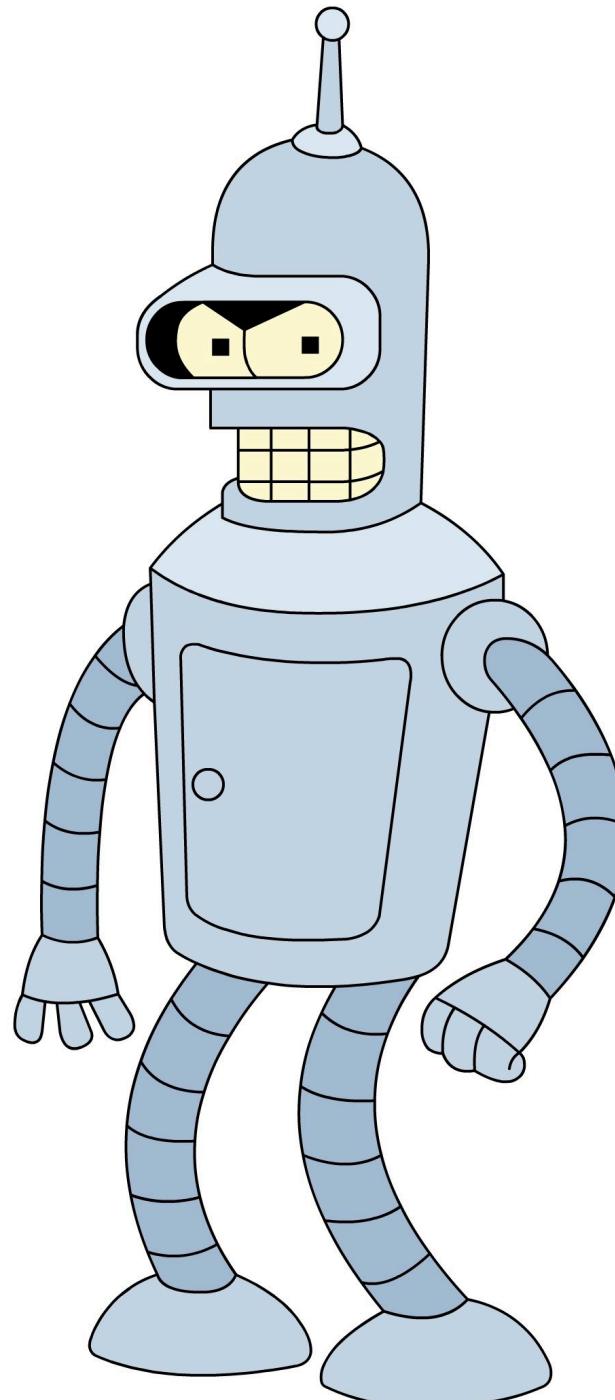


# Reinforcement Learning

HSE, winter - spring 2025

**Lecture 3: Intro to Deep RL, DQN  
and beyond**



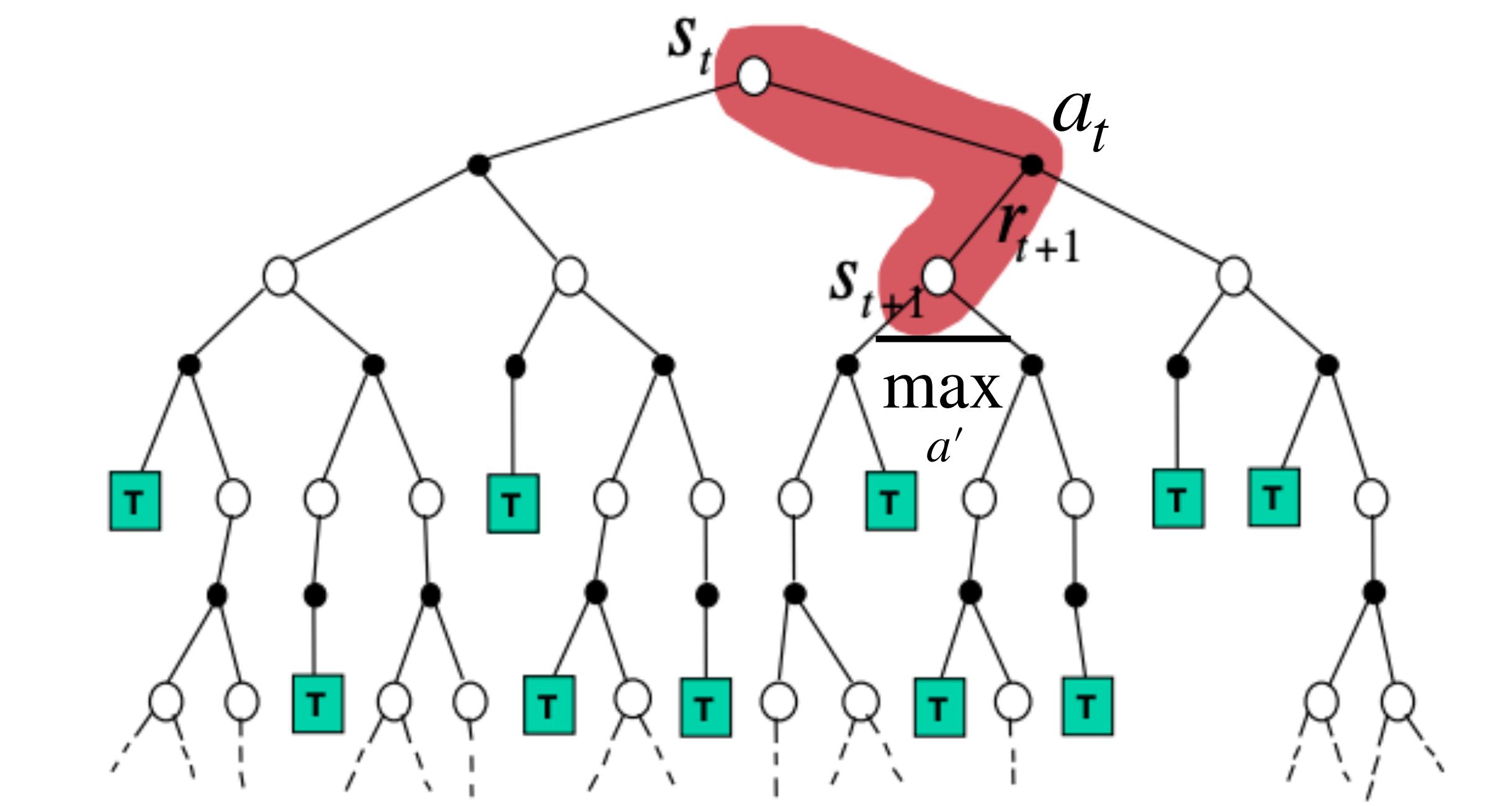
Sergei Laktionov  
[slaktionov@hse.ru](mailto:slaktionov@hse.ru)  
LinkedIn

# Recap: Q-Learning

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

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha(r + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a))$$

$$s' \sim p(\cdot | s, a)$$

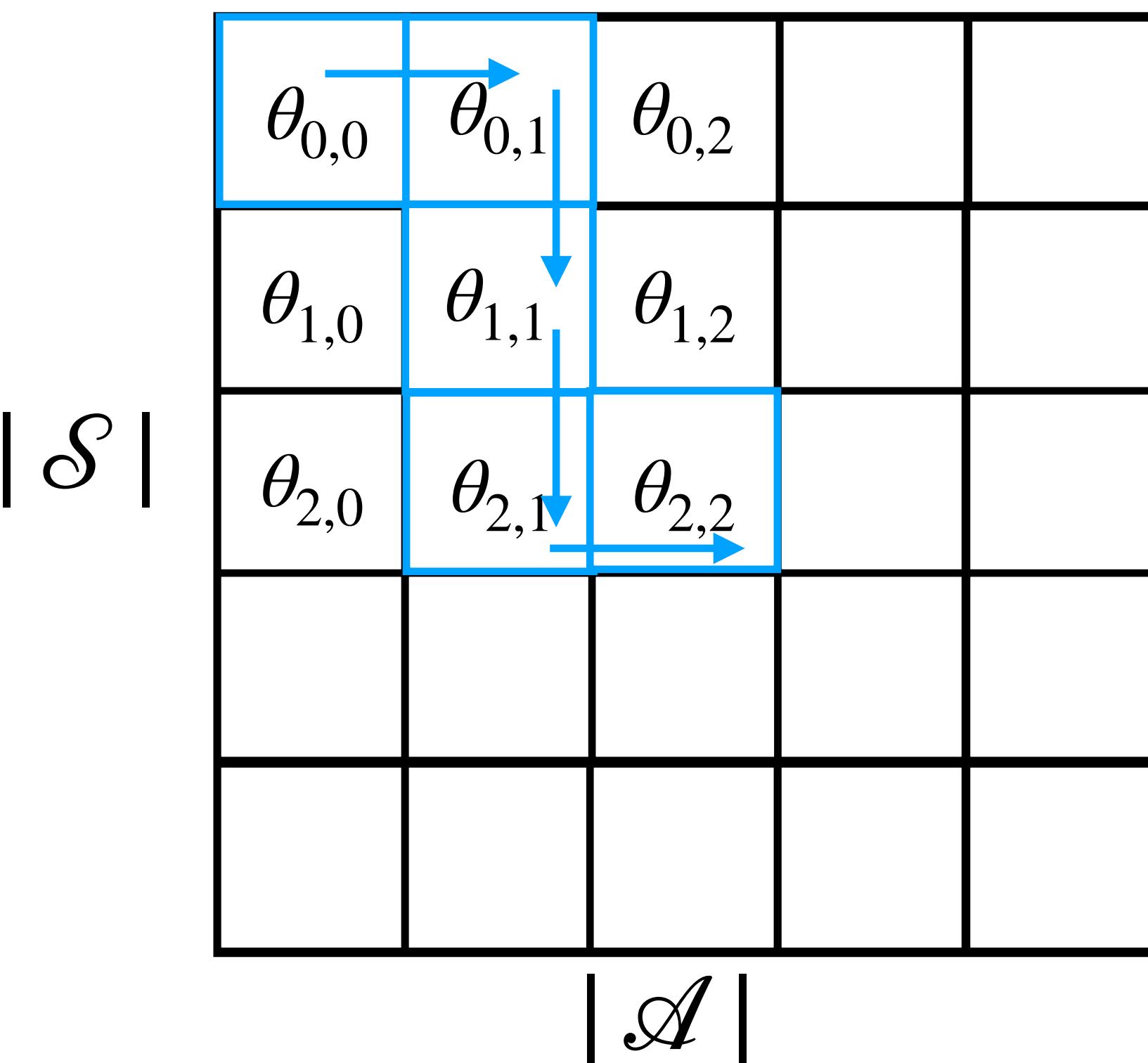


# Q-Learning

Recall the Bellman optimality equation for  $Q^*$ :

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

$$Q(s, a) \approx \theta_{s,a}$$



$$\theta_{s,a} \leftarrow \theta_{s,a} + \alpha(r + \gamma \max_{a'} \theta_{s',a'} - \theta_{s,a})$$

# Q-Learning

For target  $y_Q = r(s, a) + \gamma \max_{a'} Q(s', a')$  let's optimise MSE:

$$L(\theta) = \frac{1}{2} \mathbb{E}_{s'}[y_Q - \theta]^2 \rightarrow \min_{\theta}$$

$$\nabla_{\theta} L(\theta) = \mathbb{E}_{s'}[y_Q] - \theta = 0, \text{ so } Q(s, a) \approx \theta = \boxed{\mathbb{E}_{s'} y_Q}$$

In general infeasible

# Q-Learning

For target  $y_Q = r(s, a) + \gamma \max_{a'} Q(s', a')$  let's optimise MSE:

$$L(\theta) = \frac{1}{2} \mathbb{E}_{s'}[y_Q - \theta]^2 \rightarrow \min_{\theta}$$

$$\nabla_{\theta} L(\theta) = \mathbb{E}_{s'}[y_Q] - \theta = 0, \text{ so } Q(s, a) \approx \theta = \boxed{\mathbb{E}_{s'} y_Q}$$

In general infeasible

For each  $s'$ ,  $y_Q$  is an unbiased estimator of  $\mathbb{E}_{s'} y_Q$ , so

$\theta - y_Q$  is an unbiased estimator of  $\nabla_{\theta} L(\theta)$

# Q-Learning

For target  $y_Q = r(s, a) + \gamma \max_{a'} Q(s', a')$  let's optimise MSE:

$$L(\theta) = \frac{1}{2} \mathbb{E}_{s'}[y_Q - \theta]^2 \rightarrow \min_{\theta}$$

$$\nabla_{\theta} L(\theta) = \mathbb{E}_{s'}[y_Q] - \theta = 0, \text{ so } Q(s, a) \approx \theta = \boxed{\mathbb{E}_{s'} y_Q}$$

In general infeasible

For each  $s'$ ,  $y_Q$  is an unbiased estimator of  $\mathbb{E}_{s'} y_Q$ , so

$\theta - y_Q$  is an unbiased estimator of  $\nabla_{\theta} L(\theta)$

SGD:  $\theta_{k+1} = \theta_k + \alpha(y_Q - \theta_k)$

# Q-Learning Generalisation

$$Q(s, a) \approx Q_\theta(s, a)$$

For target  $y_Q = r(s, a) + \gamma \max_{a'} Q_\theta(s', a')$  let's optimise MSE:

$$L(\theta) = \frac{1}{2} \mathbb{E}_{s'} [y_Q - Q_\theta(s, a)]^2 \rightarrow \min_{\theta}$$

$$\nabla_{\theta} L(\theta) \approx (Q_\theta(s, a) - y_Q) \nabla_{\theta} Q_\theta(s, a)$$

# Q-Learning Generalisation

In general, for  $y_Q = r(s, a) + \gamma \max_{a'} Q_{\theta_k}(s', a')$

$$L(\theta) = \frac{1}{2} \mathbb{E}_{s, a \sim \rho(\cdot), s' \sim p(\cdot | s, a)} [y_Q - Q_\theta(s, a)]^2 \rightarrow \min_{\theta}$$

where  $\rho$  is the behaviour distribution.

$$\nabla_{\theta} L(\theta) \approx \mathbb{E}_{s, a \sim \rho(\cdot)} (Q_{\theta}(s, a) - y_Q) \nabla_{\theta} Q_{\theta}(s, a)$$

# Q-Learning Generalisation

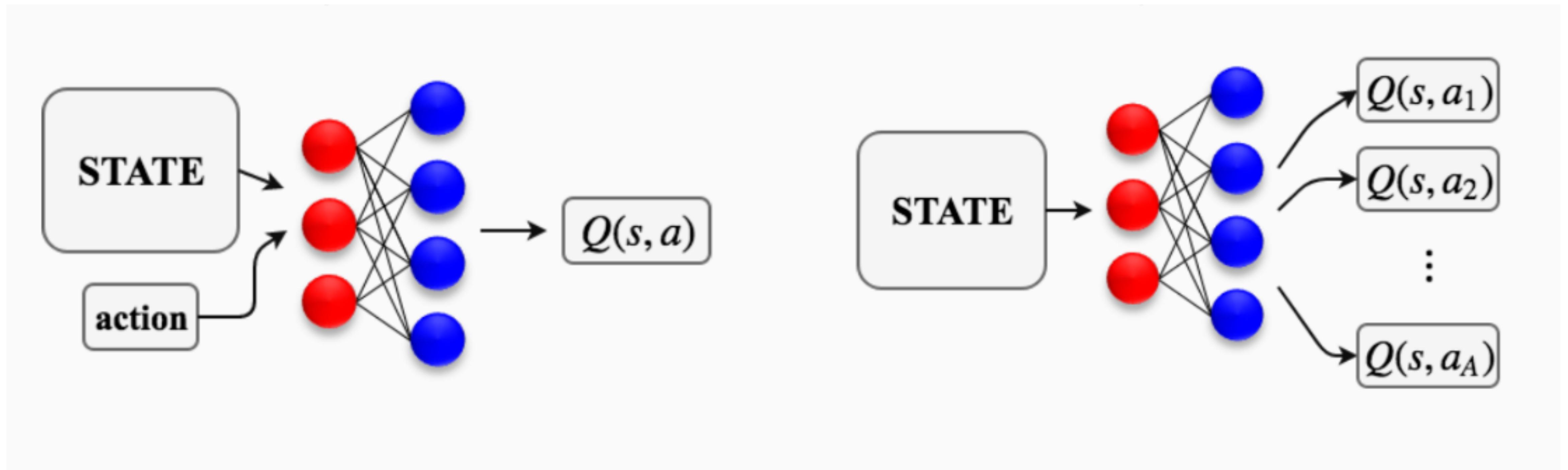
In general, for  $y_Q = r(s, a) + \gamma \max_{a'} Q_\theta(s', a')$

$$L(\theta) = \frac{1}{2} \mathbb{E}_{s, a \sim \rho(\cdot), s' \sim p(\cdot | s, a)} [y_Q - Q_\theta(s, a)]^2 \rightarrow \min_{\theta}$$

where  $\rho$  is the behaviour distribution.

