Value-based Methods

Reinforcement Learning

October 5, 2021

MSU

Reminder: Q-learning

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

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

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

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

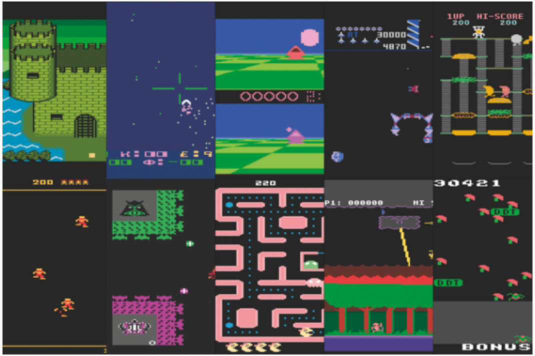
Important!

Q-learning is **off-policy**

Atari Games

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

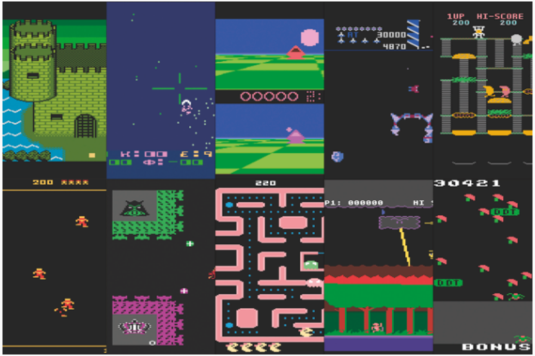
- 57 various games
- Only screen image as input.
- No game-specific features.
- Finite-state case... not quite finite.
- $|\mathcal{A}| \leqslant 18$



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 $\{-1, 0, 1\}$;

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;

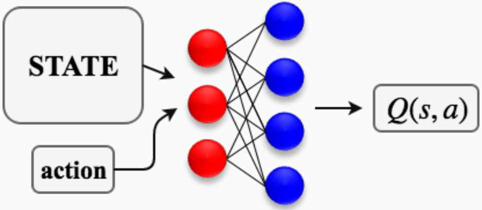
Reward preprocessing:

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

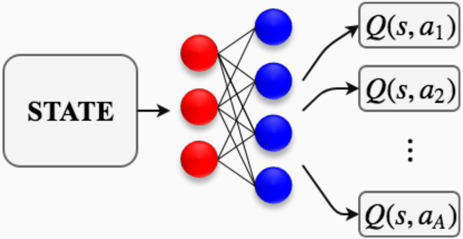
Important!

No hyperparameter tuning for each specific game!

Option 1

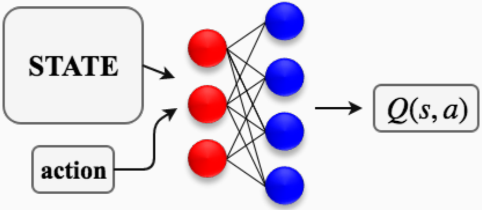


Option 2



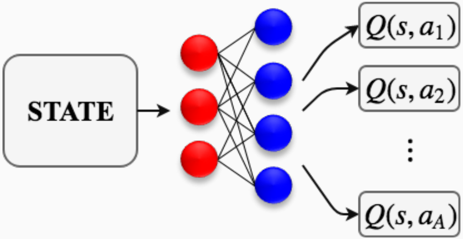
Deep Q-network

Option 1



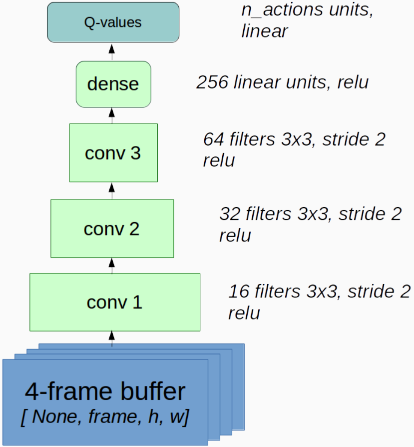
$\times$ $\underset{a}{\operatorname{argmax}}\, Q^*(s, a, \theta)$ — expensive!

Option 2



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

Idea: Approximate Dynamic Programming

Consider the following **regression task**:

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

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

Idea: Approximate Dynamic Programming

Consider the following **regression task**:

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

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

- MSE loss function: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

Idea: Approximate Dynamic Programming

Consider the following **regression task**:

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

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

- MSE loss function: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

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

Looking at the gradient

$$\nabla_\theta \frac{1}{2} \left(y - Q^*(s, a, \theta)\right)^2 =$$

Looking at the gradient

$$\nabla_\theta \frac{1}{2}\left(y - Q^*(s,a,\theta)\right)^2 = \underbrace{\left(y - Q^*(s,a,\theta)\right)}_{\text{temporal difference}} \overbrace{\nabla_\theta Q^*(s,a,\theta)}^{\substack{\text{how to increase}\\ \text{output value}}} =$$

$$=$$

Looking at the gradient

$$\nabla_\theta \frac{1}{2}\left(y - Q^*(s,a,\theta)\right)^2 = \underbrace{\left(y - Q^*(s,a,\theta)\right)}_{\text{temporal difference}} \overbrace{\nabla_\theta Q^*(s,a,\theta)}^{\text{how to increase output value}} =$$

$$= \left(r + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s',a',\theta_k) - Q^*(s,a,\theta)\right) \nabla_\theta Q^*(s,a,\theta)$$

Looking at the gradient

$$\nabla_\theta \frac{1}{2}\left(y - Q^*(s,a,\theta)\right)^2 = \underbrace{\left(y - Q^*(s,a,\theta)\right)}_{\text{temporal difference}} \overbrace{\nabla_\theta Q^*(s,a,\theta)}^{\text{how to increase output value}} =$$

$$= \left(r + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s',a',\theta_k) - Q^*(s,a,\theta)\right) \nabla_\theta Q^*(s,a,\theta) \approx$$

$$\approx \left(r + \gamma \max_{a'} Q^*(s',a',\theta_k) - Q^*(s,a,\theta)\right) \nabla_\theta Q^*(s,a,\theta),$$

where $s' \sim p(s' \mid s,a)$.

