}$$

This is a consequence of **causality**:

- ✓ match *past* estimations to *future*;
- ✗ do not match *future* estimations to *past*;

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

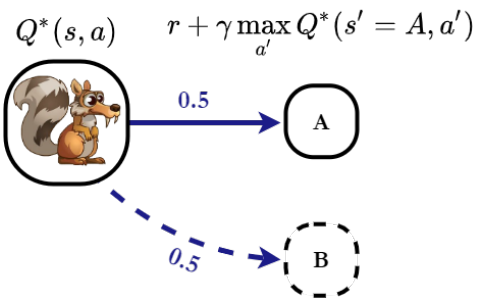


Why stop gradients from target?

$$\mathbb{E}_{s,a,r,s'}\Big(\underbrace{r + \gamma \max_{a} Q^*(s', a')}_{\text{fixed; no gradient flow!}} - Q^*(s, a)\Big)^2 \to \min_{Q^*}$$

This is a consequence of **causality**:

- ✓ match *past* estimations to *future*;
- ✗ do not match *future* estimations to *past*;



Why stop gradients from target?

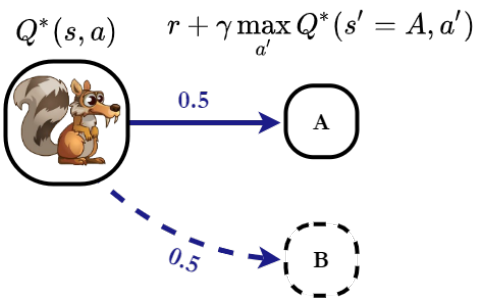
$$\mathbb{E}_{s,a,r,s'}\Big(\underbrace{r + \gamma \max_a Q^*(s', a')}_{\text{fixed; no gradient flow!}} - Q^*(s, a)\Big)^2 \to \min_{Q^*}$$

This is a consequence of **causality**:

- ✓ match *past* estimations to *future*;
- ✗ do not match *future* estimations to *past*;

Note that unlike supervised learning:

- target «depends» on our own model;
- it is still *better* that we currently have; («provides direction»)



Consider the following **regression task**:

- s, a is input;
- $y \in \mathbb{R}$ is target:

$$y(s, a) := r + \gamma \max_{a'} Q^*(s', a', \theta_k)$$

where $r = r(s, a), s' \sim p(s' \mid s, a)$;

Consider the following **regression task**:

- s, a is input;
- $y \in \mathbb{R}$ is target:

$$y(s, a) := r + \gamma \max_{a'} Q^*(s', a', \theta_k)$$

  where $r = r(s, a), s' \sim p(s' \mid s, a)$;
- MSE loss function: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

Consider the following **regression task**:

- s, a is input;
- $y \in \mathbb{R}$ is target:

$$y(s, a) := r + \gamma \max_{a'} Q^*(s', a', \theta_k)$$

  where $r = r(s, a), s' \sim p(s' \mid s, a)$;
- MSE loss function: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

$$\mathbb{E}_{(s,a,r,s')}(y - Q^*(s, a, \theta_{k+1}))^2 \to \min_{\theta_{k+1}}$$

Solving Bellman equations: deep version

$$\mathbb{E}_{(s,a,r,s')}(y - Q^*(s, a, \theta_{k+1}))^2 \to \min_{\theta_{k+1}}$$

What solution is optimal?

$$Q^*(s, a, \theta^*_{k+1}) =$$

Solving Bellman equations: deep version

$$\mathbb{E}_{(s,a,r,s')}(y - Q^*(s, a, \theta_{k+1}))^2 \to \min_{\theta_{k+1}}$$

What solution is optimal?

$$Q^*(s, a, \theta^*_{k+1}) = \mathbb{E}_{r,s'|s,a} y =$$

Solving Bellman equations: deep version

$$\mathbb{E}_{(s,a,r,s')}(y - Q^*(s, a, \theta_{k+1}))^2 \to \min_{\theta_{k+1}}$$

What solution is optimal?

$$Q^*(s, a, \theta^*_{k+1}) = \mathbb{E}_{r,s'|s,a} y = \mathbb{E}_{r,s'|s,a}\left[r + \gamma \max_{a'} Q^*(s', a', \theta_k)\right] =$$

Solving Bellman equations: deep version

$$\mathbb{E}_{(s,a,r,s')}(y - Q^*(s, a, \theta_{k+1}))^2 \to \min_{\theta_{k+1}}$$

What solution is optimal?

$$Q^*(s, a, \theta^*_{k+1}) = \mathbb{E}_{r,s'|s,a} y = \mathbb{E}_{r,s'|s,a}\left[r + \gamma \max_{a'} Q^*(s', a', \theta_k)\right] =$$

$$= \underbrace{r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \max_{a'} Q^*(s', a', \theta_k)}_{\text{value iteration update}}$$

Target network

For how long to solve one regression task?