$$\nabla_{\theta} L(\theta) \approx \mathbb{E}_{s, a \sim \rho(\cdot)} (Q_\theta(s, a) - y_Q) \nabla_{\theta} Q_\theta(s, a)$$

# Choose your fighter



Source

# Behaviour Policy

Behaviour policy is  $\varepsilon$ -greedy policy:

$$\mu = \begin{cases} \text{select random action with probability } \varepsilon \\ \text{select greedy action with probability } 1 - \varepsilon \end{cases}$$

We use  $\varepsilon$ -greedy strategies to avoid being stuck in local optima.

$$\nabla_{\theta} L(\theta) \approx (Q_{\theta}(s, a) - y_Q) \nabla_{\theta} Q_{\theta}(s, a), a \sim \mu(\cdot | s)$$

# Behaviour Policy

Behaviour policy is  $\varepsilon$ -greedy policy:

$$\mu = \begin{cases} \text{select random action with probability } \varepsilon \\ \text{select greedy action with probability } 1 - \varepsilon \end{cases}$$

We use  $\varepsilon$ -greedy strategies to avoid being stuck in local optima.

$$\nabla_{\theta} L(\theta) \approx (Q_{\theta}(s, a) - y_Q) \nabla_{\theta} Q_{\theta}(s, a), a \sim \mu(\cdot | s)$$

1. How to generate target  $y_Q$ ?
2. How to get an unbiased gradient estimator?

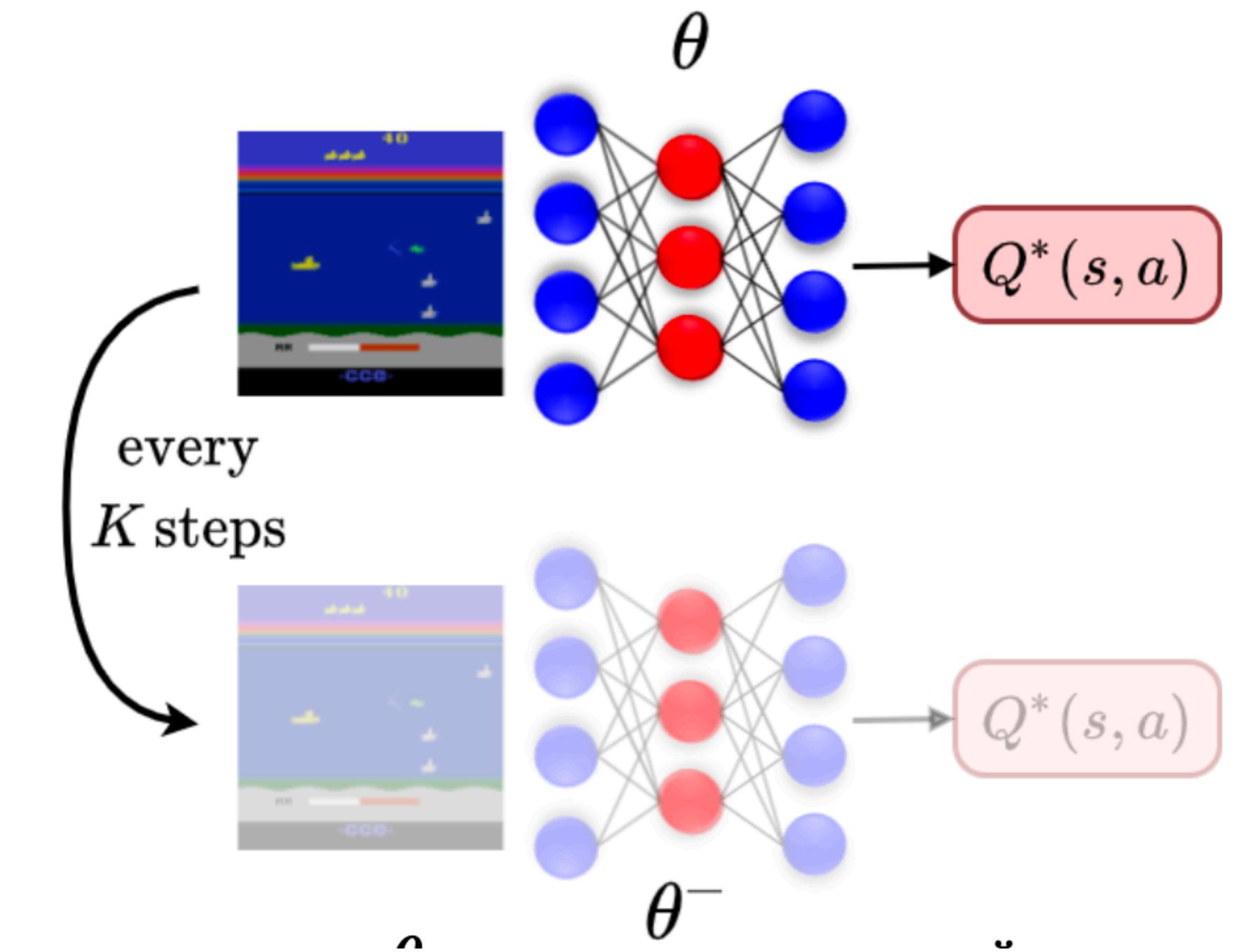
# Target Network

If we use weights from the previous step to calculate  $y_Q$  they will be highly non-stationary.

Moreover, we should get the network some time to do enough policy evaluation iterations.

A.  $\theta^- \leftarrow \theta$  every  $K$  SGD iterations

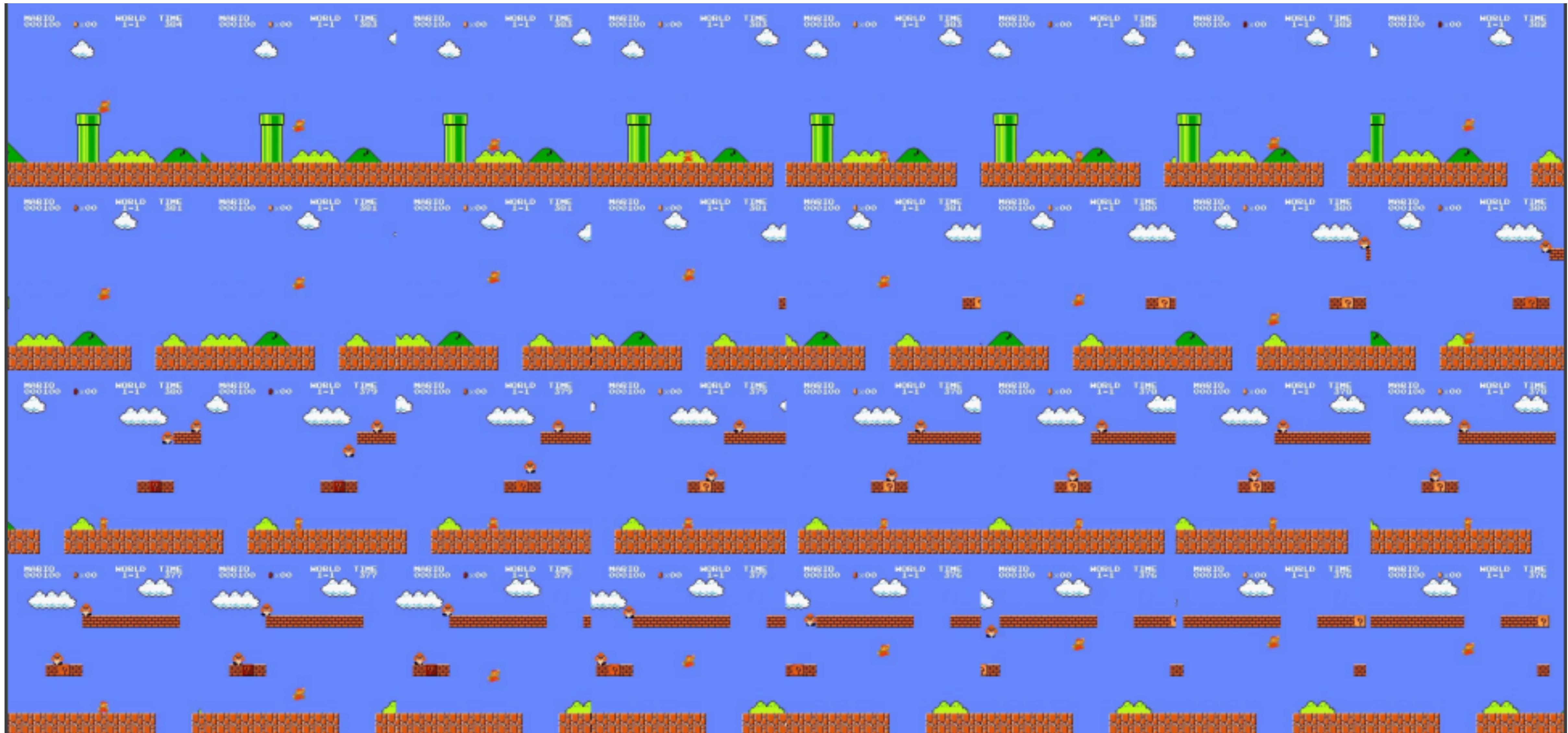
B.  $\theta^- \leftarrow (1 - \beta)\theta^- + \beta\theta$  on each iteration



$$y_Q(s, a) = r(s, a) + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

$$\nabla_{\theta} L(\theta) \approx \frac{1}{B} \sum_{j=1}^B [(Q_{\theta}(s_j, a_j) - y_j) \nabla_{\theta} Q_{\theta}(s_j, a_j)], a_j \sim \mu(\cdot | s_j)$$

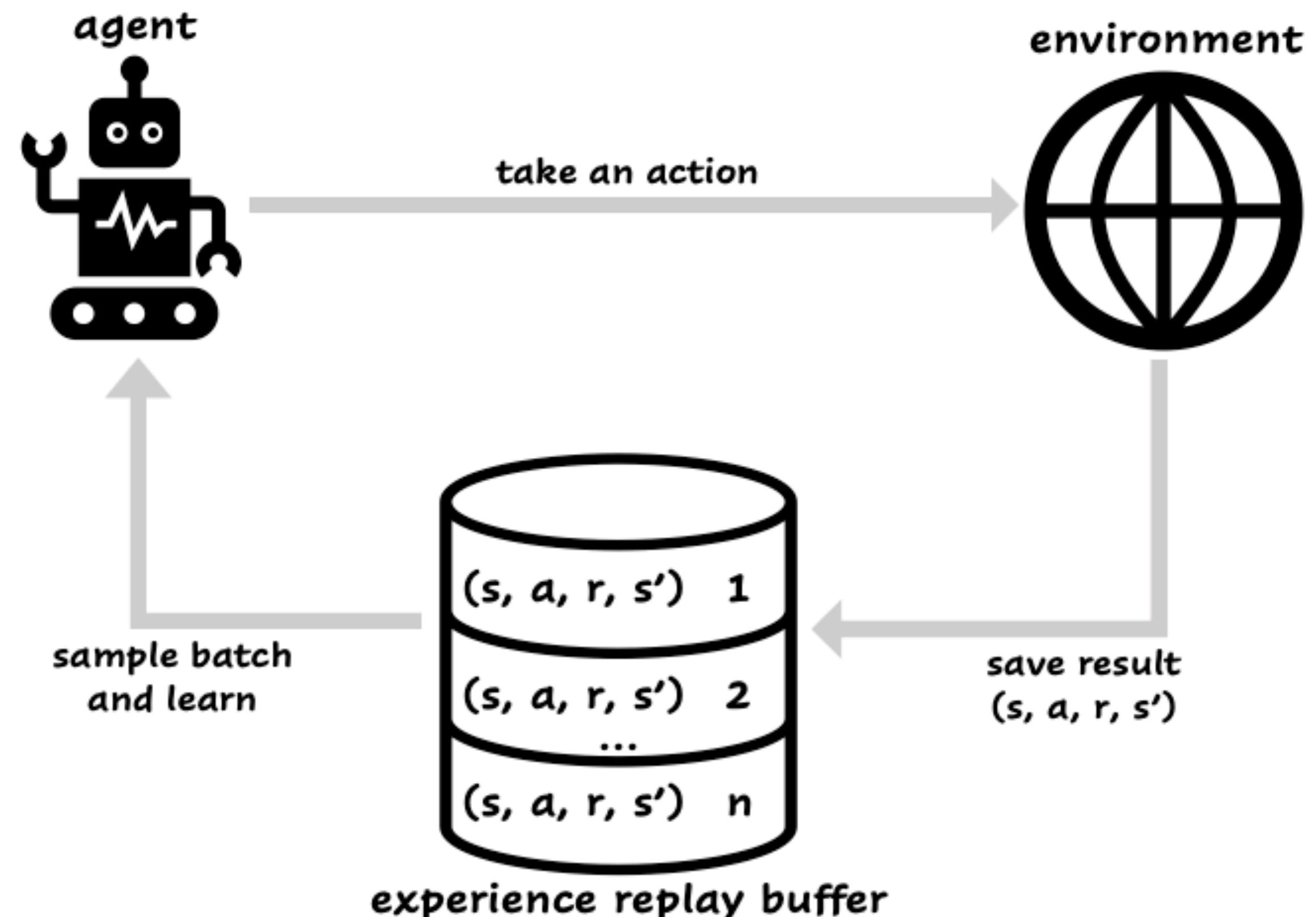
Consecutive samples are extremely correlated so gradient variance is high