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

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: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

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
  - × completely unstable :(
- Till convergence

Target network

For how long to solve one regression task?

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

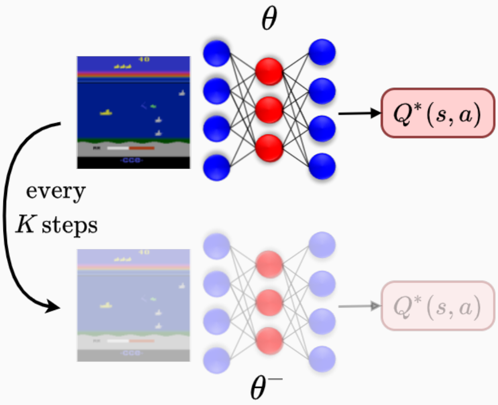
✓ ≈1000 SGD iterations

Target network

For how long to solve one regression task?

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

✓ $\approx$1000 SGD iterations

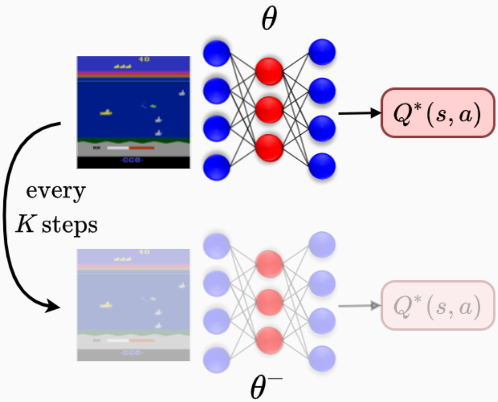


Store a copy of your old Q-network $Q^*(s, a, \theta^-)$

Target network

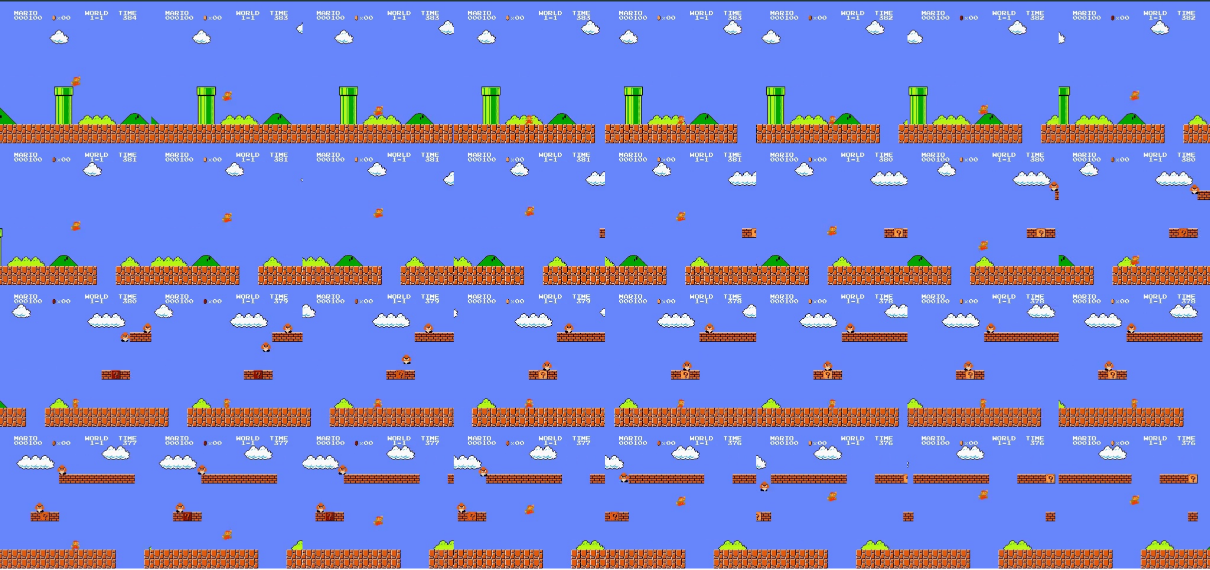
For how long to solve one regression task?

- 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 $Q^*(s, a, \theta^-)$

Sample Decorrelation



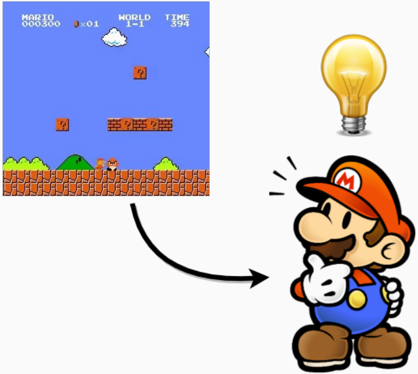
Experience Replay

Do we still need exploration?

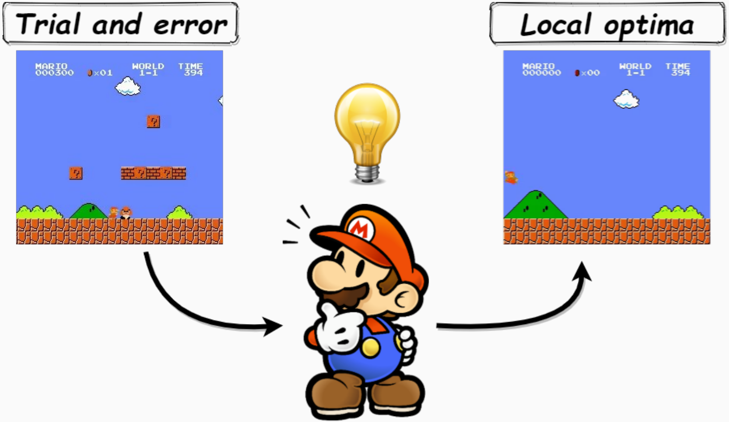
Trial and error

Do we still need exploration?