Target network

For how long to solve one regression task?

- One SGD iteration

Target network

For how long to solve one regression task?

- One SGD iteration
  - × completely unstable :(

Target network

For how long to solve one regression task?

- One SGD iteration
  - $\times$ completely unstable :(
- Till convergence

Target network

For how long to solve one regression task?

- One SGD iteration
  - × completely unstable :(
- Till convergence
  - × too long

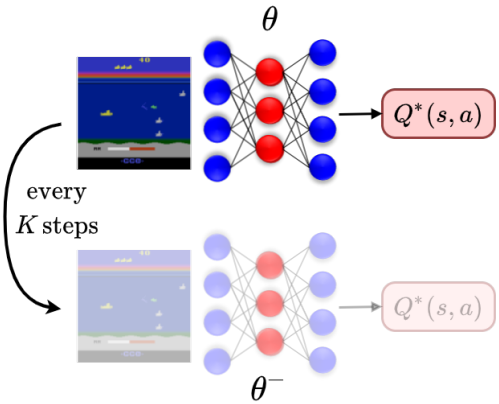
✓ ≈1000 SGD iterations

Target network

For how long to solve one regression task?

- One SGD iteration
  - × completely unstable :(
- Till convergence
  - × too long

✓ ≈1000 SGD iterations



Store a copy of your old Q-network $Q^*(s, a, \theta^-)$

Target network

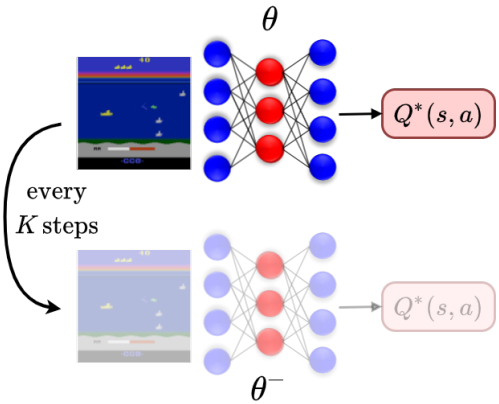
For how long to solve one regression task?

- One SGD iteration
  - × completely unstable :(
- Till convergence
  - × too long

✓ ≈1000 SGD iterations

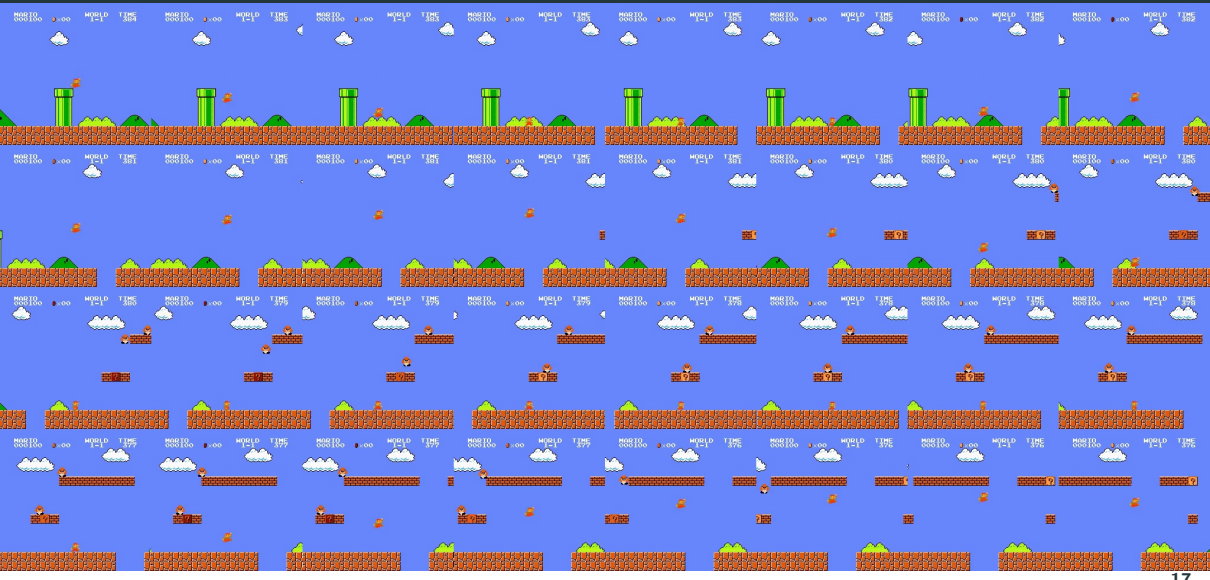
a) $\theta^- \leftarrow \theta$ every K SGD iterations