# Recap: Experience Replay Buffer

## Advantages:

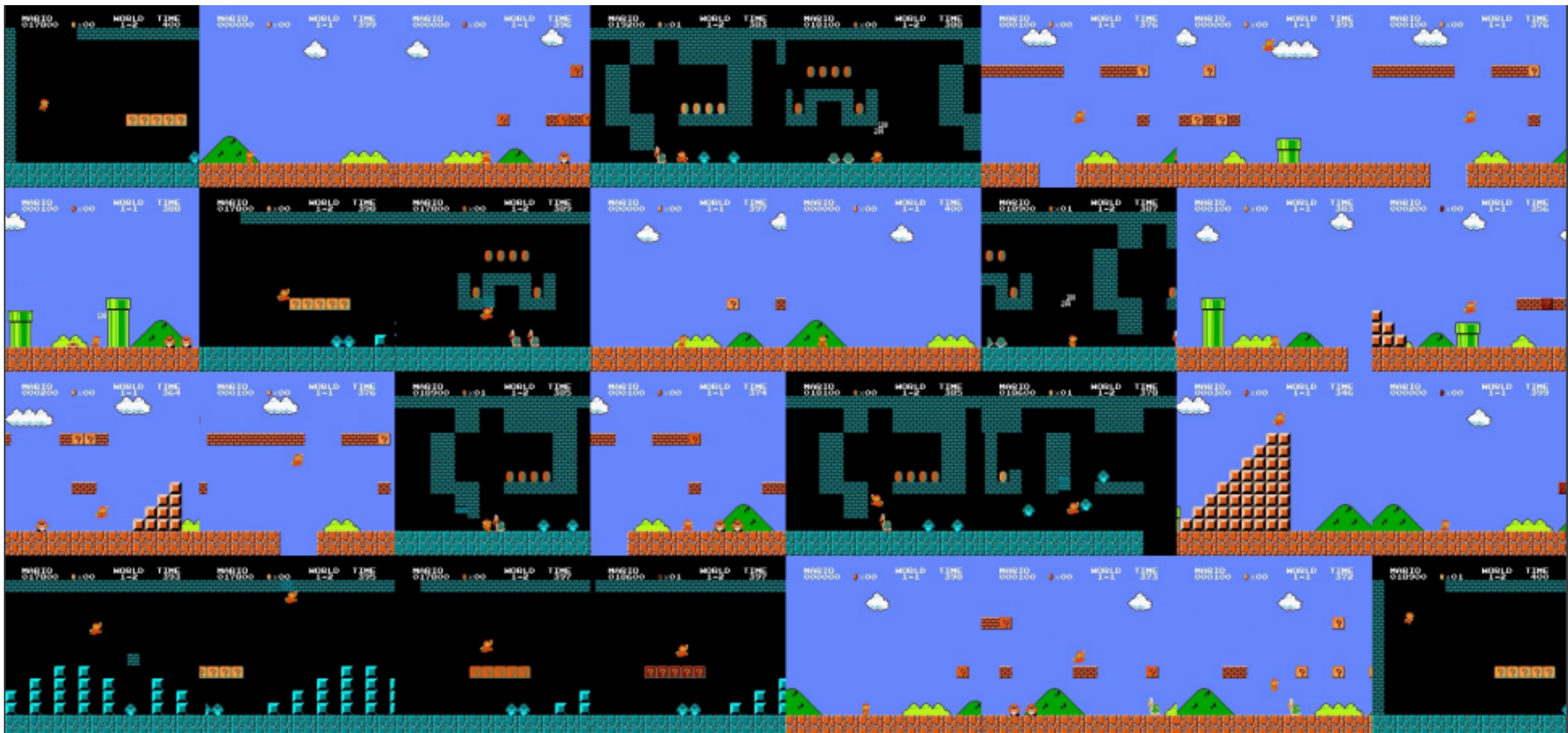
- No need to revisit same states many times
- Make the estimators consistent with the current policy, update estimators
- Decorrelate update samples to maintain i.i.d. assumption



## Disadvantages:

- Not applicable for the on-policy learning

Let's sample randomly from the Experience Replay Buffer which stores played transitions



# DQN Algorithm

**Initialise**  $\theta$ ;  $\theta^- = \theta$ ; replay buffer  $D$  is empty;

For  $k = 0, 1, \dots$

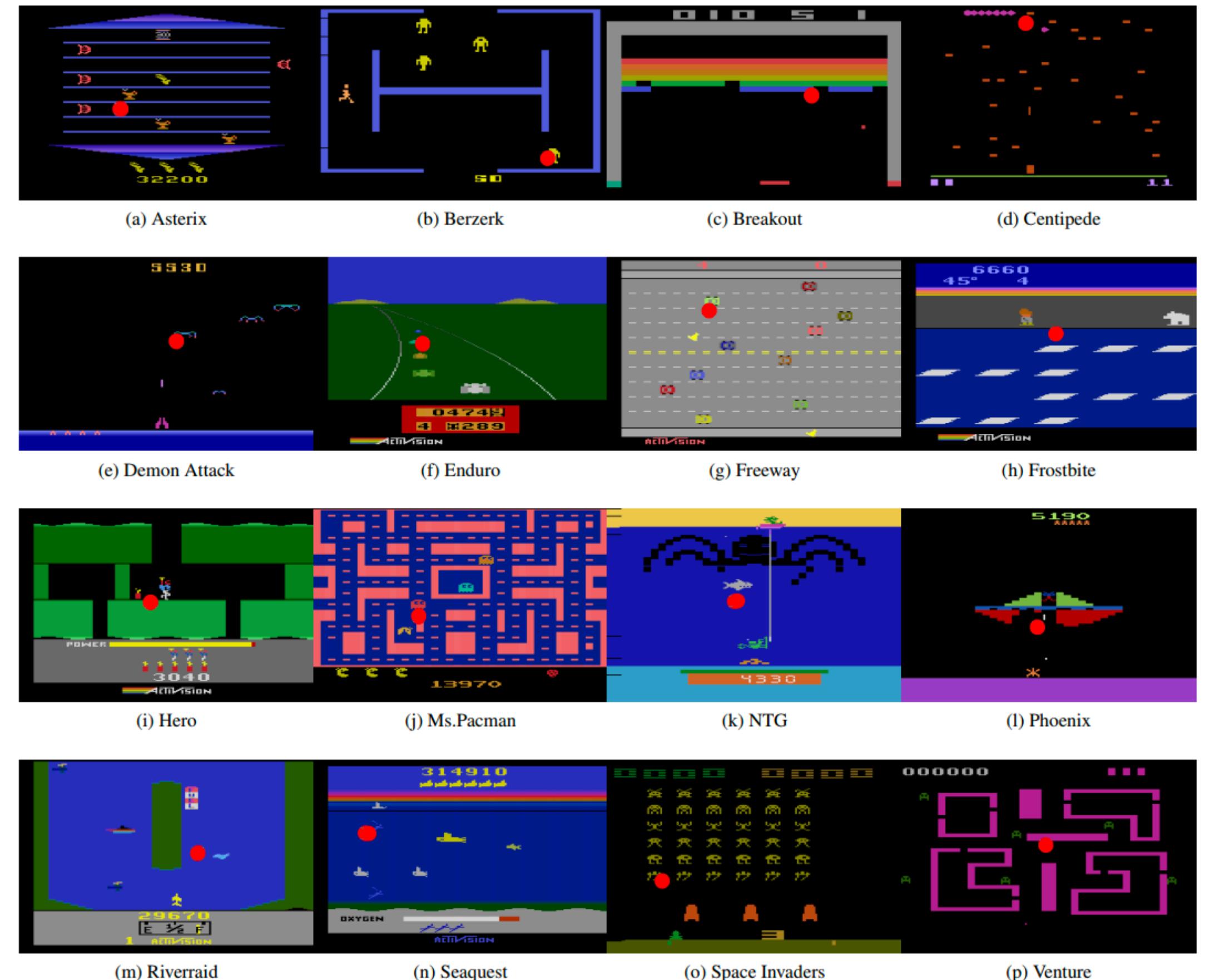
- Observe  $s$ , take action  $a$  sampled from the  $\varepsilon$ -greedy policy w.r.t.  $Q_{\theta_k}(s, a)$ ;
- Observe new  $(r, s', \text{done})$ , store  $(s, a, r, s', \text{done})$  in  $D$ ;
- If  $B > |D|$ , sample batch of transitions  $(s_j, a_j, r_j, s'_j, \text{done}_{j+1})_{j=1}^B \sim D$
- $y_j = r_j + \gamma(1 - \text{done}_{j+1}) \max_{a'} Q(s'_j, a'; \theta^-)$ ;
- Perform a gradient descent step:  $\theta_{k+1} = \theta_k - \frac{\alpha}{B} \sum_{j=1}^B \nabla_{\theta} (y_j - Q_{\theta}(a_j, s_j))^2$
- If  $k + 1 \bmod K = 0$  update target network:  $\theta^- \leftarrow \theta_{k+1}$

# Atari

?

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

1. More than 57 different games
2. Only frames are available
3. State space is actually finite ( $2^{1024}$  different RAM states) but too large to apply tabular methods
4. From 4 to 18 actions are available



[Source](#)

The ultimate goal is to build a universal algorithm which can be applied to all of these environments without modifications and specific hyperparameters.

# Is Atari Environment MDP?



[Source](#)

# MDP from Frames



Source

# Pre-Processing

## States:

- Crop Image
- Grayscale
- Observation Stack
- Frame Skip

## Actions' frequency:

- MaxAndSkip
- Sticky actions, if you want to make env stochastic

## Environment:

- EpisodicLife (can be omitted according to Machado et al. (2018))
- Noop Reset
- FireReset

## Reward:

- ClipReward ( $\{-1, 0, 1\}$ )

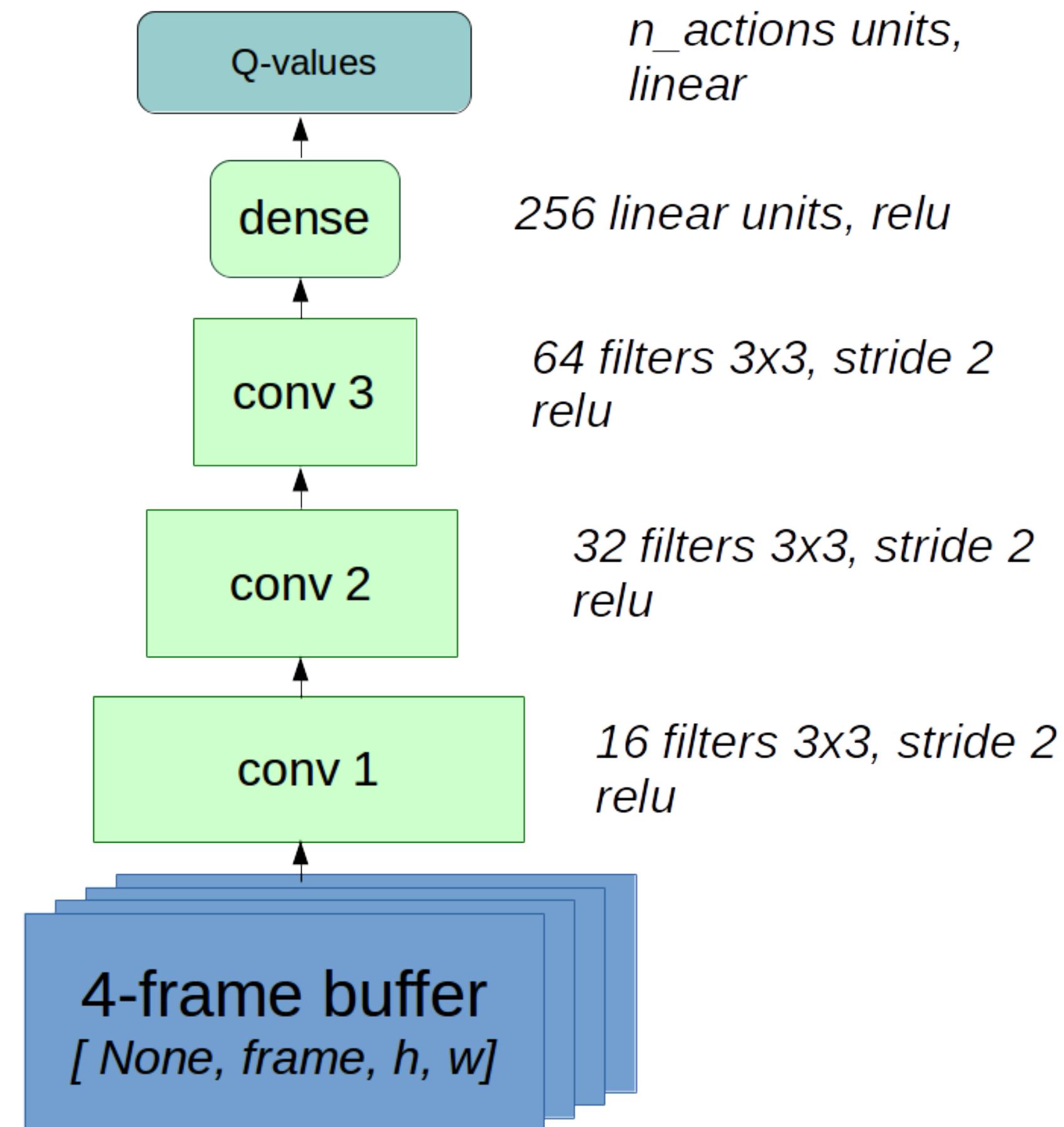
# Frame Skipping and Pre-Processing



[Source](#)

# Architecture

- No MaxPool
- No Dropout
- No BatchNorm



[Source](#)

# Overestimation Bias

1.  $U(a') : \mathbb{E}[U(a')] = 0,$   
 $\mathbb{E}[\max_{a'}[Q(s', a') + U(a')]] \geq \max_{a'} \mathbb{E}[Q(s', a') + U(a')] = \max_{a'} Q(s', a')$
2. Due to using the same samples (plays) both to determine the maximising action and to estimate its value.

$$\max_{a'} Q(s', a') = Q(s', \boxed{\text{argmax}_{a'} Q(s', a')})$$

Action selection

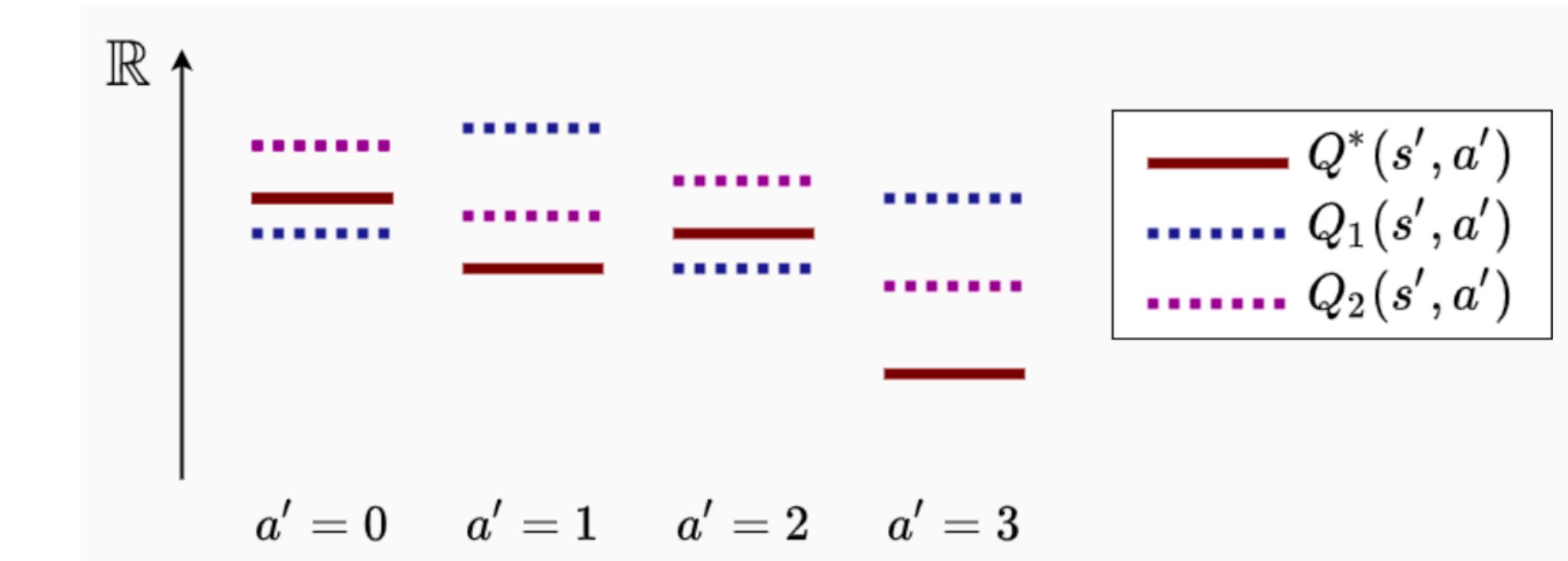
Action evaluation

# Double DQN

We can use two weakly correlated networks with independent buffers  $D_1$  and  $D_2$ :

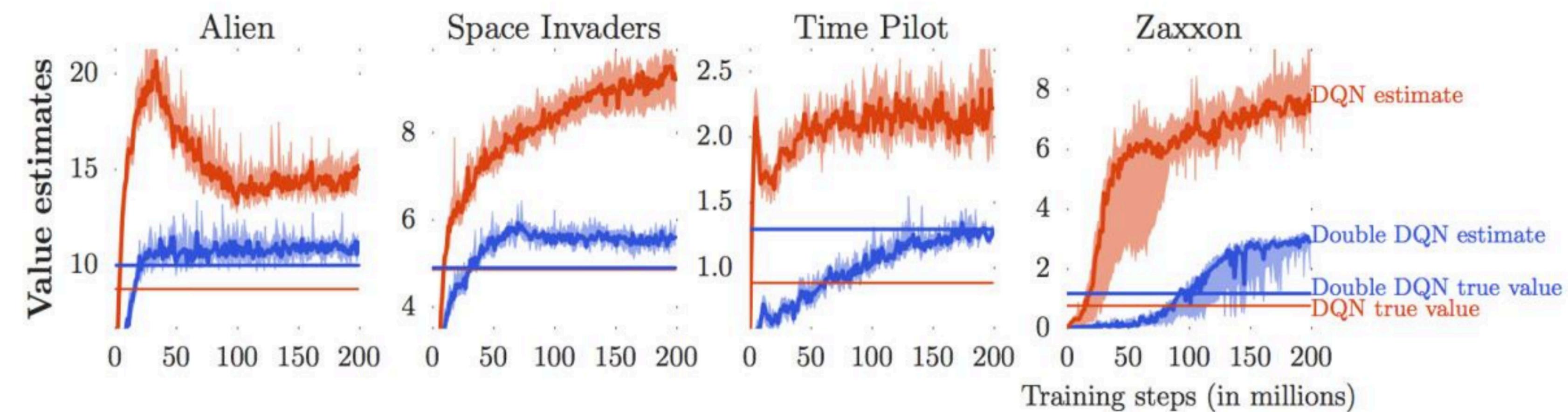
$$y_1 = r + \gamma Q_{\theta_2}(s', \text{argmax}_a Q_{\theta_1}(s', a'))$$

$$y_2 = r + \gamma Q_{\theta_1}(s', \text{argmax}_a Q_{\theta_2}(s', a'))$$



but it can be too expensive...