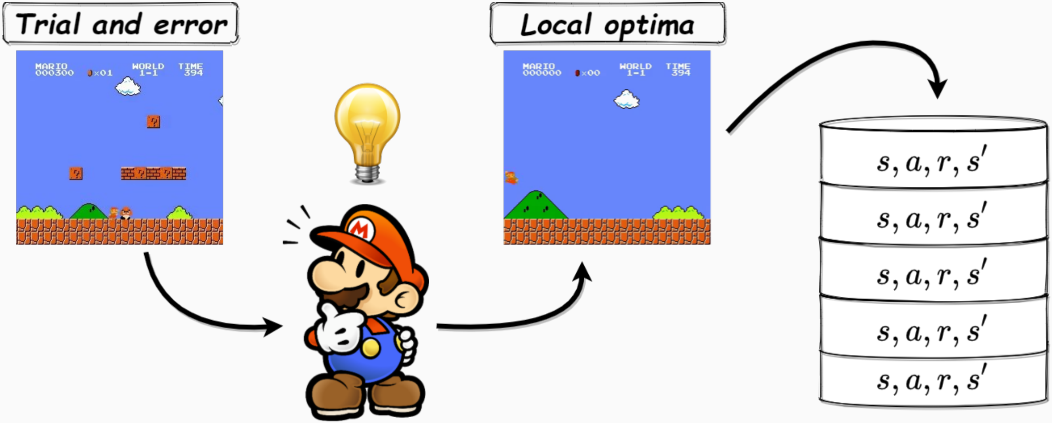
Trial and error



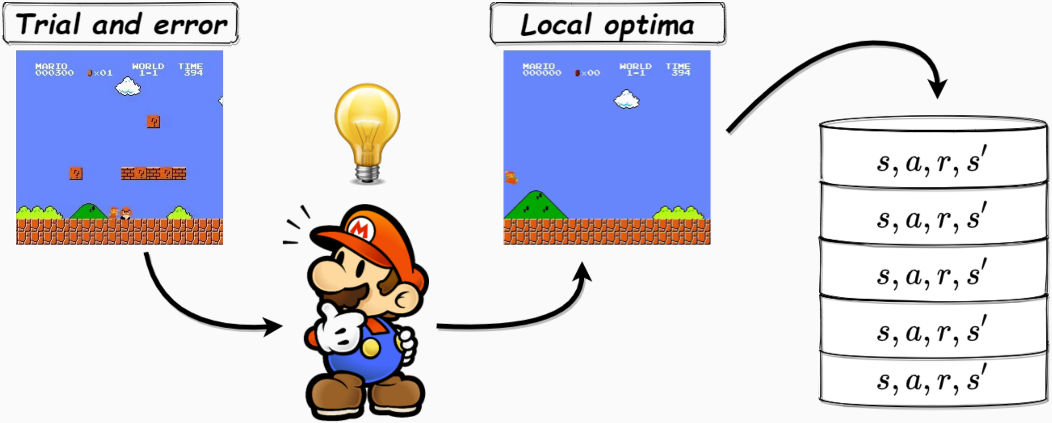
Do we still need exploration?



Do we still need exploration?



Do we still need exploration?



Important!

Diverse data is still required!

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 $k = 0, 1, 2 \dots$

- 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 $k = 0, 1, 2 \dots$

- 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 $k = 0, 1, 2 \dots$

- 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 $k = 0, 1, 2 \dots$

- 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

Problem: model often *overestimates* future reward.

Overestimation Bias and how to fight it

Problem: model often *overestimates* future reward.

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

Overestimation Bias and how to fight it

Problem: model often *overestimates* future reward.

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

Overestimation Bias and how to fight it

Problem: model often *overestimates* future reward.

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

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

Problem: model often *overestimates* future reward.

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

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 not really	Q_1^* Q_2^*	$y_1 = r + \gamma Q_?^*(s', \operatorname{argmax}_{a'} Q_?^*(s', a'))$

Action Evaluation: options

Name	Networks	Targets
Double Q-learning not really	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 not really	Q_1^* Q_2^*	$y_1 = r + \gamma Q_2^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$ $y_2 = r + \gamma Q_1^*(s', \operatorname{argmax}_{a'} Q_2^*(s', a'))$

Action Evaluation: options

Name	Networks	Targets
~~Double Q-learning~~ Twin DQN still not really	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	

Action Evaluation: options

Name	**Networks**	**Targets**
~~Double Q-learning~~ Twin DQN still not really	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^*	

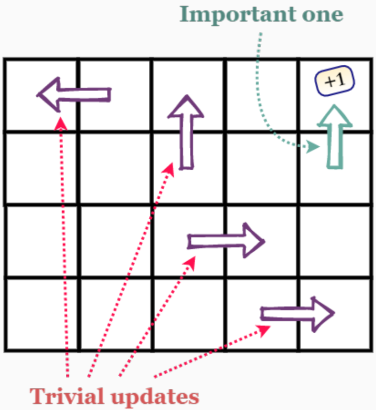
Action Evaluation: options

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^*	$y_1 = r + \gamma \min_{i=1,2} Q_i^*(s', \operatorname{argmax}_{a'} Q_1^*(s', a'))$ $y_2 = r + \gamma \min_{i=1,2} Q_i^*(s', \operatorname{argmax}_{a'} Q_2^*(s', a'))$

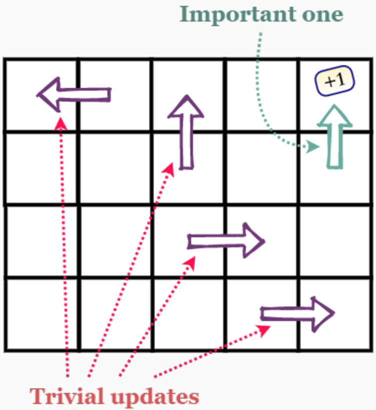
Prioritized Experience Replay



Problem: a lot of trivial updates.

