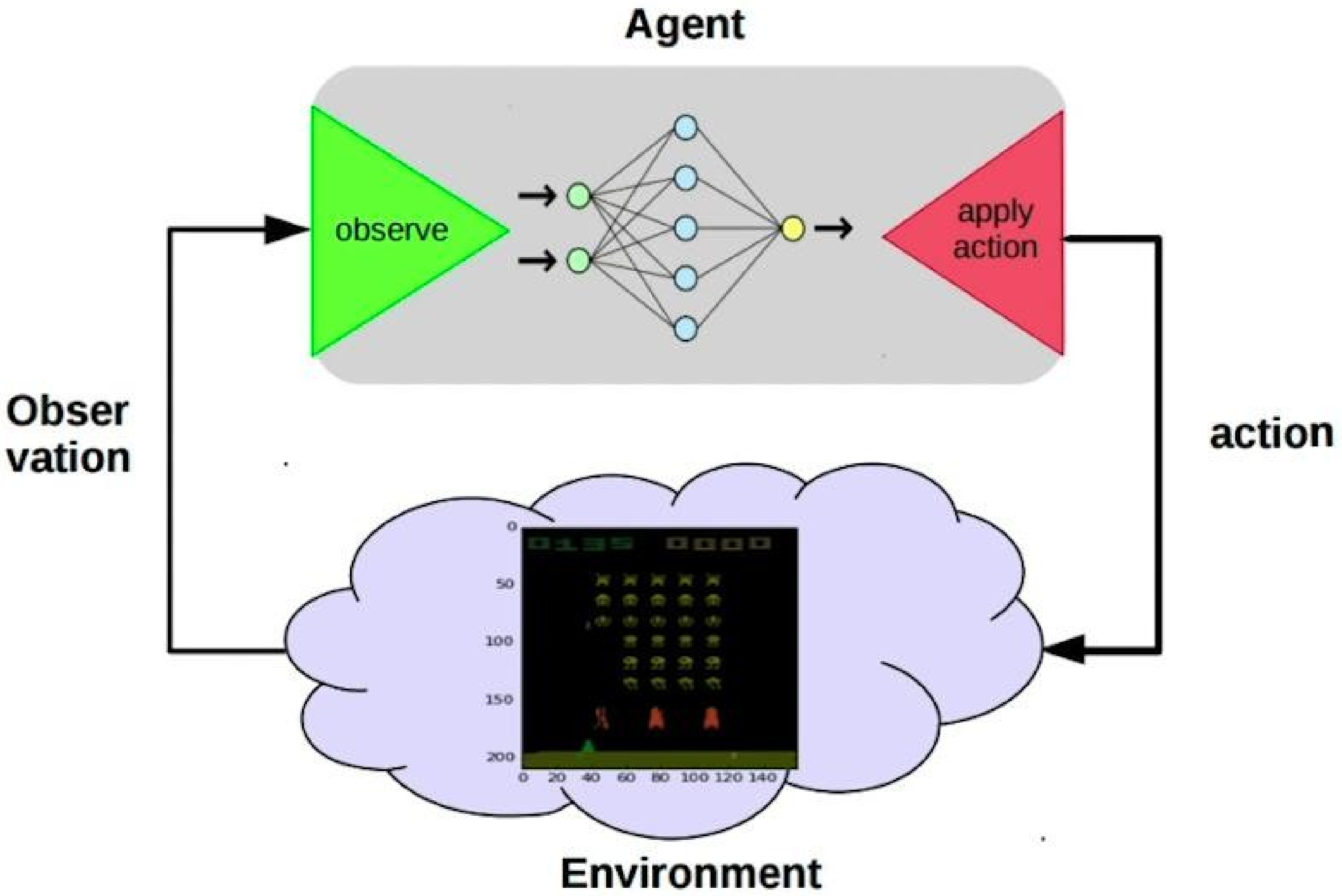
Reinforcement Learning

Episode 4

Deep Reinforcement Learning

MDP


Agent
observe
apply
action
Obser
vation
action
Environment

Basic Definitions

Basic Definitions

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

Basic Definitions

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

Basic Definitions

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \mathbb{E}_{a_t \sim \pi}[Q^\pi(s_t, a_t)]$$

Basic Definitions

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \mathbb{E}_{a_t \sim \pi}[Q^\pi(s_t, a_t)]$$

Recurrent Relations

Basic Definitions

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \mathbb{E}_{a_t \sim \pi}[Q^\pi(s_t, a_t)]$$

Recurrent Relations

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

Basic Definitions

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \mathbb{E}_{a_t \sim \pi}[Q^\pi(s_t, a_t)]$$

Recurrent Relations

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

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

Optimal Policy

Optimal Policy

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

Optimal Policy

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

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

Optimal Policy

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

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

Bellman Optimality Equation

Optimal Policy

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

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

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

Q-learning

Q-learning

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

Q-learning

Bellman Optimality Equation

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

Training Step

Q-learning

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

Training Step

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

Q-learning

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

Training Step

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

Q-learning as MSE optimization

Q-learning

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

Training Step

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

Q-learning as MSE optimization

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

Q-learning

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max\nolimits_{a'} Q^*(s_{t+1}, a')]$$

Training Step

$$\mathrm{Q}(\mathrm{s}_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max\nolimits_{a'} Q(s_{t+1}, a'))$$

Q-learning as MSE optimization

$$\mathrm{Q}(\mathrm{s}_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max\nolimits_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

$$\mathrm{L} = (\mathrm{r}_t + \gamma \max\nolimits_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))^2$$

Q-learning

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

Training Step

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

Q-learning as MSE optimization

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

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

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

Q-learning

Bellman Optimality Equation

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

Training Step

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

Q-learning as MSE optimization

$$\mathrm{Q}(\mathrm{s}_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\boxed{(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))}$$

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

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

Q-learning

Bellman Optimality Equation

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

Training Step

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

Q-learning as MSE optimization

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

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

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

What’s wrong here?

Q-learning

Bellman Optimality Equation

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

Training Step

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

Q-learning as MSE optimization

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

Const

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

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

What's wrong here?

Q-learning

Bellman Optimality Equation

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

Training Step

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

Q-learning as MSE optimization

$$\mathrm{Q}(\mathrm{s}_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \boxed{(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))}$$

Const

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

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

Stop gradient!

Q-learning

Bellman Optimality Equation

$$\mathrm{Q}^*(s_t, a) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')]$$

Training Step

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

Q-learning as MSE optimization

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

$$\text{Gradient descent on } \ \mathrm{L} = (\mathrm{r}_t + \gamma \max_{a'} Q^-(s_{t+1}, a') - Q(s_t, a_t))^2$$

“Real” world

“Real” world

“Real” world

How many states are there?

“Real” world

How many states are there?

$$\#S = 2^{210 \cdot 168 \cdot 3 \cdot 8}$$

“Real” world

How many states are there?

$$\#S = 2^{210 \cdot 168 \cdot 3 \cdot 8}$$

In fact, only 256

Problem:

Problem:

State space is usually large,
sometimes continuous

Problem:

State space is usually large,
sometimes continuous

Two solutions:

Problem:

State space is usually large,
sometimes continuous

Two solutions:

- Binarize state space (last week)

Problem:

State space is usually large,
sometimes continuous

Two solutions:

- Binarize state space (last week)
- Approximate agent with a function (Crossentropy method)

Problem:

State space is usually large, sometimes continuous

Two solutions:

- Binarize state space
- Approximate agent with a function

Problem:

State space is usually large,
sometimes continuous

Two solutions:

- Binarize state space ← Too many bins or handcrafted features

- Approximate agent with a function

Problem:

State space is usually large,
sometimes continuous

Two solutions:

- Binarize state space ← Too many bins or handcrafted features

- Approximate agent with a function ← Let's pick this one

From Tables to Approximations

Before:

Now:

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function
- For example, a linear model over state features

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function

- For example, a linear model over state features

$$\text{Gradient descent on } \mathrm{L} = (\mathrm{r}_t + \gamma \max_{a'} Q^{-}(s_{t+1}, a') - Q(s_t, a_t))^2$$

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function
- For example, a linear model over state features

$$\text{Gradient descent on } \ \mathrm{L} = (\mathrm{r}_t + \gamma \max_{a'} Q^{-}(s_{t+1}, a') - Q(s_t, a_t))^2$$

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function

- For example, a linear model over state features

$$\text{Gradient descent on } \mathrm{L} = (\mathrm{r}_t + \gamma \max_{a'} Q^{-}(s_{t+1}, a') - Q(s_t, a_t))^2$$

Is Q-learning a classification or a regression task?

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function

- For example, a linear model over state features

$$\text{Gradient descent on } \ \mathrm{L} = (\mathrm{r}_t + \gamma \max_{a'} Q^{-}(s_{t+1}, a') - Q(s_t, a_t))^2$$

Is Q-learning a classification or a regression task?

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function

- For example, a linear model over state features

$$\text{Gradient descent on } \mathrm{L} = (\mathrm{r}_t + \gamma \max_{a'} Q^{-}(s_{t+1}, a') - Q(s_t, a_t))^2$$

Note: Formally a table can be used as a functional approximator.

From Tables to Approximations

Before:

- For all states and actions remember Q(s, a)

Now:

- Approximate Q(s, a) with some function

- For example, a linear model over state features

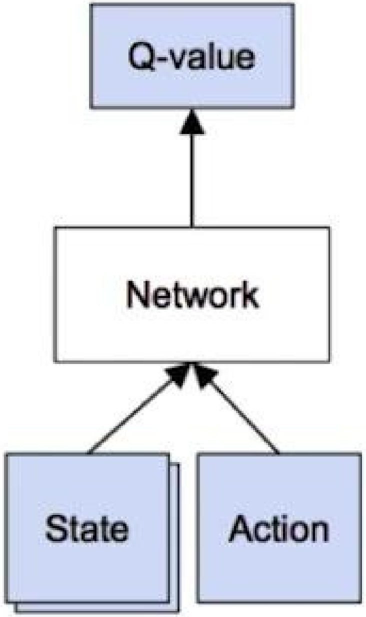
$$\text{Gradient descent on } \mathrm{L} = (\mathrm{r}_t + \gamma \max_{a'} Q^{-}(s_{t+1}, a') - Q(s_t, a_t))^2$$

Note: Formally a table can be used as a functional approximator.