b) $\theta^- \leftarrow (1-\beta)\theta^- + \beta\theta$



Store a copy of your old Q-network $Q^*(s, a, \theta^-)$

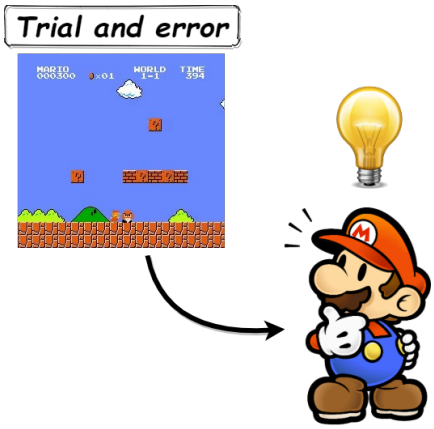
Distributional shift issue



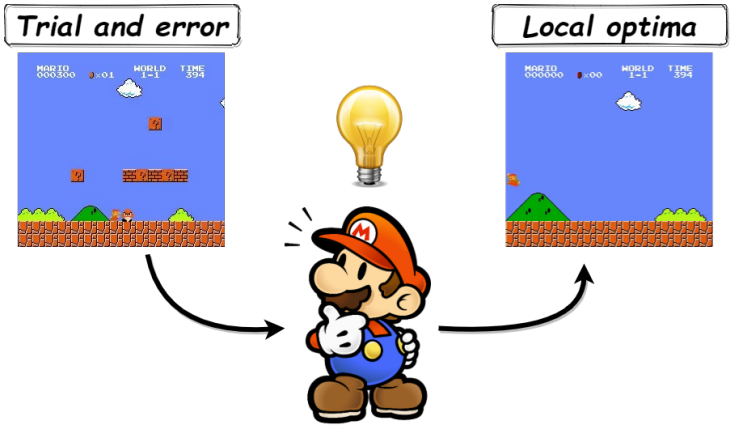
Experience Replay

Do we still need exploration?

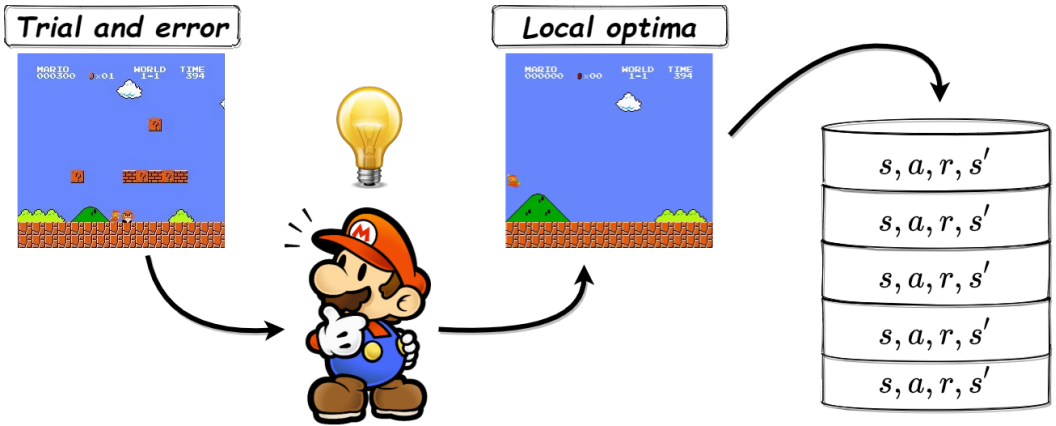
Do we still need exploration?



