girafe ai

Lecture 04: Approximate Q-learning, DQN

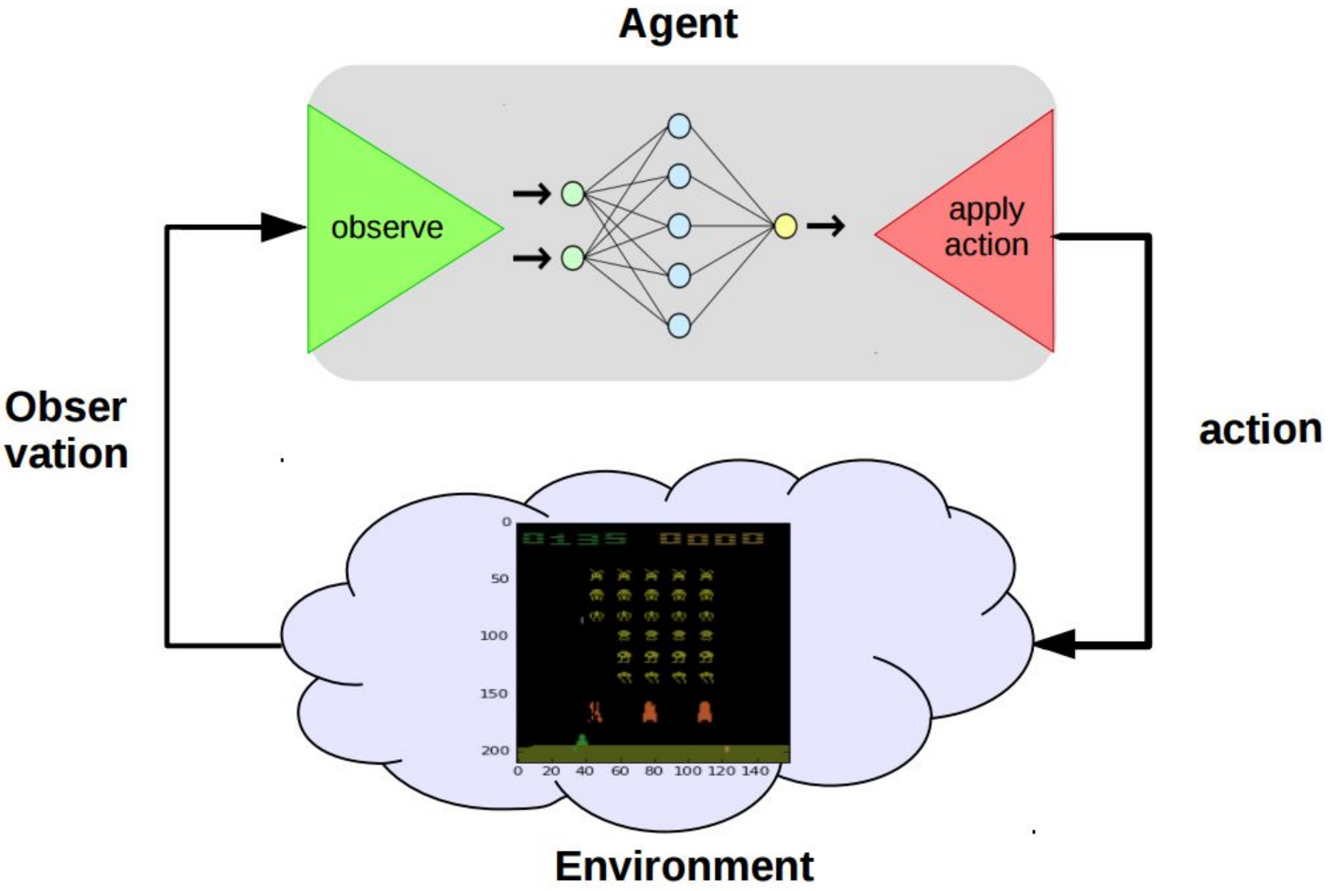
Radoslav Neychev

References

These slides are almost the exact copy of Practical RL course week 4 slides.
Special thanks to YSDA team for making them publicly available.

Original slides link: week04_approx_rl

MDP


Agent
observe
apply action
Obser vation
action
Environment
0
50
100
150
200
0 20 40 60 80 100 120 140

Basic definitions

$$G_t = \sum_{t'=t}^{T} \gamma^{(t'-t)} r_{t'}$$

$$Q^\pi(s,a) = E_\pi[G_t | s_t = s, a_t = a]$$

$$V^\pi(s) = E_\pi[G_t | s_t = s]$$

Recurrent relations

$$Q^\pi(s,a) = E_{s_{t+1}}[r_t + \gamma V^\pi(s_{t+1})]$$

$$Q^\pi(s,a) = E_{s_{t+1},a_{t+1}\sim\pi}[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1})]$$

Optimal policy

For all π, s, a: $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$

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

Bellman optimality equation

$$Q^*(s_t,a) = E_{s_{t+1}}[r_t + \max_{a'} Q^*(s_{t+1},a')]$$

Training step

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

Q-learning as MSE minimization

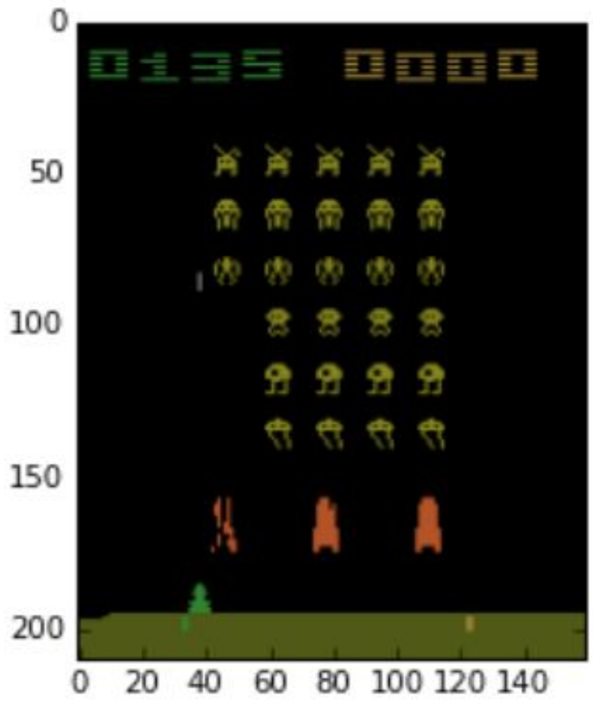
$$L = (r_t + \gamma \max_{a'} \boxed{Q(s_{t+1}, a')} - Q(s_t, a_t))^2$$

Const

$$\nabla L = 2 \cdot (r_t + \gamma \max_{a'} \boxed{Q(s_{t+1}, a')} - Q(s_t, a_t))$$

What's wrong here?