Goal: propagate reinforcement from future to past

Prioritized Experience Replay



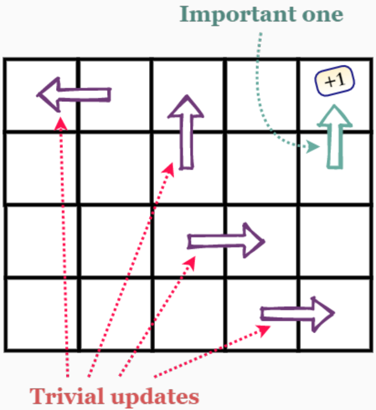
Problem: a lot of trivial updates.

Goal: propagate reinforcement from future to past

Prioritized sampling from replay buffer:

$$P(\mathbb{T}) \propto \left|y(\mathbb{T}) - Q^*(s, a, \theta)\right|^{\alpha}$$

Prioritized Experience Replay



Problem: a lot of trivial updates.

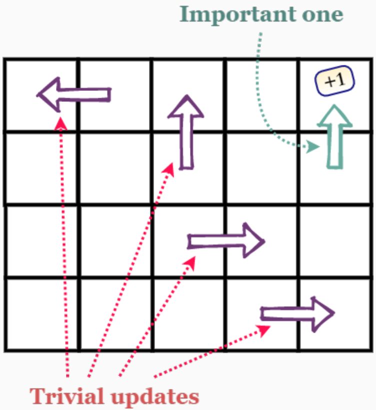
Goal: propagate reinforcement from future to past

Prioritized sampling from replay buffer:

$$P(\mathbb{T}) \propto \left|y(\mathbb{T}) - Q^*(s, a, \theta)\right|^{\alpha}$$

Issues?

Prioritized Experience Replay



Problem: a lot of trivial updates.

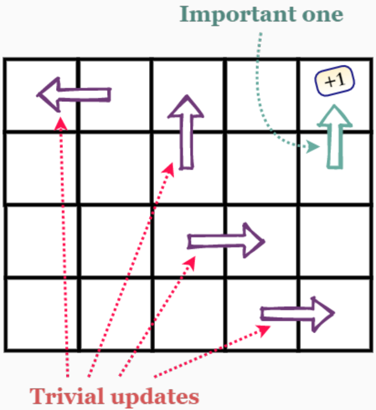
Goal: propagate reinforcement from future to past

Prioritized sampling from replay buffer:

$$P(\mathbb{T}) \propto |y(\mathbb{T}) - Q^*(s, a, \theta)|^\alpha$$

Issues? $s' \nsim p(s' \mid s, a)$ now.

Prioritized Experience Replay



Problem: a lot of trivial updates.

Goal: propagate reinforcement from future to past

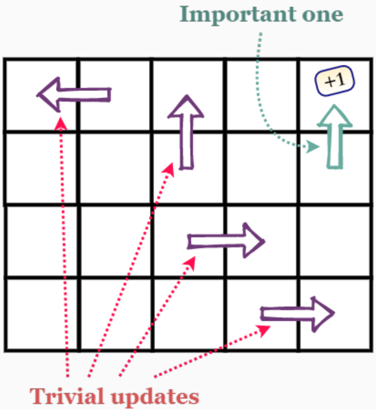
Prioritized sampling from replay buffer:

$$P(\mathbb{T}) \propto |y(\mathbb{T}) - Q^*(s, a, \theta)|^\alpha$$

Issues? $s' \not\sim p(s' \mid s, a)$ now.

$$\mathbb{E}_{\mathbb{T}\sim\text{Uniform}} \text{Loss}(\mathbb{T}) \not\propto$$

Prioritized Experience Replay



Problem: a lot of trivial updates.

Goal: propagate reinforcement from future to past

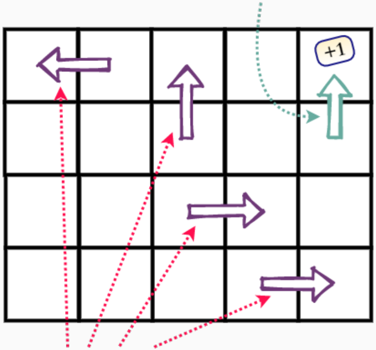
Prioritized sampling from replay buffer:

$$P(\mathbb{T}) \propto |y(\mathbb{T}) - Q^*(s, a, \theta)|^{\alpha}$$

Issues? $s' \not\sim p(s' \mid s, a)$ now.

$$\mathbb{E}_{\mathbb{T}\sim\text{Uniform}} \text{Loss}(\mathbb{T}) \propto \mathbb{E}_{\mathbb{T}\sim P(\mathbb{T})} \underbrace{\frac{1}{P(\mathbb{T})}}_{\text{weight}} \text{Loss}(\mathbb{T}),$$

Prioritized Experience Replay



Problem: a lot of trivial updates.

Goal: propagate reinforcement from future to past

Prioritized sampling from replay buffer:

$$P(\mathbb{T}) \propto |y(\mathbb{T}) - Q^*(s, a, \theta)|^\alpha$$

Issues? $s' \not\sim p(s' \mid s, a)$ now.