Do we still need exploration?



Do we still need exploration?



Full alrogithm

Deep Q-learning

Initialize $Q^*(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \varnothing$;

Full alrogithm

Deep Q-learning

Initialize $Q^*(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a, \theta))$;
- observe $r_k, s_{k+1}, \mathrm{done}_{k+1}$, store $(s_k, a_k, r_k, s_{k+1}, \mathrm{done}_{k+1})$ in $\mathcal{D}$;

Full alrogithm

Deep Q-learning

Initialize $Q^*(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a, \theta))$;
- observe $r_k, s_{k+1}, \mathrm{done}_{k+1}$, store $(s_k, a_k, r_k, s_{k+1}, \mathrm{done}_{k+1})$ in $\mathcal{D}$;
- sample batch of transitions $\mathbb{T} := (s, a, r, s', \mathrm{done})$ from $\mathcal{D}$;

Full alrogithm

Deep Q-learning

Initialize $Q^*(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a, \theta))$;
- observe $r_k, s_{k+1}, \mathrm{done}_{k+1}$, store $(s_k, a_k, r_k, s_{k+1}, \mathrm{done}_{k+1})$ in $\mathcal{D}$;
- sample batch of transitions $\mathbb{T} := (s, a, r, s', \mathrm{done})$ from $\mathcal{D}$;
- $y(\mathbb{T}) := r + \gamma(1 - \mathrm{done}) \max_{a'} Q^*(s', a', \theta^-)$;

Full alrogithm

Deep Q-learning

Initialize $Q^*(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a, \theta))$;
- observe $r_k, s_{k+1}, \mathrm{done}_{k+1}$, store $(s_k, a_k, r_k, s_{k+1}, \mathrm{done}_{k+1})$ in $\mathcal{D}$;
- sample batch of transitions $\mathbb{T} := (s, a, r, s', \mathrm{done})$ from $\mathcal{D}$;
- $y(\mathbb{T}) := r + \gamma(1 - \mathrm{done}) \max_{a'} Q^*(s', a', \theta^-)$;
- perform a step of gradient descent:

$$\theta \leftarrow \theta - \frac{\alpha}{B} \sum_{\mathbb{T}} \nabla_\theta \left(Q^*(s, a, \theta) - y(\mathbb{T}) \right)^2$$

Full alrogithm

Deep Q-learning

Initialize $Q^*(s, a, \theta)$ arbitrarily, $\theta^- := \theta$, $\mathcal{D} = \varnothing$;

observe s_0;

for $k = 0, 1, 2 \ldots$

- take action $a_k \sim \varepsilon$-greedy$(Q^*(s_k, a, \theta))$;
- observe $r_k, s_{k+1}, \mathrm{done}_{k+1}$, store $(s_k, a_k, r_k, s_{k+1}, \mathrm{done}_{k+1})$ in $\mathcal{D}$;
- sample batch of transitions $\mathbb{T} := (s, a, r, s', \mathrm{done})$ from $\mathcal{D}$;
- $y(\mathbb{T}) := r + \gamma(1 - \mathrm{done}) \max_{a'} Q^*(s', a', \theta^-)$;
- perform a step of gradient descent:

$$\theta \leftarrow \theta - \frac{\alpha}{B} \sum_{\mathbb{T}} \nabla_\theta \left(Q^*(s, a, \theta) - y(\mathbb{T})\right)^2$$

- update target network: if $k \bmod K = 0$: $\theta^- \leftarrow \theta$

There is a lot to improve

Overestimation Bias and how to fight it

$$\max_{a'}(Q^*(s', a', \theta) + \underbrace{\varepsilon(s', a')}_{\text{approximation error or luck}})$$

Overestimation Bias and how to fight it

$$\mathbb{E}_{\varepsilon} \max_{a'}(Q^*(s', a', \theta) + \underbrace{\varepsilon(s', a')}_{\text{approximation error or luck}}) \geqslant \max_{a'} Q^*(s', a')$$

Overestimation Bias and how to fight it

$$\mathbb{E}_{\varepsilon} \max_{a'}(Q^*(s', a', \theta) + \underbrace{\varepsilon(s', a')}_{\text{approximation error or luck}}) \geqslant \max_{a'} Q^*(s', a')$$