[Source](#)



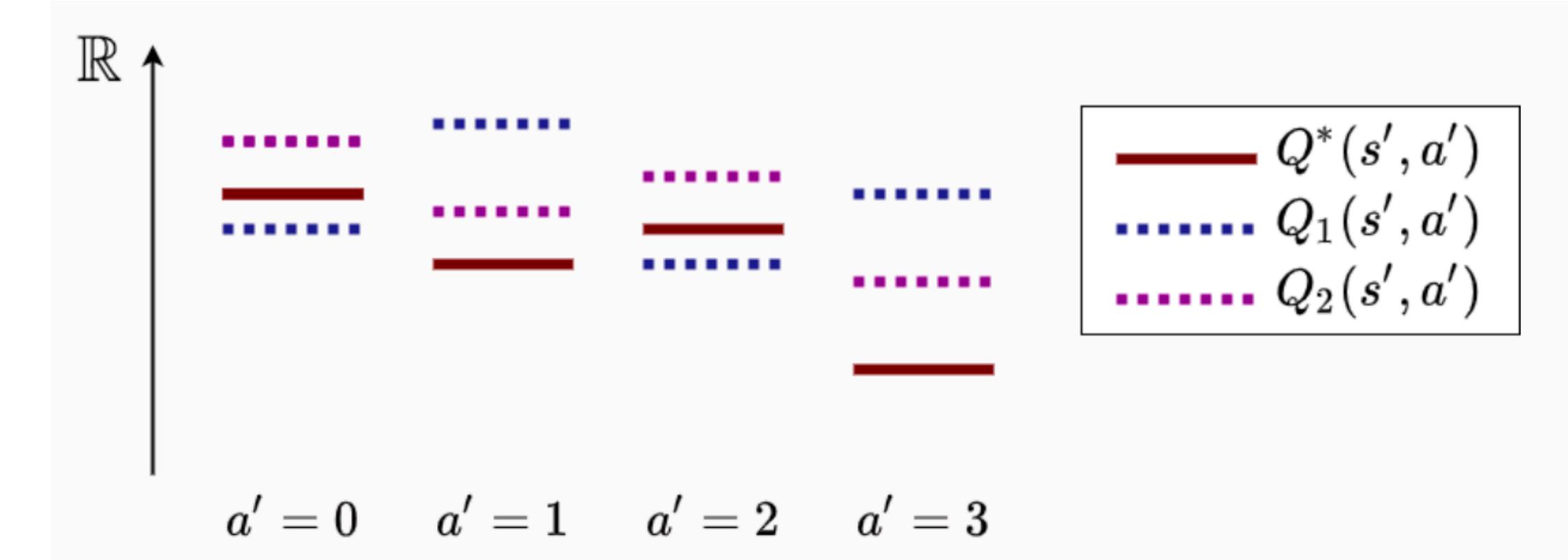
[Source](#)

# Double DQN

We can use two weakly correlated networks with independent buffers  $D_1$  and  $D_2$ :

$$y_1 = r + \gamma Q_{\theta_2}(s', \text{argmax}_a Q_{\theta_1}(s', a'))$$

$$y_2 = r + \gamma Q_{\theta_1}(s', \text{argmax}_a Q_{\theta_2}(s', a'))$$



but it can be too expensive...

[Source](#)

We can use the target network as the second network:

$$y = r + \gamma Q_{\theta^-}(s', \text{argmax}_{a'} Q_{\theta}(s', a'))$$

# Twin DQN

$$y_1 = r + \gamma \min_i Q_{\theta_i}(s', \text{argmax}_{a'} Q_{\theta_1}(s', a'))$$

$$y_2 = r + \gamma \min_i Q_{\theta_i}(s', \text{argmax}_{a'} Q_{\theta_2}(s', a'))$$

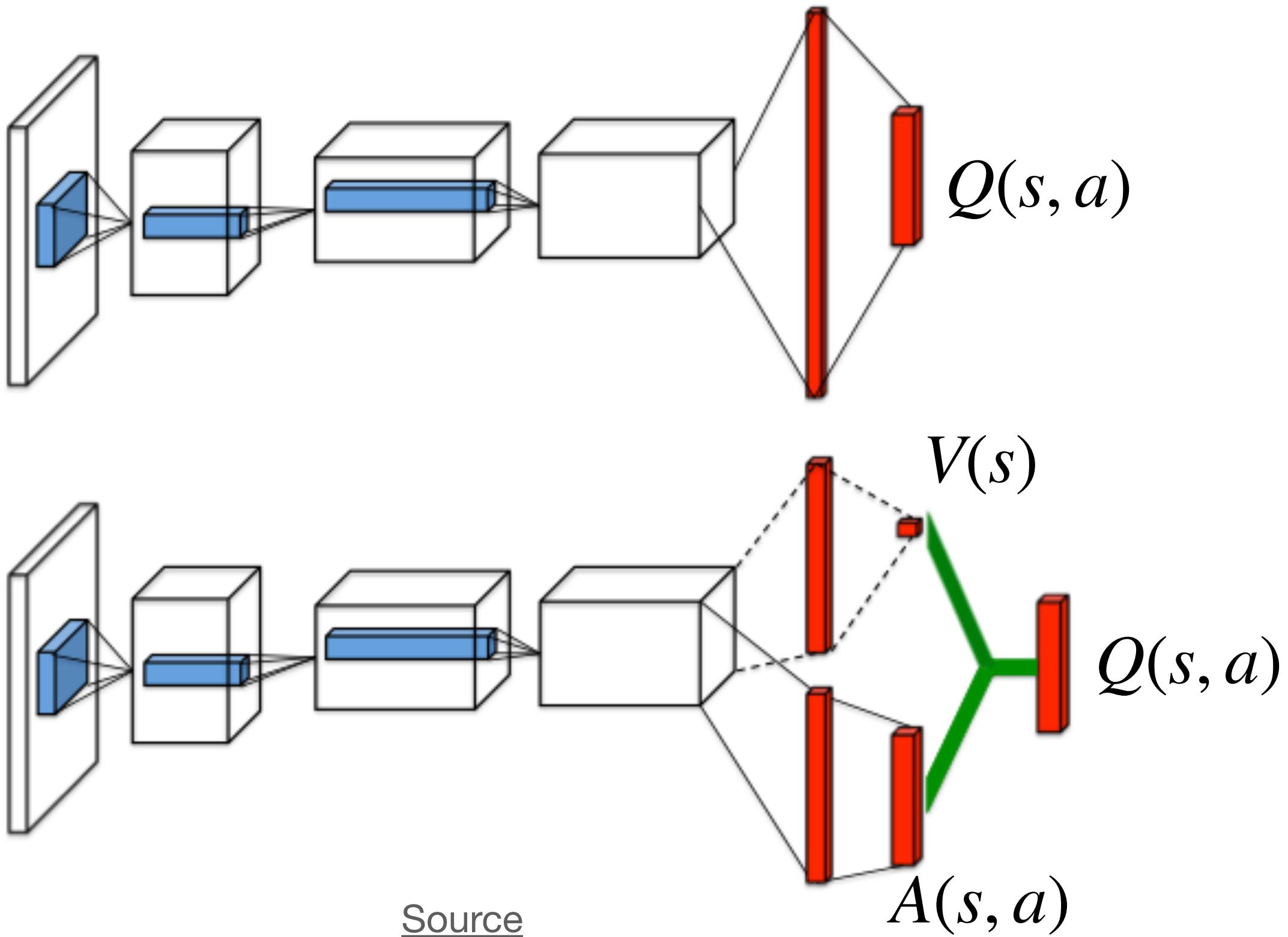
# Dueling DQN

The advantage  $A(s, a) = Q(s, a) - V(s)$  is a relative measure of the importance of each action.

Note that:

$$\mathbb{E}_{a \sim \pi(\cdot | s)} A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) = 0$$

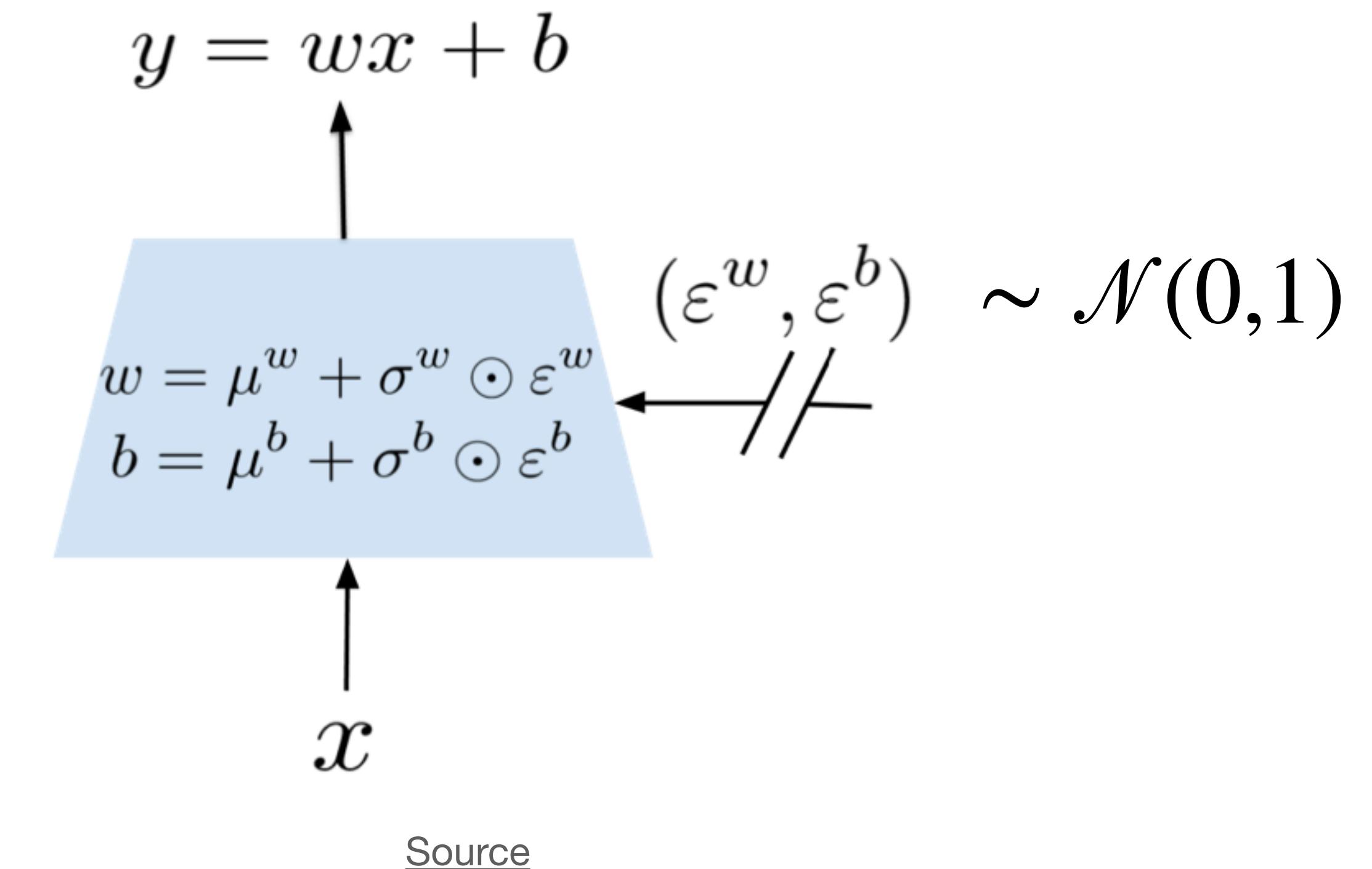
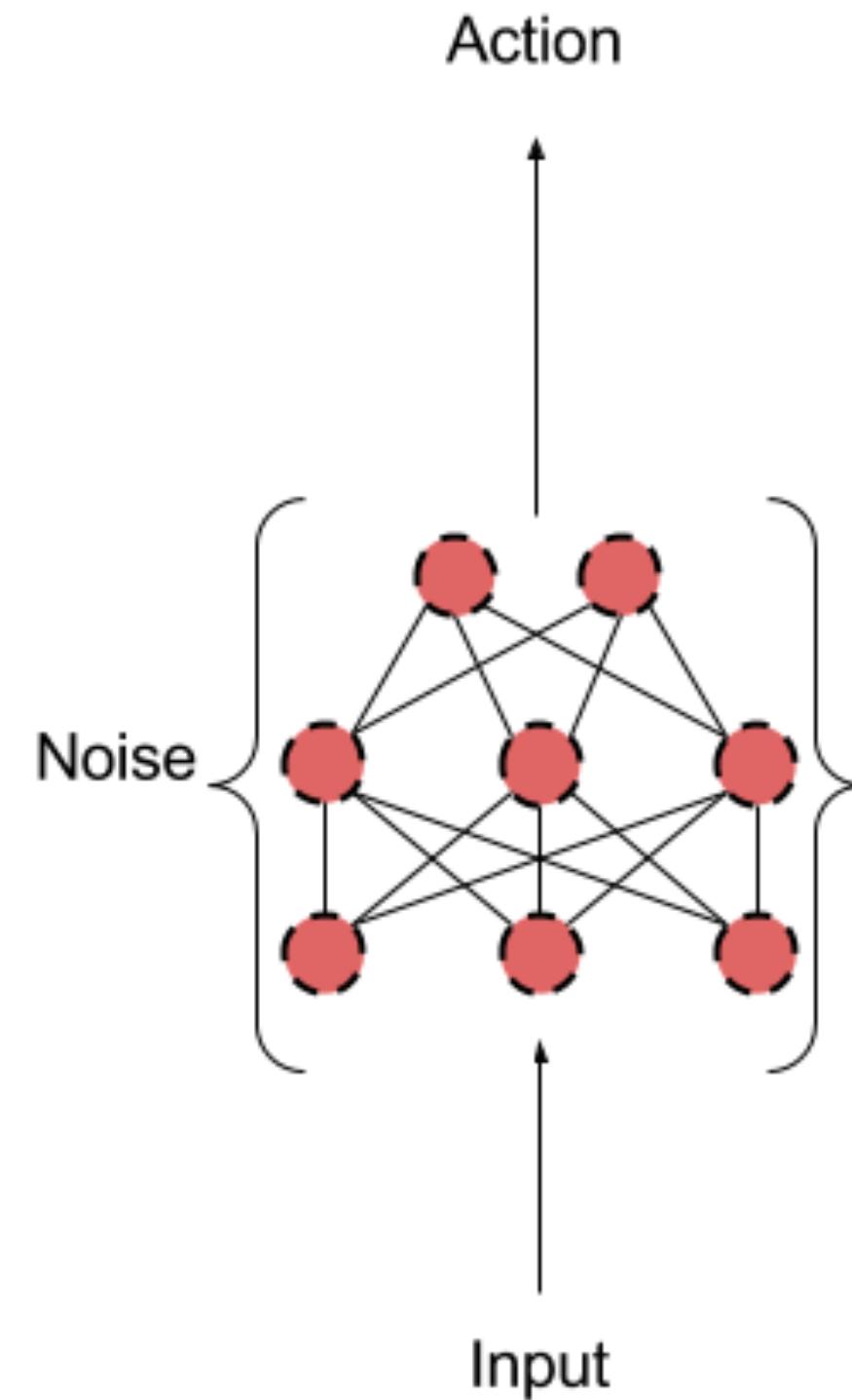
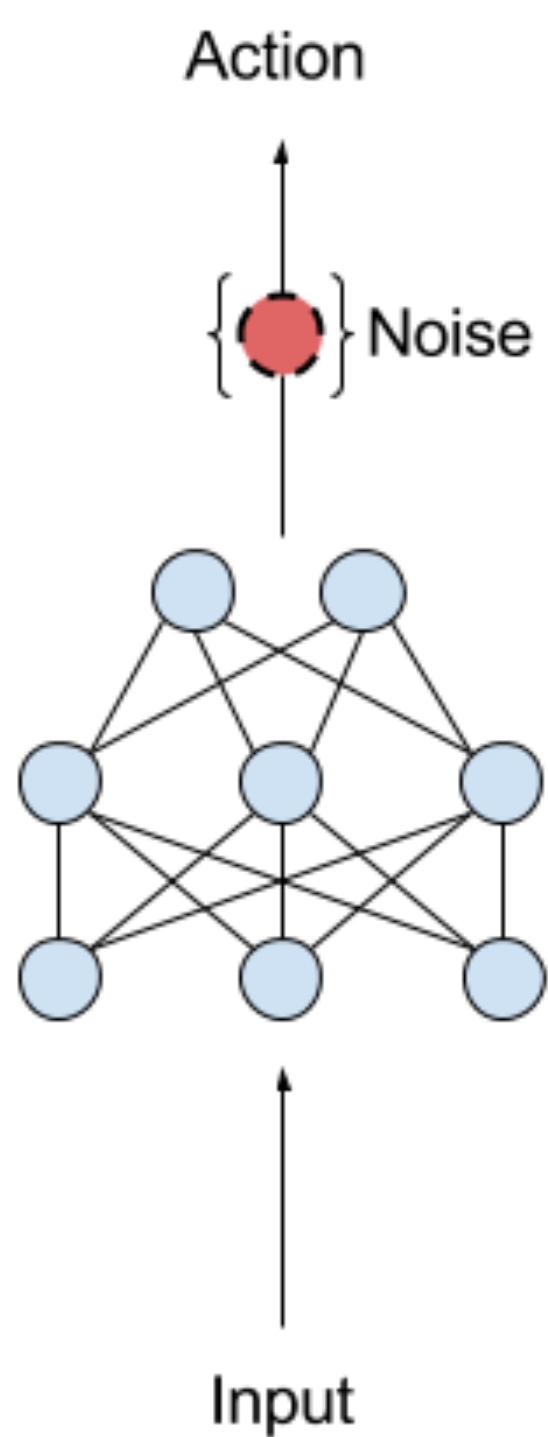
$$\max_a A^*(s, a) = Q^*(s, a) - V^*(s) = 0$$