Real world



How many states are there?

approximately

$$|S| = 2^{210 \cdot 160 \cdot 8 \cdot 3}$$

Q-learning: make it continuous

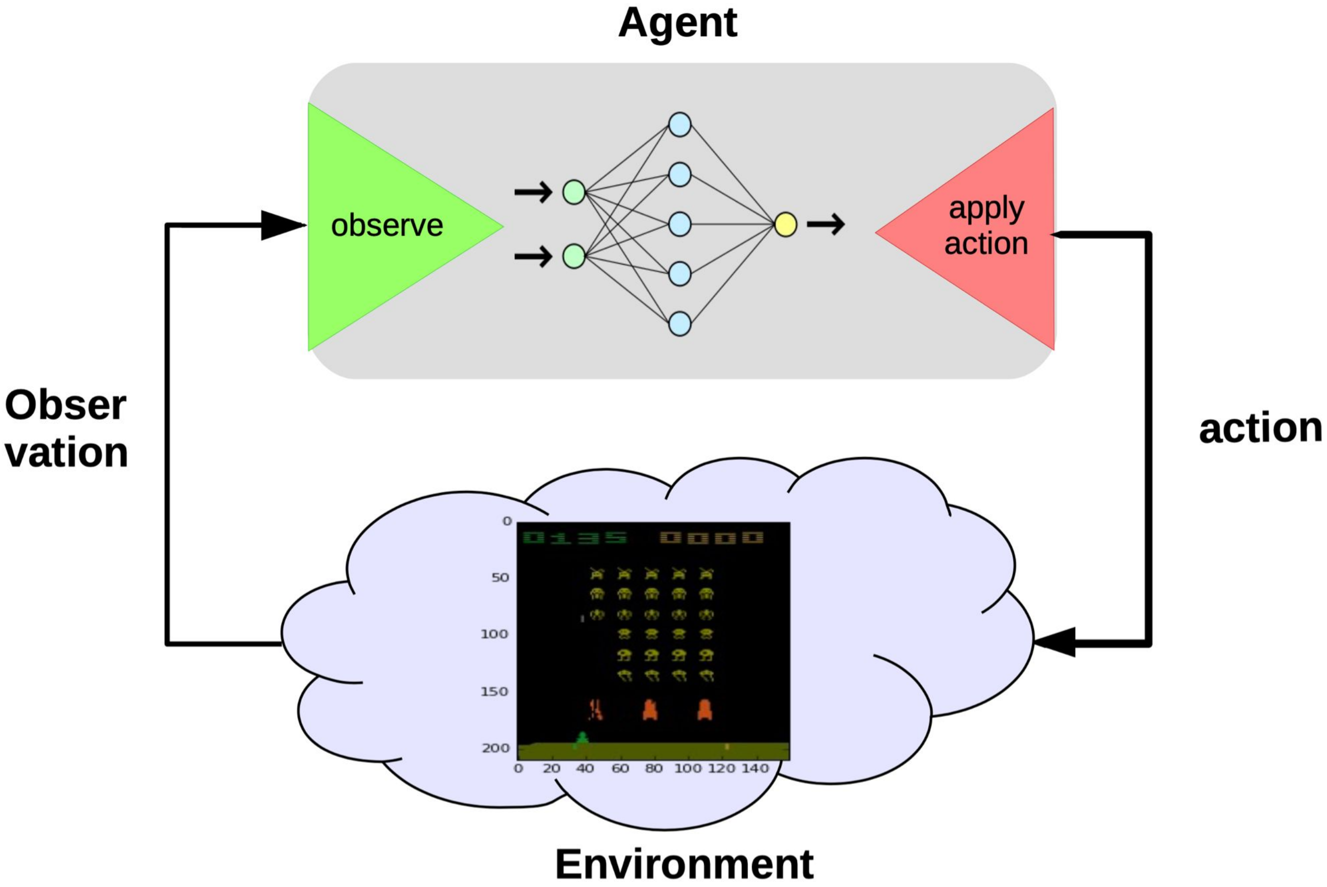
For now states and actions are discrete. What if states are **not**?

Minimize loss given <**s, a, r, s'>**

$$L = [Q(s_t, a_t) - Q^{true}(s_t, a_t)]^2$$

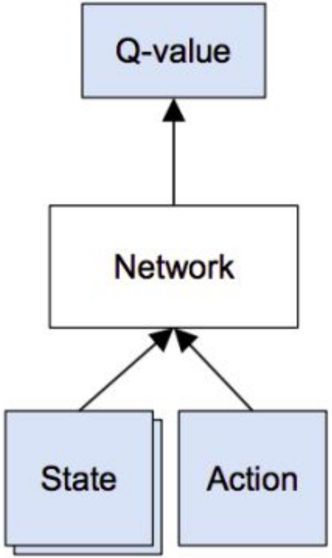
$$L \approx [Q(s_t, a_t) - (r_t + \gamma \cdot max_{a'} Q(s_{t+1}, a'))]^2$$

Approximate Q-learning

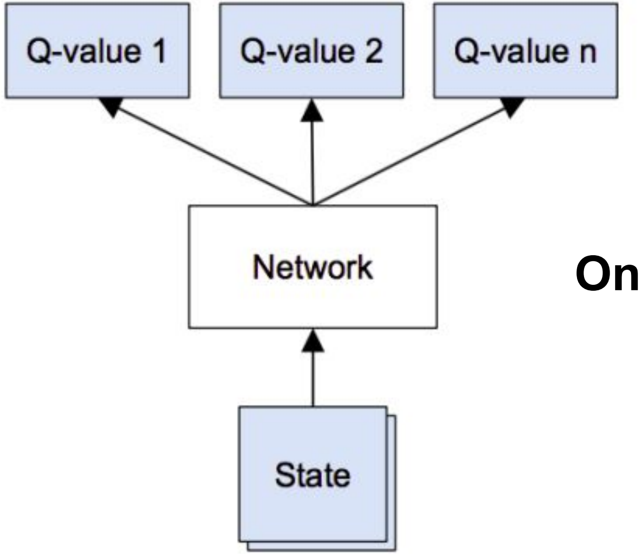


Possible architectures

Continuous control or large number of actions



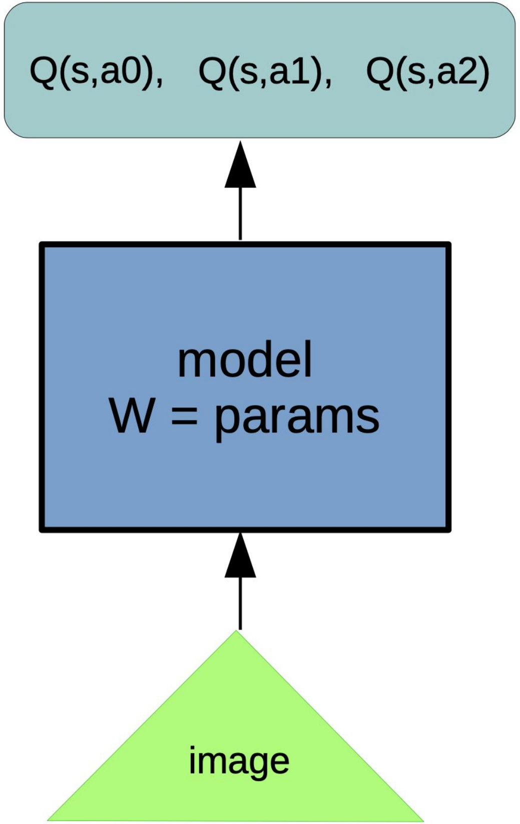
Given **(s,a)**
Predict Q(s,a)



Given **s** predict all q-values
Q(s,a0), Q(s,a1), Q(s,a2)

One pass for all actions

Approximate Q-learning



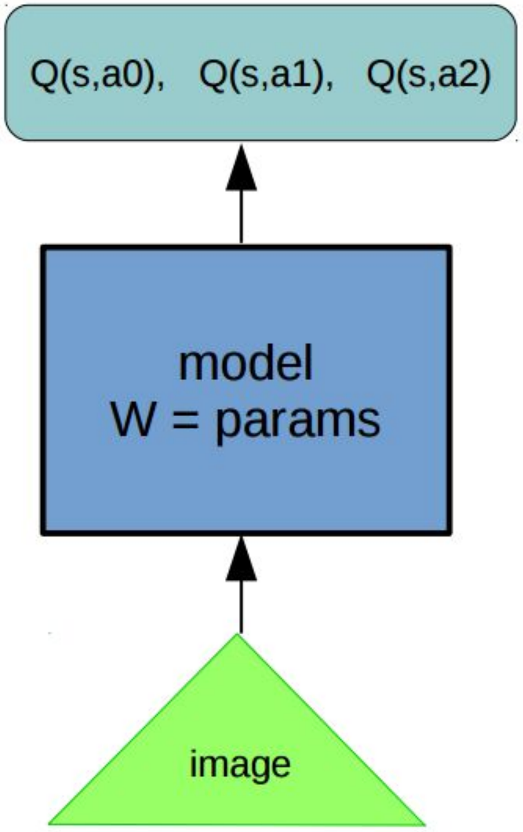
$$\hat{Q}(s_t, a_t) = r + \gamma \cdot max_{a'} \hat{Q}(s_{t+1}, a')$$

$$L = (Q(s_t, a_t) - \boxed{[r + \gamma \cdot max_{a'} Q(s_{t+1}, a')]})^2$$

Consider const

$$w_{t+1} = w_t - \alpha \cdot \frac{\delta L}{\delta w}$$

Approximate Q-learning



Objective:

$$L = \left(Q(s_t, a_t) - \underbrace{\hat{Q}(s_t, a_t)}_{\text{consider const}}\right)^2$$

