

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



یادگیری عمیق

جلسه ۲۷

# یادگیری تقویتی عمیق

Deep Reinforcement Learning

کاظم فولادی قلعه  
دانشکده مهندسی، پردیس فارابی  
دانشگاه تهران

<http://courses.fouladi.ir/deep>

یادگیری عمیق

یادگیری تقویتی عمیق

،

یادگیری  
تقویتی

SUPERVISED LEARNING

## So far... Supervised Learning

**Data:**  $(x, y)$   
x is data, y is label

**Goal:** Learn a *function* to map  $x \rightarrow y$

**Examples:** Classification,  
regression, object detection,  
semantic segmentation, image  
captioning, etc.



→ Cat

Classification

[This image is CC0 public domain](#)

## یادگیری بدون نظارت

UNSUPERVISED LEARNING

## So far... Unsupervised Learning

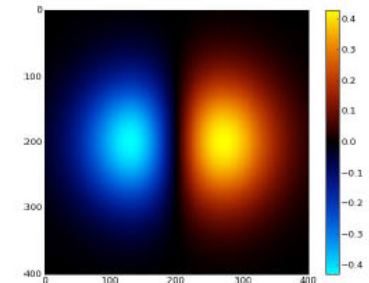
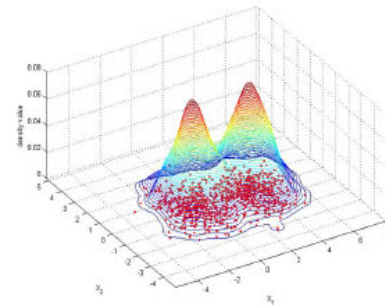
**Data:**  $x$ 

Just data, no labels!

**Goal:** Learn some underlying hidden *structure* of the data**Examples:** Clustering, dimensionality reduction, feature learning, density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



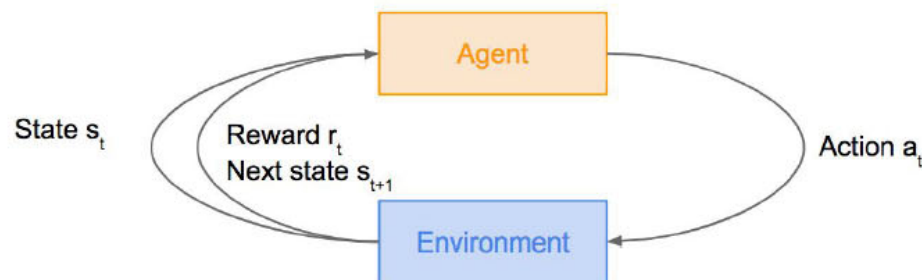
2-d density estimation

2-d density images [left](#) and [right](#) are [CC0 public domain](#)

REINFORCEMENT LEARNING

## Today: Reinforcement Learning

Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals



**Goal:** Learn how to take actions in order to maximize reward

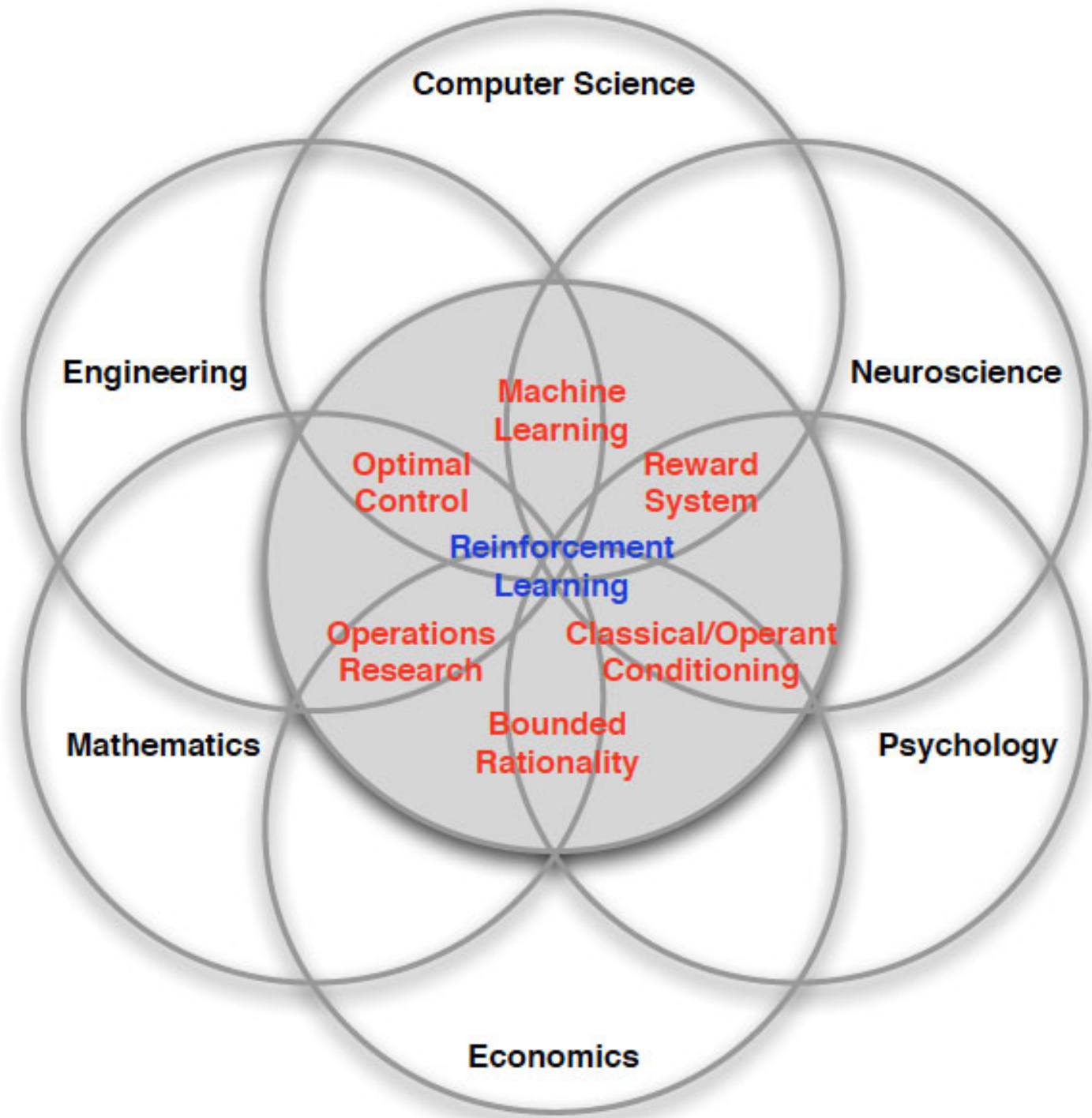
مسائل شامل یک عامل در حال اندرکنش با یک محیط که سیگنال‌های پاداش عددی را فراهم می‌کند.

**هدف:**

یادگیری چگونگی انجام کنش‌ها برای ماکزیمم‌سازی پاداش



Atari games figure copyright Volodymyr Mnih et al., 2013. Reproduced with permission.



## یادگیری تقویتی

مروری بر یادگیری تقویتی کلاسیک

### REINFORCEMENT LEARNING

- What is Reinforcement Learning?
- Markov Decision Processes
- Q-Learning
- Policy Gradients