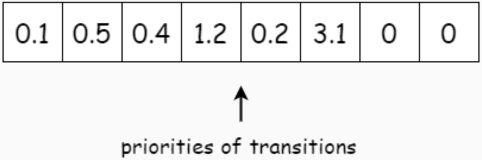
$$\mathbb{E}_{\mathbb{T}\sim \text{Uniform}} \text{Loss}(\mathbb{T}) \approx \mathbb{E}_{\mathbb{T}\sim P(\mathbb{T})} \left(\frac{1}{P(\mathbb{T})}\right)^{\beta(t)} \text{Loss}(\mathbb{T}),$$

where $\beta(t)$ anneals from 0 to 1.

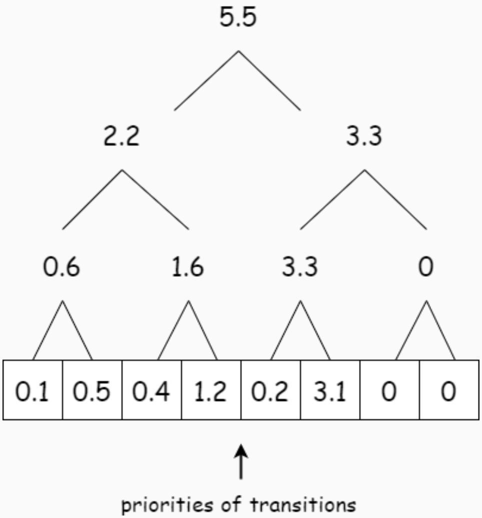
Technical note: SumTree structure

Use **SumTree** structure:

- leaf nodes are sampling priorities



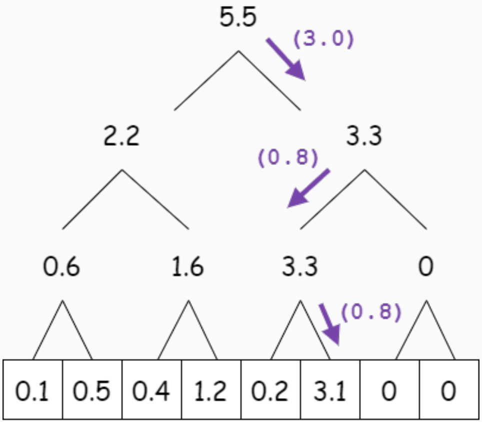
Technical note: SumTree structure



Use **SumTree** structure:

- leaf nodes are sampling priorities
- each node is sum of its two children

Technical note: SumTree structure



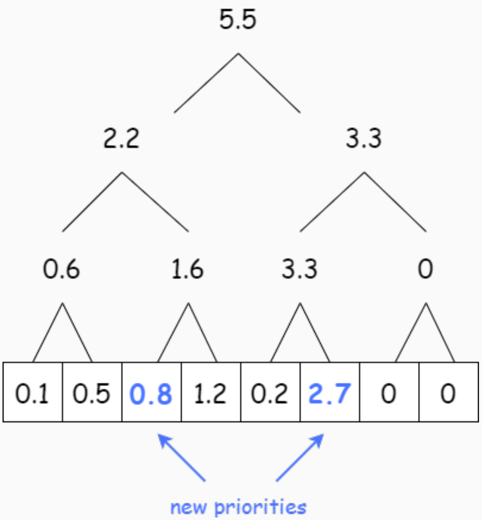
Use **SumTree** structure:

- leaf nodes are sampling priorities
- each node is sum of its two children

Prioritized buffer procedure:

- to sample from $P(\mathbb{T})$ use uniform real number from $[0, \sum P(\mathbb{T})]$;

Technical note: SumTree structure



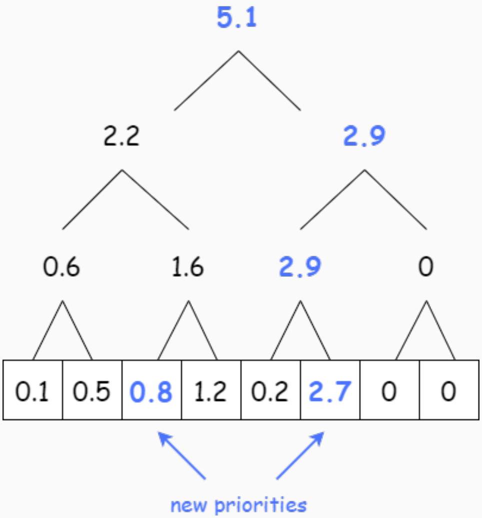
Use **SumTree** structure:

- leaf nodes are sampling priorities
- each node is sum of its two children

Prioritized buffer procedure:

- to sample from $\mathrm{P}(\mathbb{T})$ use uniform real number from $[0, \sum \mathrm{P}(\mathbb{T})]$;
- train on the batch;
- recompute priorities **for this batch only**;

Technical note: SumTree structure



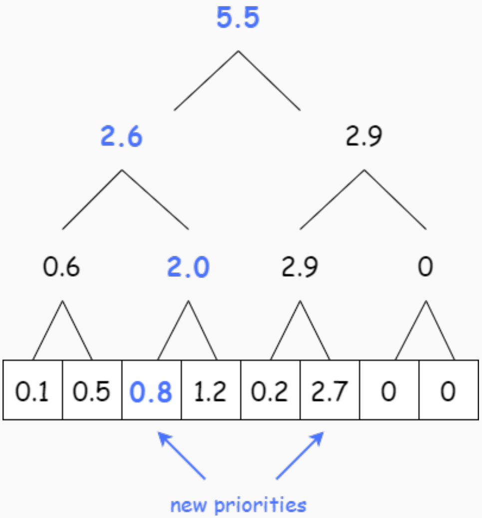
Use **SumTree** structure:

- leaf nodes are sampling priorities
- each node is sum of its two children

Prioritized buffer procedure:

- to sample from $\mathrm{P}(\mathbb{T})$ use uniform real number from $[0, \sum \mathrm{P}(\mathbb{T})]$;
- train on the batch;
- recompute priorities **for this batch only**;
- update nodes of SumTree;