Q-learning:

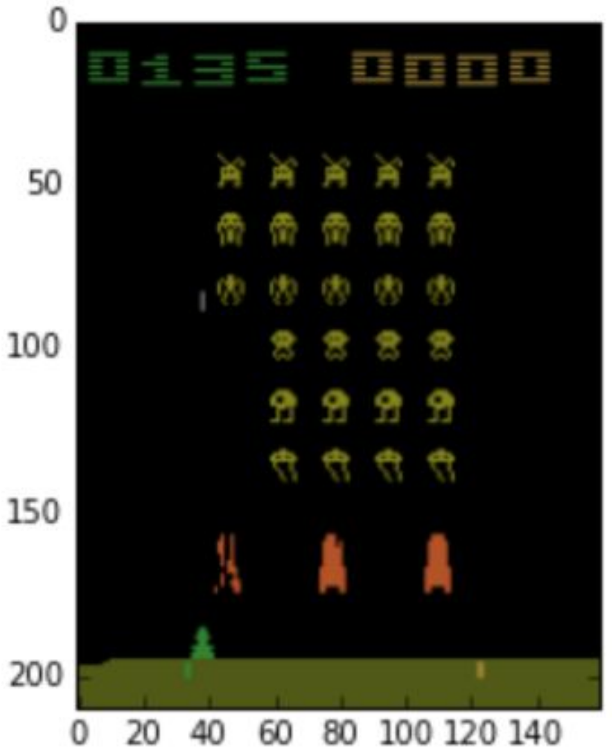
$$\hat{Q}(s_t, a_t) = r + \gamma \cdot max_{a'} Q(s_{t+1}, a')$$

SARSA:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot Q(s_{t+1}, a_{t+1})$$

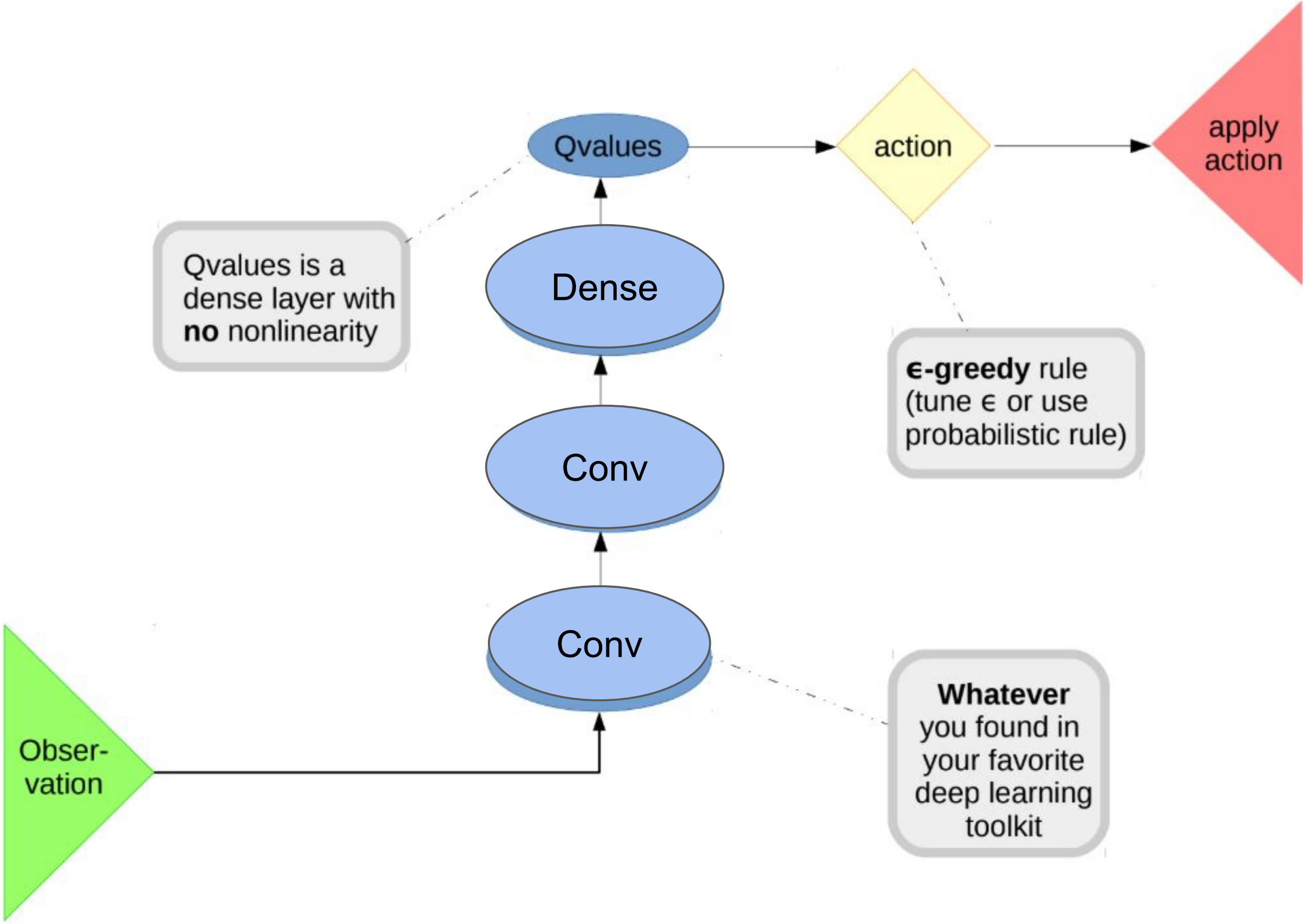
Expected Value SARSA:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \underset{a' \sim \pi(a|s)}{E} Q(s_{t+1}, a')$$

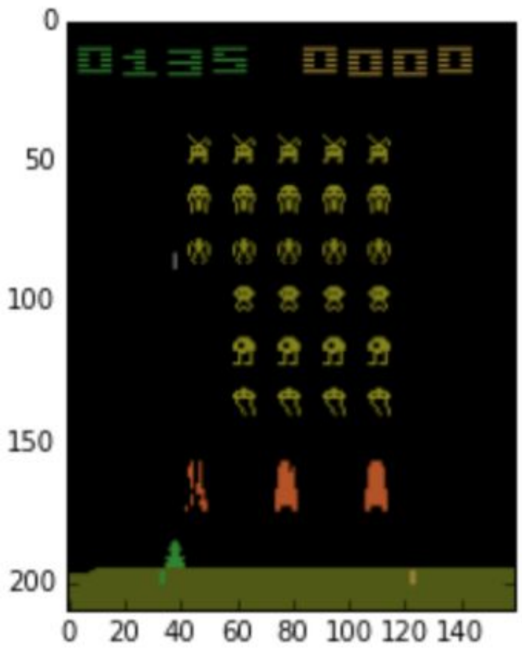


What kind of network digests images well?