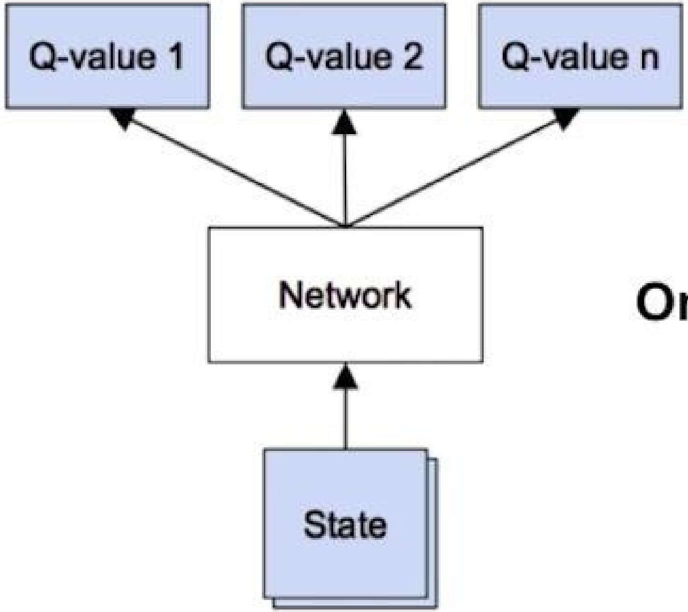
This can be helpful for theoretical analysis.

Possible architectures

Continuous control or large number of actions



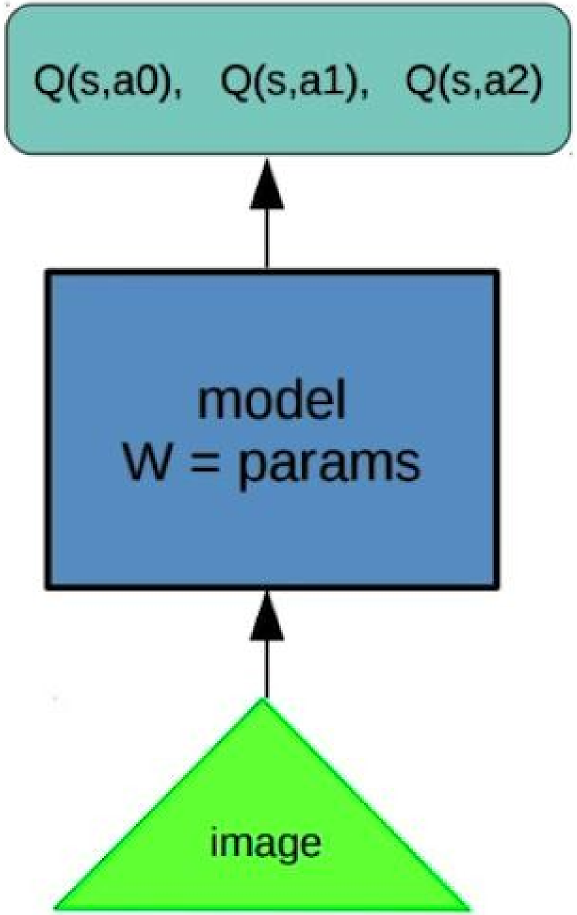
Given **(s,a)**
Predict Q(s,a)



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

One pass for all actions

Approximate Q-learning



Q-values:

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

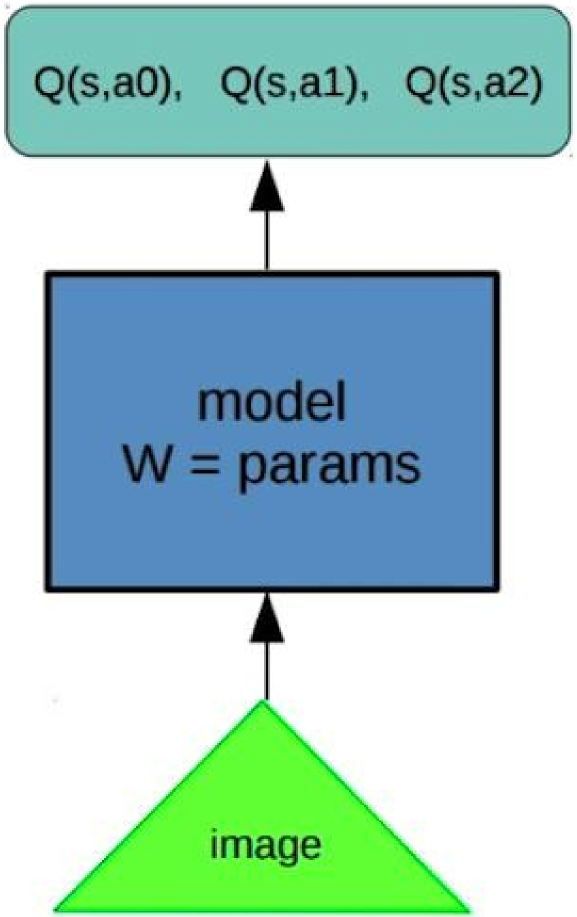
Objective:

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

Gradient step:

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

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:

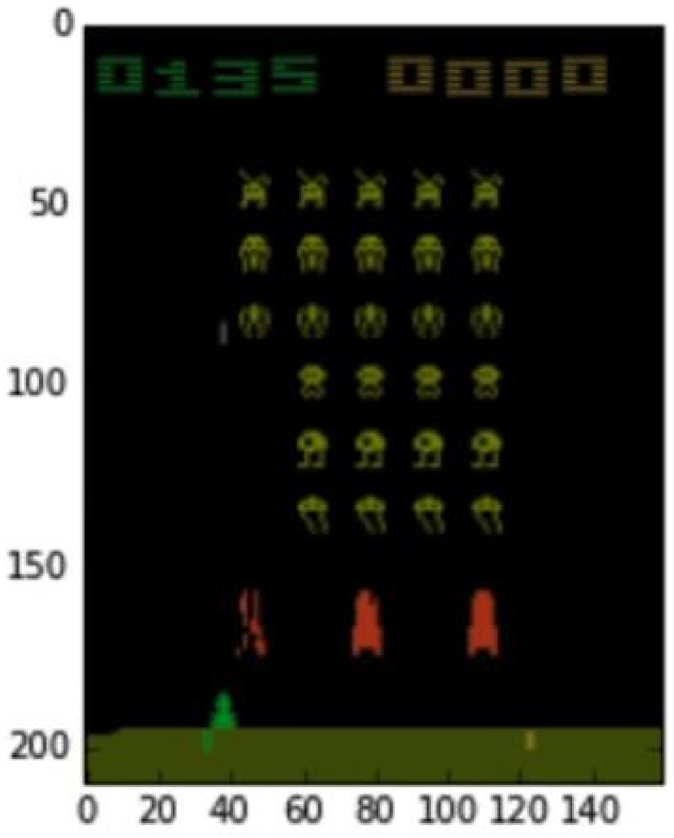
$$\hat{Q}(s_t, a_t) = r + \gamma \cdot max_{a'} Q(s_{t+1}, a')$$

SARSA:

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

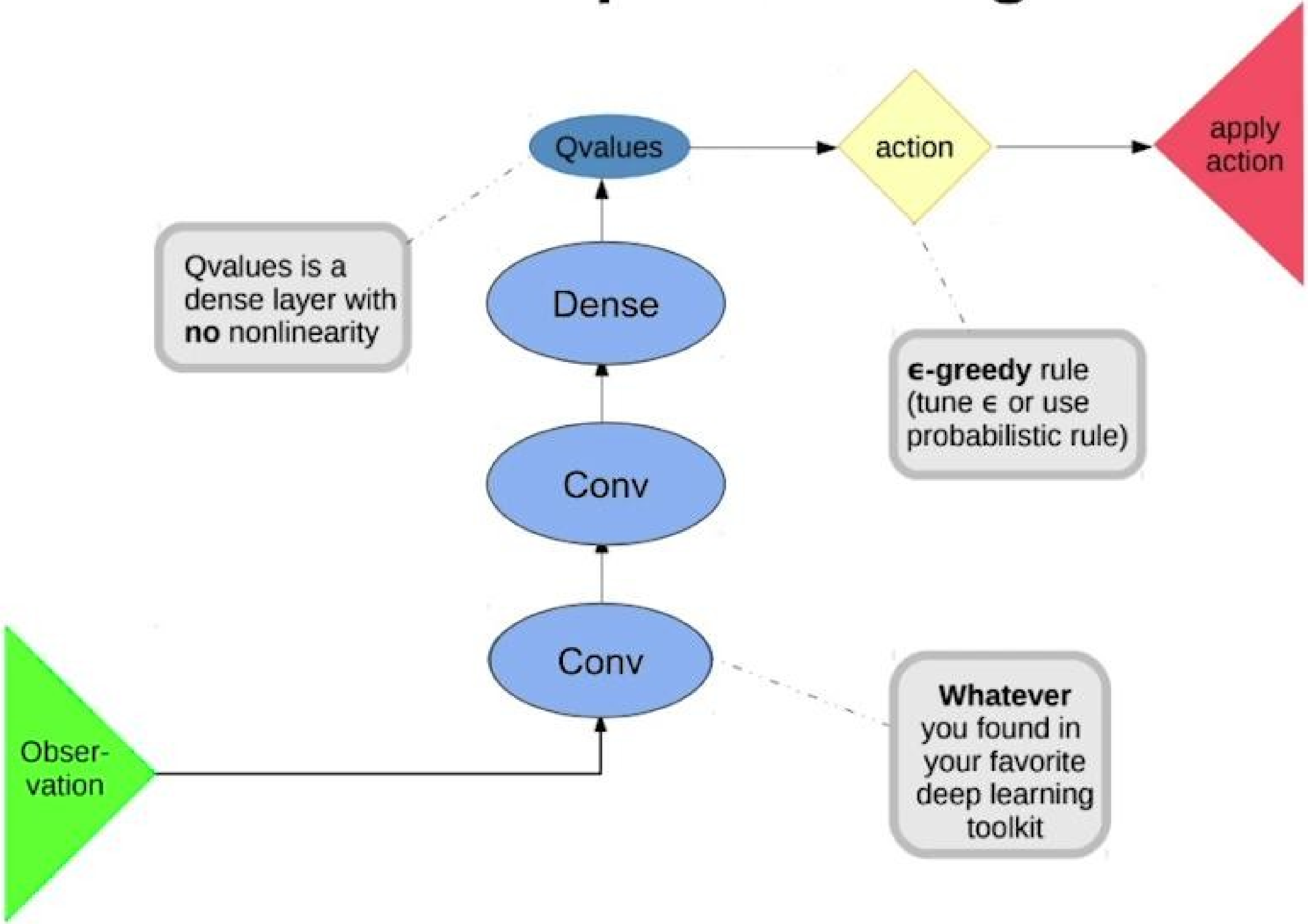
Expected Value SARSA:

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

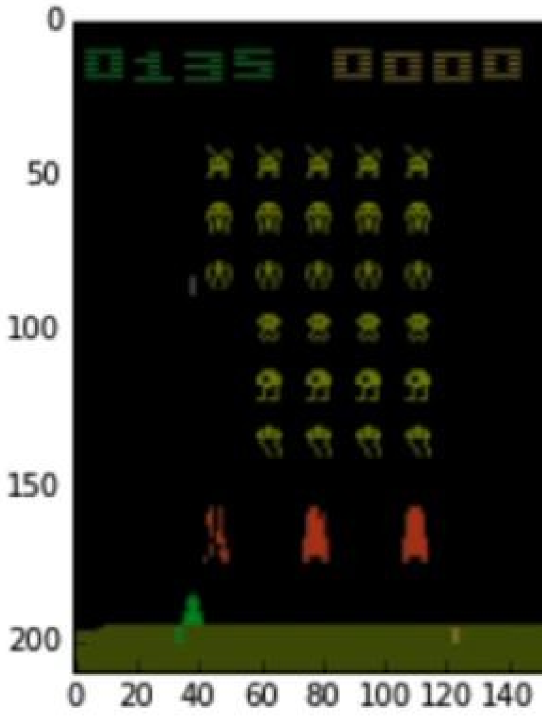


What kind of network digests images well?

Basic deep Q-learning




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

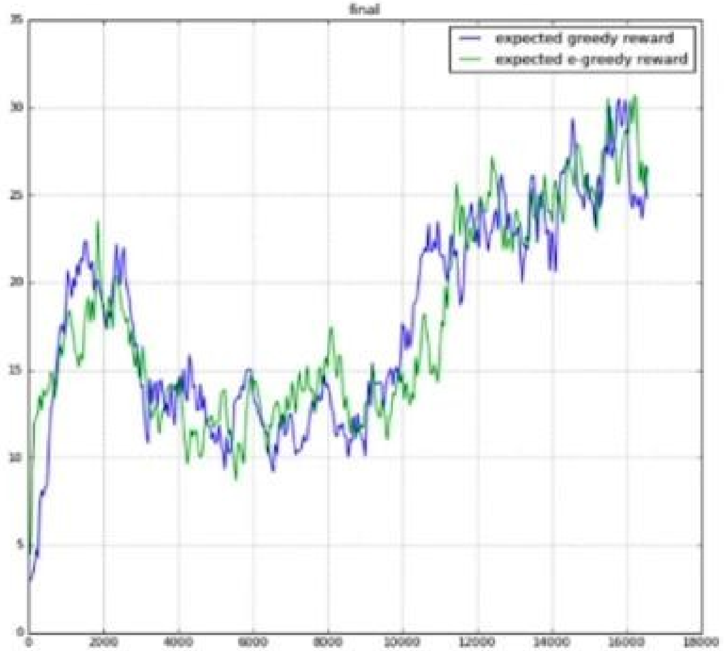


How bad it is if agent spends

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

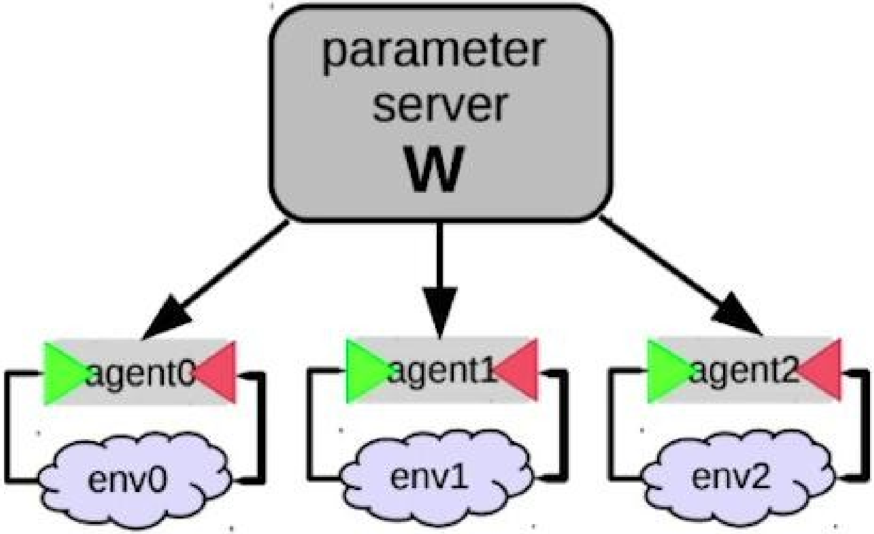
Problem

- 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 agent trick

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



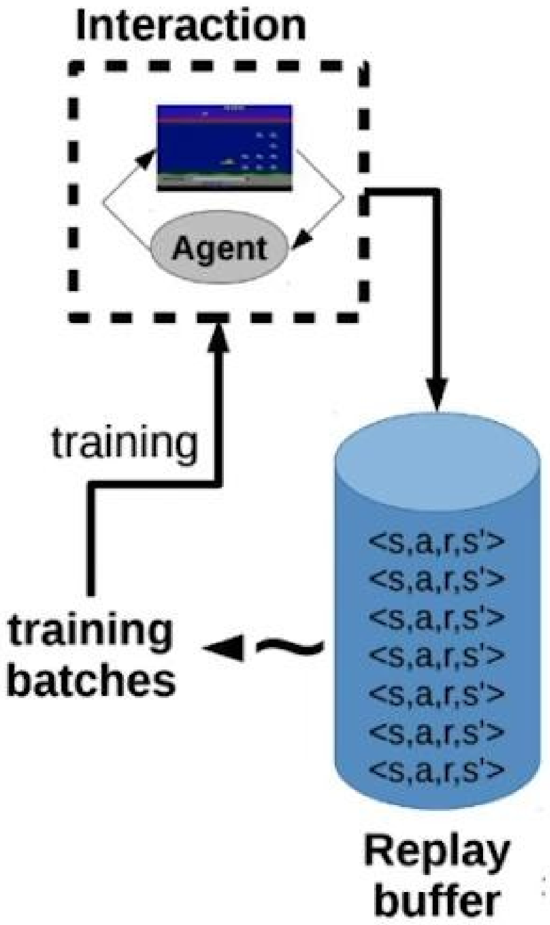
- Chances are, they will be exploring different parts of the environment,
- More stable training,
- Requires a lot of interaction

Question: your agent is a real robot car. Any problems?

Experience replay

Idea: store several past interactions
<s,a,r,s'>
Train on random subsamples

- Closer to i.i.d
  pool contains several sessions
- Older interactions were obtained
  under weaker policy



Experience Replay

Experience Replay

You approximate $Q(s, a)$ with a neural network

Experience Replay

You approximate Q(s, a) with a neural network

You use experience replay when training

Experience Replay

You approximate $Q(s, a)$ with a neural network

You use experience replay when training

Question: which of these algorithms will fail?

Experience Replay

You approximate Q(s, a) with a neural network

You use experience replay when training

Question: which of these algorithms will fail?

Q-Learning

Experience Replay

You approximate Q(s, a) with a neural network

You use experience replay when training

Question: which of these algorithms will fail?

Q-Learning

SARSA

Experience Replay

You approximate Q(s, a) with a neural network

You use experience replay when training

Question: which of these algorithms will fail?

Q-Learning

CEM

SARSA

Experience Replay

You approximate Q(s, a) with a neural network

You use experience replay when training

Question: which of these algorithms will fail?

Q-Learning

CEM

SARSA

Expected Values 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 him

Question: which of these algorithms will fail?

Q-Learning	CEM
SARSA	Expected Values 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 him

Question: which of these algorithms will fail?

Q-Learning

CEM

SARSA

Expected Values SARSA

When training with on-policy methods,

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

Left or right?

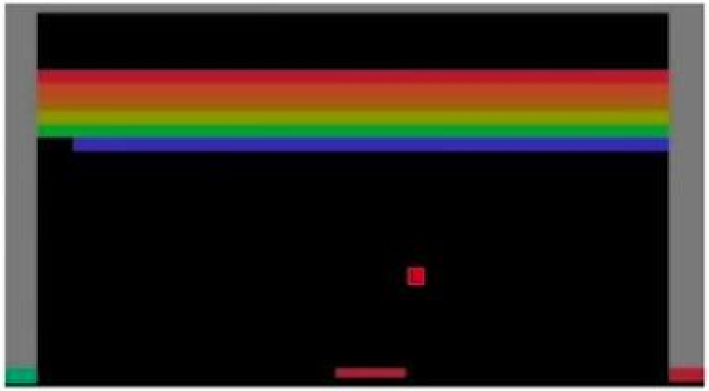
N-gram trick

Idea:

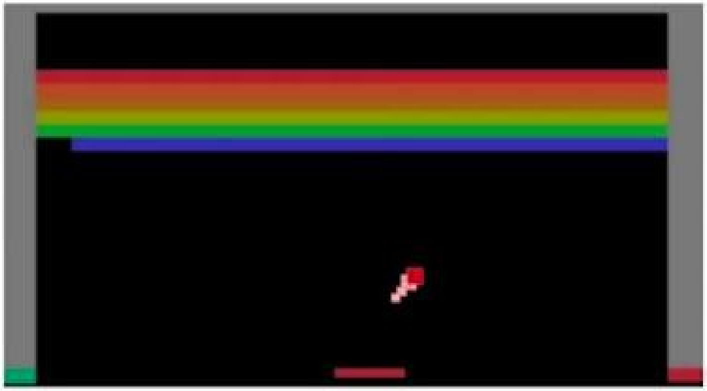
$$s_t \neq o(s_t)$$

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

e.g. ball movement in breakout



· One frame



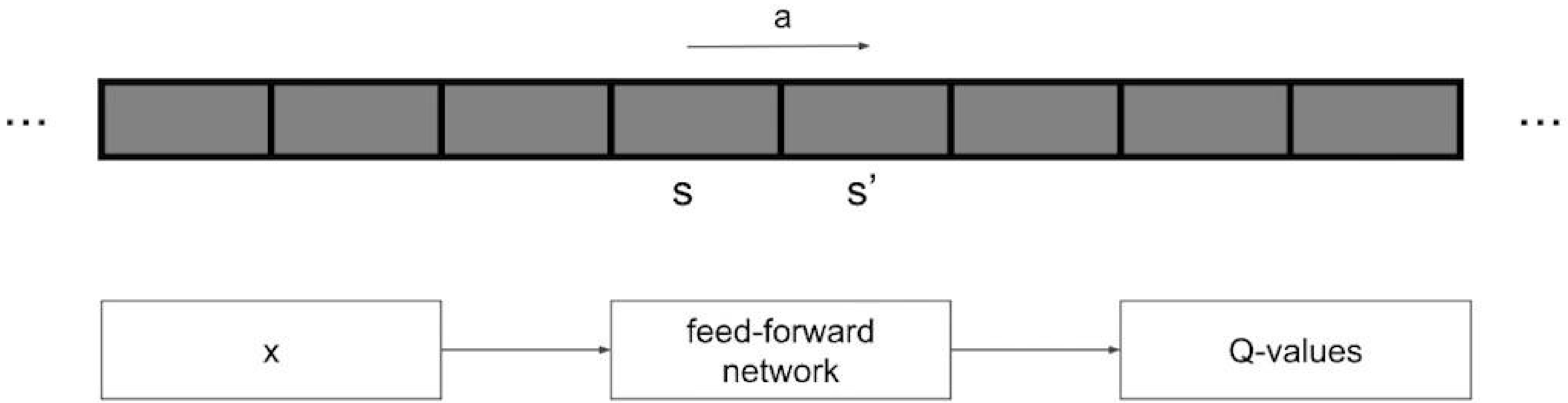
· Several frames

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:

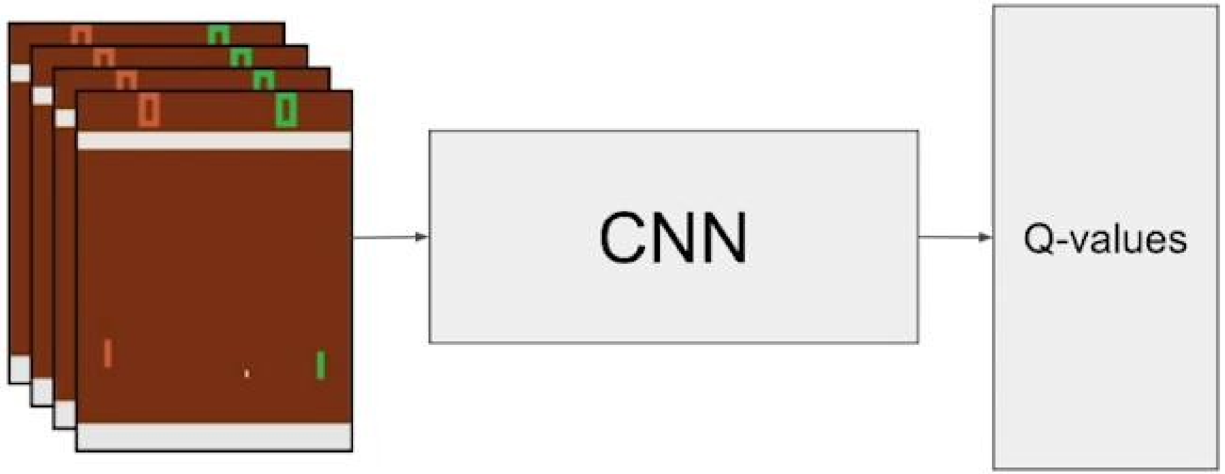
Update Θ^- every **n** steps and set its weights as Θ

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

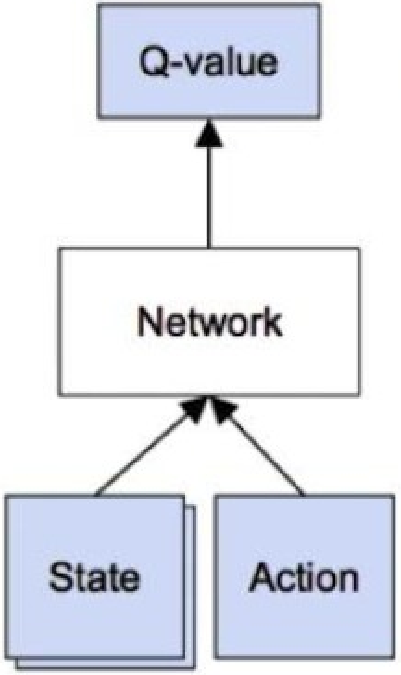
10^6 last transitions

Update weights using:

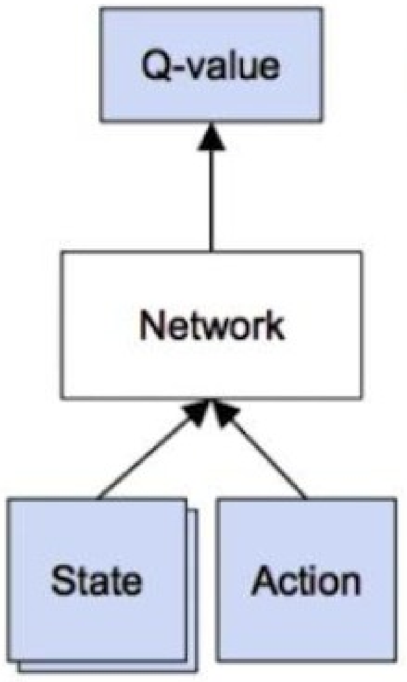
$$L(\Theta) = E_{s\sim S, a\sim A}[(Q(s,a,\Theta) - (r + \gamma \max_{a'} Q(s',a',\Theta^-)))^2]$$

Update Θ^- every 5000 train steps

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)

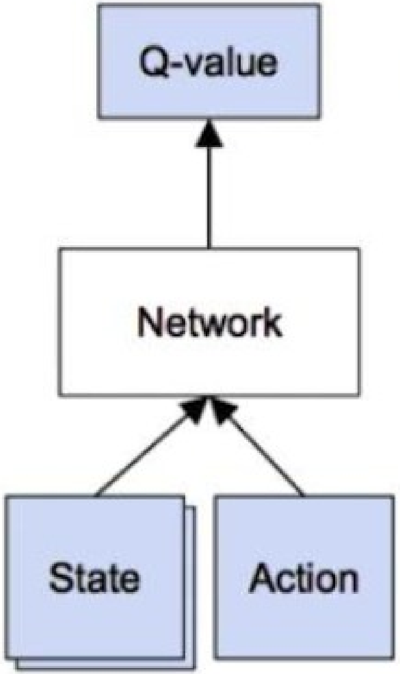


CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Q-network takes (state, action) as input and outputs 1 Q-value and is called Critic.

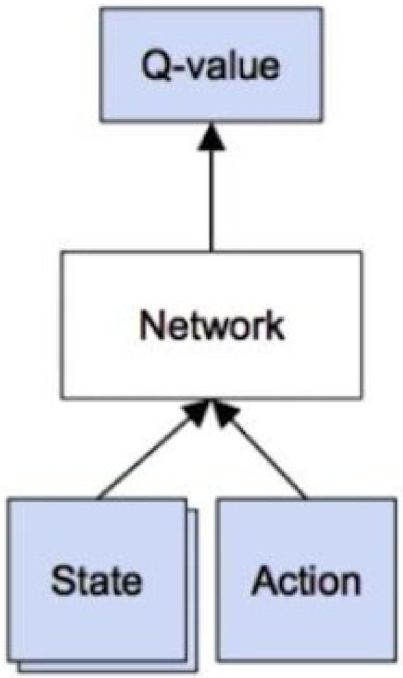
CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Q-network takes (state, action) as input and outputs 1 Q-value and is called Critic.

+1 more network, which selects an action, it is called (Actor)

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)

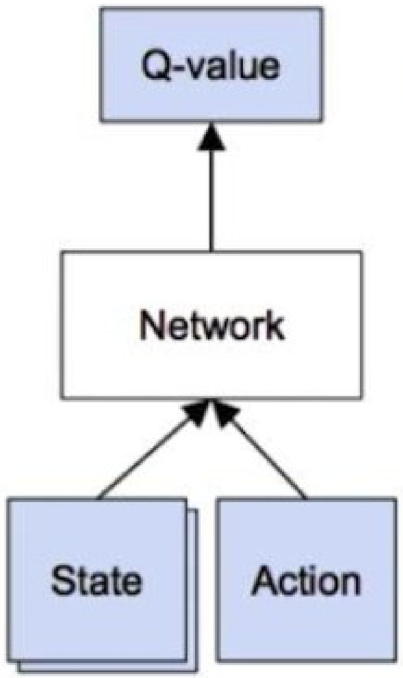


Q-network takes (state, action) as input and outputs 1 Q-value and is called Critic.

+1 more network, which selects an action, it is called (Actor)

Both Actor and Critic have a target network, so now it’s 4 networks total

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



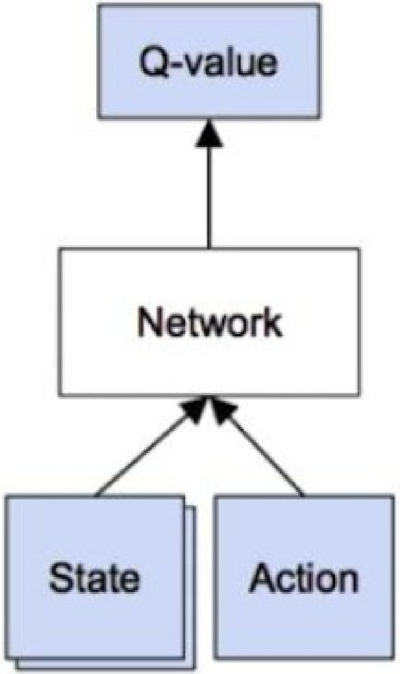
Q-network takes (state, action) as input and outputs 1 Q-value and is called Critic.