# Noisy Networks

Issues with  $\varepsilon$ -greedy exploration:

- It's naive and must be adopted during training
- State-independent exploration



# Multi-step DQN

N-step Bellman equation:  $Q^*(s, a) = \mathbb{E}_{s', a', s'', a''} [\sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \mathbb{E}_{s^N} \max_{a^N} Q^*(s^N, a^N)]$

$$y_Q = r + \gamma r' + \gamma r'' + \dots + \gamma^n \max_{a^{(N)}} Q(s^{(N)}, a^{(N)})$$

# Multi-step DQN

N-step Bellman equation:  $Q^*(s, a) = \mathbb{E}_{s', a', s'', a''} [\dots] \sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \mathbb{E}_{s^N} \max_{a^N} Q^*(s^N, a^N)$

$$y_Q = \boxed{r + \gamma r' + \gamma r'' + \dots} + \gamma^n \boxed{\max_{a^{(N)}} Q(s^{(N)}, a^{(N)})}$$

Behave non-optimal N steps

Behave optimal w.r.t. to current Q

Theoretically incorrect in off-policy case!

<https://arxiv.org/abs/1901.07510>

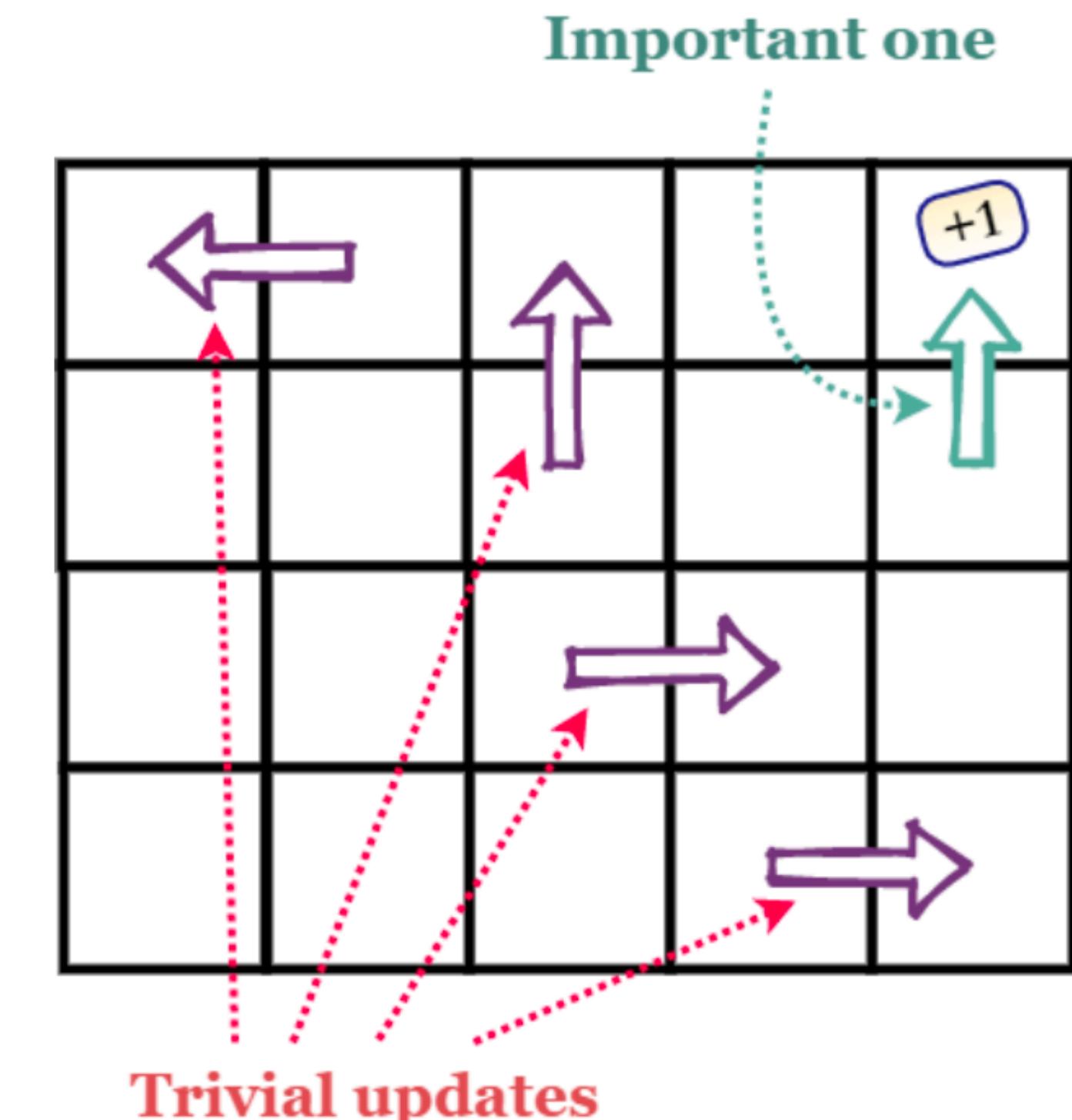
# Prioritized Experience Replay

$$\delta_i = y_i - Q_\theta(s_i, a_i) - \text{TD-error}$$

$$p_i = |\delta_i| + \epsilon, \epsilon > 0$$

$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$  is the probability of sampling transition  $i$

- we induce a bias in a  $Q$ -function approximation
- $s'$  can be not from  $p(\cdot | s, a)$



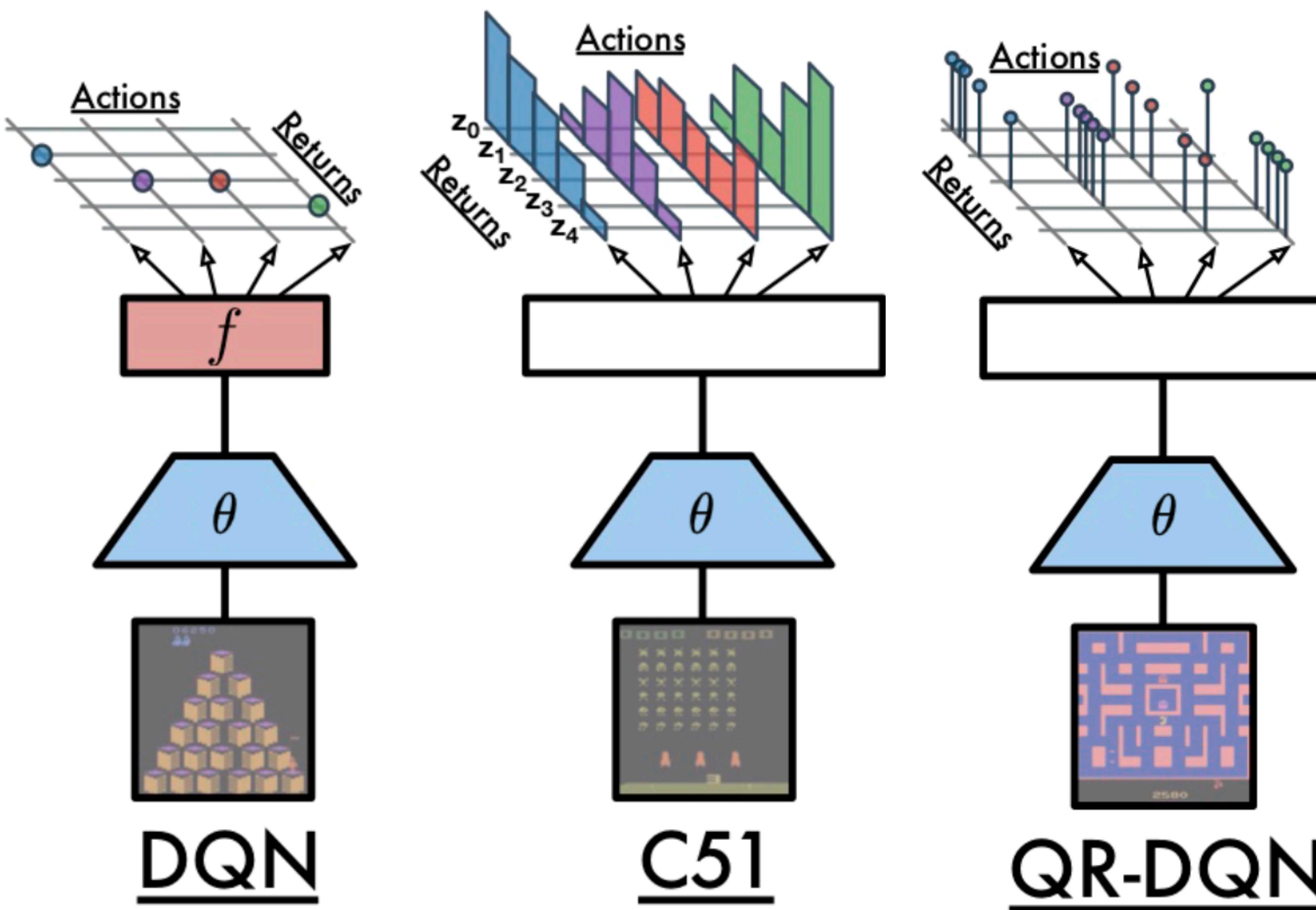
[Source](#)

We can correct this bias by using Importance-Sampling (IS) weights

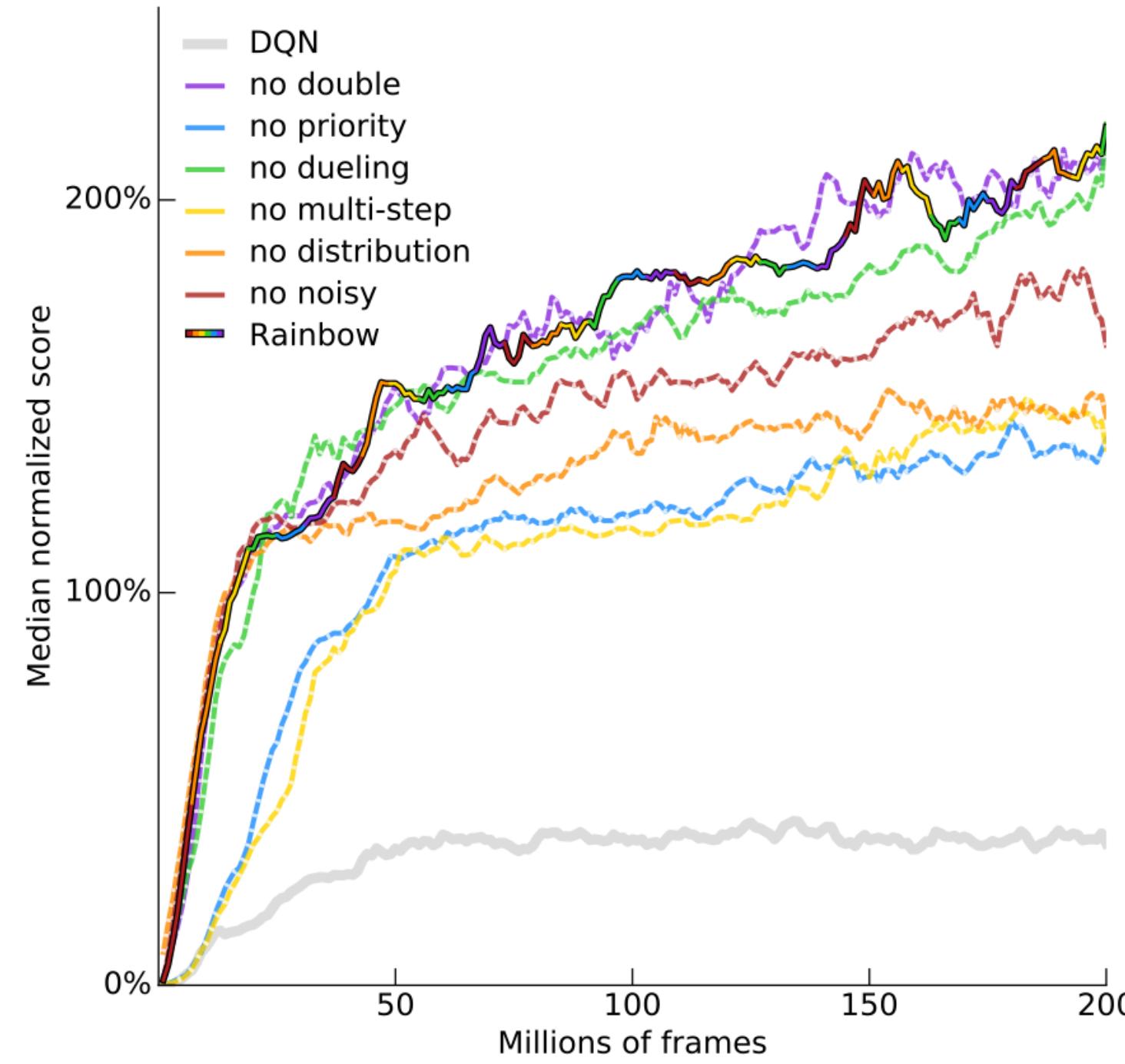
$$w_i = \left( \frac{1}{P(i)} \right)^{\beta(t)}$$

with beta annealing from 0 to 1 and  $w_i \delta_i$  instead of  $\delta_i$ .