## یادگیری تقویتی

عامل و محیط

REINFORCEMENT LEARNING

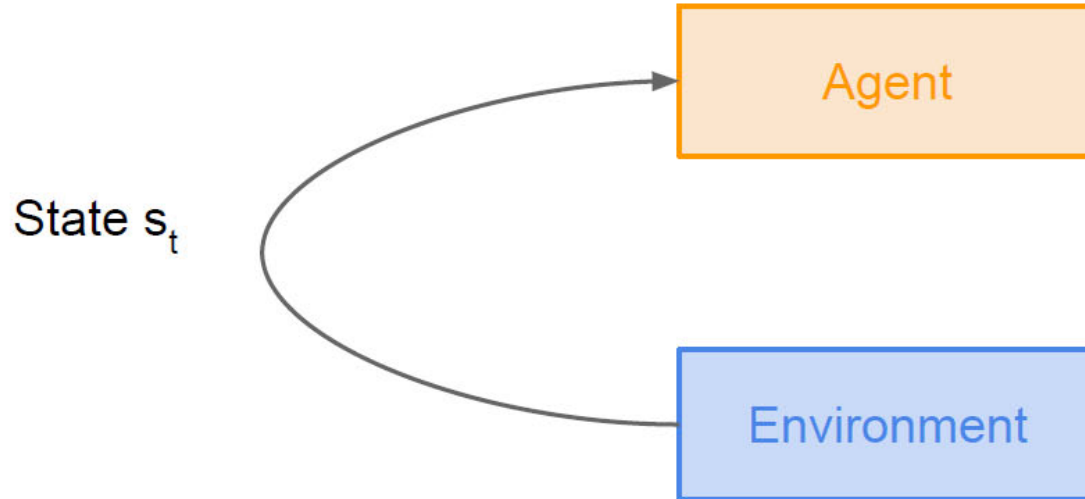
Agent

Environment



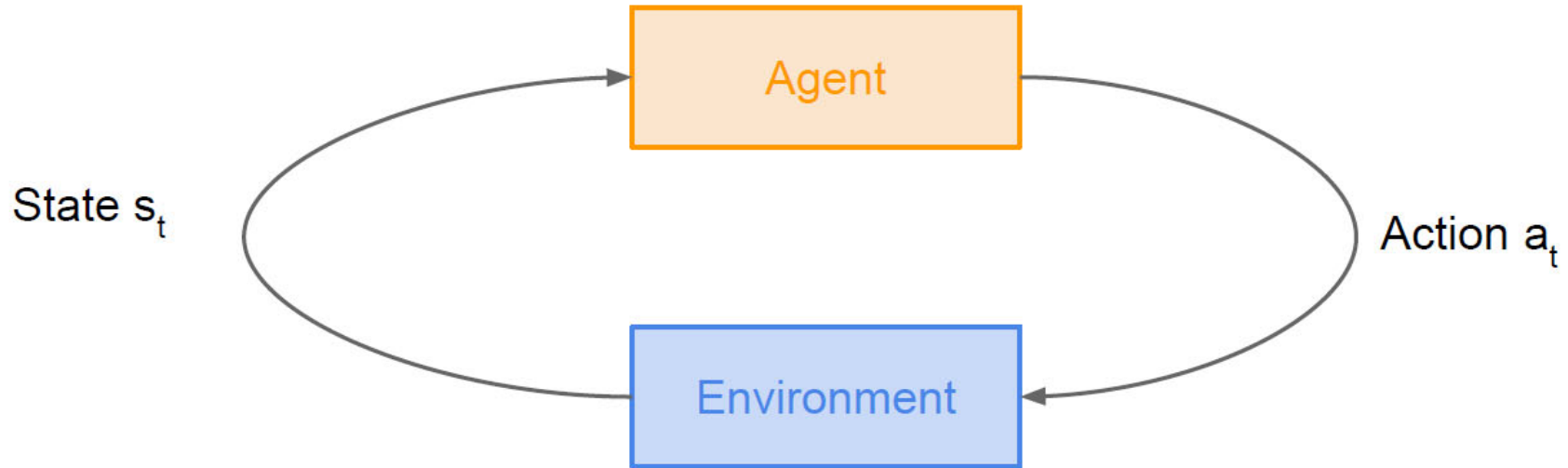
## یادگیری تقویتی

حالت محیط

REINFORCEMENT LEARNING

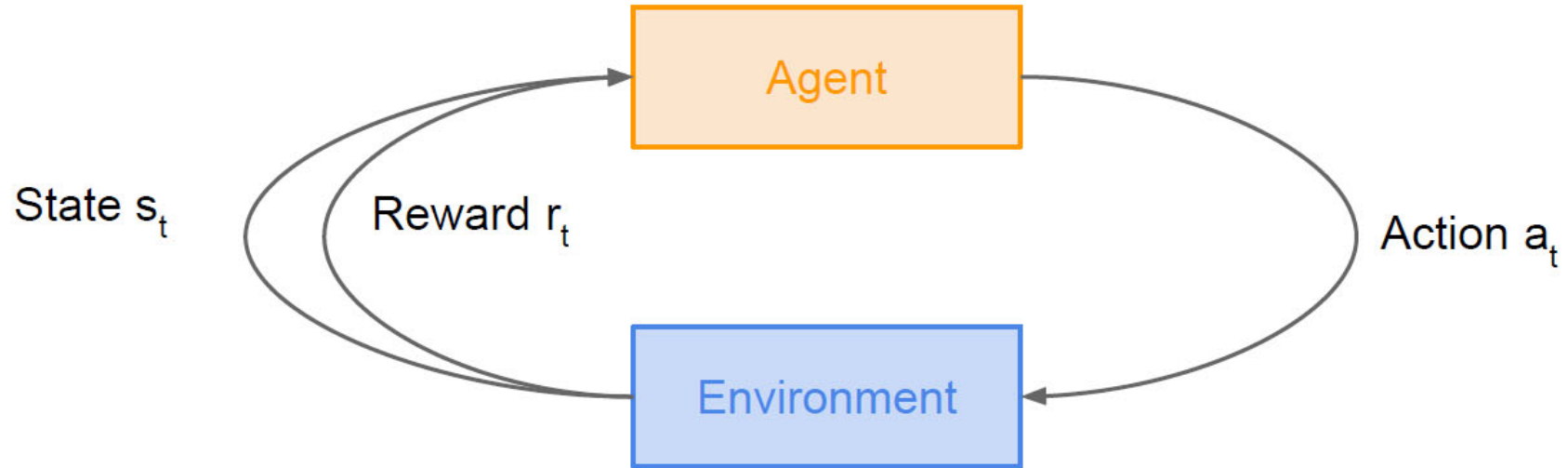
## یادگیری تقویتی

کنش عامل

REINFORCEMENT LEARNING

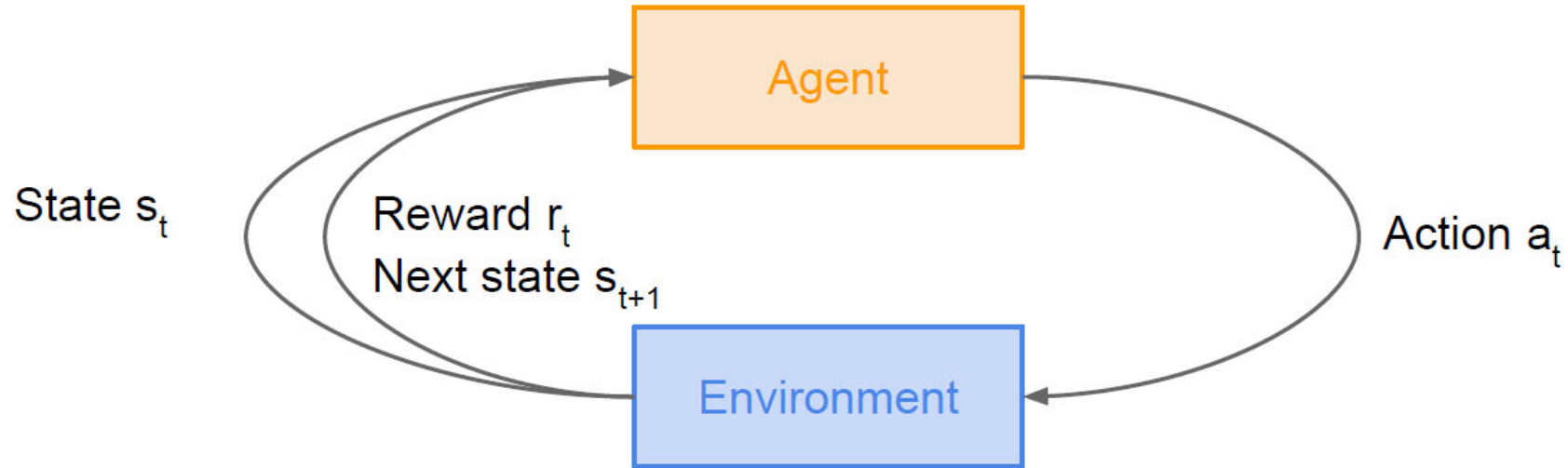
## یادگیری تقویتی

پاداش

REINFORCEMENT LEARNING

## یادگیری تقویتی

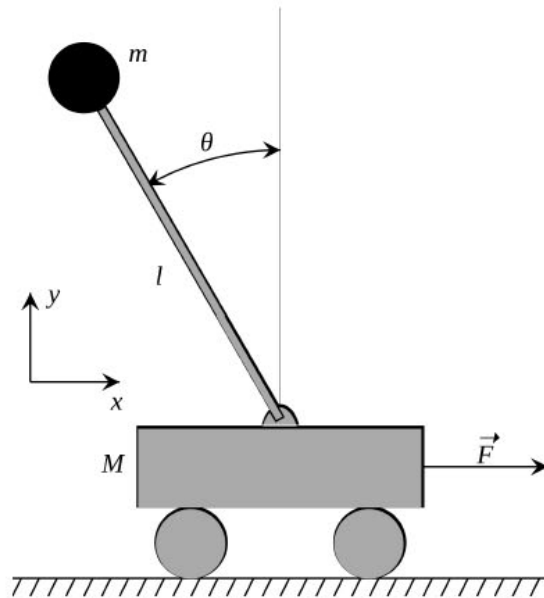
حالت بعدی

REINFORCEMENT LEARNING

## یادگیری تقویتی

مثال: مسئله‌ی ارابه-میله

## Cart-Pole Problem



**Objective:** Balance a pole on top of a movable cart

**State:** angle, angular speed, position, horizontal velocity

**Action:** horizontal force applied on the cart

**Reward:** 1 at each time step if the pole is upright

**هدف:** متعادل کردن یک میله بر بالای یک ارابه‌ی متحرک

**حالت:** زاویه، سرعت زاویه‌ای، موقعیت، سرعت افقی

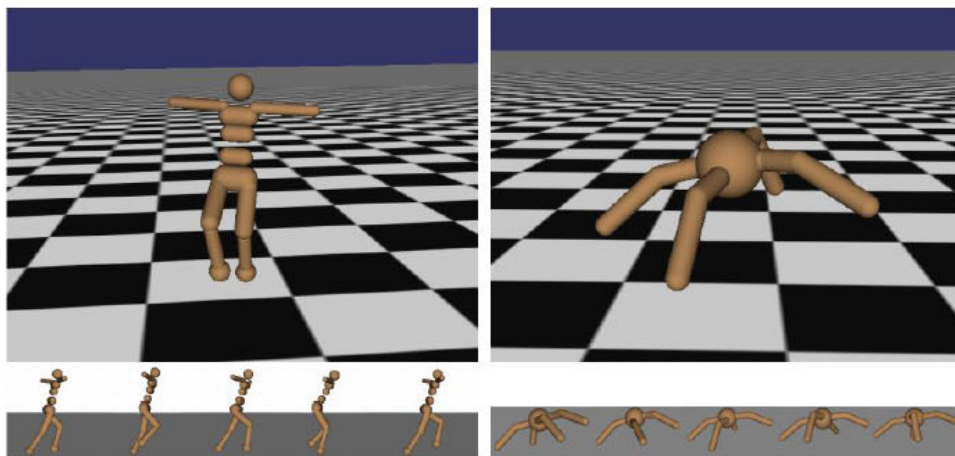
**کنش:** نیروی افقی وارد بر ارابه

**پاداش:** ۱ واحد در هر گام زمانی اگر میله به بالا ایستاده باشد

## یادگیری تقویتی

مثال: مسئله‌ی حرکت ربات

## Robot Locomotion



**Objective:** Make the robot move forward

**State:** Angle and position of the joints

**Action:** Torques applied on joints

**Reward:** 1 at each time step upright + forward movement

**هدف:** وادار کردن ربات به حرکت به جلو

**حالت:** زاویه و موقعیت مفاصل‌ها

**کنش:** گشتاورهای اعمال شده بر مفاصل‌ها

**پاداش:** ۱ واحد در هر گام زمانی اگر ربات ایستاده باشد و به جلو حرکت کند

## یادگیری تقویتی

مثال: بازی‌های آتاری

## Atari Games



**Objective:** Complete the game with the highest score

**State:** Raw pixel inputs of the game state

**Action:** Game controls e.g. Left, Right, Up, Down

**Reward:** Score increase/decrease at each time step

هدف: اتمام بازی با بالاترین امتیاز

حالت: ورودی‌های پیکسلی خام حالت بازی

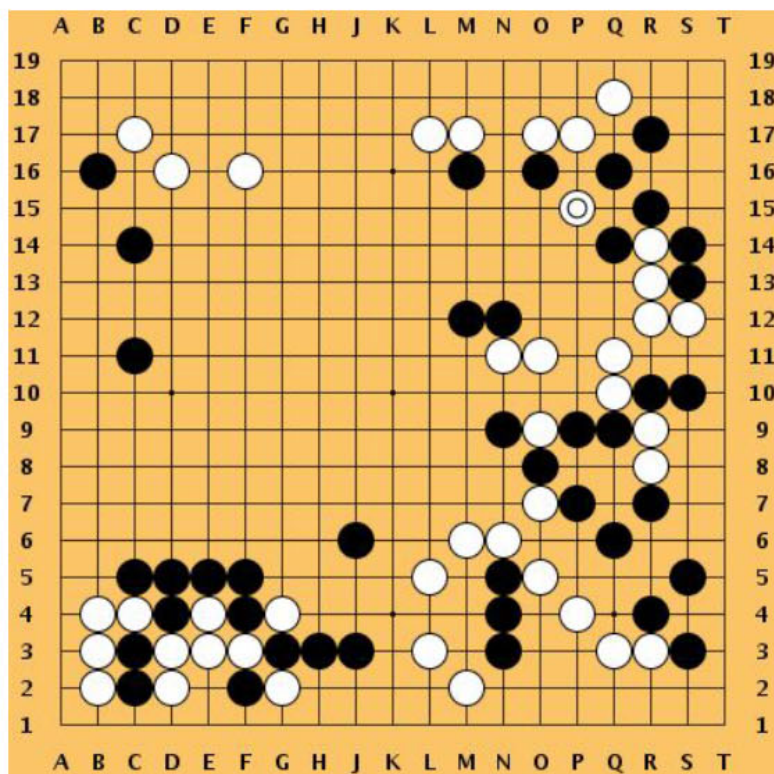
کنش: کنترل‌های بازی، مثل: چپ، راست، بالا، پایین

پاداش: افزایش/کاهش امتیاز در هر گام زمانی

## یادگیری تقویتی

مثال: بازی Go

## Go

**Objective:** Win the game!**State:** Position of all pieces**Action:** Where to put the next piece down**Reward:** 1 if win at the end of the game, 0 otherwise

هدف: بردن بازی!

حالت: موقعیت همه‌ی مهره‌ها

کنش: تعیین موقعیتی که مهره‌ی بعدی در آن باید قرار بگیرد

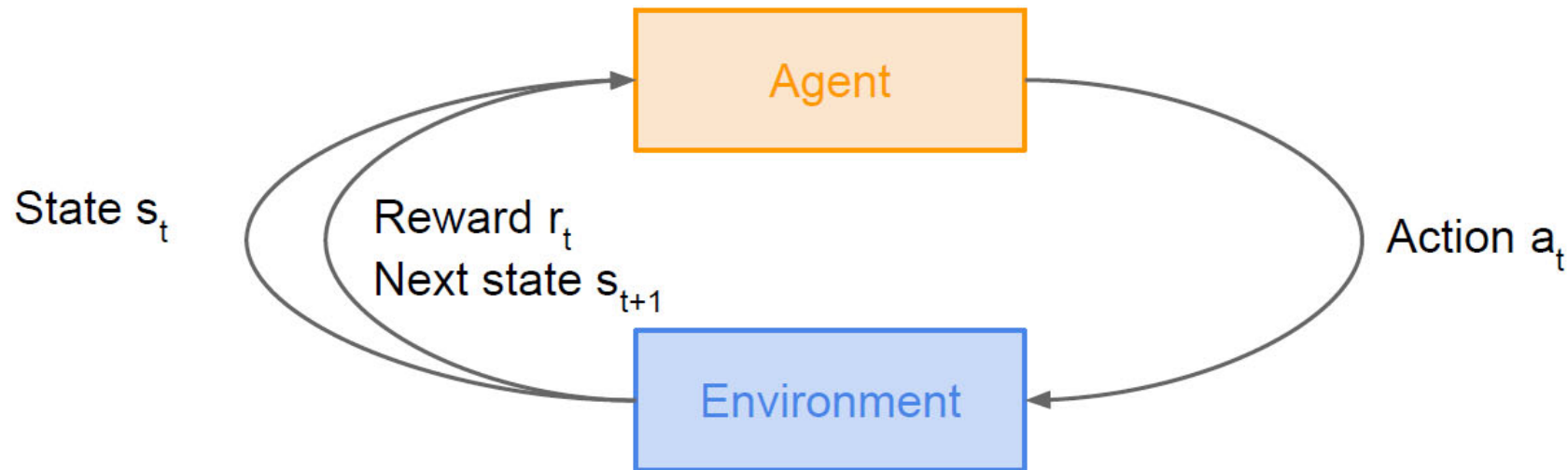
پاداش: ۱ اگر عامل در انتهای بازی ببرد، وگرنه صفر



## یادگیری تقویتی

صورت بندی ریاضی

How can we mathematically formalize the RL problem?



چگونه می توانیم مسئله ی یادگیری تقویتی را به صورت ریاضی صورت بندی کنیم؟  
با استفاده از فرآیند تصمیم مارکوف

یادگیری تقویتی عمیق

۲

فرآیند  
تصمیم  
مارکوف

## فرآیند تصمیم مارکوف

فرمول‌بندی ریاضی مسئله‌ی یادگیری تقویتی

MARKOV DECISION PROCESS (MDP)

مؤلفه‌های تعریف یک MDP

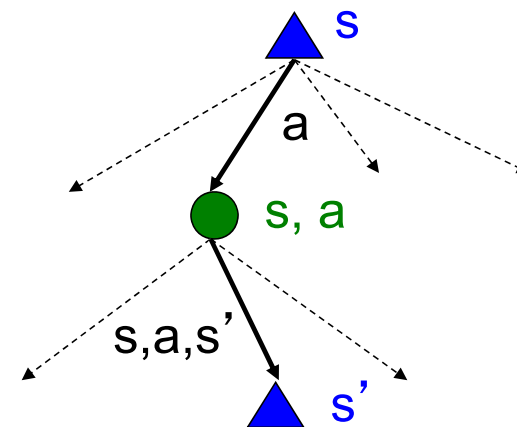
کنش‌های عامل Actions of Agent	حالت‌های محیط States of Environment	حالت آغازین Initial State	مدل گذار Transition Model	تابع پاداش Reward Function
----------------------------------	--	------------------------------	------------------------------	-------------------------------

$R(s)$        $T(s, a, s')$        $s_0$       States  $s \in S$ , actions  $a \in A$   
 $R(s, a)$   
 $R(s, a, s')$

روش‌های راه‌حل		
...	تکرار سیاست Policy Iteration (PI)	تکرار ارزش Value Iteration (VI)

## محدودیت‌ها:

- \* فضای حالت نباید زیاد بزرگ باشد.
- \* فرض شده است  $T$  و  $R$  (مدل محیط) معلوم است.