Basic deep Q-learning




WHAT IF I TOLD YOU
THAT IT'S ALL ONE GREAT BIG LIE.
imgflip.com

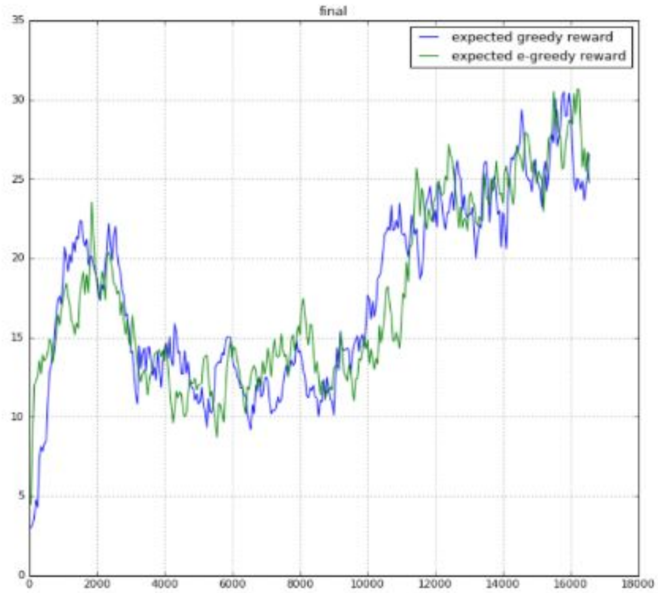


How bad it is if agent spends

next 1000 ticks under the left rock?
(while training)

Problem

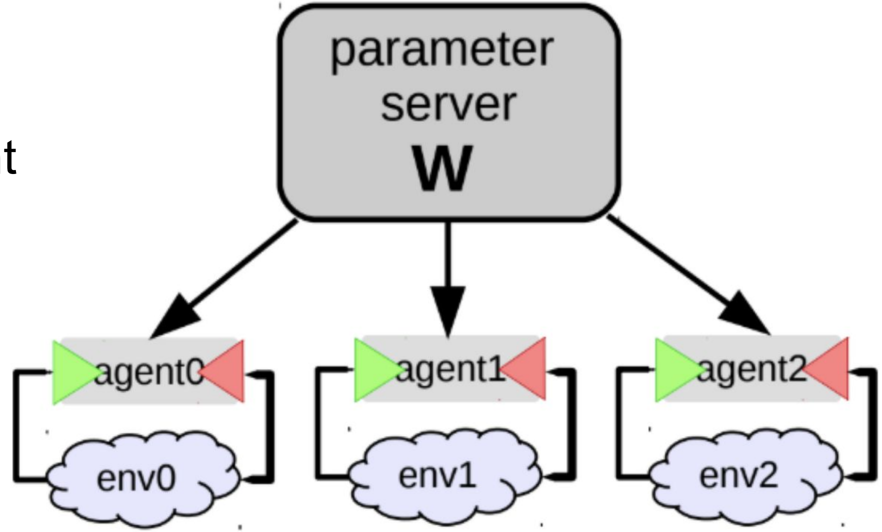
- Training samples are **not "i.i.d"**,
- Model forgets parts of environment it hasn't visited for some time
- Drops on learning curve
- **Any ideas?**



Multiple agents trick

Idea: Throw in several agents with shared **W**.

- Chances are, they will be exploring different parts of the environment
- More stable training
- Requires a lot of interaction



***Question:** your agent is a real robot car. Will there be any problems?*

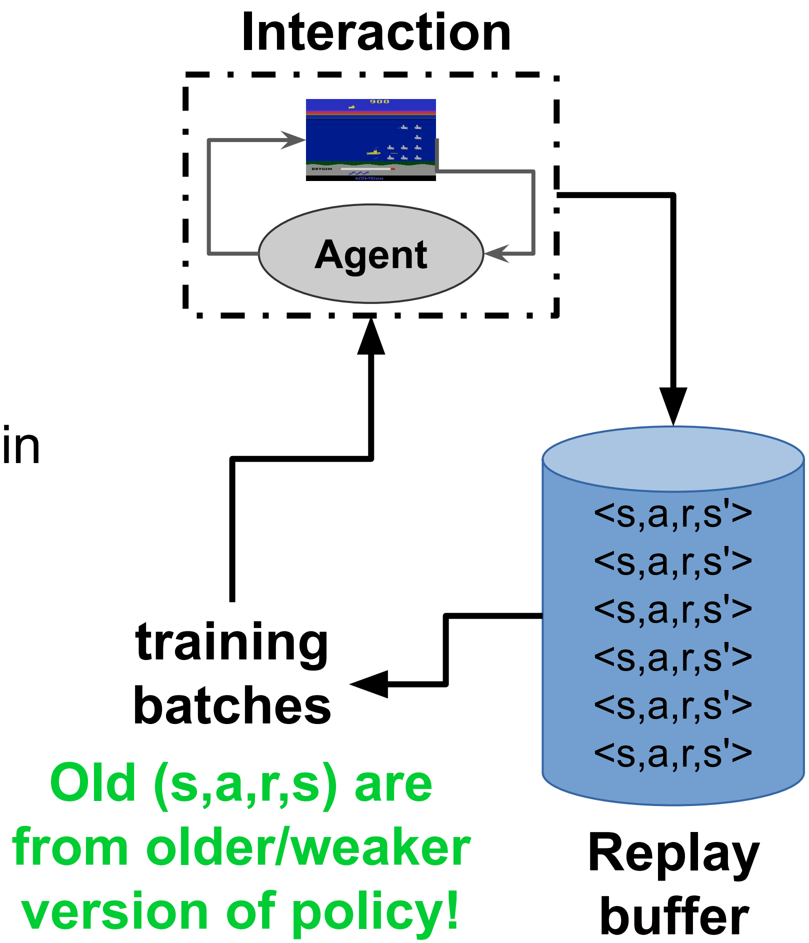
Experience replay

Idea:

Store several past interactions

<s,a,r,s'>

Train on random subsamples



Training curriculum:

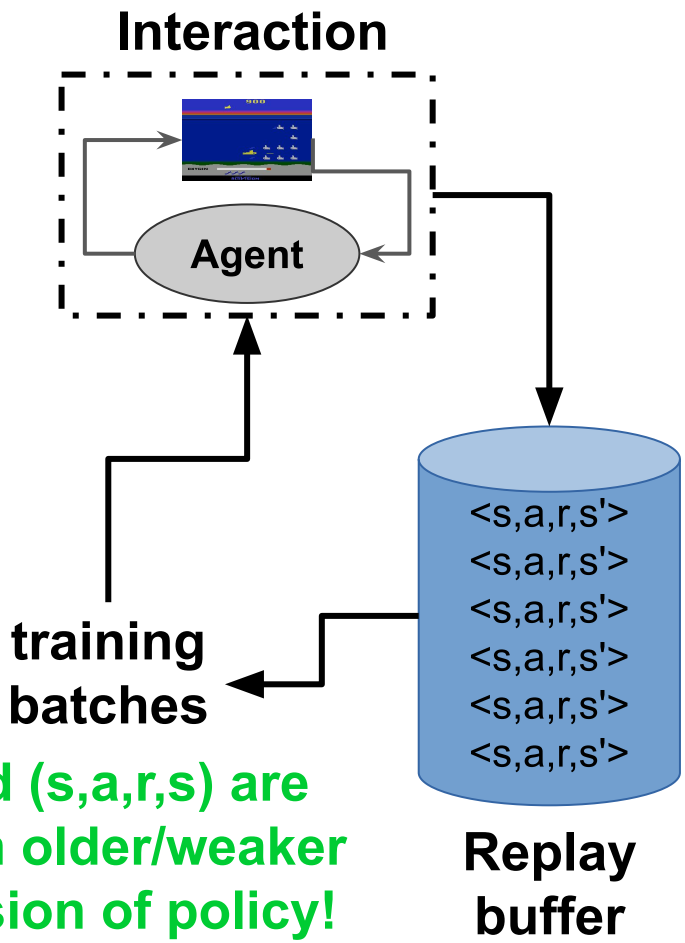
- Play 1 step and record it
- Pick N random transitions to train

Profit:

you don't need to revisit same (s,a) many times to learn it.