# Distributional RL

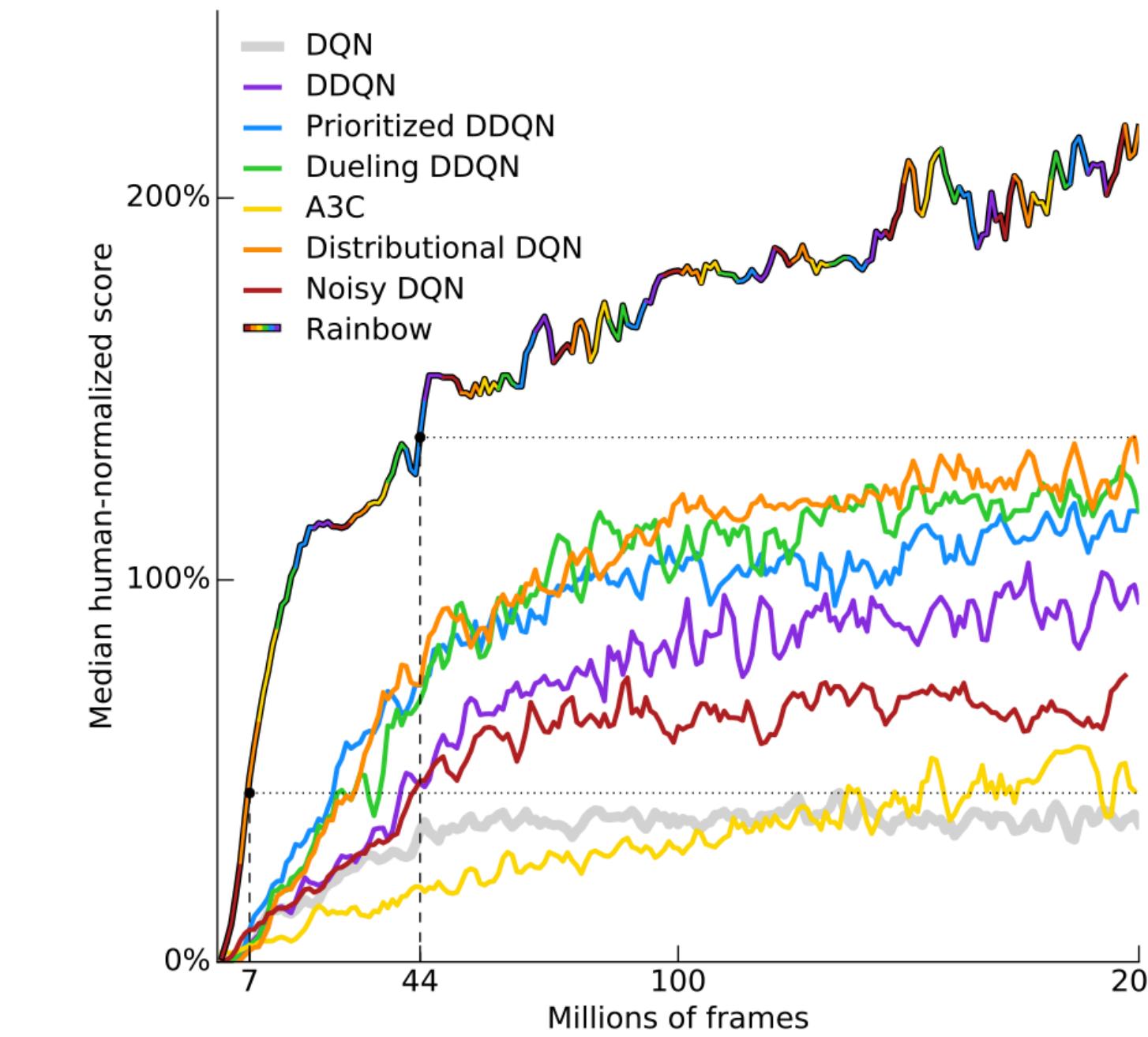


# Ablation Study



**Figure 3: Median human-normalized performance** across 57 Atari games, as a function of time. We compare our integrated agent (rainbow-colored) to DQN (gray) and to six different ablations (dashed lines). Curves are smoothed with a moving average over 5 points.

[Source](#)



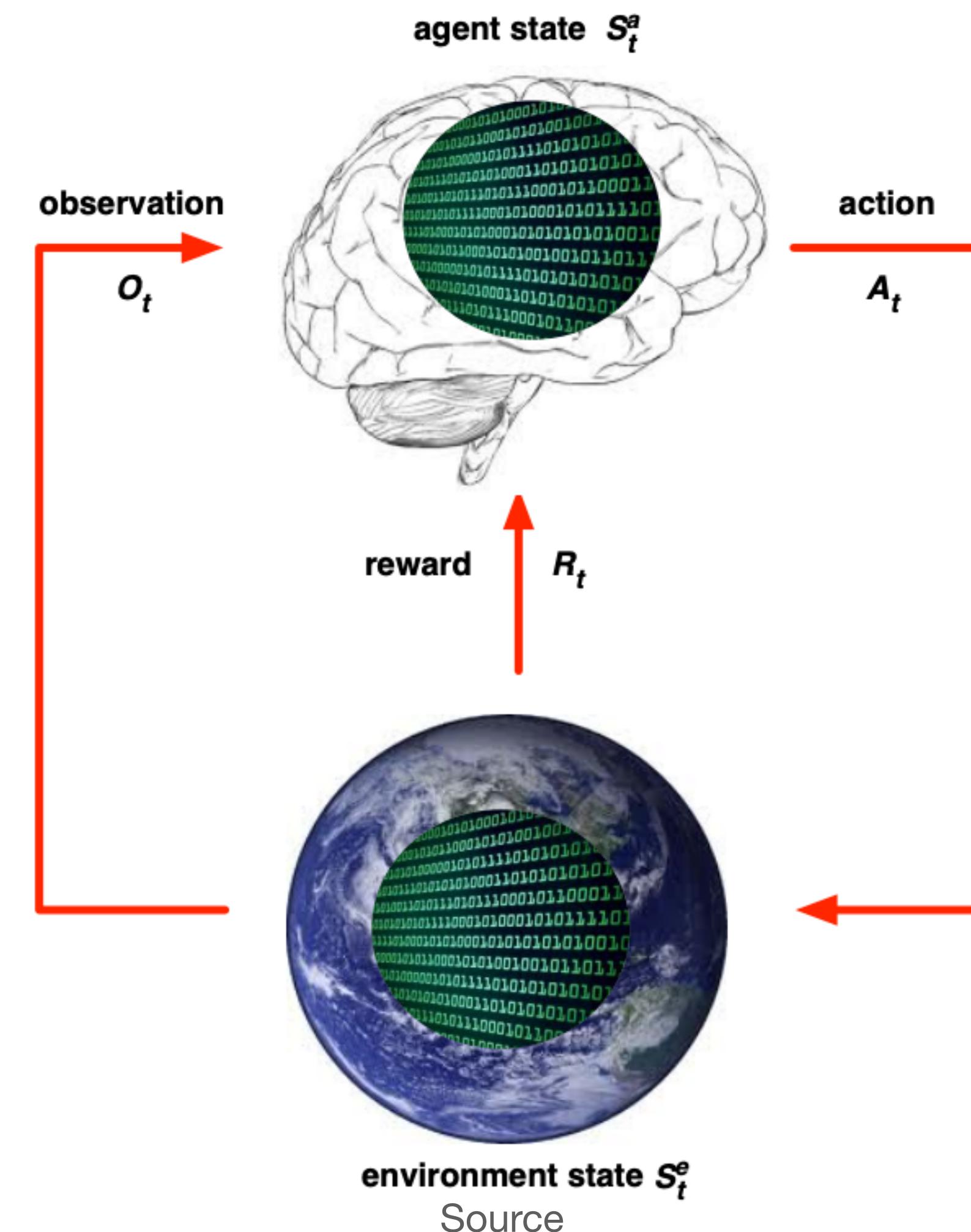
**Figure 1: Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

[Source](#)

# POMDP

Partially observable:  $S_t^e \neq S_t^a$

1. MDP +
2.  $\mathcal{O}$  is a set of possible observations
3.  $p(o|s) = \mathbb{P}(O_t = o | S_t = s)$  is a probability to get the observation given the state

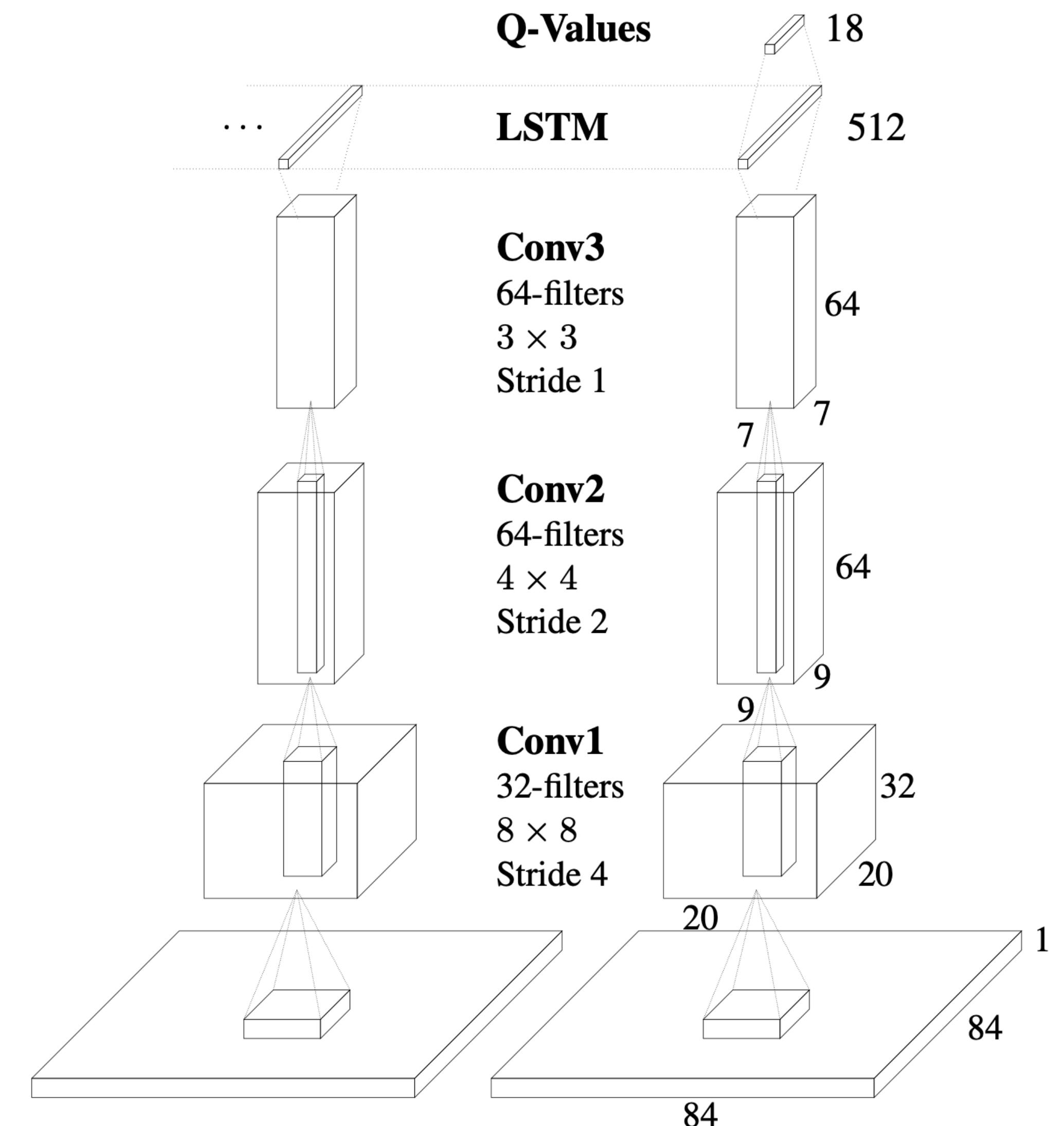
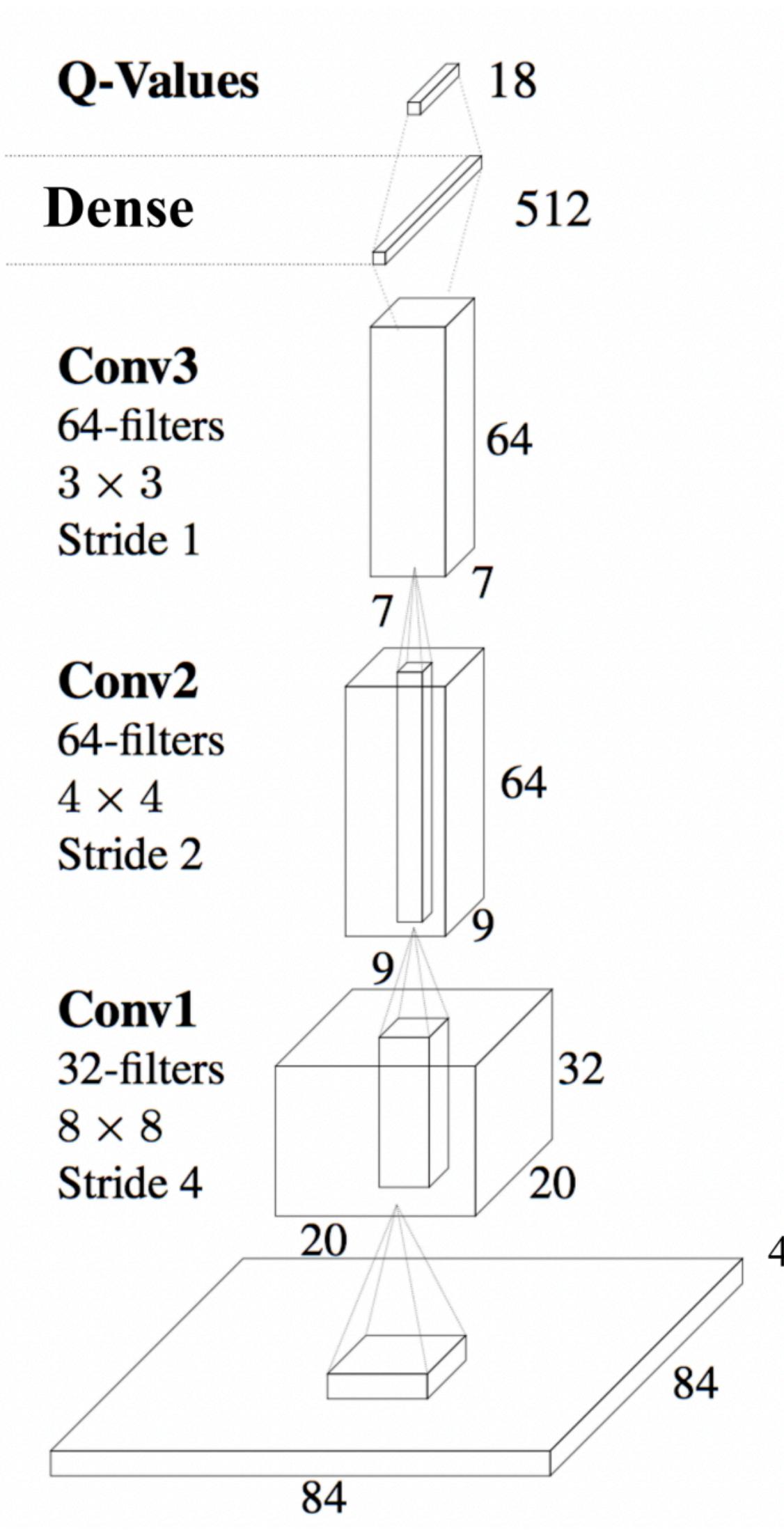


# DRQN

In the general case, estimating a Q-value from an observation can be arbitrarily bad since  $Q_\theta(o, a) \neq Q_\theta(s, a)$ .