Technical note: SumTree structure



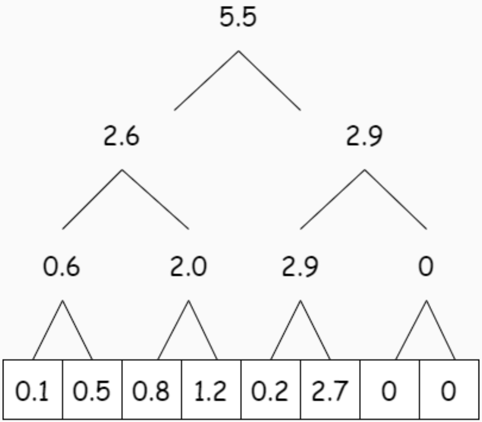
Use **SumTree** structure:

- leaf nodes are sampling priorities
- each node is sum of its two children

Prioritized buffer procedure:

- to sample from $\mathrm{P}(\mathbb{T})$ use uniform real number from $[0, \sum \mathrm{P}(\mathbb{T})]$;
- train on the batch;
- recompute priorities **for this batch only**;
- update nodes of SumTree;

Technical note: SumTree structure



Use **SumTree** structure:

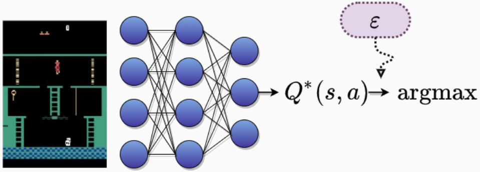
- leaf nodes are sampling priorities
- each node is sum of its two children

Prioritized buffer procedure:

- to sample from $\mathrm{P}(\mathbb{T})$ use uniform real number from $[0, \sum \mathrm{P}(\mathbb{T})]$;
- train on the batch;
- recompute priorities **for this batch only**;
- update nodes of SumTree;

Complexity: $O(\log N)$, where N is buffer size.

Noisy Networks

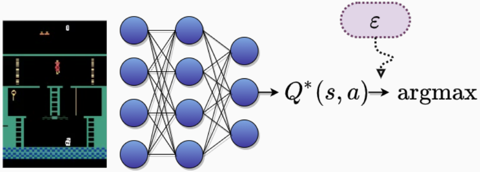


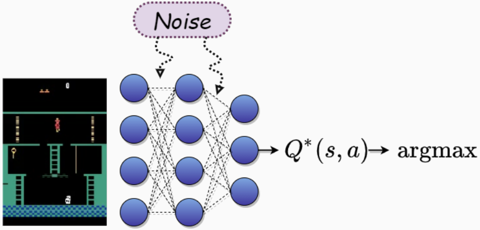
Problem: ε-greedy is naive

× inconvenient hyperparameter

× state-independent exploration

Noisy Networks





Problem: ε-greedy is naive

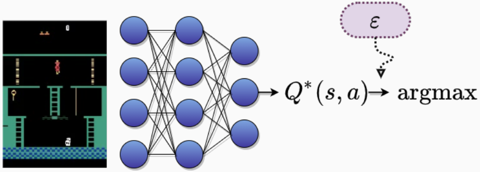
× inconvenient hyperparameter

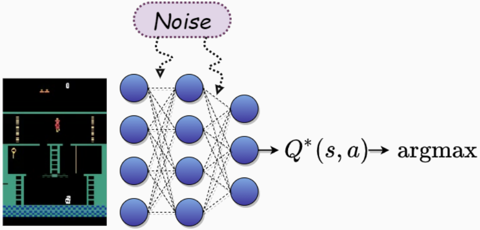
× state-independent exploration

Add noise to the network weights

$$w := \mu + \sigma\varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 1)$$

Noisy Networks





Problem: ε-greedy is naive

× inconvenient hyperparameter

× state-independent exploration

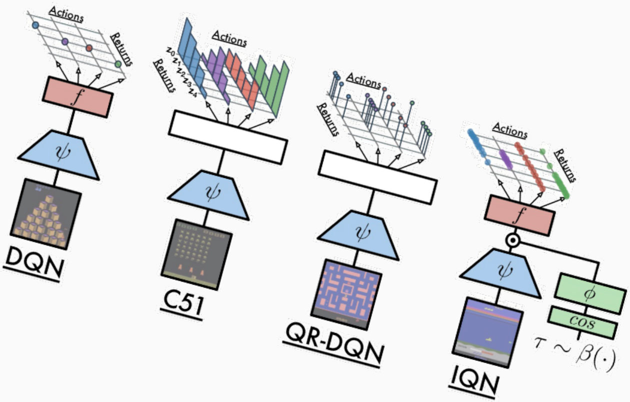
Add noise to the network weights

$$w := \mu + \sigma\varepsilon, \quad \varepsilon \sim \mathcal{N}(0,1)$$

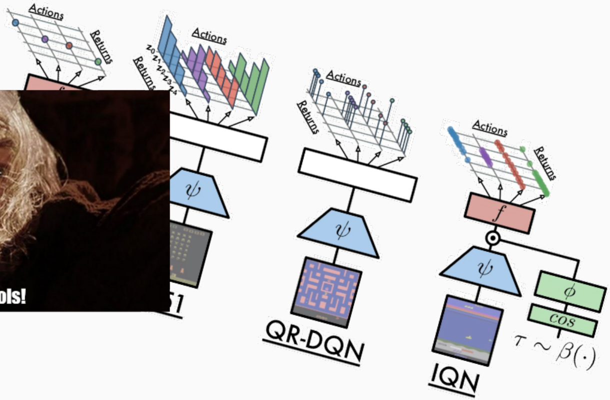
✓ no hyperparameters (initialization of σ?)