Experience replay

- Atari DQN > 10^5 interactions
- Closer to i.i.d.
  pool contains several sessions
- Older interactions were
  obtained under weaker policy



Old (s,a,r,s) are from older/weaker version of policy!

Experience replay

- You approximate Q(s, a) with a neural network
- You use **experience replay** when training

Question: which of those algorithms will fail?

- Q-learning
- SARSA
- CEM
- Expected Value SARSA

Experience replay

- You approximate Q(s, a) with a neural network
- You use **experience replay** when training

Agent trains off-policy on an older version of himself

Question: which of those algorithms will fail?

Off-policy methods work, **On-policy methods are super slow (fail)**

- **Q-learning**
- **SARSA**
- **CEM**
- **Expected Value SARSA**

When training with on-policy methods,

- use no (or small) experience replay
- compensate with parallel game sessions

Left or right?

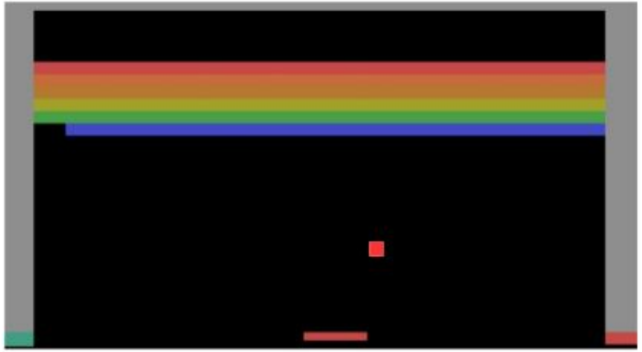
N-gram trick

Idea:

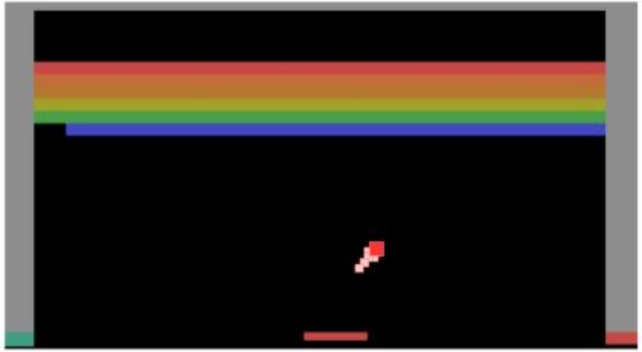
$$s_t \neq o(s_t)$$

$$s_t \approx (o(s_{t-n}), a_{t-n}, \ldots, o(s_{t-1}), a_{t-1}, o(s_t))$$

e.g. ball movement in breakout



· One frame

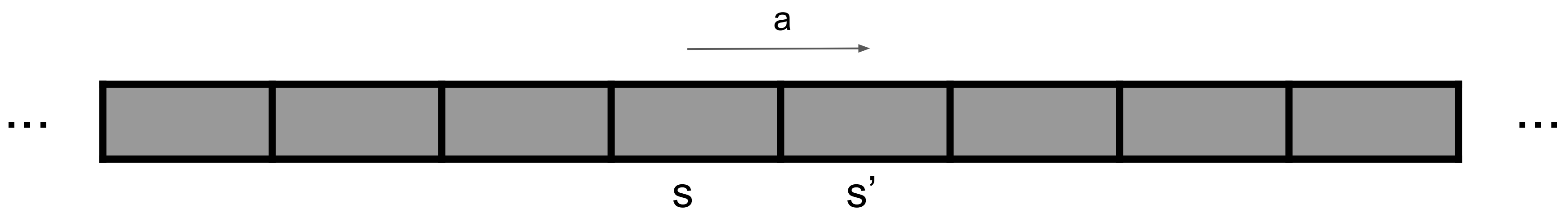


· Several frames 48

N-gram trick

- Nth-order markov assumption
- Works for velocity/timers
- Fails for anything longer that N frames
- Impractical for large N

Autocorrelation



Target is based on prediction

Q(s, a) correlates with Q(s', a)

Target network

Idea: use network with frozen weights to compute the target

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

where Θ^- is the frozen weights

Hard target network:

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

Target network

Idea: use network with frozen weights to compute the target

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

where Θ^- is the frozen weights

Const

Hard target network:

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

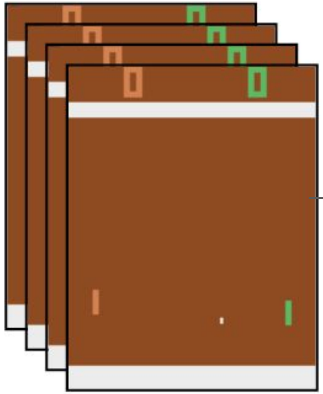
Soft target network:

Update Θ^- every step:

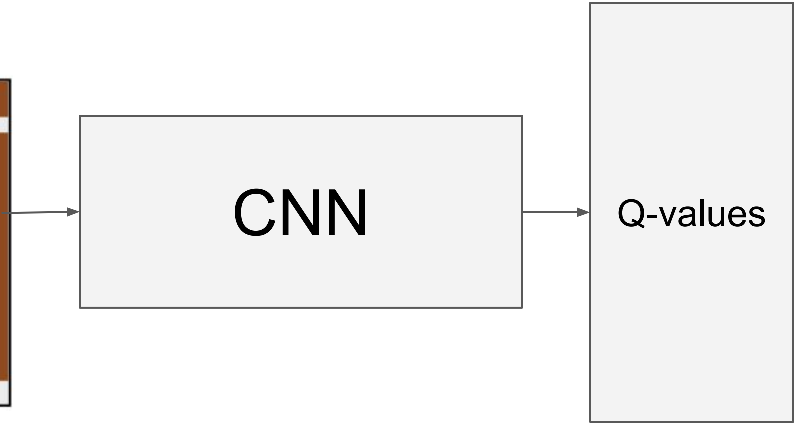
$$\Theta^- = (1-\alpha)\Theta^- + \alpha\Theta$$

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

Outro

- Q-learning can be applied to solve the environments with continuous states as well
- But we haven’t said anything about continuous **actions**!
- Remember what Q(s, a) and V(s) functions do, we will need them even outside the Q-learning
- Remember about the i.i.d. property!
    - RL algorithms usually affect the environment, hence the distribution of the data they are trained on