راه‌حل: روش‌های یادگیری تقویتی (Reinforcement Learning)

## فرآیند تصمیم مارکوف

فرمول‌بندی ریاضی مسئله‌ی یادگیری تقویتی

MARKOV DECISION PROCESS (MDP)

# Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

خاصیت مارکوف: حالت فعلی به‌طور کامل حالت دنیا را مشخص می‌کند.

Defined by:  $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$  : set of possible states

$\mathcal{A}$  : set of possible actions

$\mathcal{R}$  : distribution of reward given (state, action) pair

$\mathbb{P}$  : transition probability i.e. distribution over next state given (state, action) pair

$\gamma$  : discount factor

## فرآیند تصمیم مارکوف

مراحل

MARKOV DECISION PROCESS (MDP)

## Markov Decision Process

- در گام زمانی  $t = 0$ ، محیط حالت آغازین را نمونه‌گیری می‌کند.

- At time step  $t=0$ , environment samples initial state  $s_0 \sim p(s_0)$
- Then, for  $t=0$  until done:
  - سپس، برای  $t = 0$  تا پایان کار:
    - عامل کنش  $a_t$  را انتخاب می‌کند
    - محیط از پاداش نمونه‌گیری می‌کند
    - محیط از حالت بعدی نمونه‌گیری می‌کند.
    - عامل پاداش و حالت بعدی را دریافت می‌کند.
  - Agent selects action  $a_t$
  - Environment samples reward  $r_t \sim R(\cdot | s_t, a_t)$
  - Environment samples next state  $s_{t+1} \sim P(\cdot | s_t, a_t)$
  - Agent receives reward  $r_t$  and next state  $s_{t+1}$

- A policy  $\pi$  is a function from  $S$  to  $A$  that specifies what action to take in each state

- **Objective:** find policy  $\pi^*$  that maximizes cumulative discounted reward:  $\sum_{t \geq 0} \gamma^t r_t$

سیاست  $\pi$ : تابعی از  $S$  به  $A$  است که مشخص می‌کند کدام کنش در هر حالت باید انتخاب شود.

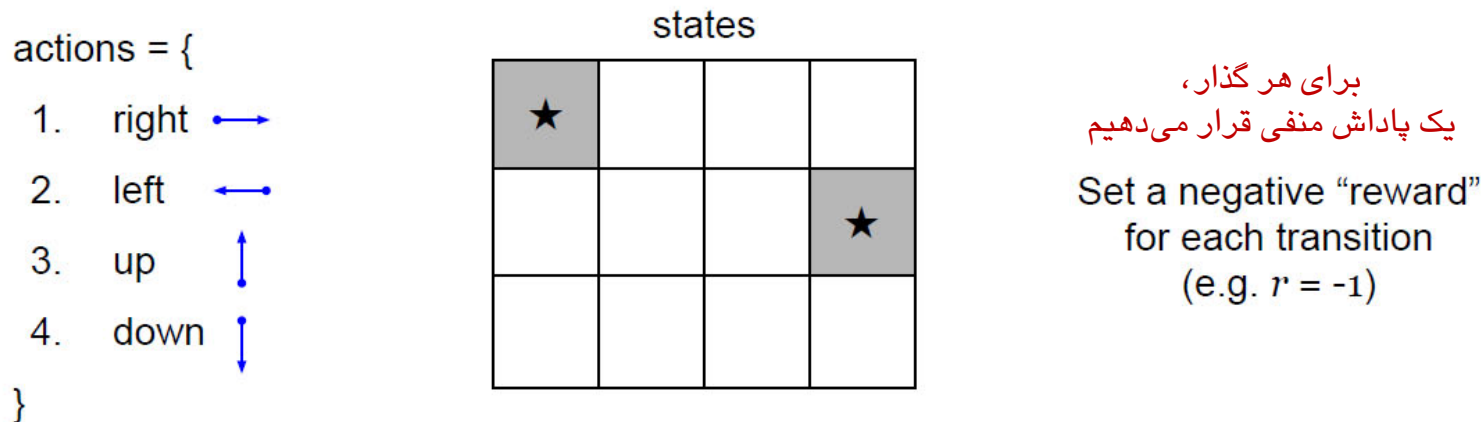
هدف: یافتن سیاست بهینه  $\pi^*$  که پاداش تخفیف‌یافته‌ی تجمعی را ماکزیمم کند.

## فرآیند تصمیم مارکوف

یک فرآیند تصمیم مارکوف ساده: دنیای توری

MARKOV DECISION PROCESS (MDP)

## A simple MDP: Grid World



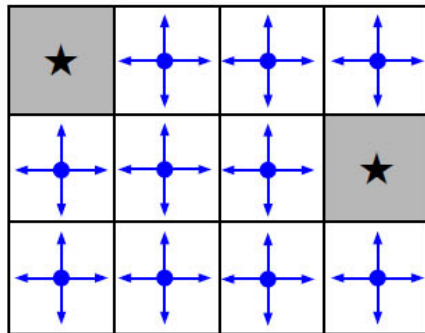
**Objective:** reach one of terminal states (greyed out) in  
least number of actions

هدف: رسیدن به یک حالت پایانی (خاکستری) با حداقل تعداد کنش‌ها

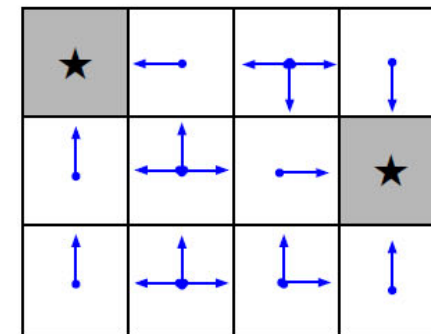
## فرآیند تصمیم مارکوف

یک فرآیند تصمیم مارکوف ساده: دنیای توری

### A simple MDP: Grid World



Random Policy



Optimal Policy

راه حل

# ۳

مؤلفه‌های  
یک عامل  
یادگیرنده‌ی  
تقویتی



## مؤلفه‌های یک عامل یادگیرنده تقویتی

### COMPONENTS OF AN RL AGENT

مؤلفه‌های یک عامل یادگیرنده تقویتی <i>Components of an RL Agent</i>		
مدل <i>Model</i>	تابع ارزش <i>Value Function</i>	سیاست <i>Policy</i>
بازنمایی عامل از محیط چگونه است؟	هر <b>حالت</b> و / یا زوج <b>حالت-کنش</b> چه قدر خوب است؟	یک عامل چگونه رفتار می‌کند؟

## مؤلفه‌های یک یادگیرنده‌ی تقویتی

سیاست

POLICY

سیاست: یک عامل چگونه رفتار می‌کند؟  
نگاشت از حالت‌ها به کنش‌ها

e.g.

State	Action
A	→ 2
B	→ 1

Deterministic policy:  $a = \pi(s)$

Stochastic policy:  $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

## مؤلفه‌های یک عامل یادگیرنده‌ی تقویتی

سیاست بهینه‌ی  $\pi^*$ The optimal policy  $\pi^*$ 

We want to find optimal policy  $\pi^*$  that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability...)?

**سیاست  $\pi^*$ :** سیاست بهینه که مجموع پاداش‌ها (ی تخفیف یافته) را ماکزیمم می‌کند.  
هدف، یافتن سیاست بهینه است.

چگونه می‌توانیم تصادفی بودن را اداره کنیم (حالت آغازین، احتمال گذار، ...)?

## مؤلفه‌های یک عامل یادگیرنده‌ی تقویتی

سیاست بهینه‌ی  $\pi^*$

### The optimal policy $\pi^*$

We want to find optimal policy  $\pi^*$  that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability...)?

Maximize the **expected sum of rewards!**

**سیاست  $\pi^*$ :** سیاست بهینه که مجموع پاداش‌ها (ی تخفیف یافته) را ماکزیمم می‌کند.  
هدف، یافتن سیاست بهینه است.

$$\text{Formally: } \pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right] \text{ with } s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$$

چگونه می‌توانیم تصادفی بودن را اداره کنیم (حالت آغازین، احتمال گذار، ...)?  
از طریق ماکزیمم‌سازی امید مجموع پاداش‌ها

## مؤلفه‌های یک عامل یادگیرنده‌ی تقویتی

تابع ارزش و تابع ارزش-Q

## Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths)  $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

دنبال کردن یک سیاست تراجکتوری‌ها (یا مسیرها)ی نمونه را تولید می‌کند.

## مؤلفه‌های یک عامل یادگیرنده‌ی تقویتی

تابع ارزش و تابع ارزش-Q

## Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths)  $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

How good is a state?

یک حالت چه قدر خوب است؟

The **value function** at state  $s$ , is the expected cumulative reward from following the policy from state  $s$ :

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$

یک تابع ارزش در حالت  $S$ : امید پاداش تجمعی حاصل از دنبال کردن این سیاست از حالت  $S$

## مؤلفه‌های یک عامل یادگیرنده‌ی تقویتی

تابع ارزش و تابع ارزش-Q

### Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths)  $s_0, a_0, r_0, s_1, a_1, r_1, \dots$

How good is a state?

The **value function** at state  $s$ , is the expected cumulative reward from following the policy from state  $s$ :

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$

How good is a state-action pair?

یک جفت حالت-کنش چه قدر خوب است؟

The **Q-value function** at state  $s$  and action  $a$ , is the expected cumulative reward from taking action  $a$  in state  $s$  and then following the policy:

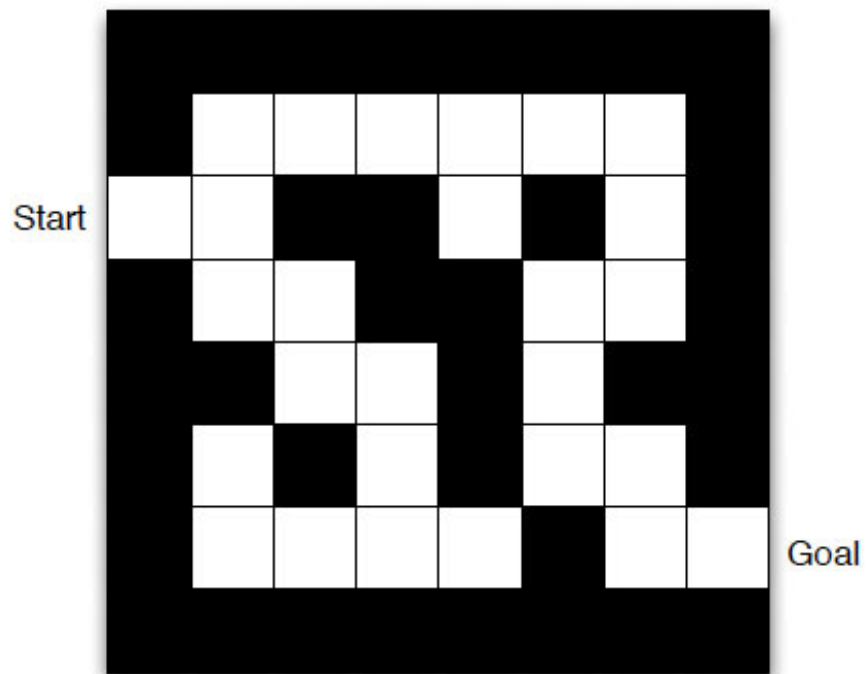
$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

یک تابع ارزش-Q در حالت  $s$  و کنش  $a$ : امید پاداش تجمعی حاصل از انتخاب کنش  $a$  در حالت  $s$  و سپس دنبال کردن این سیاست