+1 more network, which selects an action, it is called (Actor)

Both Actor and Critic have a target network, so now it’s 4 networks total

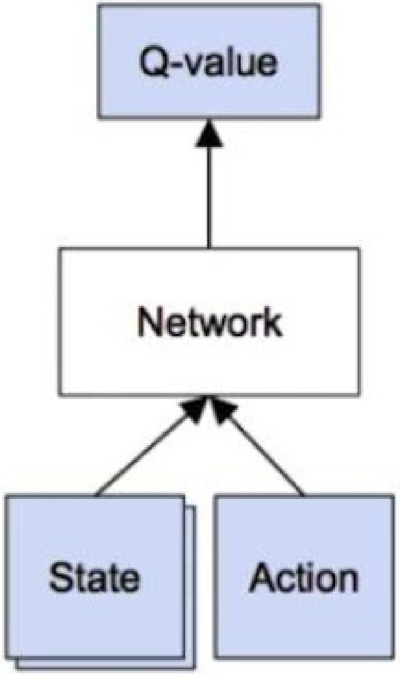
Environment interaction step: action chosen by Critic + noise

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

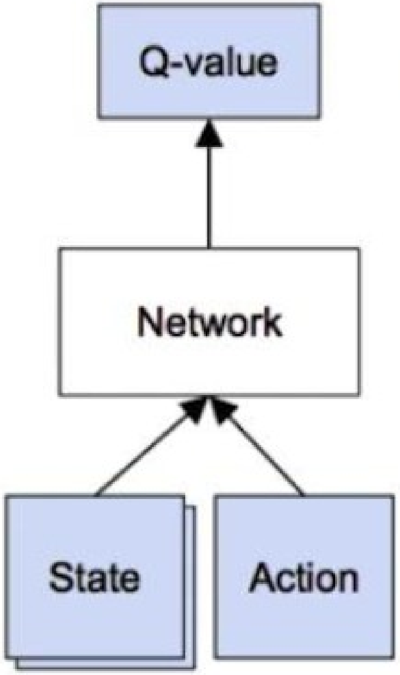
CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Training step

Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)

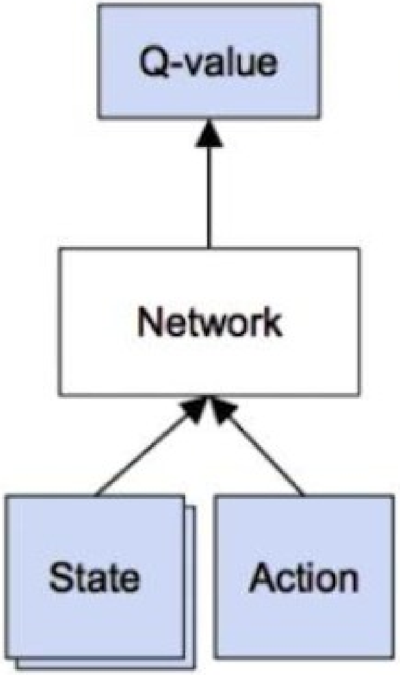


Training step

1. Use the Critic to select the optimal action, using target networks:

Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



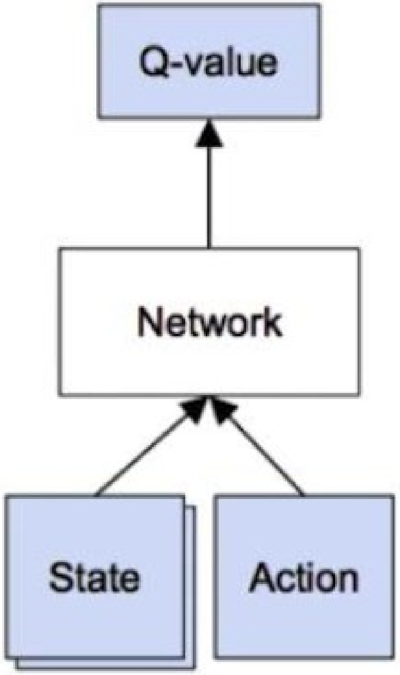
Training step

1. Use the Critic to select the optimal action, using target networks:

$$\mathrm{Q}_{target} = r + \gamma Q'(s_{next}, \mu'(s_{next}))$$

Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Training step

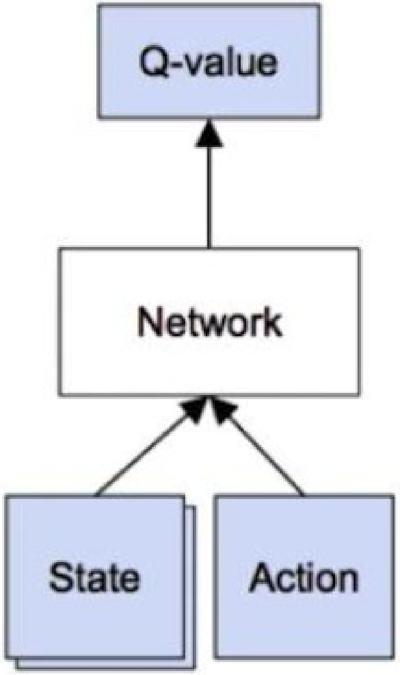
1. Use the Critic to select the optimal action, using target networks:

$$\mathrm{Q}_{target} = r + \gamma Q'(s_{next}, \mu'(s_{next}))$$

2. Make a gradient descent step for Critic by the regular DQN loss:

Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Training step

1. Use the Critic to select the optimal action, using target networks:

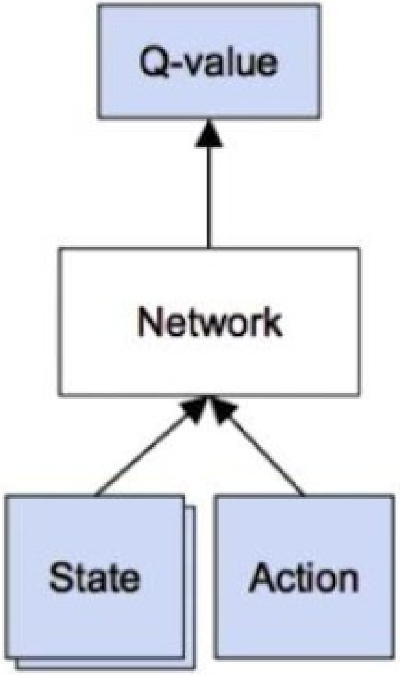
$$\mathrm{Q}_{target} = r + \gamma Q'(s_{next}, \mu'(s_{next}))$$

2. Make a gradient descent step for Critic by the regular DQN loss:

$$\mathrm{L}_{Critic} = (Q(s,a) - Q^{-}_{target})^2$$

Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Training step

1. Use the Critic to select the optimal action, using target networks:

$$\mathrm{Q}_{target} = r + \gamma Q'(s_{next}, \mu'(s_{next}))$$

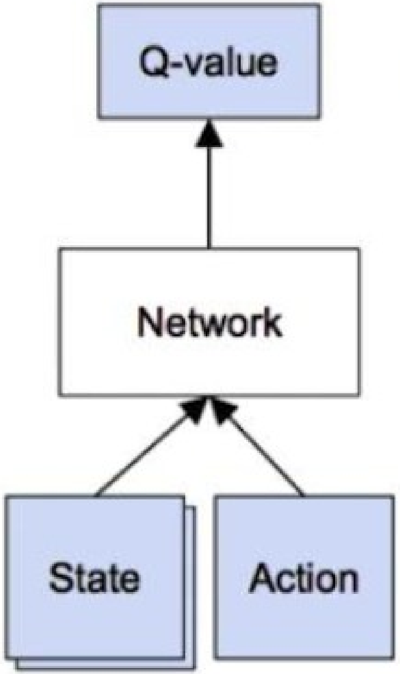
2. Make a gradient descent step for Critic by the regular DQN loss:

$$\mathrm{L}_{Critic} = (Q(s,a) - Q^{-}_{target})^2$$

3. Make a gradient descent step for Actor by the by the loss maximizing Q:

Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING (DDPG)



Training step

1. Use the Critic to select the optimal action, using target networks:

$$\mathrm{Q}_{target} = r + \gamma Q'(s_{next}, \mu'(s_{next}))$$

2. Make a gradient descent step for Critic by the regular DQN loss:

$$\mathrm{L}_{Critic} = (Q(s,a) - Q^{-}_{target})^2$$

3. Make a gradient descent step for Actor by the by the loss maximizing Q:

$$\mathrm{L}_{Actor} = -Q(s, \mu(s))$$

Notation. Actor: $\boldsymbol{\mu}$, Critic: $\boldsymbol{Q}$

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)

$$\mathbb{E} \max_{i=1..n} \xi_i \geq \max_{i=1..n} \mathbb{E}\xi_i$$

For any set of random variables $\{\xi_i\}_{i=1}^n$.

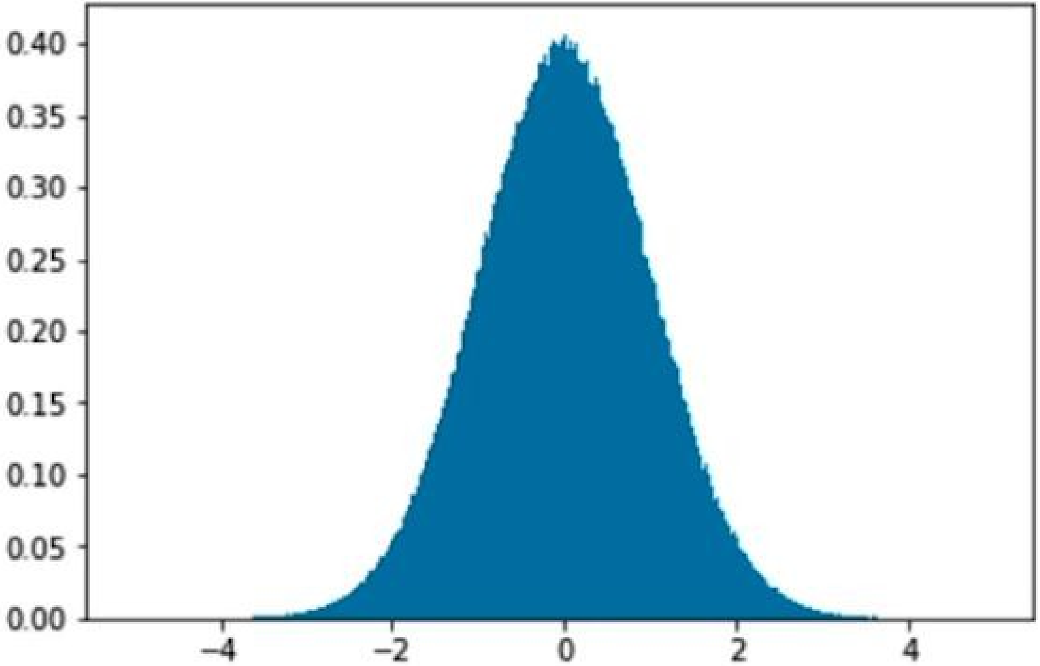
Equality is only reached when one of them (let it be ξ_1) is greater than all the others with $\mathbb{P} = 1$, i.e.