Backlog

Problem of overestimation

We use “max” operator to compute the target

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

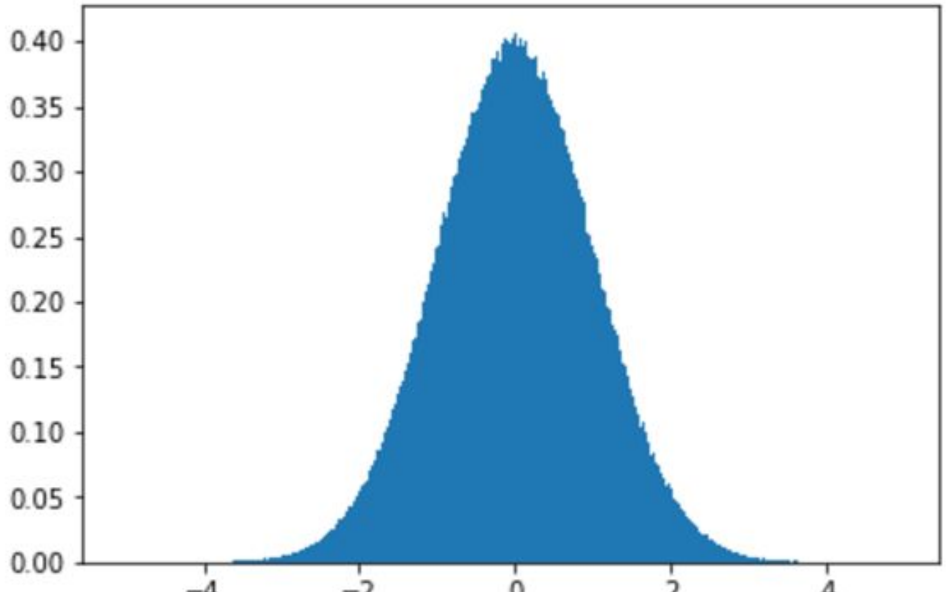
We have a problem

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

Problem of overestimation

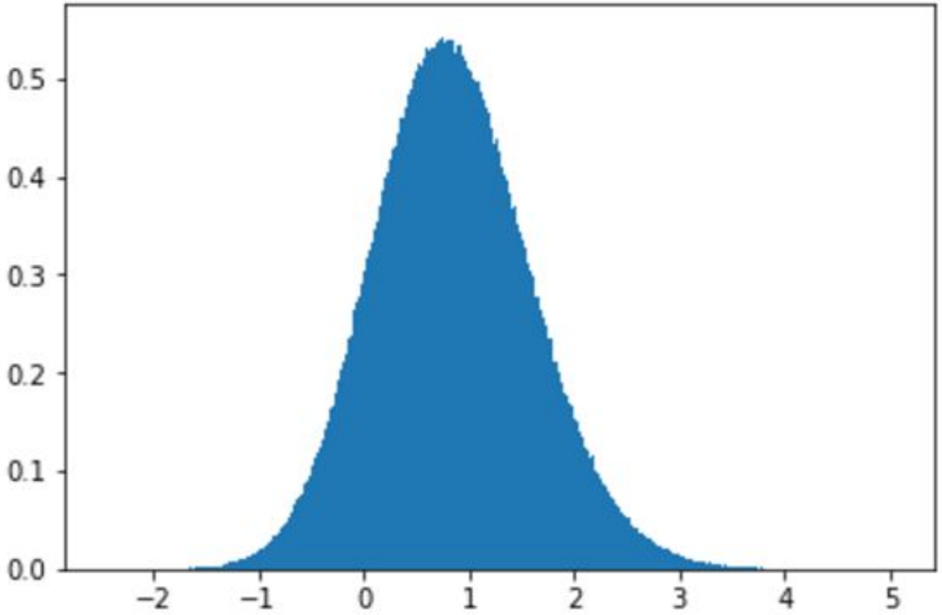
Normal distribution
$3*10^6$ samples

mean: ~0.0004



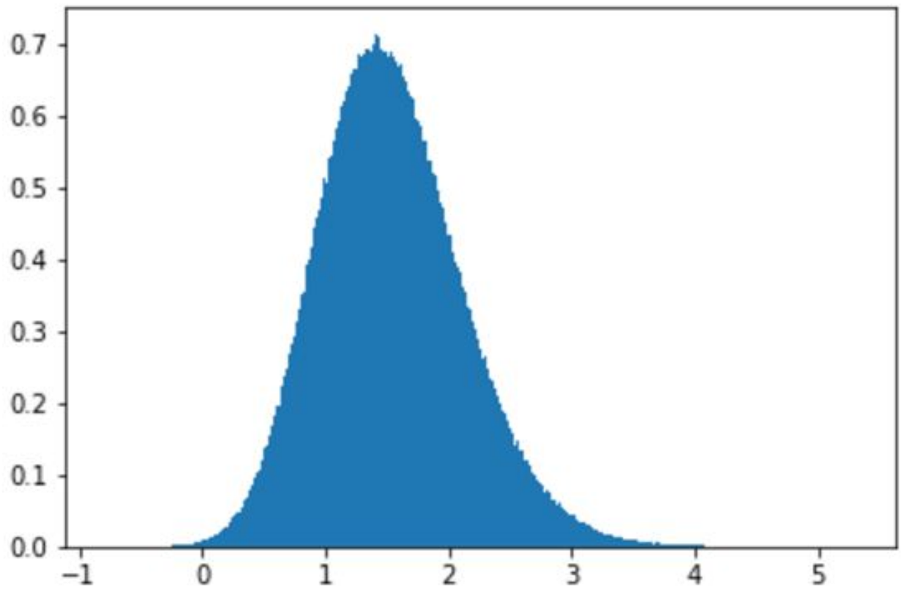
Problem of overestimation

Normal distribution
$3*10^6$ x 3 samples
Then take maximum of every tuple
mean: ~0.8467

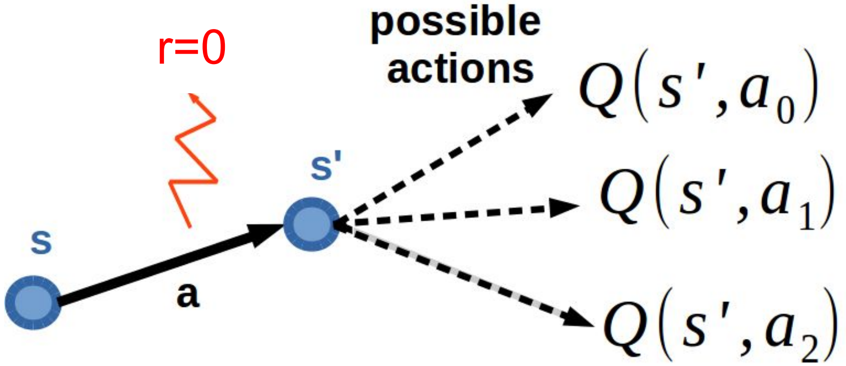


Problem of overestimation

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



Problem of overestimation

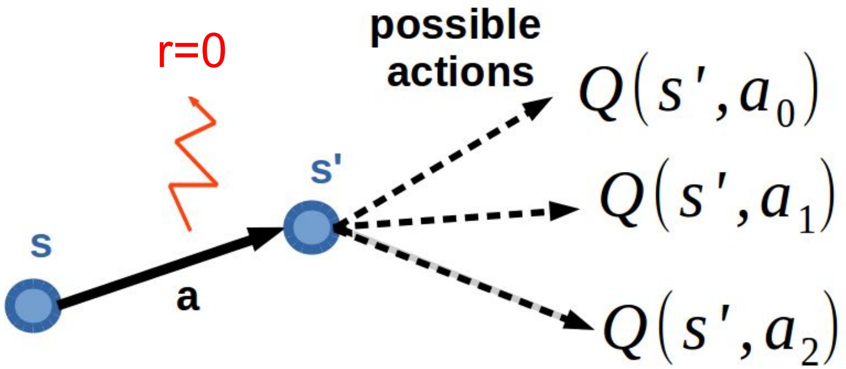


Suppose true $Q(s', a')$ are equal to **0** for all a'

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

So $Q(s, a)$ should be equal to **0**

Problem of overestimation



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

we will have overestimated $Q(s,a)$ > **0** because

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

Double Q-learning

(NIPS 2010)

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

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

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!

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

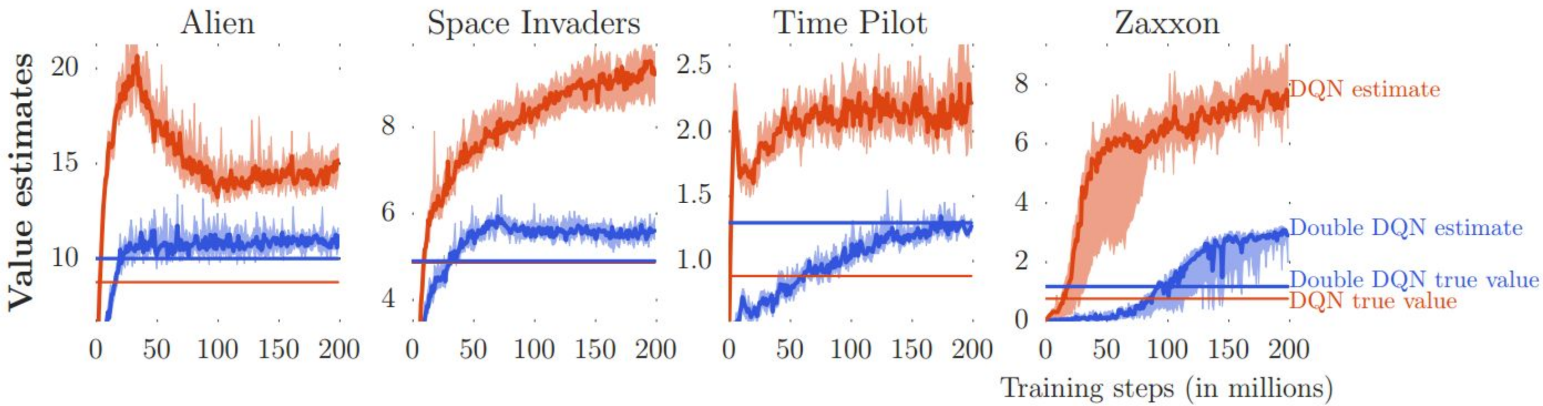
Can we combine this algorithm with DQN?