## مؤلفه‌های یک یادگیرنده تقویتی

مثال : مسئله‌ی ماز

### MAZE PROBLEM



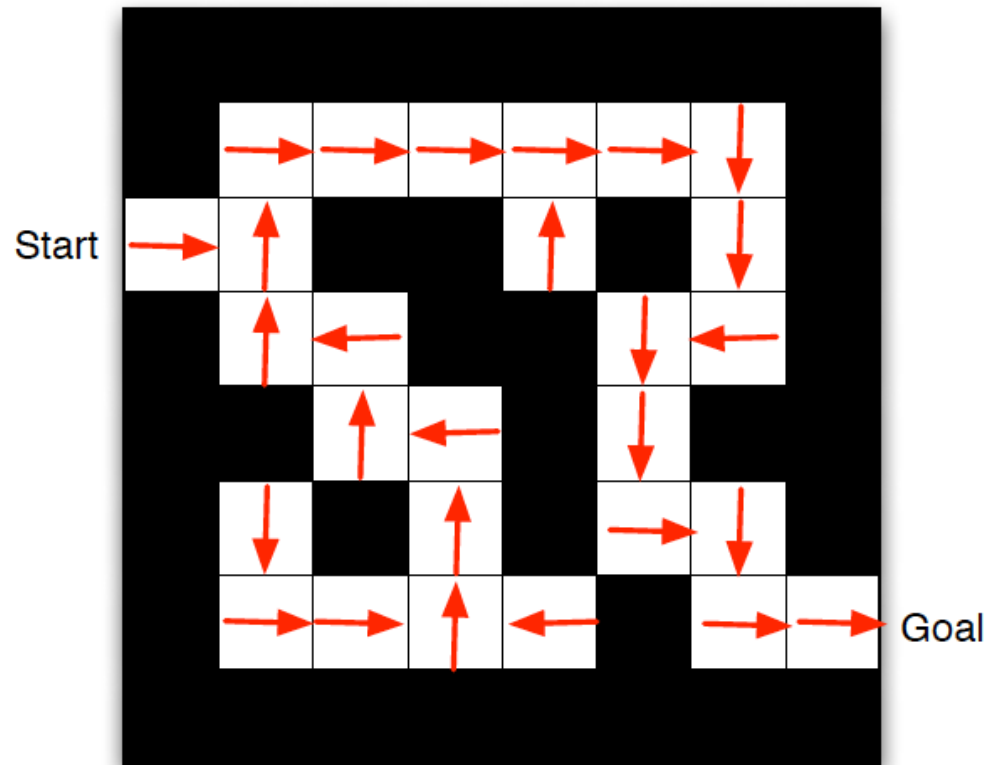
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location



## مؤلفه‌های یک عامل یادگیرنده تقویتی

مثال: مسئله‌ی ماز: سیاست

### POLICY

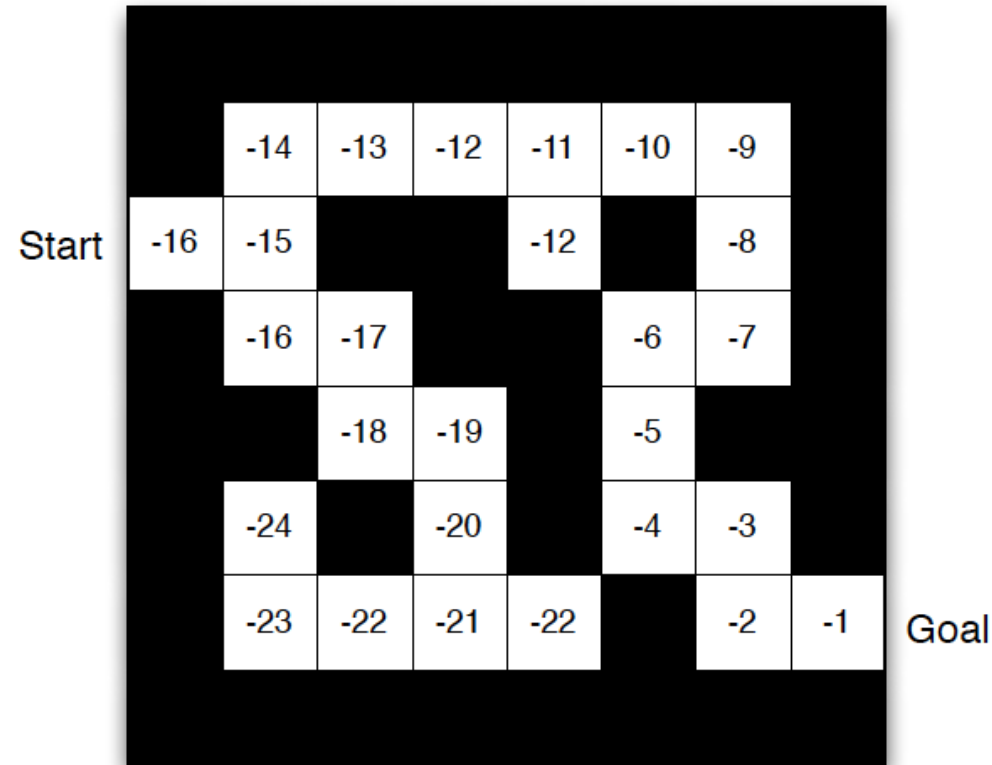


- Arrows represent policy  $\pi(s)$  for each state  $s$

## مؤلفه‌های یک عامل یادگیرنده تقویتی

مثال: مسئله‌ی ماز: تابع ارزش

### VALUE FUNCTION

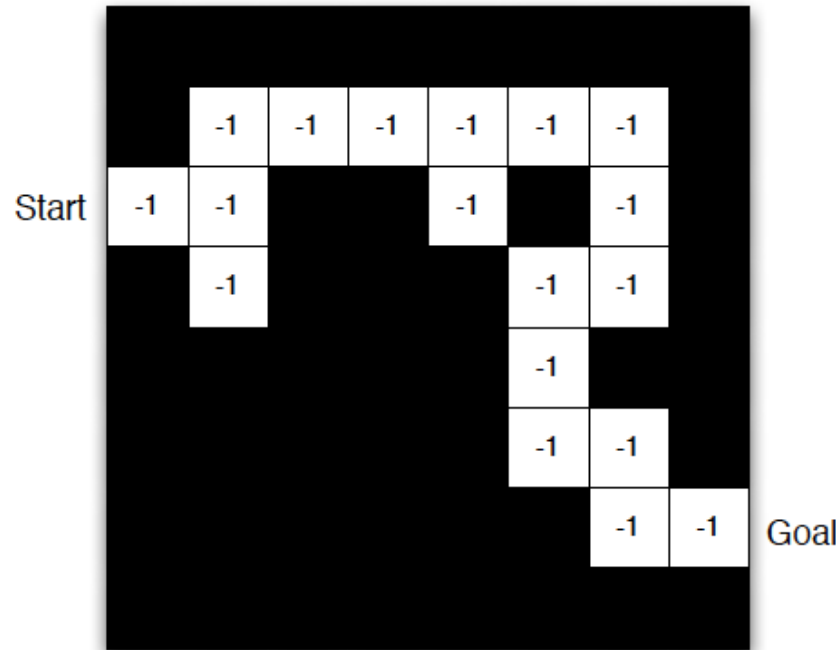


- Numbers represent value  $v_{\pi}(s)$  of each state  $s$

## مؤلفه‌های یک عامل یادگیرنده تقویتی

مثال: مسئله‌ی ماز: مدل

### MODEL



- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model  $\mathcal{P}_{ss'}^a$
- Numbers represent immediate reward  $\mathcal{R}_s^a$  from each state  $s$  (same for all  $a$ )

یادگیری تقویتی عمیق

۴

یادگیری  
تقویتی  
عمیق

## یادگیری تقویتی

روی کردها

روی کردهای یادگیری تقویتی <i>RL Approaches</i>		
مبتنی بر مدل <i>Model-Based RL</i>	مبتنی بر ارزش <i>Value-Based RL</i>	مبتنی بر سیاست <i>Policy-Based RL</i>
ساخت یک مدل از دنیا گذار حالت، احتمالات پاداش، ... طرح ریزی با استفاده از مدل	تخمین تابع بهینه‌ی کنش-ارزش	جستجوی مستقیم به دنبال سیاست بهینه $\pi^*$

## یادگیری تقویتی عمیق

## DEEP RL

روی‌کردهای یادگیری تقویتی عمیق <i>Deep RL Approaches</i>		
مبتنی بر مدل <i>Model-Based RL</i>	مبتنی بر ارزش <i>Value-Based RL</i>	مبتنی بر سیاست <i>Policy-Based RL</i>
ساخت یک مدل از دنیا گذار حالت، احتمالات پاداش، ... طرح‌ریزی با استفاده از مدل	تخمین تابع بهینه‌ی کنش-ارزش	جستجوی مستقیم به دنبال سیاست بهینه $\pi^*$
استفاده از شبکه‌های عصبی برای بازنمایی و یادگیری مدل	استفاده از شبکه‌های عصبی برای بازنمایی تابع ارزش Q $Q(s, a; \theta)$ $Q(s, a; \theta^*) \approx Q^*(s, a)$	استفاده از شبکه‌های عصبی برای بازنمایی سیاست $\pi_\theta$ $\pi_{\theta^*} \approx \pi^*$

## Bellman equation

The optimal Q-value function  $Q^*$  is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

بهینه‌ی تابع ارزش-Q یعنی  $Q^*$ ، ماکزیمم امید پاداش تجمعی قابل حصول از یک زوج (حالت، کنش) داده شده است.

## معادله‌ی بلمن

## اصل بهینگی

## Bellman equation

The optimal Q-value function  $Q^*$  is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

$Q^*$  satisfies the following **Bellman equation**: این معادله‌ی بلمن را ارضا می‌کند:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

**Intuition:** if the optimal state-action values for the next time-step  $Q^*(s', a')$  are known, then the optimal strategy is to take the action that maximizes the expected value of  $r + \gamma Q^*(s', a')$

شهود (اصل بهینگی): اگر مقادیر بهینه‌ی حالت-کنش برای گام زمانی بعدی یعنی  $Q^*(s', a')$  معلوم باشد، آن‌گاه استراتژی بهینه انتخاب کنشی است که مقدار امید  $r + \gamma Q^*(s', a')$  را ماکزیمم می‌کند.



## معادله‌ی بلمن

سیاست بهینه

## Bellman equation

The optimal Q-value function  $Q^*$  is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

$Q^*$  satisfies the following **Bellman equation**:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

**Intuition:** if the optimal state-action values for the next time-step  $Q^*(s', a')$  are known, then the optimal strategy is to take the action that maximizes the expected value of  $r + \gamma Q^*(s', a')$

The optimal policy  $\pi^*$  corresponds to taking the best action in any state as specified by  $Q^*$

سیاست بهینه  $\pi^*$  متناظر است با: انتخاب بهترین کنش در هر حالت آن‌گونه که با  $Q^*$  مشخص شده است.

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم تکرار ارزش

VALUE ITERATION ALGORITHM

## Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

$Q_i$  will converge to  $Q^*$  as  $i \rightarrow \infty$

الگوریتم تکرار ارزش: استفاده از معادله‌ی بلمن به صورت به روزرسانی تکراری  $Q_i$  به  $Q^*$  همگرا می‌شود وقتی  $i$  به بی‌نهایت میل کند.

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم تکرار ارزش

VALUE ITERATION ALGORITHM

# Solving for the optimal policy

**Value iteration algorithm:** Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

$Q_i$  will converge to  $Q^*$  as  $i \rightarrow \infty$

What's the problem with this?

مشکل این روش؟

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم تکرار ارزش

VALUE ITERATION ALGORITHM

# Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

$Q_i$  will converge to  $Q^*$  as  $i \rightarrow \infty$

What's the problem with this?

Not scalable. Must compute  $Q(s,a)$  for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

مشکل این روش؟

مشکل این روش این است که مقیاس پذیر نیست.

$Q(s,a)$  باید برای هر جفت حالت-کنش محاسبه شود.

مثلاً اگر حالت، پیکسل‌های حالت فعلی بازی باشد، محاسبه برای کل فضای حالت به لحاظ محاسباتی امکان‌ناپذیر است!

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم تکرار ارزش

VALUE ITERATION ALGORITHM

# Solving for the optimal policy

**Value iteration algorithm:** Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E} \left[ r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

$Q_i$  will converge to  $Q^*$  as  $i \rightarrow \infty$

**What's the problem with this?**

Not scalable. Must compute  $Q(s,a)$  for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

**Solution:** use a function approximator to estimate  $Q(s,a)$ . E.g. a neural network!

راه حل:

استفاده از یک تقریب‌زن تابع برای تخمین  $Q(s,a)$ .  
مثلاً یک شبکه‌ی عصبی!

## حل معادله‌ی بلمن برای سیاست بهینه

## الگوریتم یادگیری Q

Q-LEARNING

# Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

یادگیری Q:

استفاده از یک تقریبزن تابع برای تخمین تابع ارزش-کنش  $Q(s,a)$ .

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم یادگیری Q عمیق

DEEP Q-LEARNING

# Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

If the function approximator is a deep neural network => **deep q-learning!**

یادگیری Q عمیق:

اگر تقریب‌زن تابع یک شبکه‌ی عصبی عمیق باشد.

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم یادگیری Q عمیق

### DEEP Q-LEARNING

# Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning!**

یادگیری Q عمیق:

اگر تقریب‌زن تابع یک شبکه‌ی عصبی عمیق باشد.



## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم یادگیری Q عمیق

DEEP Q-LEARNING

# Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

یادآوری: می‌خواهیم یک تابع Q بیابیم که معادله‌ی بلمن را ارضا کند.

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم یادگیری Q عمیق

DEEP Q-LEARNING

# Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

## Forward Pass

Loss function:  $L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} [(y_i - Q(s, a; \theta_i))^2]$ where  $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$ 

گذر پیش‌رو: محاسبه‌ی تابع اتلاف

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم یادگیری Q عمیق

DEEP Q-LEARNING

# Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

## Forward Pass

Loss function:  $L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} [(y_i - Q(s, a; \theta_i))^2]$ where  $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$ 

## Backward Pass

Gradient update (with respect to Q-function parameters  $\theta$ ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right] \nabla_{\theta_i} Q(s, a; \theta_i)$$

گذر پس‌رو: به روزرسانی گرادیان (نسبت به پارامترهای تابع Q یعنی  $\theta$ )

## حل معادله‌ی بلمن برای سیاست بهینه

الگوریتم یادگیری Q عمیق

DEEP Q-LEARNING

## Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

## Forward Pass

$$\text{Loss function: } L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right]$$

$$\text{where } y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Iteratively try to make the Q-value close to the target value ( $y_i$ ) it should have, if Q-function corresponds to optimal  $Q^*$  (and optimal policy  $\pi^*$ )

## Backward Pass

Gradient update (with respect to Q-function parameters  $\theta$ ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right] \nabla_{\theta_i} Q(s, a; \theta_i)$$

به صورت تکراری سعی می‌کنیم تا مقادیر Q به مقادیر تارگت ( $y_i$ ) که باید داشته باشند، نزدیک شوند. مقادیر تارگت: اگر تابع Q متناظر با  $Q^*$  بهینه (و سیاست بهینه‌ی  $\pi^*$ ) باشد.