✓ state-dependent (not interpretable though)

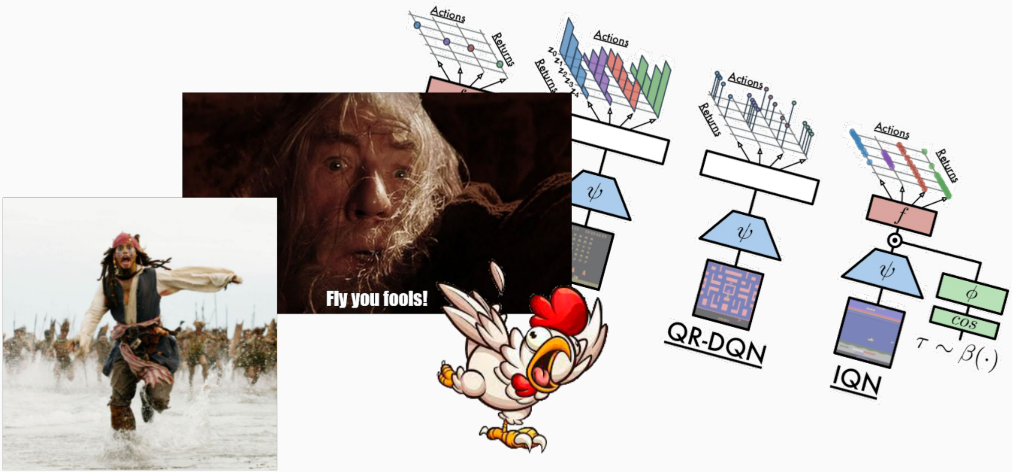
× expensive in wall-clock time


Actions
Returns
f
ψ
DQN
Actions
z_0 z_1 z_2 z_3 z_4
Returns
ψ
C51
Actions
Returns
ψ
QR-DQN
Actions
Returns
f
ψ
ϕ
cos
$\tau \sim \beta(\cdot)$
IQN

Distributional RL

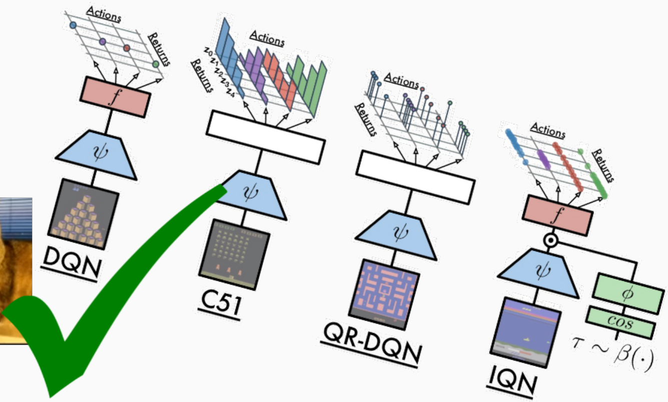


Distributional RL



Distributional RL

Distributional RL

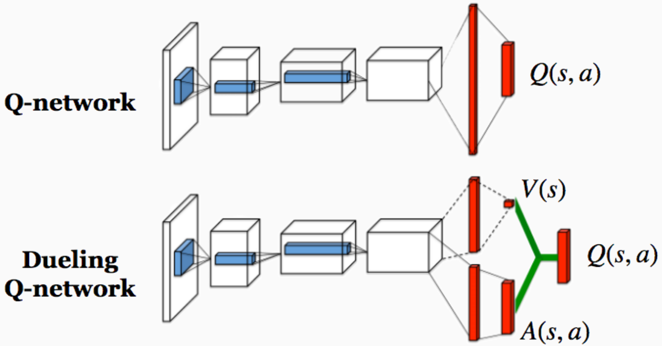


Dueling DQN

Problem: state value estimation requires knowledge about *all* actions

Problem: state value estimation requires knowledge about *all* actions

$$Q^*(s,a) := V^*(s) + \underbrace{A^*(s,a)}_{\text{not arbitrary}}$$



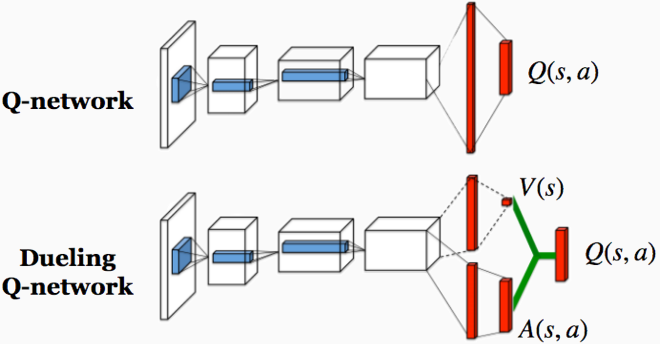
Dueling DQN

Problem: state value estimation requires knowledge about *all* actions

$$Q^*(s,a) := V^*(s) + \underbrace{A^*(s,a)}_{\text{not arbitrary}}$$

Danger!

This introduces one **extra dimension of freedom**!



Dueling DQN

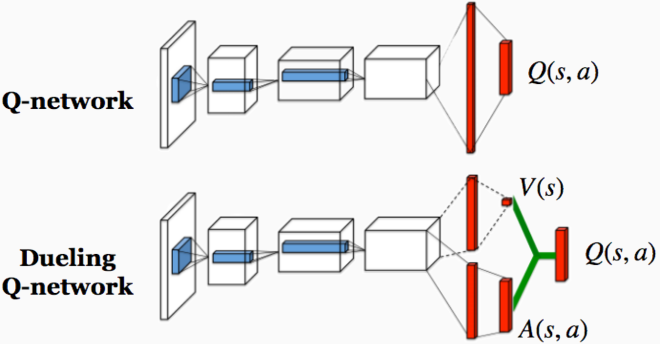
Problem: state value estimation requires knowledge about *all* actions

$$Q^*(s,a) := V^*(s) + \underbrace{A^*(s,a)}_{\text{not arbitrary}}$$

Danger!