- Let's equip agent with memory  $h_t$
- $Q(s_t, a_t) \approx Q(o_t, h_{t-1}, a_t)$
- $h_t = LSTM(o_t, h_{t-1})$

# DQN vs DRQN

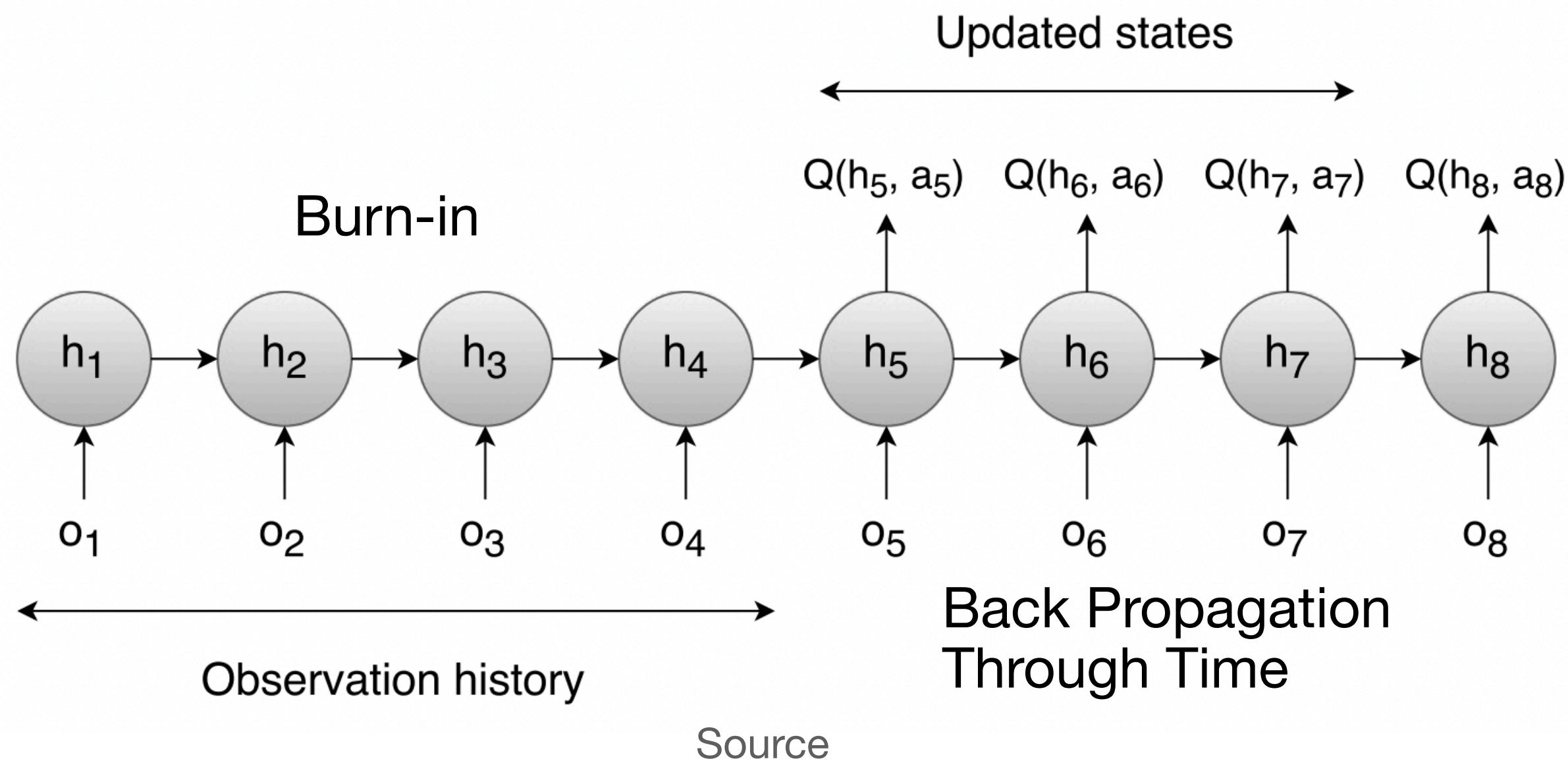


Original paper

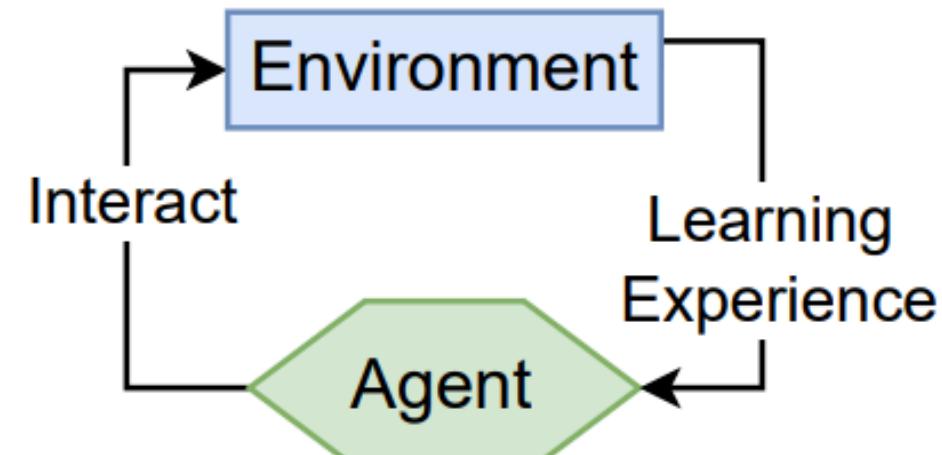
Original paper

# Recurrent Experience Replay

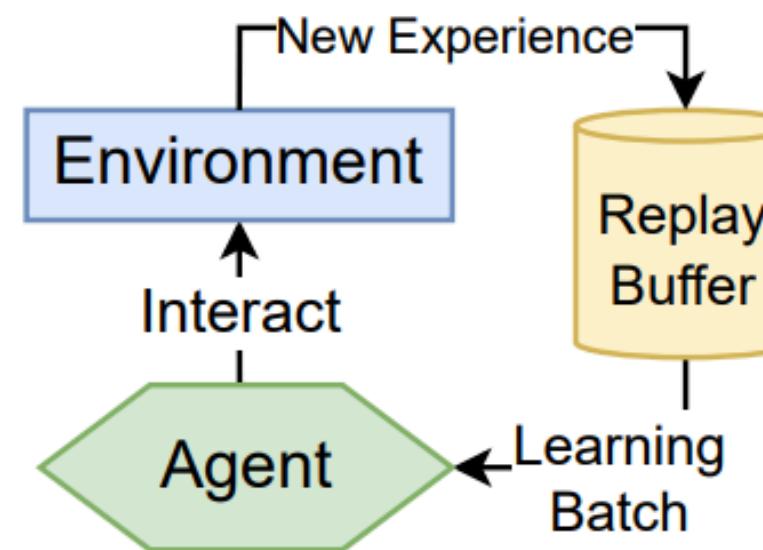
- Sample random time step
- Consider  $N$  consecutive transitions
- Update only last  $M$