## یادگیری تقویتی عمیق

مطالعه‌ی موردی: انجام بازی‌های آتاری

DEEP Q-LEARNING

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Case Study: Playing Atari Games



**Objective:** Complete the game with the highest score

**State:** Raw pixel inputs of the game state

**Action:** Game controls e.g. Left, Right, Up, Down

**Reward:** Score increase/decrease at each time step

هدف: اتمام بازی با بالاترین امتیاز

حالت: ورودی‌های پیکسلی خام حالت بازی

کنش: کنترل‌های بازی، مثل: چپ، راست، بالا، پایین

پاداش: افزایش/کاهش امتیاز در هر گام زمانی

## شبکه‌ی Q

معماری

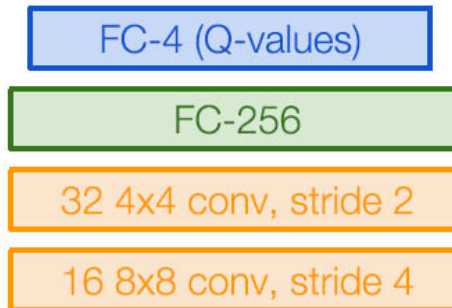
Q-NETWORK: ARCHITECTURE

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Q-network Architecture

$Q(s, a; \theta)$  :  
neural network  
with weights  $\theta$

شبکه‌ی عصبی  
با وزن‌های  $\theta$



**Current state  $s_t$ : 84x84x4 stack of last 4 frames**  
(after RGB->grayscale conversion, downsampling, and cropping)

حالت جاری: یک پشته  $۸۴ \times ۸۴ \times ۴$  از چهار فریم آخر  
(پس از تبدیل رنگی RGB به مقیاس خاکستری، نمونه‌برداری رو به پایین، و کراپ کردن)

## شبکه‌ی Q

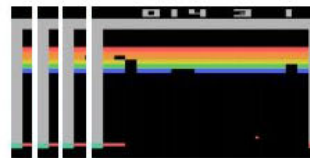
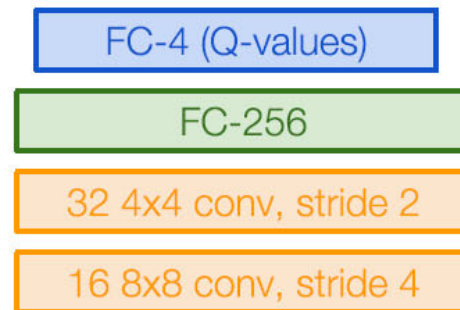
معماری

Q-NETWORK: ARCHITECTURE

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Q-network Architecture

$Q(s, a; \theta)$ :  
neural network  
with weights  $\theta$



← Input: state  $s_t$

**Current state  $s_t$ : 84x84x4 stack of last 4 frames**  
(after RGB->grayscale conversion, downsampling, and cropping)

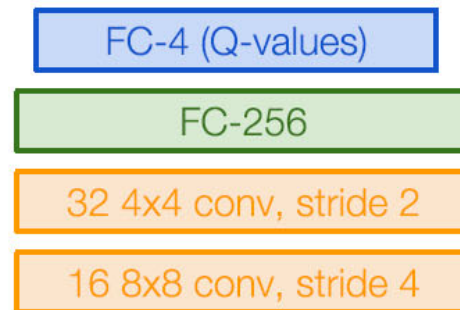
## شبکه‌ی Q

معماری

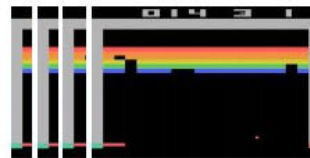
Q-NETWORK: ARCHITECTURE*[Mnih et al. NIPS Workshop 2013; Nature 2015]*

## Q-network Architecture

$Q(s, a; \theta)$ :  
neural network  
with weights  $\theta$



← Familiar conv layers,  
FC layer



**Current state  $s_t$ : 84x84x4 stack of last 4 frames**  
(after RGB->grayscale conversion, downsampling, and cropping)



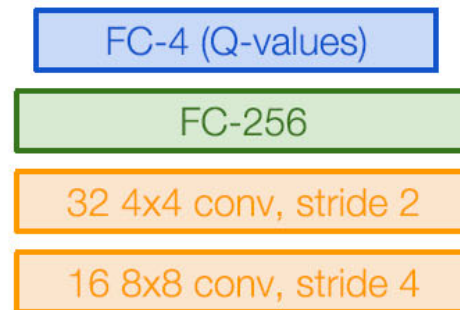
## شبکه‌ی Q

معماری

Q-NETWORK*[Mnih et al. NIPS Workshop 2013; Nature 2015]*

## Q-network Architecture

$Q(s, a; \theta)$ :  
neural network  
with weights  $\theta$



Last FC layer has 4-d output (if 4 actions), corresponding to  $Q(s_t, a_1)$ ,  $Q(s_t, a_2)$ ,  $Q(s_t, a_3)$ ,  $Q(s_t, a_4)$



**Current state  $s_t$ : 84x84x4 stack of last 4 frames**  
(after RGB->grayscale conversion, downsampling, and cropping)

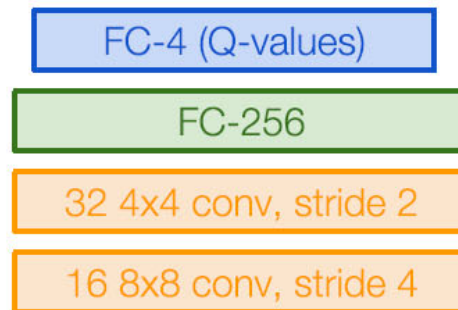
## شبکه‌ی Q

معماری

Q-NETWORK: ARCHITECTURE*[Mnih et al. NIPS Workshop 2013; Nature 2015]*

## Q-network Architecture

$Q(s, a; \theta)$ :  
neural network  
with weights  $\theta$



← Last FC layer has 4-d output (if 4 actions), corresponding to  $Q(s_t, a_1)$ ,  $Q(s_t, a_2)$ ,  $Q(s_t, a_3)$ ,  $Q(s_t, a_4)$

Number of actions between 4-18 depending on Atari game



**Current state  $s_t$ : 84x84x4 stack of last 4 frames**  
(after RGB->grayscale conversion, downsampling, and cropping)

## شبکه‌ی Q

معماری

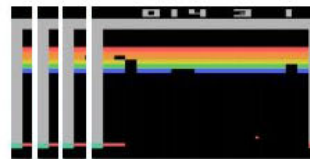
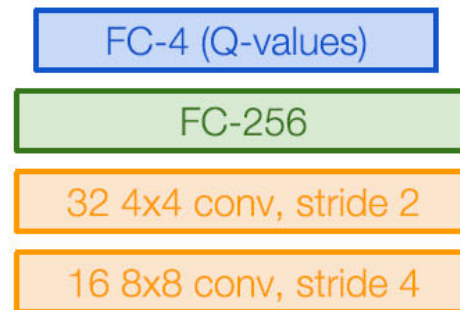
Q-NETWORK: ARCHITECTURE

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Q-network Architecture

$Q(s, a; \theta)$ :  
neural network  
with weights  $\theta$

A single feedforward pass  
to compute Q-values for all  
actions from the current  
state => efficient!



**Current state  $s_t$ : 84x84x4 stack of last 4 frames**  
(after RGB->grayscale conversion, downsampling, and cropping)

← Last FC layer has 4-d  
output (if 4 actions),  
corresponding to  $Q(s_t, a_1)$ ,  $Q(s_t, a_2)$ ,  $Q(s_t, a_3)$ ,  
 $Q(s_t, a_4)$

Number of actions between 4-18  
depending on Atari game

## شبکه‌ی Q

آموزش: تابع اتلاف

Q-NETWORK: TRAINING

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Training the Q-network: Loss function (from before)

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

## Forward Pass

Loss function:  $L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right]$ where  $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$ 

Iteratively try to make the Q-value close to the target value ( $y_i$ ) it should have, if Q-function corresponds to optimal  $Q^*$  (and optimal policy  $\pi^*$ )

## Backward Pass

Gradient update (with respect to Q-function parameters  $\theta$ ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right] \nabla_{\theta_i} Q(s, a; \theta_i)$$

## شبکه‌ی Q

آموزش: بازانجام تجربه

Q-NETWORK: TRAINING: EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

# Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

یادگیری از دسته‌های حاوی نمونه‌های متوالی، مشکل‌ساز است:

- نمونه‌ها همبسته هستند  $\Leftarrow$  یادگیری ناکارآمد
- پارامترهای شبکه‌ی Q فعلی، نمونه‌های آموزشی بعدی را تعیین می‌کنند.  $\Leftarrow$  می‌تواند منجر به حلقه‌های فیدبک بد شود. (برای مثال، اگر کنش ماکزیمم‌ساز حرکت به چپ باشد، نمونه‌های آموزشی مغلوب نمونه‌های دست چپ خواهند شد.)

## شبکه‌ی Q

آموزش: بازانجام تجربه

Q-NETWORK: TRAINING: EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

# Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**

- Continually update a **replay memory** table of transitions  $(s_t, a_t, r_t, s_{t+1})$  as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

مشکلات فوق با استفاده از بازانجام تجربه رفع می‌شود:

- یک حافظه‌ی بازانجام (جدولی از گذارهای  $(s_t, a_t, r_t, s_{t+1})$ ) در حین انجام اپیزودهای بازی (تجربه) مدام به‌روزرسانی می‌شود.
- شبکه‌ی Q بر روی مینی‌بچ‌های تصادفی از گذارها از حافظه‌ی بازانجام آموزش داده می‌شود (به جای نمونه‌های متوالی).

## شبکه‌ی Q

آموزش: بازانجام تجربه

Q-NETWORK: TRAINING: EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

# Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:

- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**

- Continually update a **replay memory** table of transitions  $(s_t, a_t, r_t, s_{t+1})$  as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

هر گذار می‌تواند در چند به‌روزرسانی وزن مشارکت داشته باشد.  
 ← کارآمدی بالاتر داده‌ها

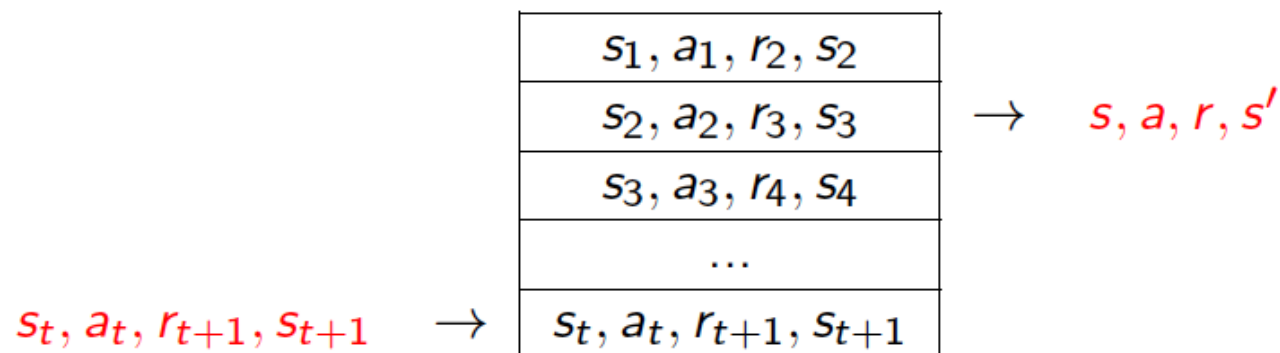
Each transition can also contribute  
 to multiple weight updates  
 => greater data efficiency

## شبکه‌ی Q

آموزش: بازانجام تجربه

Q-NETWORK: TRAINING: EXPERIENCE REPLAY

To remove correlations, build data-set from agent's own experience



برای حذف همبستگی‌ها، از تجربه‌ی شخصی عامل یک مجموعه داده می‌سازیم.