$$\mathbb{E}(\xi_1) = \mathbb{E} \max_{i=1..n} \xi_i \iff \mathbb{P}\{\xi_1 = \max_{i=1..n} \xi_i\} = 1$$

Problem of overestimation

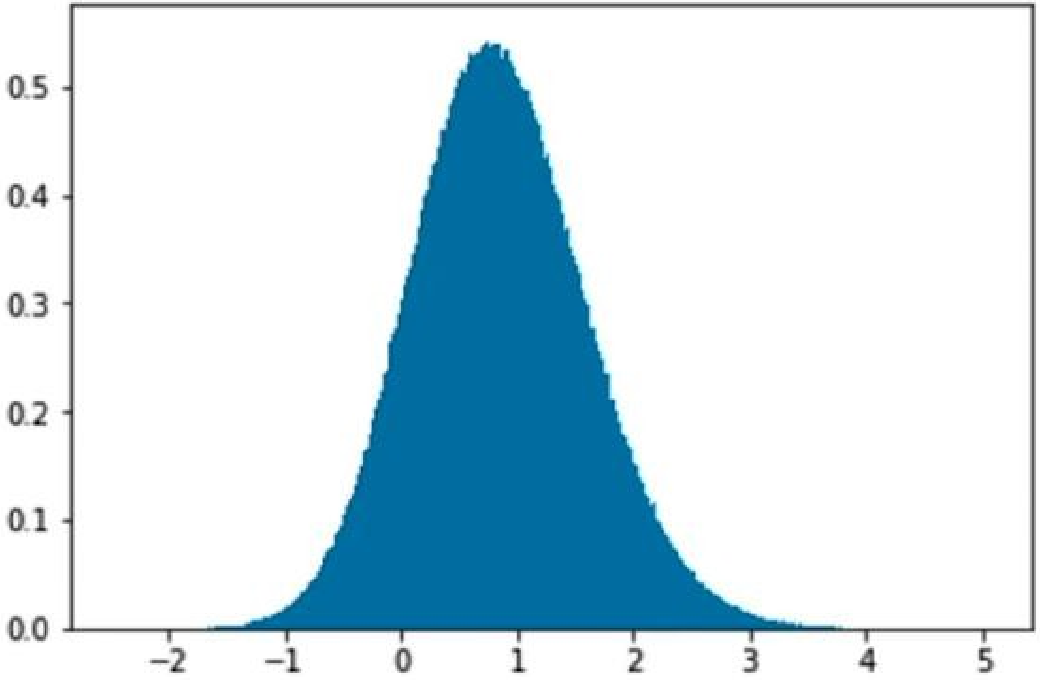
Normal distribution
$3*10^6$ samples

mean: ~0.0004



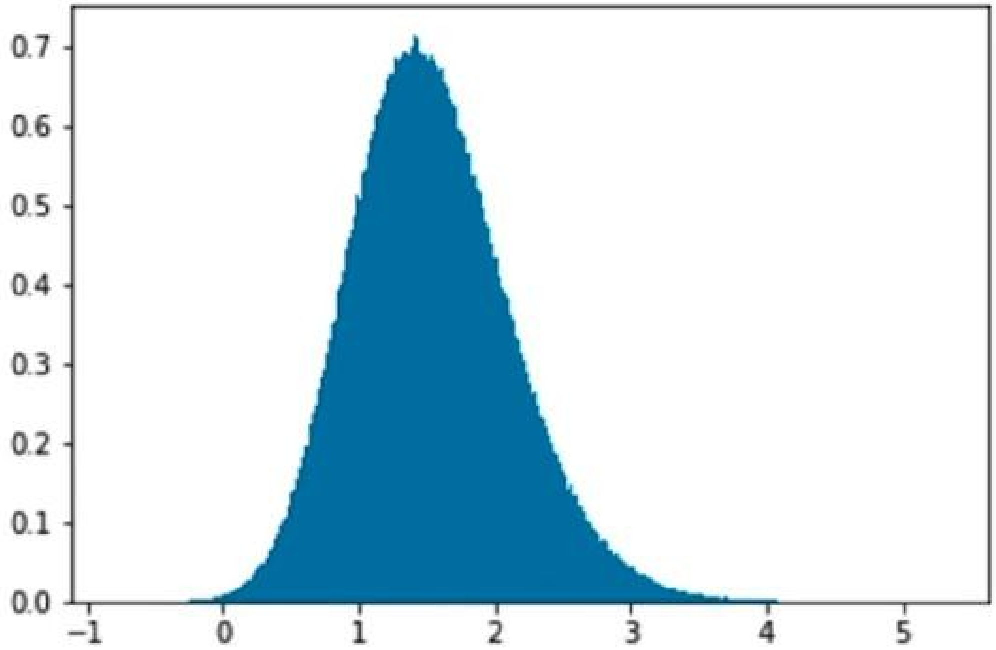
Problem of overestimation

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

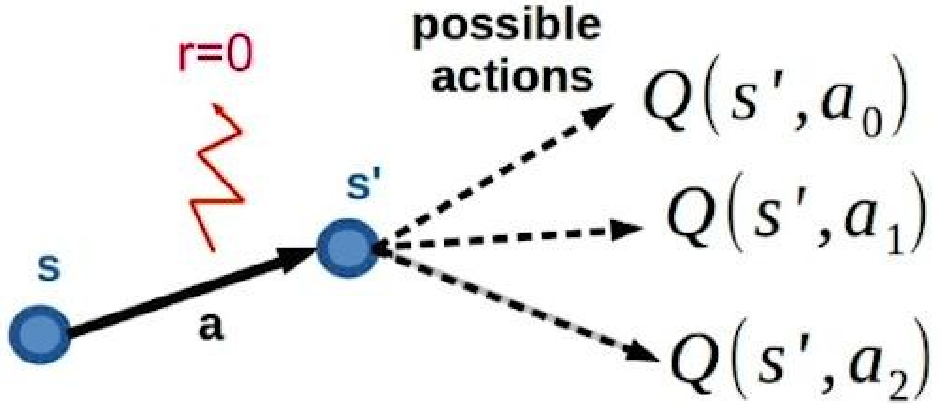


Problem of overestimation

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



Problem of overestimation

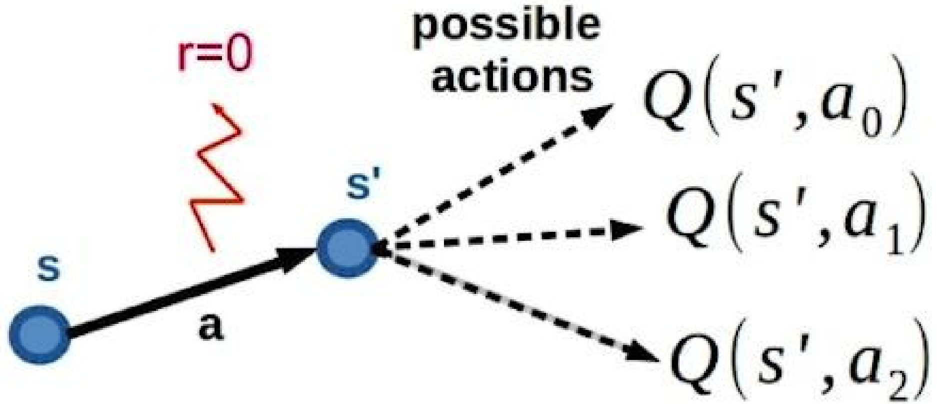


Suppose true $Q(s', a')$ are equal to **0** for all a'

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



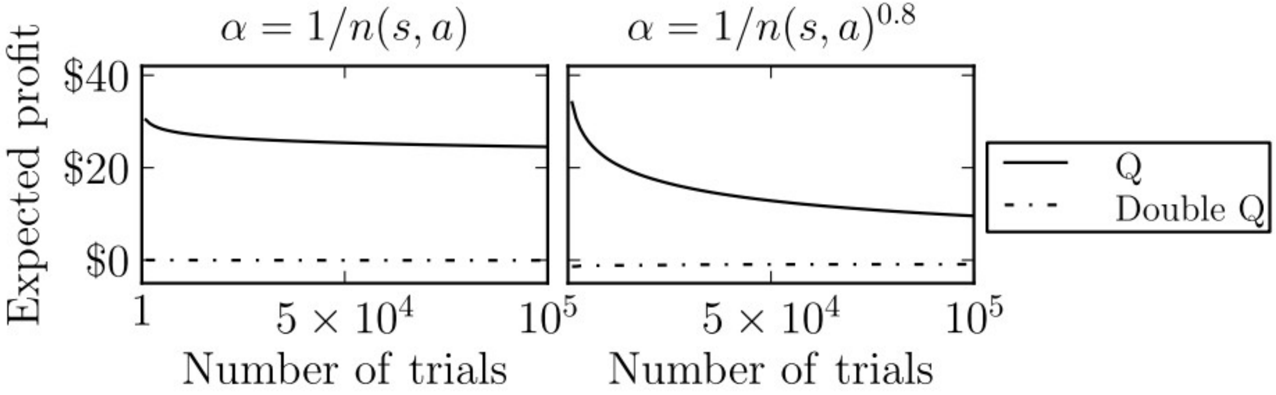
But if we update $Q(s,a)$ towards $r + \gamma \max_{a'} Q(s',a')$

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

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

Overestimation Example: Roulette

A tabular environment:
1 non-terminal state,
170 betting actions,
Betting $1 (infinite budget) or stop playing (yielding $0)
Expected gain per $1 bet: -$0.053



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

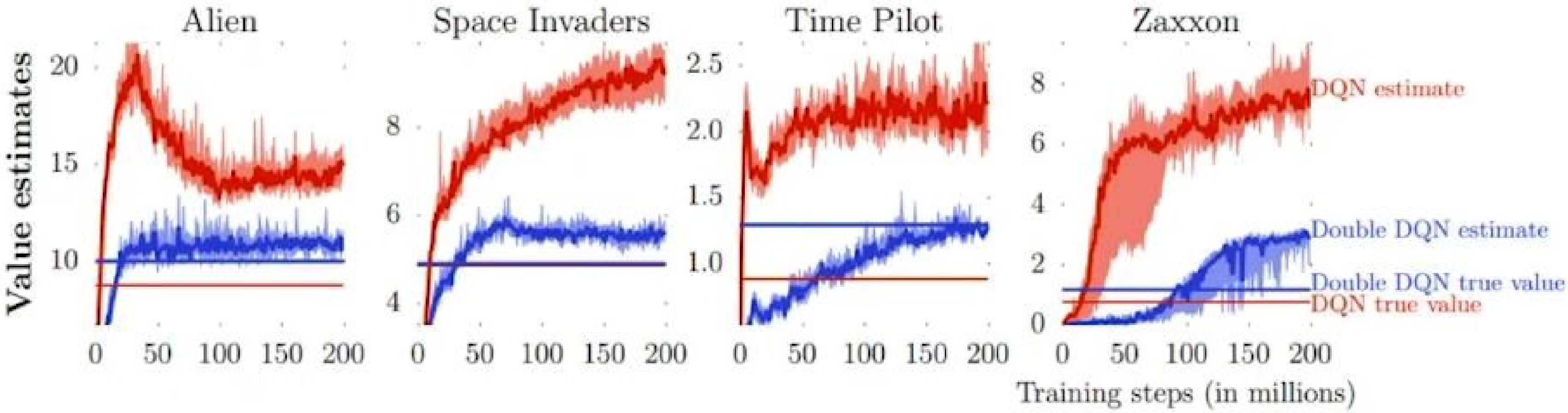
Initialize Q^A,Q^B,s
repeat
  Choose a, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe r, s'
  Choose (e.g. random) either UPDATE(A) or UPDATE(B)
  if UPDATE(A) **then**
    Define $a^* = \arg\max_a Q^A(s', a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\left(r + \gamma Q^B(s',a^*) - Q^A(s,a)\right)$
  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 Reinforcement Learning with Double Q-learning (Deepmind, 2015)

Idea: use main network to choose action!