Deep RL 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%

Double Q-learning visualization

Experience Replay

State	Action	Reward	Next state
s_0	a_0	0	s_1
s_1	a_1	0	s_2
...	...	...	...
s_(n-1)	a_(n-1)	0	s_n
s_n	**a_n**	**100**	**s_(n+1)**
s_(n+1)	a_(n+1)	0	s_(n+2)
...	...	...	...

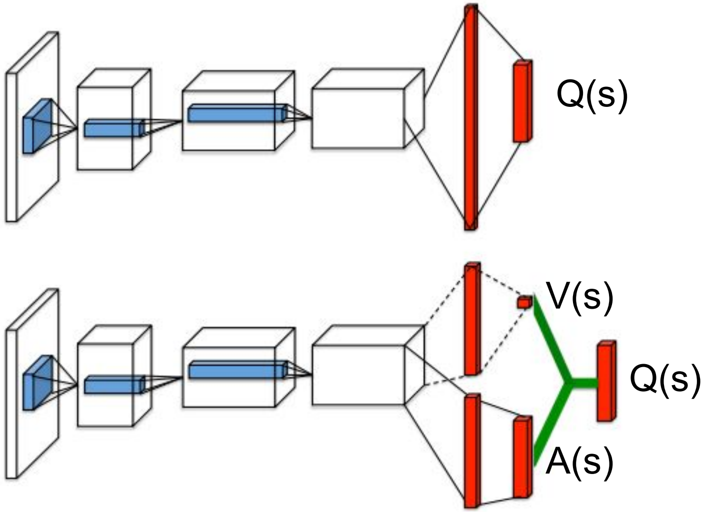
Dueling Network Architectures for Deep RL

(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)



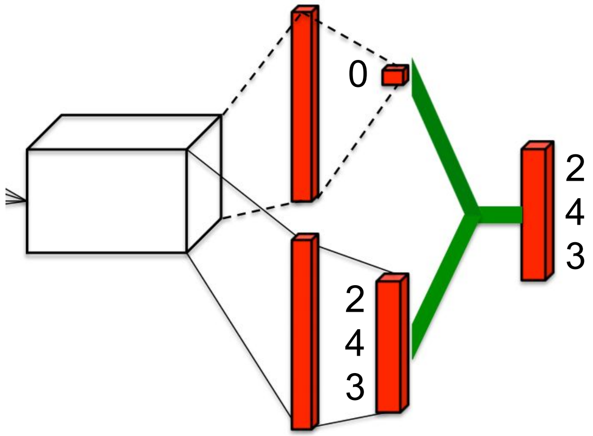
Do you see the problem?

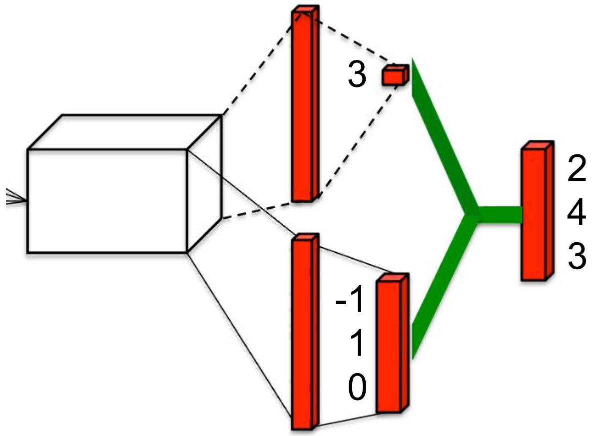
Dueling Network Architectures for Deep RL

(2016, Deepmind)

Here is one extra freedom degree!

Example:





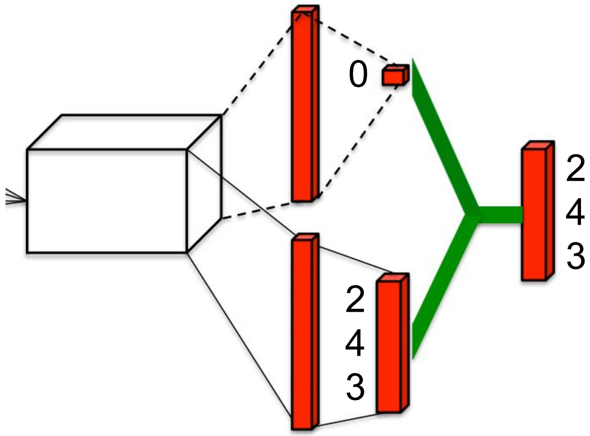
Which one is good?

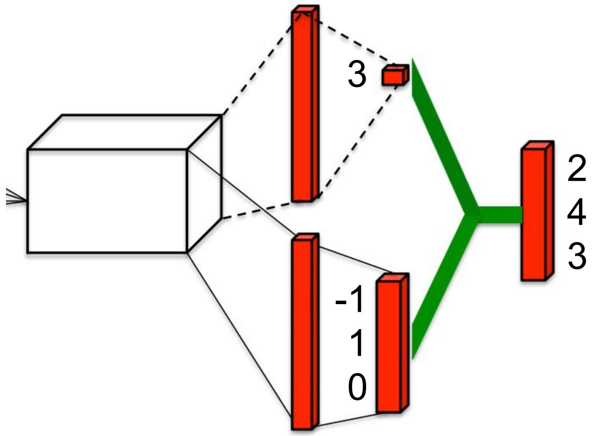
Dueling Network Architectures for Deep RL

(2016, Deepmind)

Here is one extra freedom degree!

Example:





No one

Dueling Network Architectures for Deep RL

(2016, Deepmind)

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

So the **Q-function** is computed as:

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

Dueling Network Architectures for Deep RL

(2016, Deepmind)

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

So the **Q-function** is computed as:

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

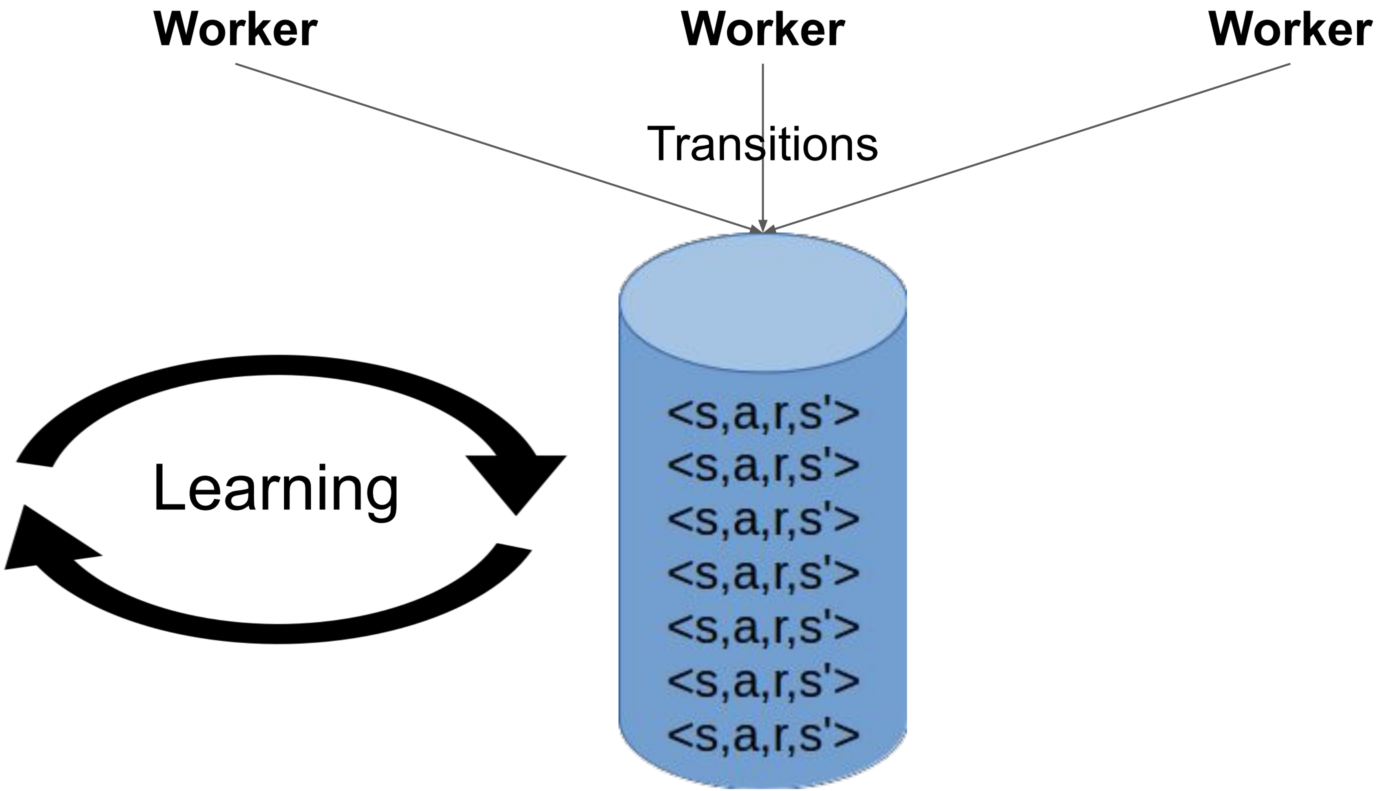
Authors of this papers also introduced this way to compute Q-values:

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

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

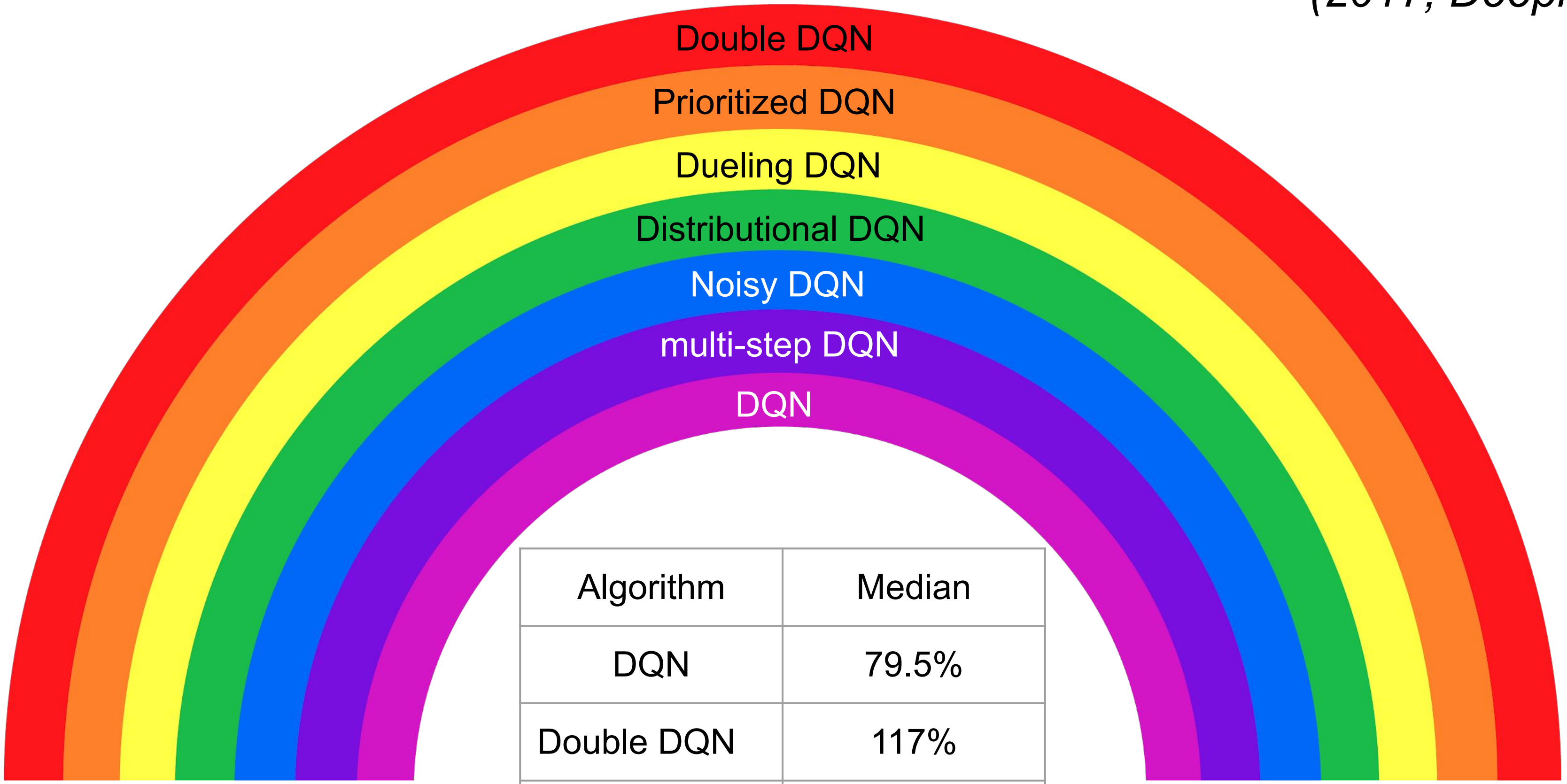
Asynchronous Methods for Deep RL

(2016, Deepmind)



Rainbow

(2017, Deepmind)



Algorithm	Median
DQN	79.5%
Double DQN	117%
Rainbow	223%

R2D2

(2018, Deepmind)

LSTM

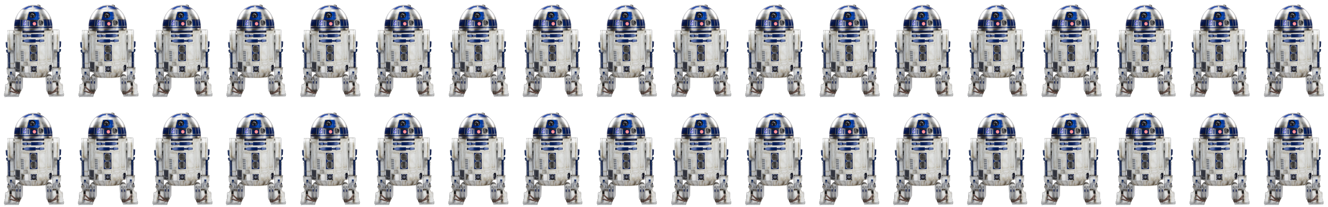
Distributed Prioritized Experience Replay

n-step DQN

Reward re-scaling

Double DQN

Dueling DQN



Median performance: **1920%** of human performance!

Prioritized Experience Replay

(2016, Deepmind)

Idea: sample transitions from xp-replay cleverly

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?

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

So we sample using $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ and multiply error by w_i

Prioritized Experience Replay

(2016, Deepmind)

Additional details

We also normalize weights by $1/\max_i w_i$ (here is no mathematical reason)

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