## الگوریتم یادگیری Q عمیق

با بازانجام تجربه

DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

**Algorithm 1** Deep Q-learning with Experience ReplayInitialize replay memory  $\mathcal{D}$  to capacity  $N$ Initialize action-value function  $Q$  with random weights**for** episode = 1,  $M$  **do**    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$     **for**  $t = 1, T$  **do**        With probability  $\epsilon$  select a random action  $a_t$         otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$         Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$         Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$         Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$         Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$         Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$         Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3    **end for****end for**

## الگوریتم یادگیری Q عمیق

با بازانجام تجربه

DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

**Algorithm 1** Deep Q-learning with Experience Replay

---

```

Initialize replay memory  $\mathcal{D}$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode = 1,  $M$  do
  Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$ 
  for  $t = 1, T$  do
    With probability  $\epsilon$  select a random action  $a_t$ 
    otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ 
    Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
    Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
    Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$ 
    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$ 
    Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ 
    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3
  end for
end for

```

---

← Initialize replay memory, Q-network

مقداردهی آغازین حافظه‌ی بازانجام،  
شبکه‌ی Q

## الگوریتم یادگیری Q عمیق

با بازانجام تجربه

DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

**Algorithm 1** Deep Q-learning with Experience ReplayInitialize replay memory  $\mathcal{D}$  to capacity  $N$ Initialize action-value function  $Q$  with random weights**for** episode = 1,  $M$  **do**    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$     **for**  $t = 1, T$  **do**        With probability  $\epsilon$  select a random action  $a_t$         otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$         Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$         Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$         Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$         Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$         Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$         Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3    **end for****end for**

← Play M episodes (full games)

انجام M اپیزود از بازی (بازی‌های کامل)

## الگوریتم یادگیری Q عمیق

با بازانجام تجربه

### DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

---

### Algorithm 1 Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$  ←

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

**end for**

**end for**

---

Initialize state  
(starting game  
screen pixels) at the  
beginning of each  
episode

مقداردهی آغازین حالت  
(پیکسل‌های صفحه‌ی  
بازی آغازین) در شروع  
هر اپیزود

## الگوریتم یادگیری Q عمیق

با بازانجام تجربه

DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

**Algorithm 1** Deep Q-learning with Experience ReplayInitialize replay memory  $\mathcal{D}$  to capacity  $N$ Initialize action-value function  $Q$  with random weights**for** episode = 1,  $M$  **do**  Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$   **for**  $t = 1, T$  **do**    With probability  $\epsilon$  select a random action  $a_t$     otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$     Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$     Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$     Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$     Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$     Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$     Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3  **end for****end for**For each timestep  $t$   
of the gameبرای هر گام زمانی  $t$   
از بازی

# الگوریتم یادگیری Q عمیق

با بازانجام تجربه

## DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

### Algorithm 1 Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$   
 Initialize action-value function  $Q$  with random weights  
**for** episode = 1,  $M$  **do**  
   Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$   
   **for**  $t = 1, T$  **do**  
     With probability  $\epsilon$  select a random action  $a_t$   
     otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$   
     Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$   
     Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$   
     Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$   
     Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$   
     Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$   
     Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3  
   **end for**  
**end for**

---

← With small probability, select a random action (explore), otherwise select greedy action from current policy

با یک احتمال کوچک،  
 یک کنش تصادفی (اکتشاف)  
 انتخاب می‌کنیم؛  
 وگرنه، کنش حریصانه از  
 سیاست فعلی انتخاب می‌شود.

## الگوریتم یادگیری Q عمیق

با بازانجام تجربه

DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

**Algorithm 1** Deep Q-learning with Experience ReplayInitialize replay memory  $\mathcal{D}$  to capacity  $N$ Initialize action-value function  $Q$  with random weights**for** episode = 1,  $M$  **do**  Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$   **for**  $t = 1, T$  **do**    With probability  $\epsilon$  select a random action  $a_t$     otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$     Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$     Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$     Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$     Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$     Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$     Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3  **end for****end for**

Take the action ( $a_t$ ),  
and observe the  
reward  $r_t$  and next  
state  $s_{t+1}$

کنش ( $a_t$ ) را انتخاب می‌کنیم  
و پاداش  $r_t$  و  
حالت بعدی  $s_{t+1}$   
را مشاهده می‌کنیم.

# الگوریتم یادگیری Q عمیق

با بازانجام تجربه

## DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

---

### Algorithm 1 Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

**end for**

**end for**

---

Store transition in  
replay memory

گذار را در  
حافظه‌ی بازانجام  
ذخیره می‌کنیم.



## الگوریتم یادگیری Q عمیق

با بازانجام تجربه

DEEP Q-LEARNING WITH EXPERIENCE REPLAY

[Mnih et al. NIPS Workshop 2013; Nature 2015]

## Putting it together: Deep Q-Learning with Experience Replay

**Algorithm 1** Deep Q-learning with Experience Replay

---

```

Initialize replay memory  $\mathcal{D}$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode = 1,  $M$  do
  Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$ 
  for  $t = 1, T$  do
    With probability  $\epsilon$  select a random action  $a_t$ 
    otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ 
    Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
    Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
    Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$ 
    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$ 
    Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ 
    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3
  end for
end for

```

---

Experience Replay:  
Sample a random minibatch of transitions from replay memory and perform a gradient descent step

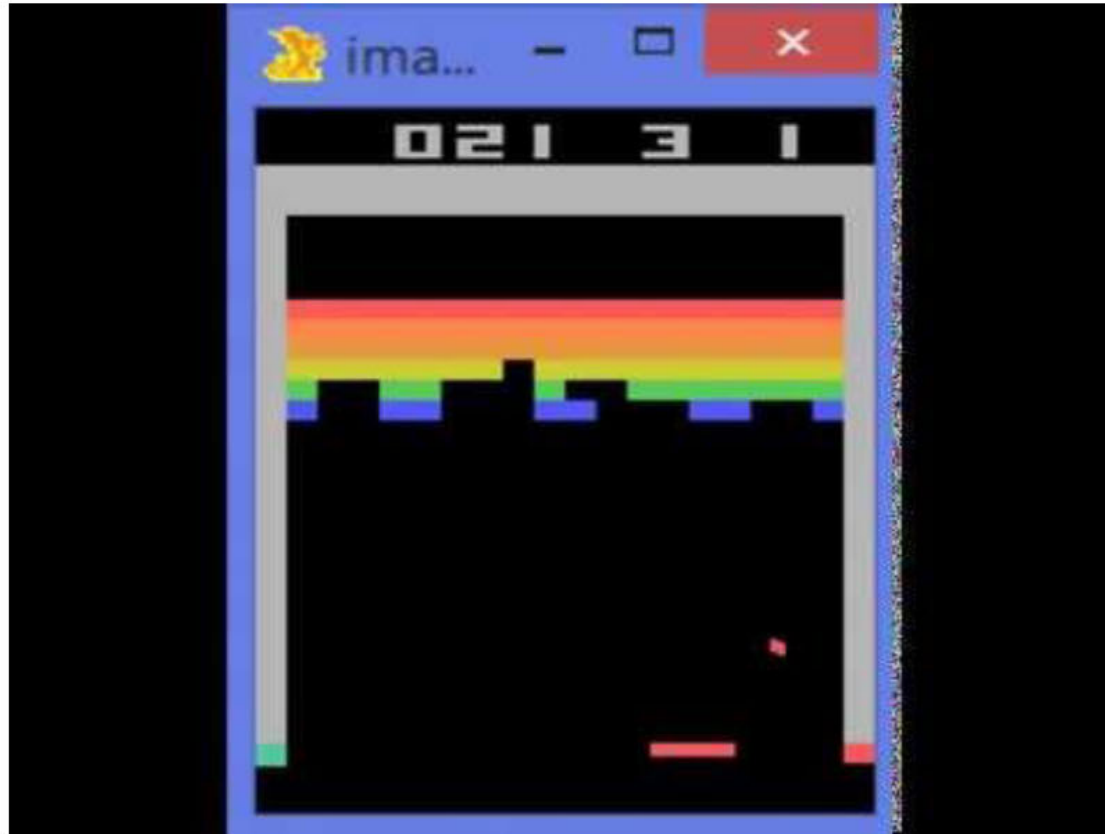
بازانجام تجربه:

یک مینی بچ تصادفی از گذارها از حافظه‌ی بازانجام نمونه‌گیری می‌کنیم و یک گام کاهش گرادیان روی آن انجام می‌دهیم.

## الگوریتم یادگیری Q عمیق

با بازانجام تجربه: مثال

### DEEP Q-LEARNING WITH EXPERIENCE REPLAY: EXAMPLE



<https://www.youtube.com/watch?v=V1eYniJ0Rnk>

Video by Károly Zsolnai-Fehér. Reproduced with permission.

## الگوریتم یادگیری Q عمیق

گرایان‌های سیاست

DEEP Q-LEARNING: POLICY GRADIENTS

# Policy Gradients

What is a problem with Q-learning?

The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

مشکل Q-Learning چیست؟

تابع Q می‌تواند بسیار پیچیده باشد!

مثال: یک ربات که یک شیء را چنگ می‌زند، دارای حالتی بسیار ابعاد-بالا است  
← یادگیری مقدار دقیق هر جفت (حالت، کنش) سخت است.

## الگوریتم یادگیری Q عمیق

گرایان‌های سیاست

### DEEP Q-LEARNING: POLICY GRADIENTS

# Policy Gradients

What is a problem with Q-learning?

The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

But the policy can be much simpler: just close your hand

Can we learn a policy directly, e.g. finding the best policy from a collection of policies?

مشکل Q-Learning چیست؟

تابع Q می‌تواند بسیار پیچیده باشد!

اما سیاست می‌تواند بسیار ساده‌تر باشد: دقیقاً نزدیک دست ما.

آیا می‌توانیم یک سیاست را مستقیماً یاد بگیریم؟ (مثلاً: یافتن بهترین سیاست از یک گرایه از سیاست‌ها)

## الگوریتم یادگیری Q عمیق

گرادیان‌های سیاست

### DEEP Q-LEARNING: POLICY GRADIENTS

# Policy Gradients

Formally, let's define a class of parameterized policies:  $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$   
 به صورت صوری، یک کلاس از سیاست‌های پارامتری شده را تعریف می‌کنیم.

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

برای هر سیاست، ارزش آن را تعریف می‌کنیم.

# الگوریتم یادگیری Q عمیق

گرادیان‌های سیاست

## DEEP Q-LEARNING: POLICY GRADIENTS

# Policy Gradients

Formally, let's define a class of parameterized policies:  $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$   
 به صورت صوری، یک کلاس از سیاست‌های پارامتری شده را تعریف می‌کنیم.

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

برای هر سیاست، ارزش آن را تعریف می‌کنیم.

We want to find the optimal policy  $\theta^* = \arg \max_{\theta} J(\theta)$

می‌خواهیم سیاست بهینه را پیدا کنیم.

How can we do this?

چگونه می‌توانیم این کار را انجام دهیم؟

# الگوریتم یادگیری Q عمیق

گرادیان‌های سیاست

## DEEP Q-LEARNING: POLICY GRADIENTS

# Policy Gradients

Formally, let's define a class of parameterized policies:  $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$   
 به صورت صوری، یک کلاس از سیاست‌های پارامتری شده را تعریف می‌کنیم.

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

برای هر سیاست، ارزش آن را تعریف می‌کنیم.

We want to find the optimal policy  $\theta^* = \arg \max_{\theta} J(\theta)$

می‌خواهیم سیاست بهینه را پیدا کنیم.

How can we do this?

Gradient ascent on policy parameters!

با «افزایش گرادیانی» بر روی پارامترهای سیاست!

## الگوریتم تقویت

برای یادگیری سیاست بهینه

### REINFORCE ALGORITHM

# REINFORCE algorithm

Mathematically, we can write:

به صورت ریاضی می‌توانیم بنویسیم:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \int_{\tau} r(\tau) p(\tau; \theta) d\tau \end{aligned}$$

که در آن  $r(\tau)$  پاداش یک تراجکتوری است:

Where  $r(\tau)$  is the reward of a trajectory  $\tau = (s_0, a_0, r_0, s_1, \dots)$



## الگوریتم تقویت

REINFORCE ALGORITHM

## REINFORCE algorithm

Expected reward: 
$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$$
$$= \int_{\tau} r(\tau) p(\tau; \theta) d\tau$$

امید پاداش:

## الگوریتم تقویت

REINFORCE ALGORITHM

## REINFORCE algorithm

Expected reward:  $J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$

امید پاداش:

$$= \int_{\tau} r(\tau) p(\tau; \theta) d\tau$$

Now let's differentiate this:  $\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau$

از آن مشتق می‌گیریم:

## الگوریتم تقویت

REINFORCE ALGORITHM

## REINFORCE algorithm

Expected reward:  $J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$

امید پاداش:

$$= \int_{\tau} r(\tau) p(\tau; \theta) d\tau$$

Now let's differentiate this:  $\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau$

Intractable! Gradient of an expectation is problematic when  $p$  depends on  $\theta$

دنبال ناپذیر است:

گرادیان یک امید مشکل ساز است  
وقتی توزیع  $p$  وابسته به پارامتر  $\theta$  است.

## الگوریتم تقویت

REINFORCE ALGORITHM

## REINFORCE algorithm

Expected reward:  $J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$

امید پاداش:

$$= \int_{\tau} r(\tau) p(\tau; \theta) d\tau$$

Now let's differentiate this:  $\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau$

Intractable! Gradient of an expectation is problematic when  $p$  depends on  $\theta$

However, we can use a nice trick:  $\nabla_{\theta} p(\tau; \theta) = p(\tau; \theta) \frac{\nabla_{\theta} p(\tau; \theta)}{p(\tau; \theta)} = p(\tau; \theta) \nabla_{\theta} \log p(\tau; \theta)$

با این وجود، می‌توانیم از یک ترفند ظریف استفاده کنیم.

## الگوریتم تقویت

REINFORCE ALGORITHM

## REINFORCE algorithm

Expected reward:  $J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$

امید پاداش:

$$= \int_{\tau} r(\tau) p(\tau; \theta) d\tau$$

Now let's differentiate this:  $\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau$

Intractable! Gradient of an expectation is problematic when  $p$  depends on  $\theta$

However, we can use a nice trick:  $\nabla_{\theta} p(\tau; \theta) = p(\tau; \theta) \frac{\nabla_{\theta} p(\tau; \theta)}{p(\tau; \theta)} = p(\tau; \theta) \nabla_{\theta} \log p(\tau; \theta)$

If we inject this back:

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_{\theta} \log p(\tau; \theta)] \end{aligned}$$

با این وجود، می‌توانیم از یک ترفند ظریف استفاده کنیم.