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



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

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)
...	...	...	...

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!

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

$$p = |\delta|$$

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

Do you see the problem?

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!

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

$$p = |\delta|$$

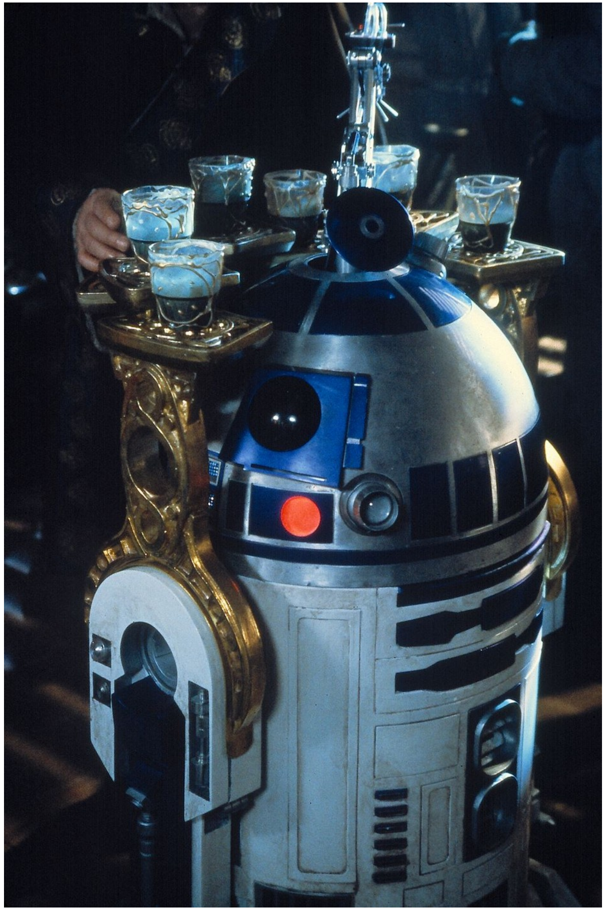
$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$ where α is the priority parameter (when α is 0 it's the uniform case)

Do you see the problem?

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

Prioritized Experience Replay Example: Roulette

A replay buffer generates a distribution over transitions ***(s, a, r, s')***.
Off-policy algorithms can handle an arbitrary distribution of ***s*** and ***a***.
(s, a) pairs come from a playing policy, initial state selection is possible.
The distribution of **(s, a)** will roughly affect only which part of the environment the agent adapts to.
However, <u>***r*** and ***s'*** are required to be unbiased</u>.



Prioritized Replay Buffer Example: Roulette

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Single non-terminal state, 2 actions: bet $1 or quit with no loss.

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Single non-terminal state, 2 actions: bet \$1 or quit with no loss.

On a \$1 bet a prize of \$100 is won with a 0.001 chance.

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Single non-terminal state, 2 actions: bet \$1 or quit with no loss.

On a \$1 bet a prize of \$100 is won with a 0.001 chance.

After a bet or a quit the environment enters the terminal state.

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Single non-terminal state, 2 actions: bet $1 or quit with no loss.

On a $1 bet a prize of $100 is won with a 0.001 chance.

After a bet or a quit the environment enters the terminal state.

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Single non-terminal state, 2 actions: bet $1 or quit with no loss.

On a $1 bet a prize of $100 is won with a 0.001 chance.

After a bet or a quit the environment enters the terminal state.

Suppose the buffer is infinitely large, and the approximator is a table.

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Single non-terminal state, 2 actions: bet \$1 or quit with no loss.

On a \$1 bet a prize of \$100 is won with a 0.001 chance.

After a bet or a quit the environment enters the terminal state.

Suppose the buffer is infinitely large, and the approximator is a table.

Which Q-function will the learner converge to?

Prioritized Replay Buffer Example: Roulette

Example: a toy roulette

Single non-terminal state, 2 actions: bet $1 or quit with no loss.

On a $1 bet a prize of $100 is won with a 0.001 chance.

After a bet or a quit the environment enters the terminal state.

Suppose the buffer is infinitely large, and the approximator is a table.

Which Q-function will the learner converge to?

Which policy will it produce?

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 $p_t = \max_{i<t} p_i$

Let’s watch a video
https://www.youtube.com/watch?v=UXurvvDY93o

Let’s watch a video
https://www.youtube.com/watch?v=UXurvvDY93o

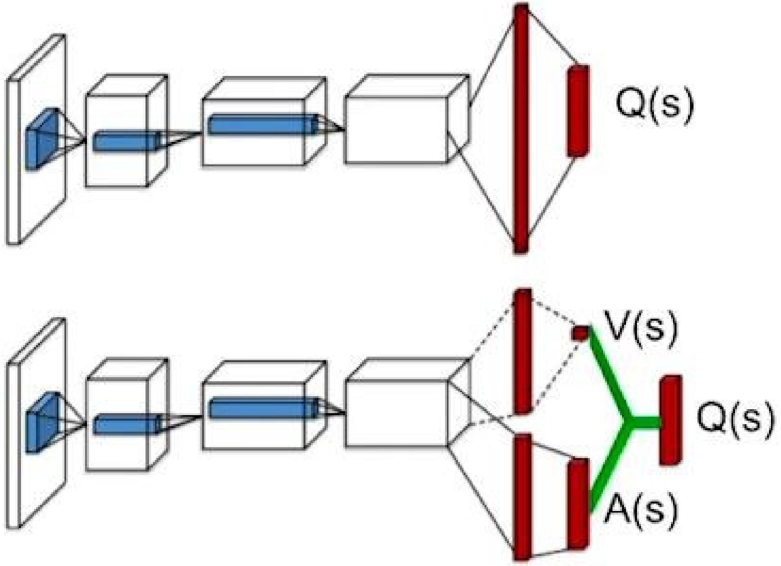
You will have it in your homework assignment to observe the spread between Q-values in a state

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

Idea: change the network's architecture.

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

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

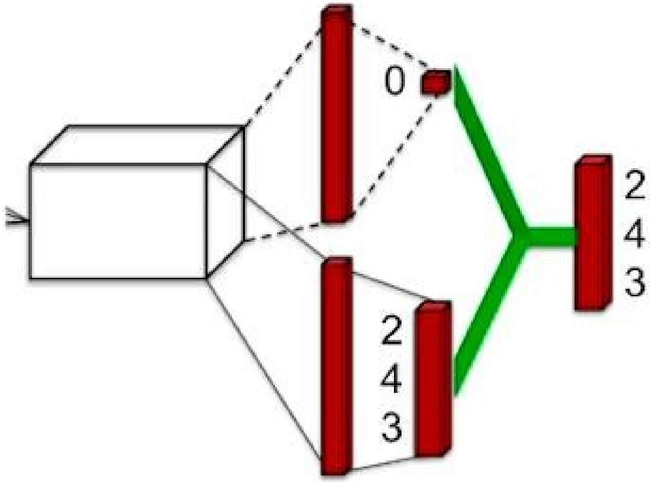


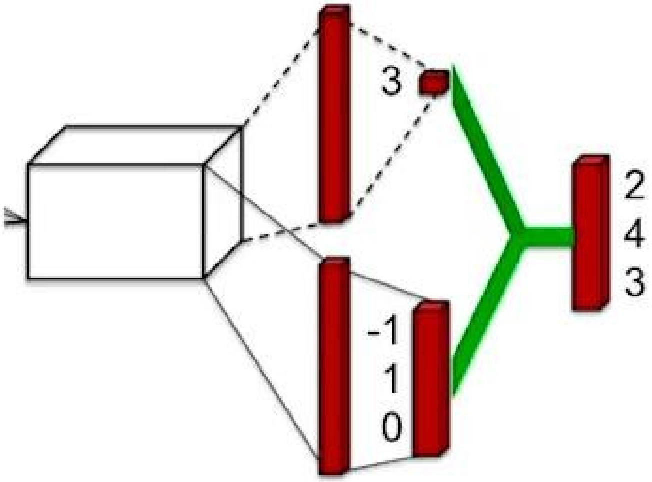
Do you see the problem?

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

Here is one extra freedom degree!

Example:





Which one is good?