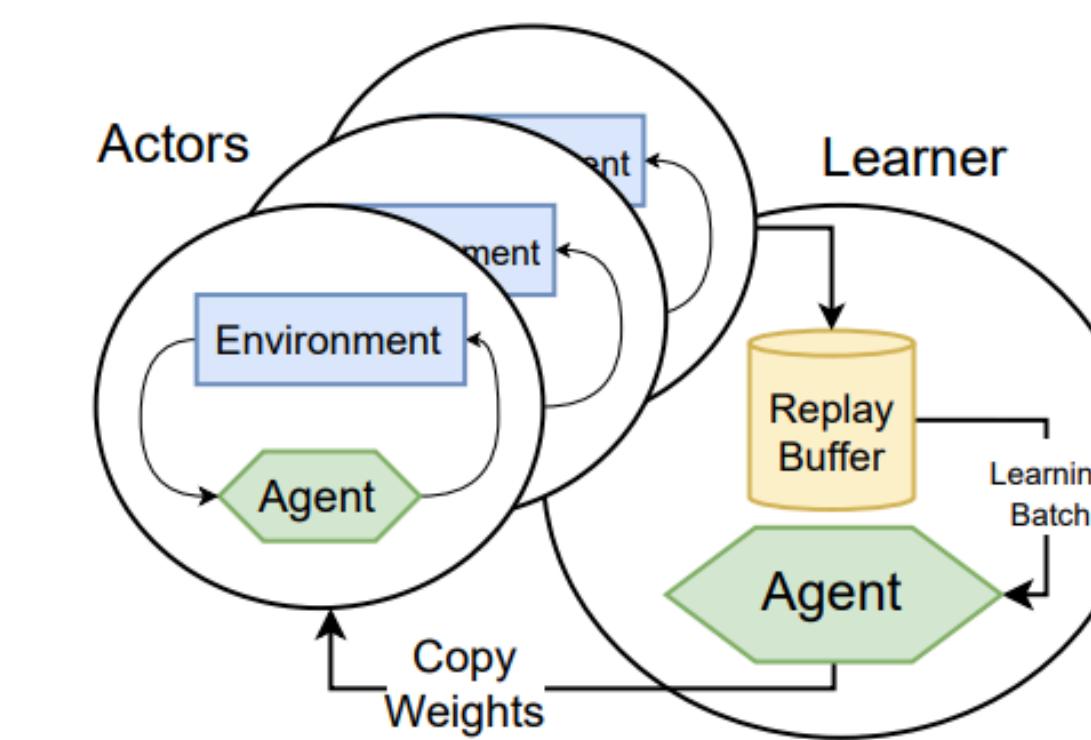
# Distributed DQN



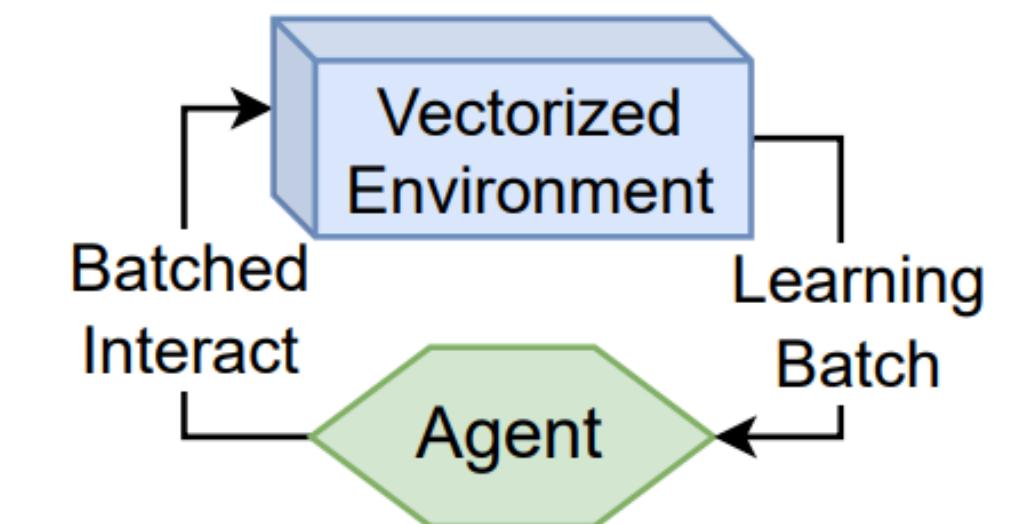
(a) Online  $Q$ -Learning



(b) DQN



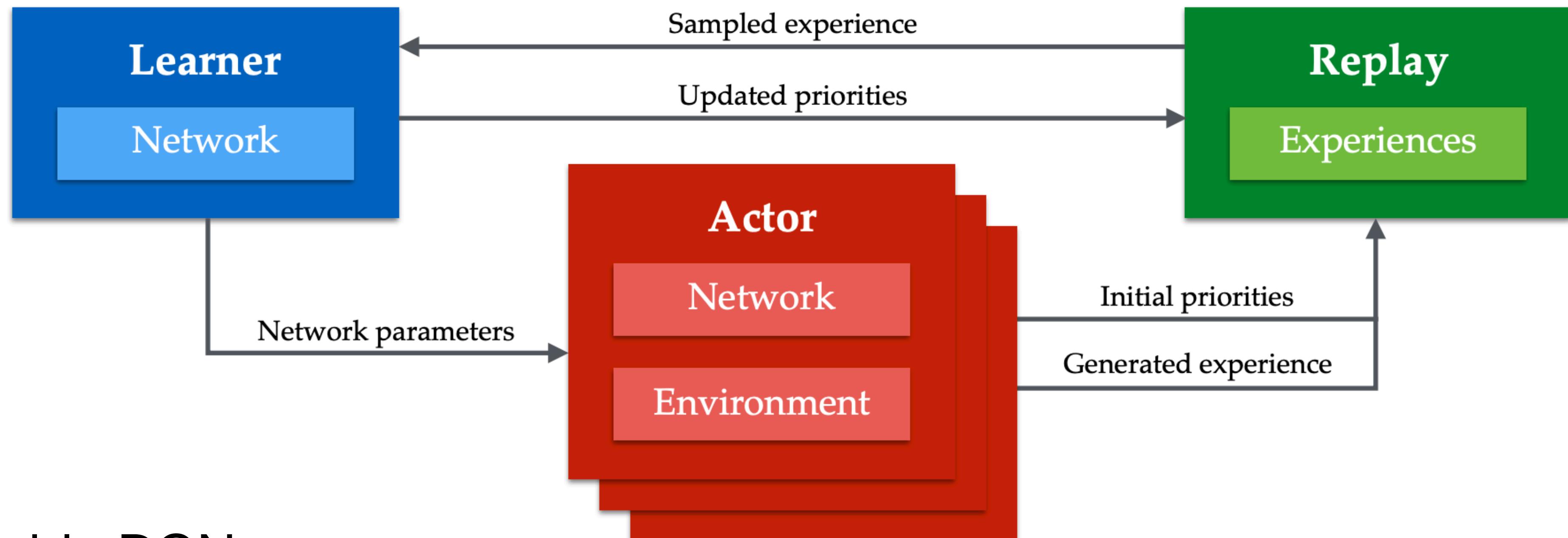
(c) Distributed DQN



(d) PQN

Source

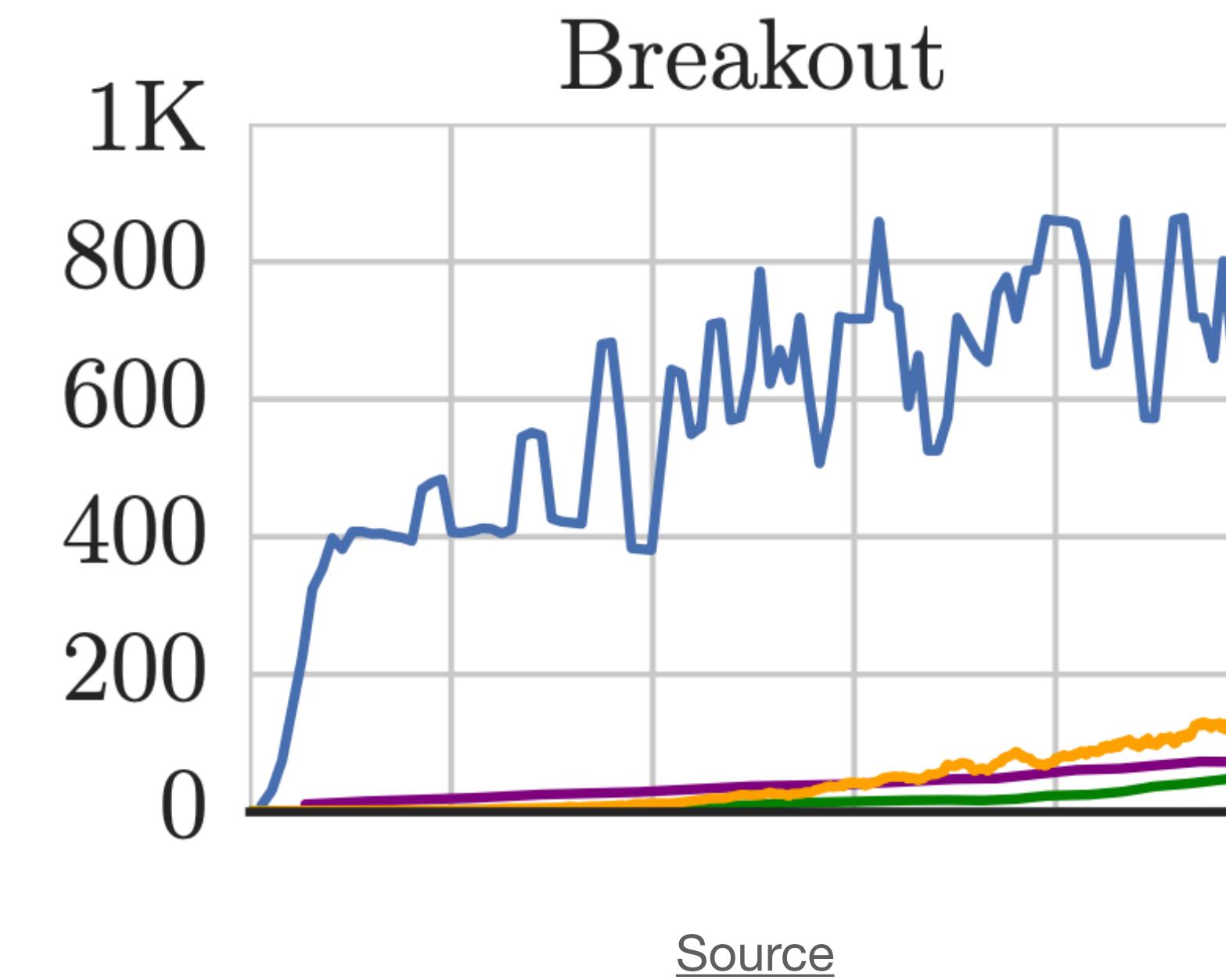
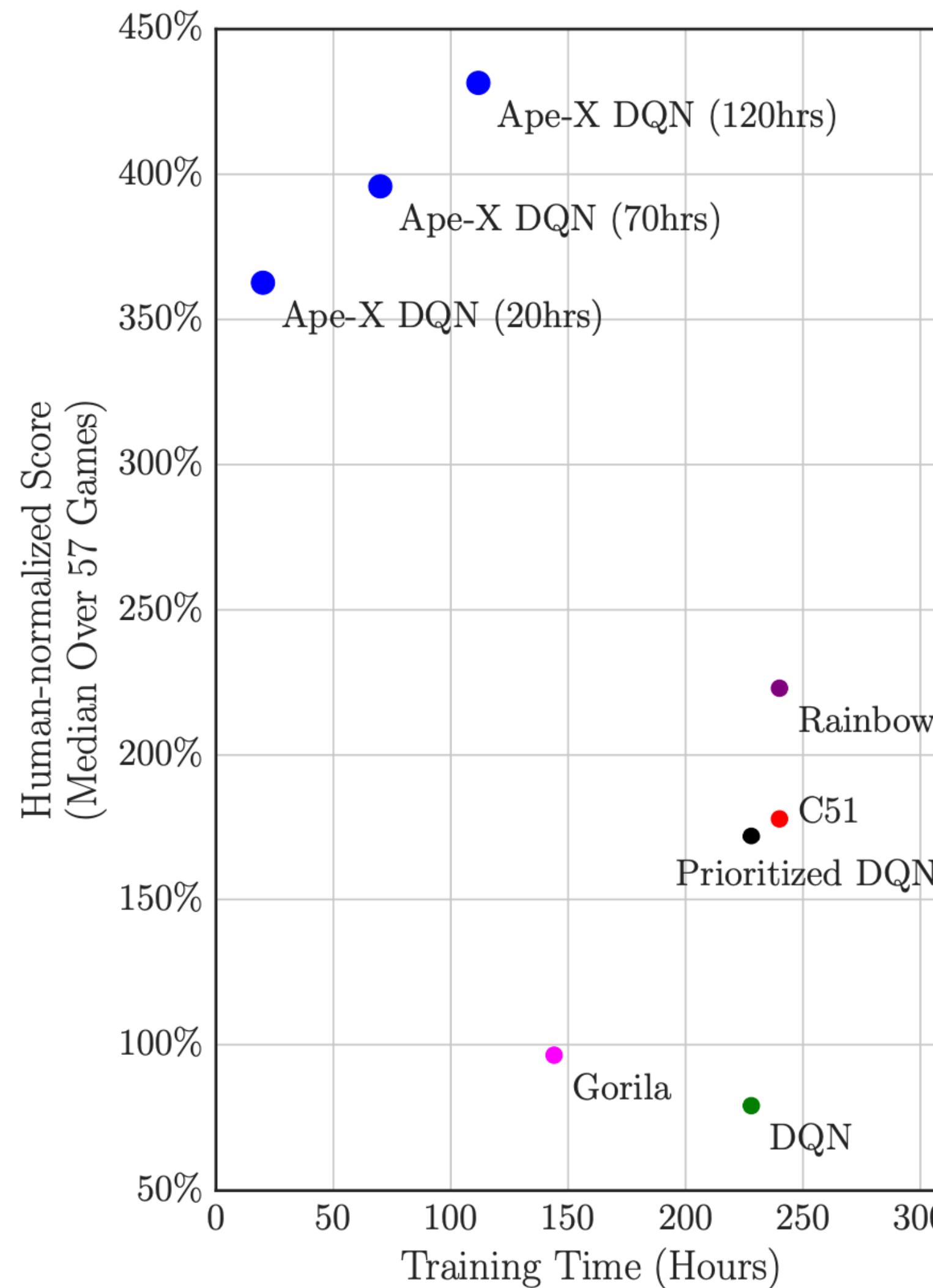
# Ape-X: Distributed Prioritized Experience Replay



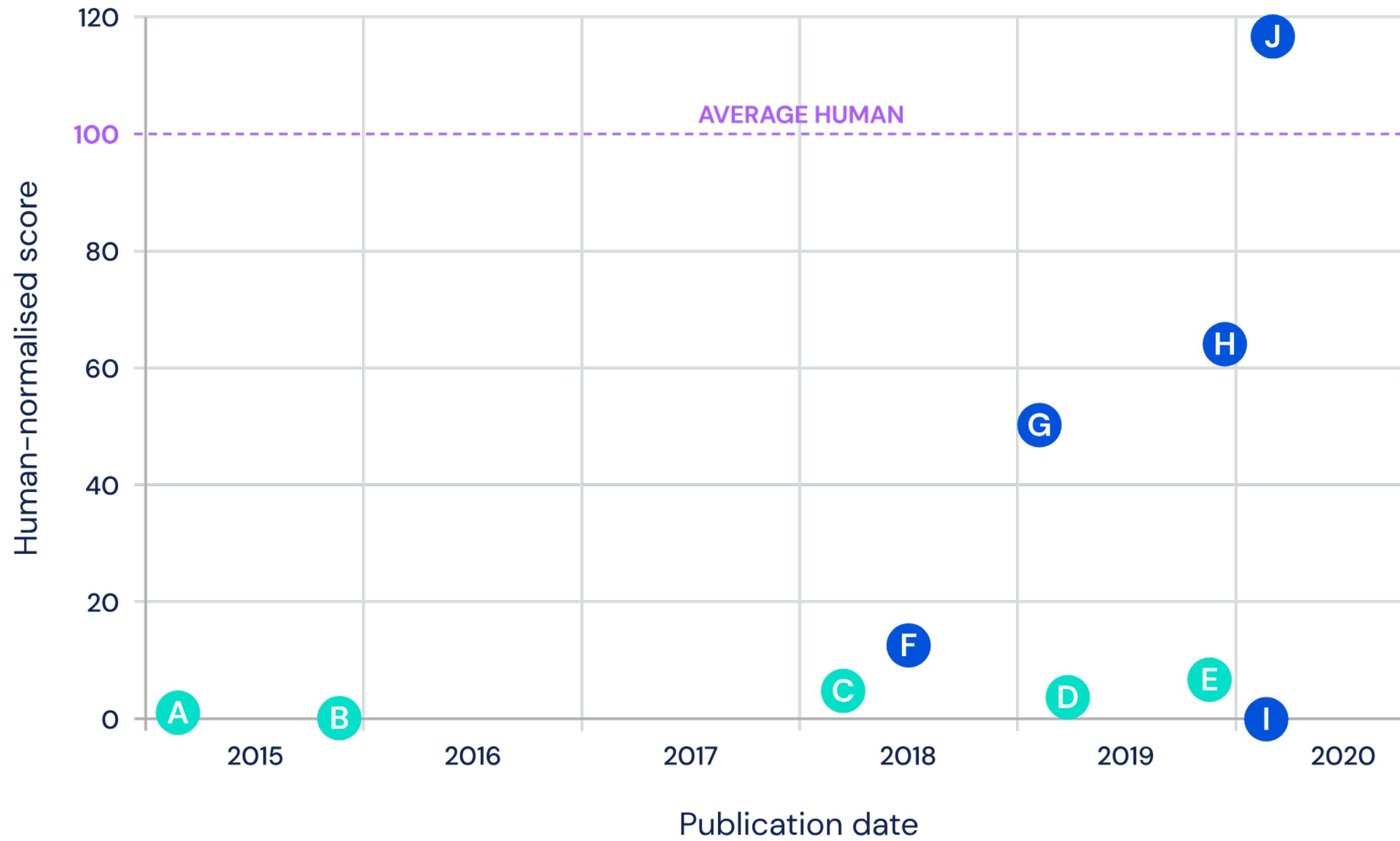
1. Double DQN
2. 5-step target
3. Dueling DQN

Source

# Ape-X: Distributed Prioritized Experience Replay



# Atari-57 5th percentile performance



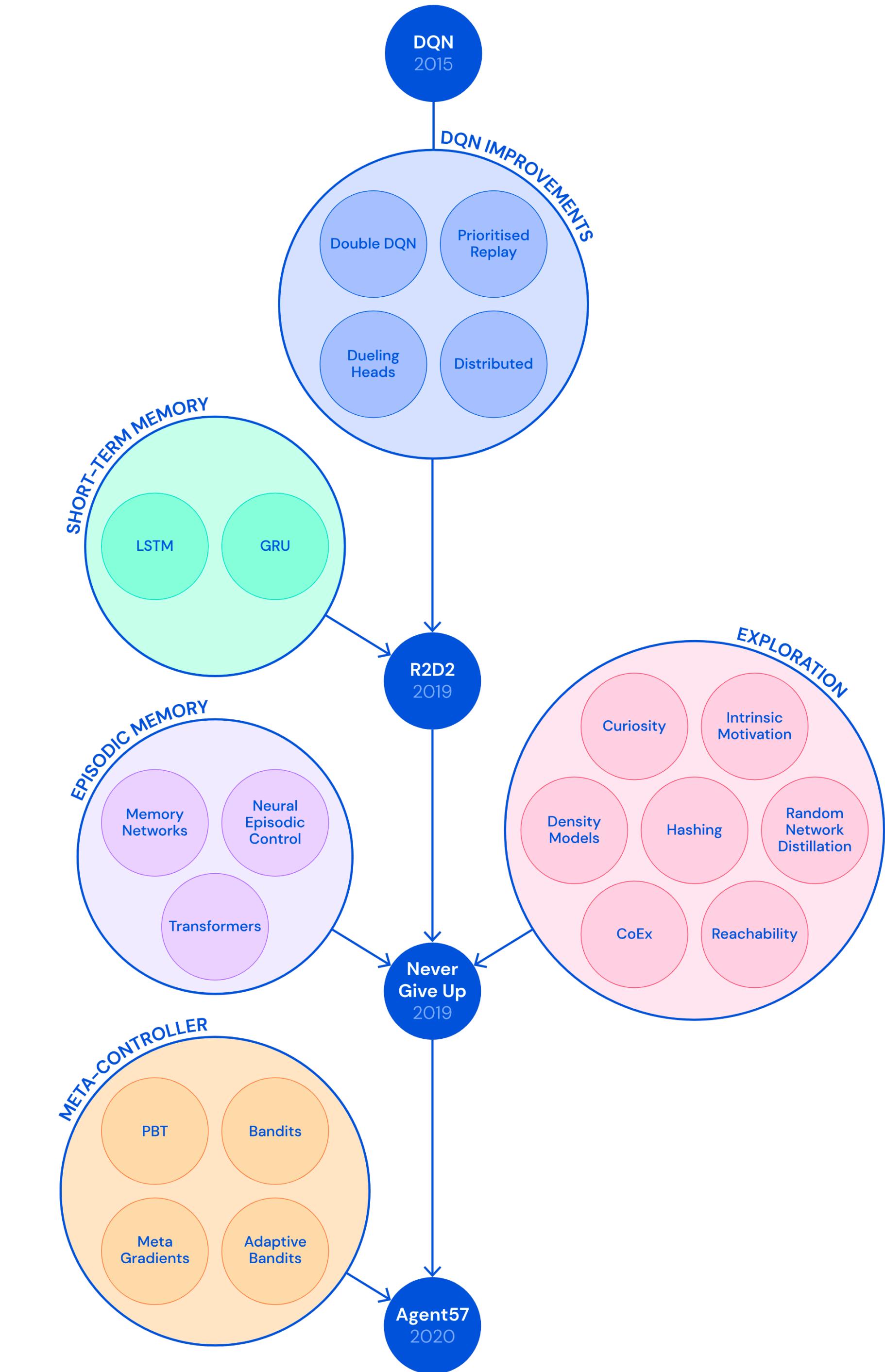
## Single-actor agents

- (A) DQN
- (B) Prioritised Dueling
- (C) Rainbow
- (D) C51-IDS
- (E) FQF

## Distributed agents

- (F) ApeX
- (G) R2D2
- (H) NGU
- (I) MuZero
- (J) Agent57

<https://deepmind.google/discover/blog/agent57-outperforming-the-human-atari-benchmark/>



# Background

1. Practical RL course by YSDA, week 4
2. Reinforcement Learning Textbook (in Russian): 4

**Thank you for your attention!**