Can estimate with Monte Carlo sampling

با جایگذاری در قبلی؛ می‌توانیم با استفاده از نمونه‌گیری مونت کارلو آن را تخمین بزنیم.

## الگوریتم تقویت

REINFORCE ALGORITHM

## REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: 
$$p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$$

می‌توانیم آن کمیت‌ها را بدون دانستن احتمالات گذار محاسبه کنیم؟

## الگوریتم تقویت

REINFORCE ALGORITHM

# REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have:  $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$

Thus:  $\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_{\theta}(a_t | s_t)$

REINFORCE ALGORITHM

## REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have:  $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$

Thus:  $\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_{\theta}(a_t | s_t)$

And when differentiating:  $\nabla_{\theta} \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

Doesn't depend on transition probabilities!

به احتمالات گذار وابسته نیست!



## الگوریتم تقویت

REINFORCE ALGORITHM

## REINFORCE algorithm

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_{\theta} \log p(\tau; \theta)]\end{aligned}$$

Can we compute those quantities without knowing the transition probabilities?

We have:  $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$

Thus:  $\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_{\theta}(a_t | s_t)$

And when differentiating:  $\nabla_{\theta} \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

Doesn't depend on transition probabilities!

Therefore when sampling a trajectory  $\tau$ , we can estimate  $J(\theta)$  with

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

بنابراین، هنگام نمونه‌گیری از یک تراجکتوری  $\tau$ ، می‌توانیم  $J(\theta)$  را با رابطه‌ی فوق تخمین بزنیم.

## الگوریتم تقویت

شهود

REINFORCE ALGORITHM: INTUITION

## Intuition

Gradient estimator:  $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$  تخمین زن گرادیان:

**Interpretation:**

- If  $r(\tau)$  is high, push up the probabilities of the actions seen
- If  $r(\tau)$  is low, push down the probabilities of the actions seen

**تفسیر:**

- \* اگر  $r(\tau)$  بالا باشد، احتمالات کنش‌های دیده شده را بالا می‌بریم.
- \* اگر  $r(\tau)$  پایین باشد، احتمالات کنش‌های دیده شده را پایین می‌بریم.

## الگوریتم تقویت

شهود

REINFORCE ALGORITHM: INTUITION

## Intuition

Gradient estimator: 
$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

**Interpretation:**

- If  $r(\tau)$  is high, push up the probabilities of the actions seen
- If  $r(\tau)$  is low, push down the probabilities of the actions seen

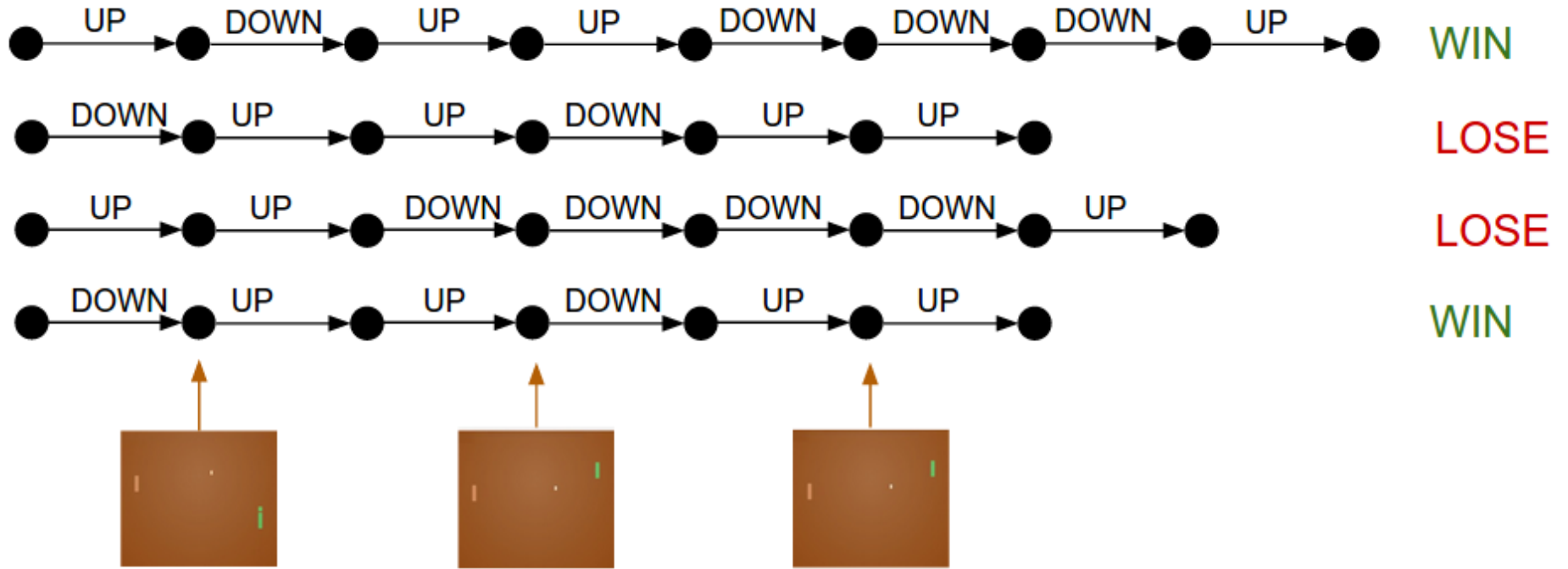
Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

ممکن است با نگاه ساده‌انگارانه بگوییم که  
اگر یک تراجکتوری خوب باشد، آن‌گاه همه‌ی کنش‌های آن خوب است.  
اما در امید ریاضی، از آن میانگین‌گیری می‌شود.

# الگوریتم تقویت

شهود

## REINFORCE ALGORITHM: INTUITION



## الگوریتم تقویت

شهود

REINFORCE ALGORITHM: INTUITION

## Intuition

Gradient estimator: 
$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

**Interpretation:**

- If  $r(\tau)$  is high, push up the probabilities of the actions seen
- If  $r(\tau)$  is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

**However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?**

با این وجود، این کار از واریانس بالا رنج می‌برد،  
 زیرا، انتساب اعتبار [به کنش‌های یک تراجکتوری] واقعاً سخت است.  
 آیا می‌توانیم به تخمین‌زن کمک کنیم؟

## الگوریتم تقویت

کاهش واریانس

REINFORCE ALGORITHM: VARIANCE REDUCTION

## Variance reduction

Gradient estimator:  $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$  تخمین زن گرادیان:

## الگوریتم تقویت

کاهش واریانس

REINFORCE ALGORITHM: VARIANCE REDUCTION

## Variance reduction

Gradient estimator:  $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$  تخمین زن گرادیان:

**First idea:** Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

**ایدهی اول:**

احتمالات یک کنش دیده شده را بالا ببریم، فقط توسط پاداش تجمعی آینده با شروع از آن حالت (به جای تنها پاداش فعلی).

## الگوریتم تقویت

کاهش واریانس

REINFORCE ALGORITHM: VARIANCE REDUCTION

## Variance reduction

Gradient estimator:  $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

**First idea:** Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

**Second idea:** Use discount factor  $\gamma$  to ignore delayed effects

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

ایده‌ی دوم:

استفاده از فاکتور تخفیف  $\gamma$  برای نادیده گرفتن اثرات تاخیر یافته



## الگوریتم تقویت

کاهش واریانس: خط پایه

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## Variance reduction: Baseline

**Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

**مشکل:**

مقدار ارزش خام یک تراجکتوری لزوماً معنادار نیست. برای مثال، اگر پاداش‌ها همگی مثبت باشند، بالا بردن احتمالات کنش‌ها را ادامه می‌دهیم.

## الگوریتم تقویت

کاهش واریانس: خط پایه

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## Variance reduction: Baseline

**Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

**What is important then?** Whether a reward is better or worse than what you expect to get

حال، چه چیزی مهم است؟  
اینکه یک پاداش بهتر یا بدتر از چیزی است که انتظار داریم به دست آورده شود.

## الگوریتم تقویت

کاهش واریانس: خط پایه

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## Variance reduction: Baseline

**Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

**What is important then?** Whether a reward is better or worse than what you expect to get

**Idea:** Introduce a baseline function dependent on the state. Concretely, estimator is now:

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

ایده:

وارد کردن یک تابع خط پایه وابسته به حالت. انضماماً، تخمین زن اکنون به صورت فوق می شود.

## الگوریتم تقویت

کاهش واریانس : خط پایه : انتخاب

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## How to choose the baseline?

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

یک خط پایه‌ی ساده:

میانگین متحرک ثابت از پاداش‌های تجربه شده تاکنون از روی همه‌ی تراجکتوری‌ها.

## الگوریتم تقویت

کاهش واریانس : خط پایه : انتخاب

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## How to choose the baseline?

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in “Vanilla REINFORCE”

تکنیک‌های کاهش واریانس دیده شده تاکنون، معمولاً در الگوریتم REINFORCE ساده استفاده می‌شوند.

## الگوریتم تقویت

کاهش واریانس : خط پایه : انتخاب

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

یک خط پایه‌ی بهتر:

می‌خواهیم احتمال یک کنش از یک حالت را بالا ببریم،  
اگر این کنش بهتر از مقدار مورد انتظار (امید) آنچه باید از آن حالت به دست آوریم باشد.

## الگوریتم تقویت

کاهش واریانس: خط پایه: انتخاب

REINFORCE ALGORITHM: VARIANCE REDUCTION

## How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

این شما را به یاد چه می‌اندازد؟

## الگوریتم تقویت

کاهش واریانس : خط پایه : انتخاب

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

این شما را به یاد چه می اندازد؟

A: Q-function and value function!

تابع Q و تابع ارزش



## الگوریتم تقویت

کاهش واریانس: خط پایه: انتخاب

### REINFORCE ALGORITHM: VARIANCE REDUCTION

## How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action  $a_t$  in a state  $s_t$  if  $Q^\pi(s_t, a_t) - V^\pi(s_t)$  is large. On the contrary, we are unhappy with an action if it's small.

به طور شهودی، با یک کنش  $a_t$  در یک حالت  $s_t$  خوشحال می شویم اگر مقدار

$$Q^\pi(s_t, a_t) - V^\pi(s_t)$$

بزرگ باشد.

برخلاف آن، با یک کنش ناراحت می شویم اگر این مقدار کوچک باشد.

## الگوریتم تقویت

کاهش واریانس: خط پایه: انتخاب

REINFORCE ALGORITHM: VARIANCE REDUCTION

## How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action  $a_t$  in a state  $s_t$  if  $Q^\pi(s_t, a_t) - V^\pi(s_t)$  is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator: 
$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

با استفاده از این، به تخمین زدن فوق دست پیدا می‌کنیم.

ACTOR-CRITIC ALGORITHM

# Actor-Critic Algorithm

**Problem:** we don't know  $Q$  and  $V$ . Can we learn them?

مسئله:

$Q$  و  $V$  را نمی‌دانیم. آیا می‌توانیم آنها را یاد بگیریم؟

## الگوریتم بازیگر-نقاد

ACTOR-CRITIC ALGORITHM

# Actor-Critic Algorithm

**Problem:** we don't know  $Q$  and  $V$ . Can we learn them?

**Yes**, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

**بله:**

با استفاده از یادگیری  $Q$ !  
می‌توانیم گرادینان‌های سیاست و یادگیری  $Q$  را ترکیب کنیم؛  
با آموزش هر دوی یک **بازیگر** (سیاست) و یک **نقاد** (تابع  $Q$ )

## الگوریتم بازیگر-نقاد

ACTOR-CRITIC ALGORITHM

# Actor-Critic Algorithm

**Problem:** we don't know  $Q$  and  $V$ . Can we learn them?

**Yes**, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay

- بازیگر تصمیم می‌گیرد که کدام کنش را انتخاب کند، و نقاد به بازیگر می‌گوید که کنش او چه قدر خوب بوده است و چگونه باید آن را اصلاح کند.
- همچنین وظیفه‌ی نقاد را سبک می‌کند، به این صورت که او تنها باید ارزش جفت (حالت، کنش) تولید شده توسط سیاست را یاد بگیرد.
- همچنین می‌تواند ترفندهای یادگیری  $Q$  مانند «بازانجام تجربه» را دخیل کند.