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

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

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

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

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

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

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

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

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

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

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

Distributional Q-learning

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

$$\mathrm{Z}^{\pi}(s, a) = [G_t | s_t = s, a_t = a]$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

$$\mathrm{Z}^{\pi}(s,a) = [G_t | s_t = s, a_t = a]$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$ Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$ Number

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

$$\mathrm{Z}^{\pi}(s, a) = [G_t | s_t = s, a_t = a]$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$ Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$ Number

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$ Number

$$\mathrm{Z}^{\pi}(s,a) = [G_t | s_t = s, a_t = a]$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$ Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$ Number

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$ Number

$$\mathrm{Z}^{\pi}(s,a) = [G_t | s_t = s, a_t = a]$$ Random variable

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$ Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$ Number

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$ Number

$$\mathrm{Z}^{\pi}(s, a) = [G_t | s_t = s, a_t = a]$$ Random variable

Recurrent Relation

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$ Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$ Number

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$ Number

$$\mathrm{Z}^{\pi}(s, a) = [G_t | s_t = s, a_t = a]$$ Random variable

Recurrent Relation

$$Z^{\pi}(x, a) \stackrel{D}{=} R(x, a) + \gamma Z^{\pi}(X', A')$$

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$ Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$ Number

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$ Number

$$\mathrm{Z}^{\pi}(s,a) = [G_t | s_t = s, a_t = a]$$ Random variable

Recurrent Relation

$$Z^{\pi}(x,a) \stackrel{D}{=} R(x,a) + \gamma Z^{\pi}(X', A')$$

Bellman Operator

Distributional Q-learning

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$ Random variable

$$\mathrm{V}^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$ Number

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$ Number

$$\mathrm{Z}^{\pi}(s,a) = [G_t | s_t = s, a_t = a]$$ Random variable

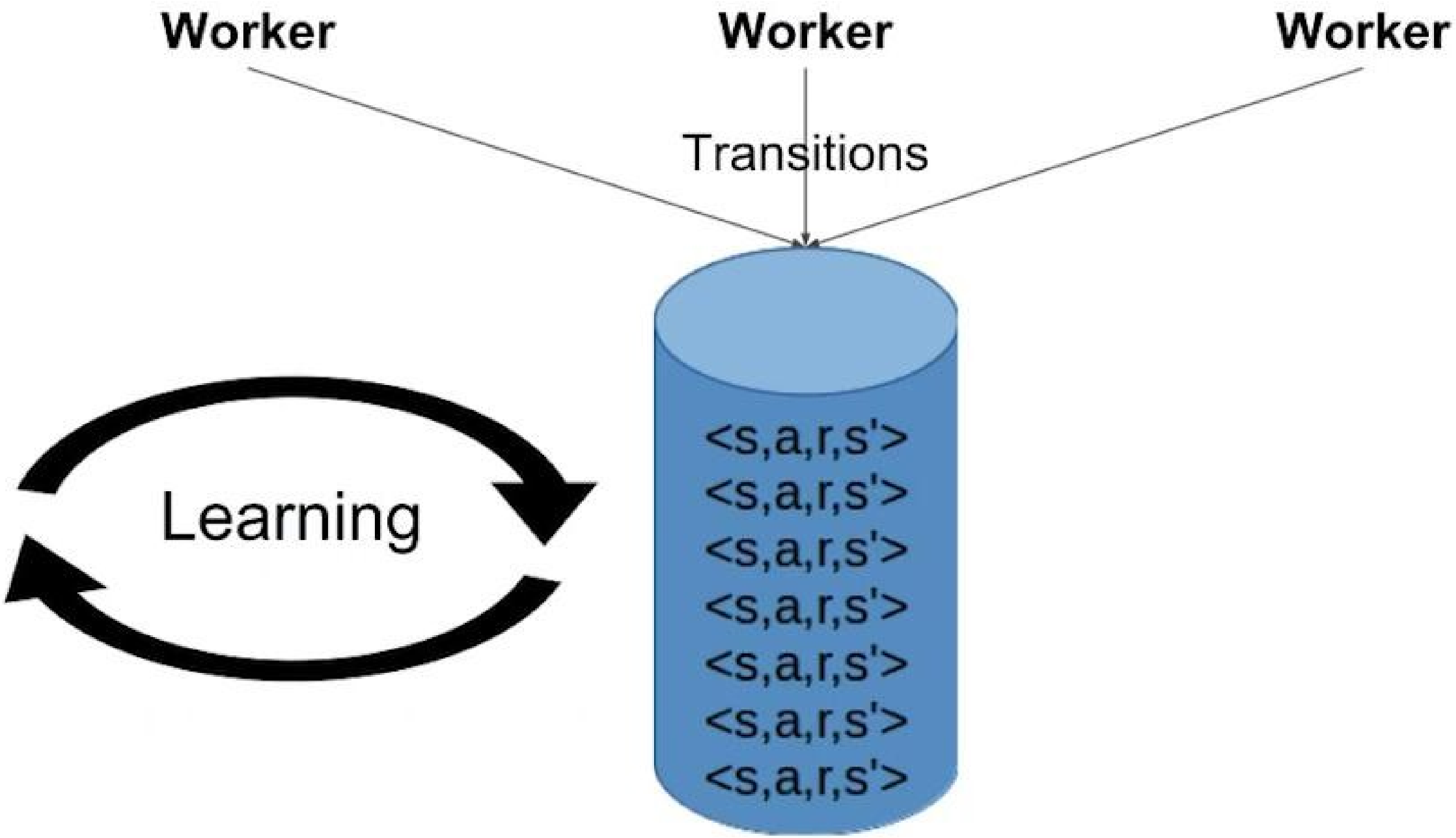
Recurrent Relation

$$Z^{\pi}(x,a) \overset{D}{=} R(x,a) + \gamma Z^{\pi}(X', A')$$

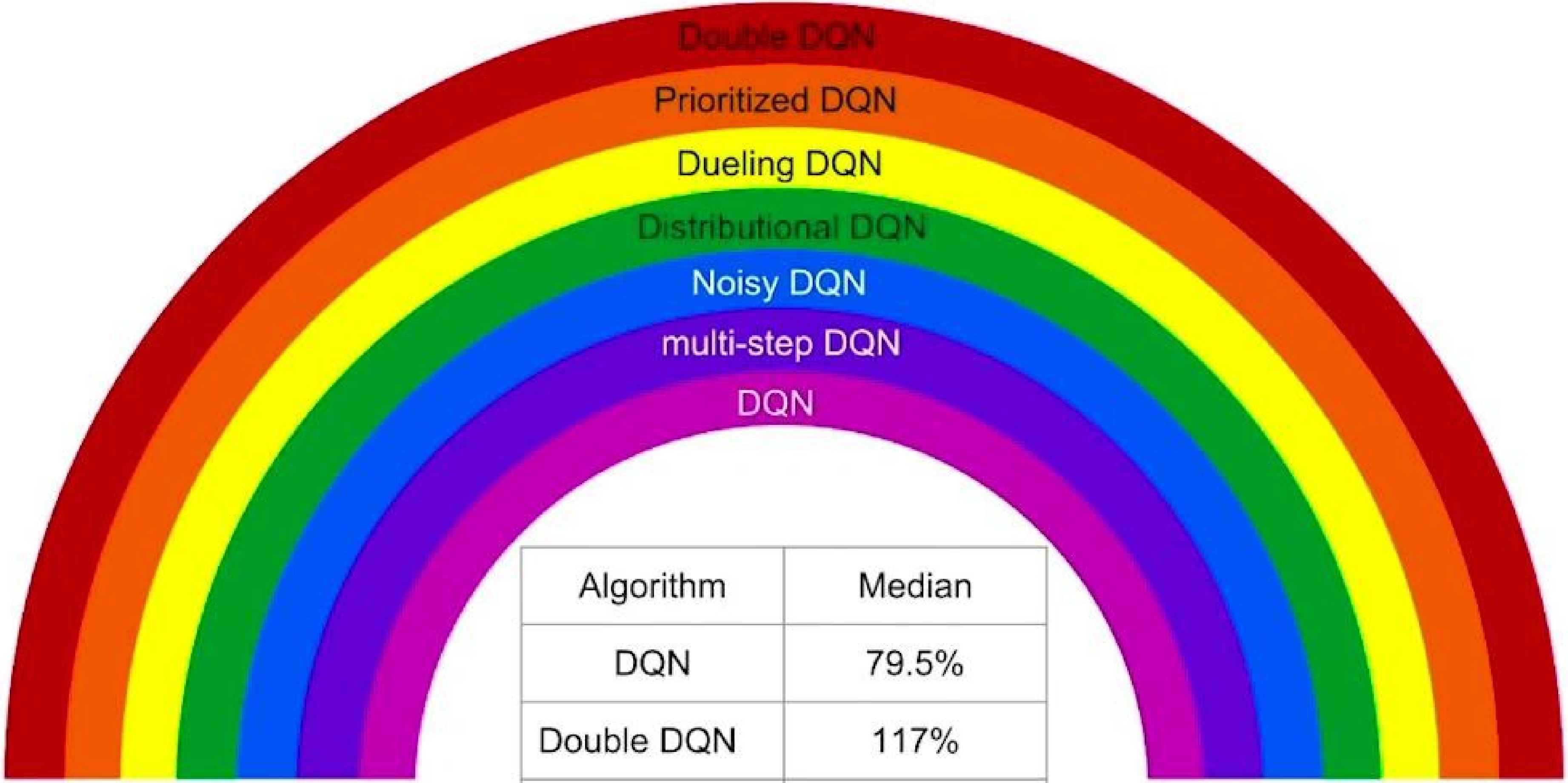
Bellman Operator

$$\mathcal{T} Z(x,a) \overset{D}{:=} R(x,a) + \gamma Z(X', \arg\max_{a' \in \mathcal{A}} \mathbb{E}\, Z(X', a'))$$

Asynchronous Methods for Deep Reinforcement Learning (2016, Deepmind)



Rainbow (2017, Deepmind)



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

R2D2 (2018, Deepmind)

LSTM

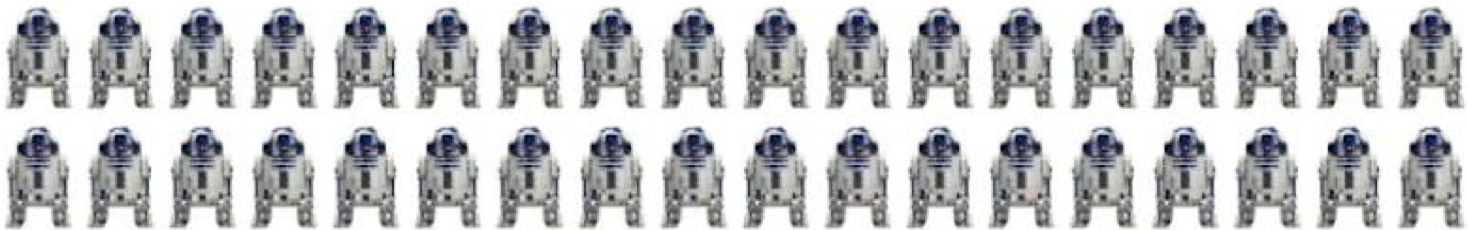
Reward re-scaling

Distributed Prioritized Experience Replay

Double DQN

n-step DQN

Dueling DQN



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

Thanks for your attention!