Decouple **action selection** and **action evaluation**:

$$\max_{a'} Q^*(s', a') = \overbrace{Q^*(s', \underbrace{\operatorname{argmax}_{a'} Q^*(s', a')}_{\text{action selection}})}^{\text{action evaluation}}$$

Overestimation Bias and how to fight it

$$\mathbb{E}_{\varepsilon} \max_{a'}(Q^*(s', a', \theta) + \underbrace{\varepsilon(s', a')}_{\text{approximation error or luck}}) \geqslant \max_{a'} Q^*(s', a')$$

Decouple **action selection** and **action evaluation**:

$$\max_{a'} Q^*(s', a') = \overbrace{Q^*(s', \underbrace{\operatorname{argmax}_{a'} Q^*(s', a')}_{\text{action selection}})}^{\text{action evaluation}} \approx Q_1^*(s', \operatorname*{argmax}_{a'} Q_2^*(s', a'))$$

Action Evaluation: options

Name	Networks	Targets
Double Q-learning	Q_1^* Q_2^*	$y_1 = r + \gamma Q_?^*(s', \underset{a'}{\operatorname{argmax}}\, Q_?^*(s', a'))$

Action Evaluation: options

Name	Networks	Targets
Double Q-learning	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \underset{a'}{\operatorname{argmax}}\, Q_1^*(s', a'))$

Action Evaluation: options

Name	Networks	Targets
Double Q-learning	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \underset{a'}{\operatorname{argmax}}\, Q_1^*(s', a'))$ $y_2 = r + \gamma Q_1^*(s', \underset{a'}{\operatorname{argmax}}\, Q_2^*(s', a'))$

Action Evaluation: options

Name	Networks	Targets
~~Double Q-learning~~ Twin DQN	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \underset{a'}{\operatorname{argmax}}\, Q_1^*(s', a'))$ $y_2 = r + \gamma Q_1^*(s', \underset{a'}{\operatorname{argmax}}\, Q_2^*(s', a'))$
Double DQN	Q^* Q_-^* — target network	

Action Evaluation: options

Name	Networks	Targets
~~Double Q-learning~~ Twin DQN	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$ $y_2 = r + \gamma Q_1^*(s', \operatorname{argmax}_{a'} Q_2^*(s', a'))$
Double DQN	Q^* Q_-^* — target network	$y = r + \gamma Q_-^*(s', \operatorname{argmax}_{a'} Q^*(s', a'))$

Action Evaluation: options

Name	Networks	Targets
~~Double Q-learning~~ ~~Twin DQN~~ ?!?	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$ $y_2 = r + \gamma Q_1^*(s', \operatorname{argmax}_{a'} Q_2^*(s', a'))$
Double DQN	Q^* Q_-^* — target network	$y = r + \gamma Q_-^*(s', \operatorname{argmax}_{a'} Q^*(s', a'))$
Twin DQN («Clipped Double DQN»)	Q_1^* Q_2^*	

Action Evaluation: options

Name	Networks	Targets
~~Double Q-learning~~ ~~Twin DQN~~ ?!?	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$ $y_2 = r + \gamma Q_1^*(s', \operatorname{argmax}_{a'} Q_2^*(s', a'))$
Double DQN	Q^* Q_-^* — target network	$y = r + \gamma Q_-^*(s', \operatorname{argmax}_{a'} Q^*(s', a'))$
Twin DQN («Clipped Double DQN»)	Q_1^* Q_2^*	$y_1 = r + \gamma \min_{i=1,2} Q_i^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$

Action Evaluation: options

Name	Networks	Targets
~~Double Q-learning~~ ~~Twin DQN~~ ?!?	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$ $y_2 = r + \gamma Q_1^*(s', \operatorname{argmax}_{a'} Q_2^*(s', a'))$
Double DQN	Q^* Q_-^* — target network	$y = r + \gamma Q_-^*(s', \operatorname{argmax}_{a'} Q^*(s', a'))$
Twin DQN («Clipped Double DQN»)	Q_1^* Q_2^*	$y_1 = r + \gamma \min_{i=1,2} Q_i^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$ $y_2 = r + \gamma \min_{i=1,2} Q_i^*(s', \operatorname{argmax}_{a'} Q_2^*(s', a'))$

Literature

- Playing Atari with Deep Reinforcement Learning (2013);
- Deep Reinforcement Learning with Double Q-learning (2015);