This introduces one **extra dimension of freedom**!

$$Q^*_\theta(s,a) := V^*_\theta(s) + A^*_\theta(s,a) - \underbrace{-\operatorname*{mean}_a A^*_\theta(s,a)}_{\text{theory: take max}}$$



Partially Observable MDP

Problem: no access to full state description, only to **observations**:

$$o_t \sim p(o_t \mid s_t)$$

Partially Observable MDP

Problem: no access to full state description, only to **observations**:

$$o_t \sim p(o_t \mid s_t)$$

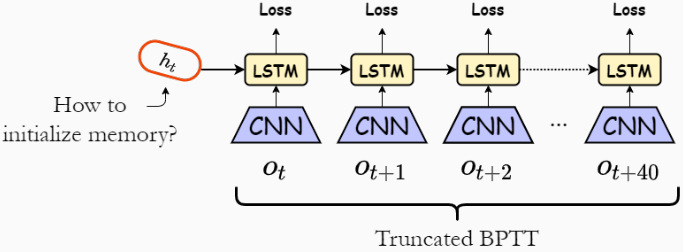
Theory: see **PoMDP** (Partially Observable MDP);
Practice: LSTM

Partially Observable MDP

Problem: no access to full state description, only to **observations**:

$$o_t \sim p(o_t \mid s_t)$$

Theory: see **PoMDP** (Partially Observable MDP);
Practice: LSTM

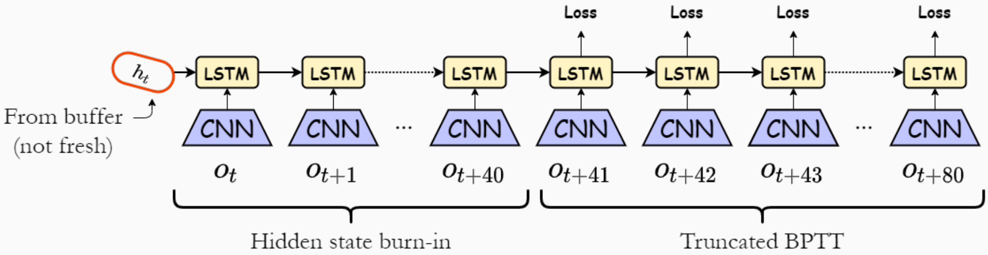


Partially Observable MDP

Problem: no access to full state description, only to **observations**:

$$o_t \sim p(o_t \mid s_t)$$

Theory: see **PoMDP** (Partially Observable MDP);
Practice: LSTM



Multi-step DQN

Problem: delayed rewards + compounding error

Multi-step DQN

Problem: delayed rewards + compounding error

$$y := \underbrace{r + \gamma r' + \gamma^2 r'' + \dots}_{\approx \text{«optimal behavior»?}} + \gamma^N \max_{a^{(N)}} Q^*(s^{(N)}, a^{(N)})$$

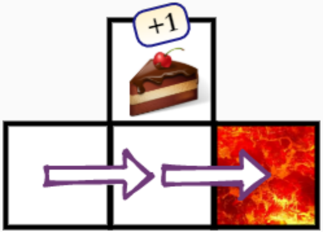
Multi-step DQN

Problem: delayed rewards + compounding error

$$y := \underbrace{r + \gamma r' + \gamma^2 r'' + \dots}_{\approx \text{«optimal behavior»?}} + \gamma^N \max_{a^{(N)}} Q^*(s^{(N)}, a^{(N)})$$

Important!

- ✓ can *significantly* help with this issue;
- × theoretically can't be done off-policy;



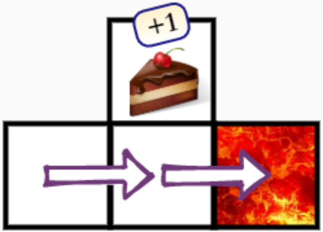
Multi-step DQN

Problem: delayed rewards + compounding error

$$y := \underbrace{r + \gamma r' + \gamma^2 r'' + \ldots}_{\approx \text{«optimal behavior»?}} + \gamma^N \max_{a^{(N)}} Q^*(s^{(N)}, a^{(N)})$$

Important!

- ✓ can *significantly* help with this issue;
- × theoretically can't be done off-policy;



Implementation: just store in replay buffer

$$\left(s, \quad a, \quad \sum_{n=0}^{N-1} \gamma^n r^{(n)}, \quad s^{(N)}, \quad \bigvee_{n=1}^{N} \mathrm{done}_n\right)$$

Rainbow DQN
(2017)

- Double DQN
- Prioritized buffer
- Multi-step DQN
- Noisy Nets
- Dueling DQN
- *Distributional RL (next time)*

Frontiers

Rainbow DQN
(2017)

- Double DQN
- Prioritized buffer
- Multi-step DQN
- Noisy Nets
- Dueling DQN
- *Distributional RL (next time)*

R2D2
(2018)

- Double DQN
- Prioritized buffer
- Multi-step DQN

\+ LSTM

\+ Massive parallelization

Frontiers

Rainbow DQN (2017)

- Double DQN
- Prioritized buffer
- Multi-step DQN
- Noisy Nets
- Dueling DQN
- *Distributional RL (next time)*

R2D2 (2018)

- Double DQN
- Prioritized buffer
- Multi-step DQN

\+ LSTM

\+ Massive parallelization

Agent 57 (2020)

- Double DQN
- Prioritized buffer
- Multi-step DQN
- LSTM
- Massive parallelization

\+ Retrace

\+ Intrinsic motivation

\+ Meta-controller for hyperparameters

Literature

- Playing Atari with Deep Reinforcement Learning (2013);
- *For DQN modifications use links on previous slide*;