ACTOR-CRITIC ALGORITHM

# Actor-Critic Algorithm

**Problem:** we don't know Q and V. Can we learn them?

**Yes,** using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay
- **Remark:** we can define by the **advantage function** how much an action was better than expected

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

**توضیح:** می‌توانیم با یک تابع برتری تعریف کنیم که یک کنش بهتر از مقدار مورد انتظار بوده است.

## الگوریتم بازیگر-نقاد

شبه کد

ACTOR-CRITIC ALGORITHM

# Actor-Critic Algorithm

Initialize policy parameters  $\theta$ , critic parameters  $\phi$ **For** iteration=1, 2 ... **do**

Sample m trajectories under the current policy

 $\Delta\theta \leftarrow 0$ **For** i=1, ..., m **do****For** t=1, ..., T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_{t'}^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi \|A_t^i\|^2$$

$$\theta \leftarrow \alpha \Delta\theta$$

$$\phi \leftarrow \beta \Delta\phi$$

**End for**

## الگوریتم تقویت در کنش

مدل توجه بازگشتی

### RECURRENT ATTENTION MODEL (RAM)

## REINFORCE in action: Recurrent Attention Model (RAM)

**Objective:** Image Classification

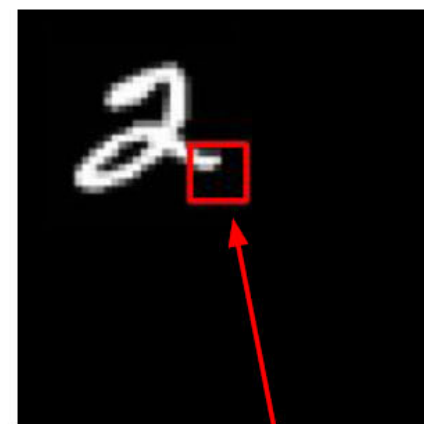
Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

**State:** Glimpses seen so far

**Action:** (x,y) coordinates (center of glimpse) of where to look next in image

**Reward:** 1 at the final timestep if image correctly classified, 0 otherwise



glimpse

**هدف:** طبقه بندی تصویر

یک دنباله از گلیمپس‌ها که به طور انتخابی بر روی نواحی یک تصویر تمرکز می‌کنند، را دریافت می‌کنیم تا کلاس را پیش‌بینی کنیم.

\* الهام از ادراک انسان و حرکات چشم، \* ذخیره‌ی منابع محاسباتی؛ مقیاس‌پذیری، قادر به نادیده گرفتن درهم‌ریختگی‌ها و نواحی نامربوط تصویر

**حالت:** گلیمپس‌های تاکنون دیده شده

**کنش:** مختصات مرکز گلیمپس بعدی که باید در تصویر به آن نگاه کنیم.

**پاداش:** ۱ در آخرین گام زمانی اگر تصویر درست طبقه بندی شود، وگرنه صفر.

[Mnih et al. 2014]



## الگوریتم تقویت در کنش

مدل توجه بازگشتی

### RECURRENT ATTENTION MODEL (RAM)

## REINFORCE in action: Recurrent Attention Model (RAM)

**Objective:** Image Classification

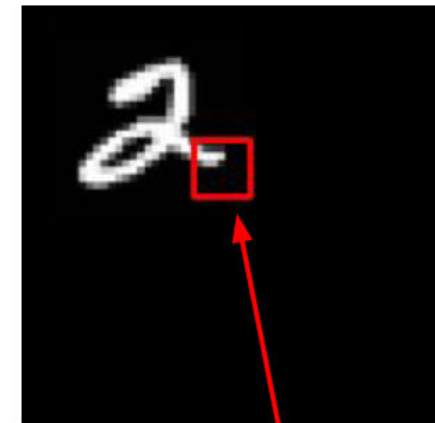
Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

**State:** Glimpses seen so far

**Action:** (x,y) coordinates (center of glimpse) of where to look next in image

**Reward:** 1 at the final timestep if image correctly classified, 0 otherwise



glimpse

Glimpsing is a non-differentiable operation => learn policy for how to take glimpse actions using REINFORCE  
Given state of glimpses seen so far, use RNN to model the state and output next action

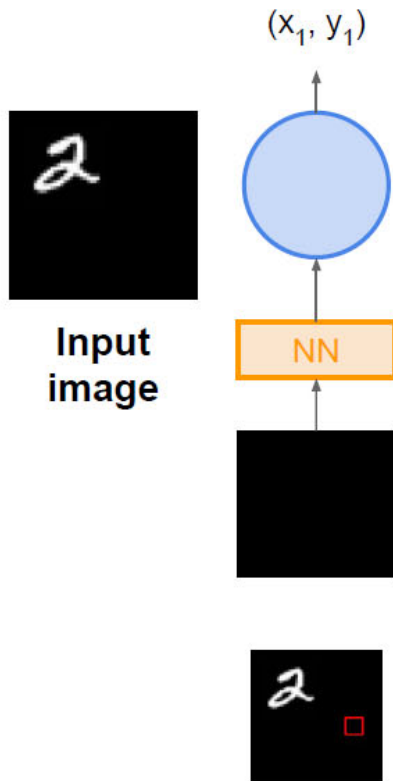
گلیمپسینگ یک عملیات مشتق پذیر نیست ←  
یادگیری سیاست برای چگونگی انتخاب کنش‌های گلیمپس با استفاده از REINFORCE با داشتن حالت گلیمپس‌های تاکنون دیده شده،  
استفاده از RNN برای مدل‌سازی حالت و خروجی کنش بعدی

# الگوریتم تقویت در کنش

مدل توجه بازگشتی

## RECURRENT ATTENTION MODEL (RAM)

### REINFORCE in action: Recurrent Attention Model (RAM)



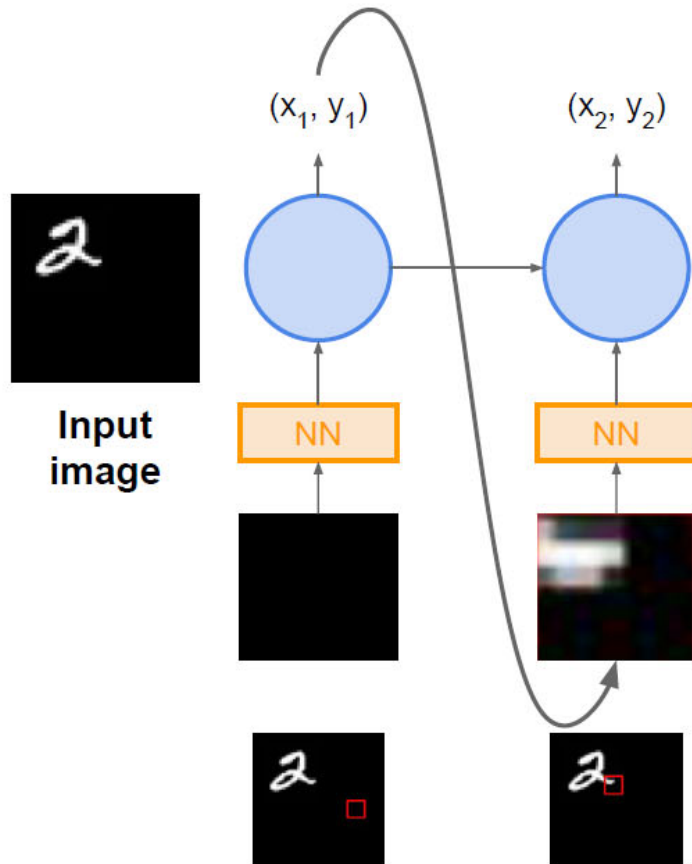
[Mnih et al. 2014]

## الگوریتم تقویت در کنش

مدل توجه بازگشتی

RECURRENT ATTENTION MODEL (RAM)

## REINFORCE in action: Recurrent Attention Model (RAM)



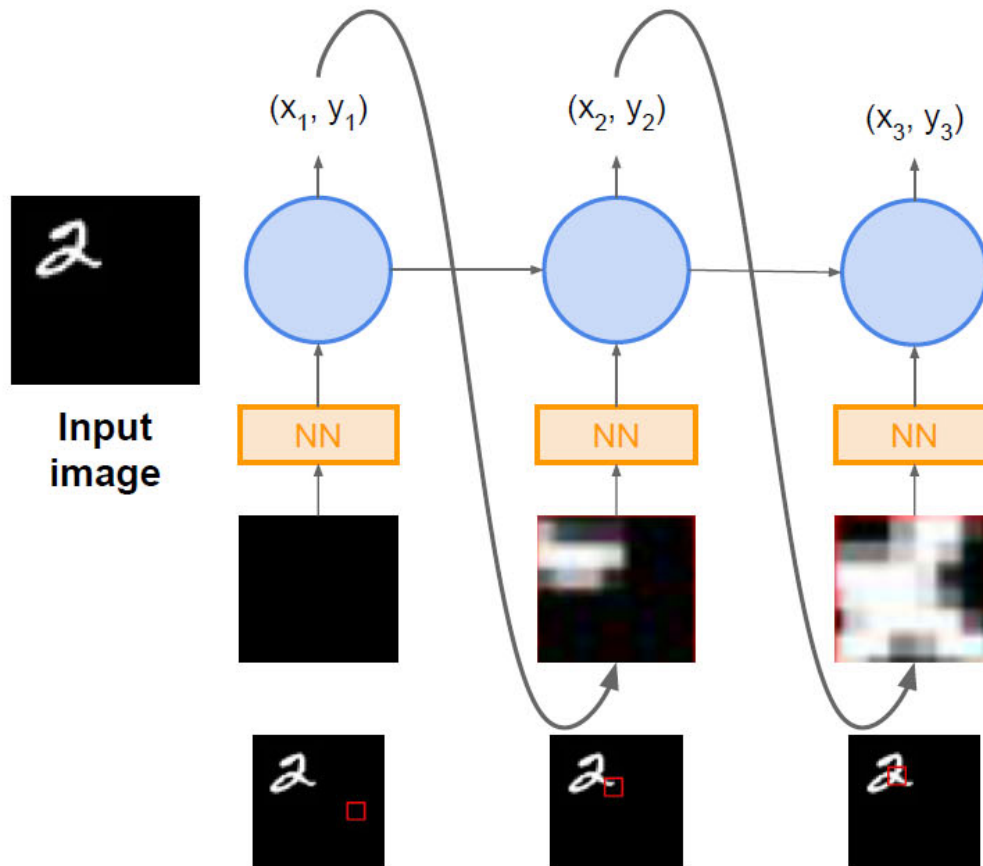
[Mnih et al. 2014]

## الگوریتم تقویت در کنش

مدل توجه بازگشتی

RECURRENT ATTENTION MODEL (RAM)

## REINFORCE in action: Recurrent Attention Model (RAM)



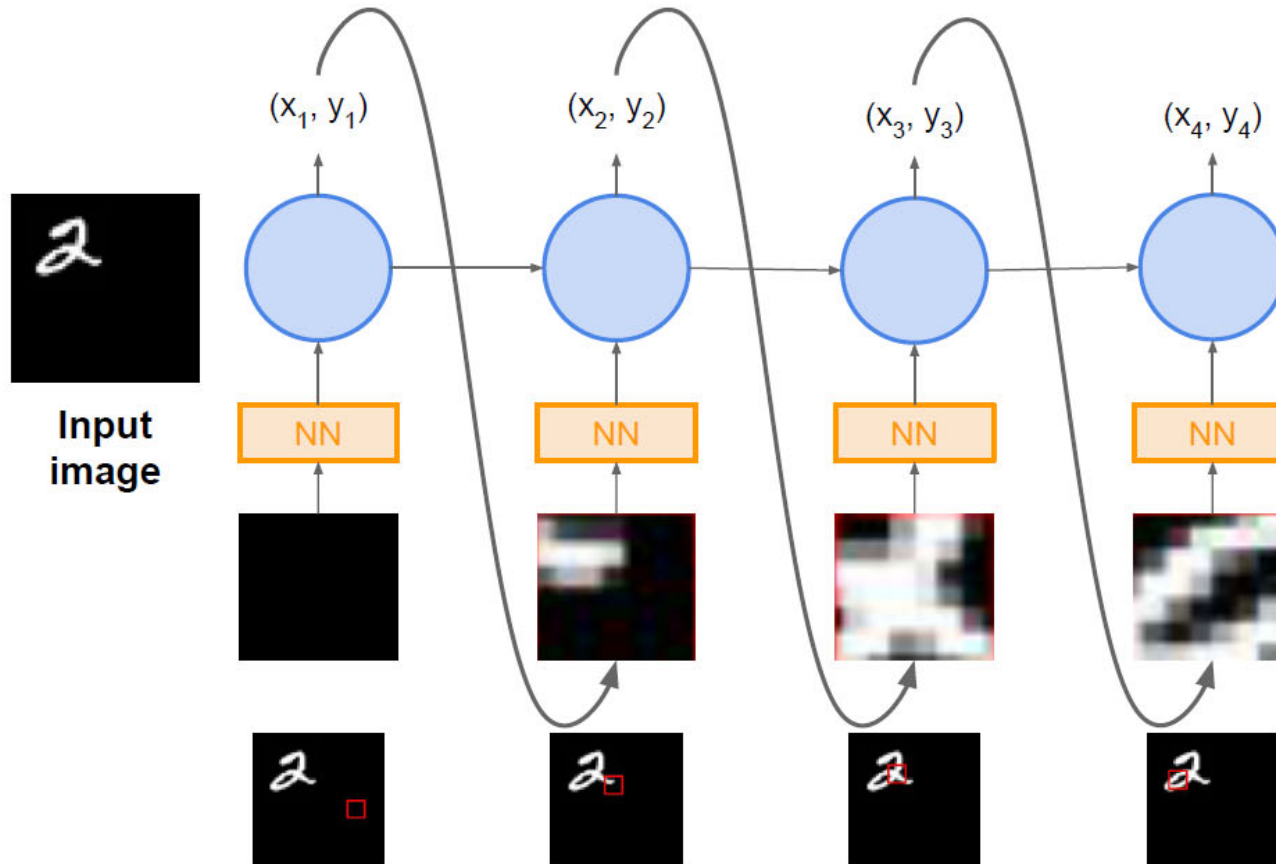
[Mnih et al. 2014]

## الگوریتم تقویت در کنش

مدل توجه بازگشتی

RECURRENT ATTENTION MODEL (RAM)

## REINFORCE in action: Recurrent Attention Model (RAM)



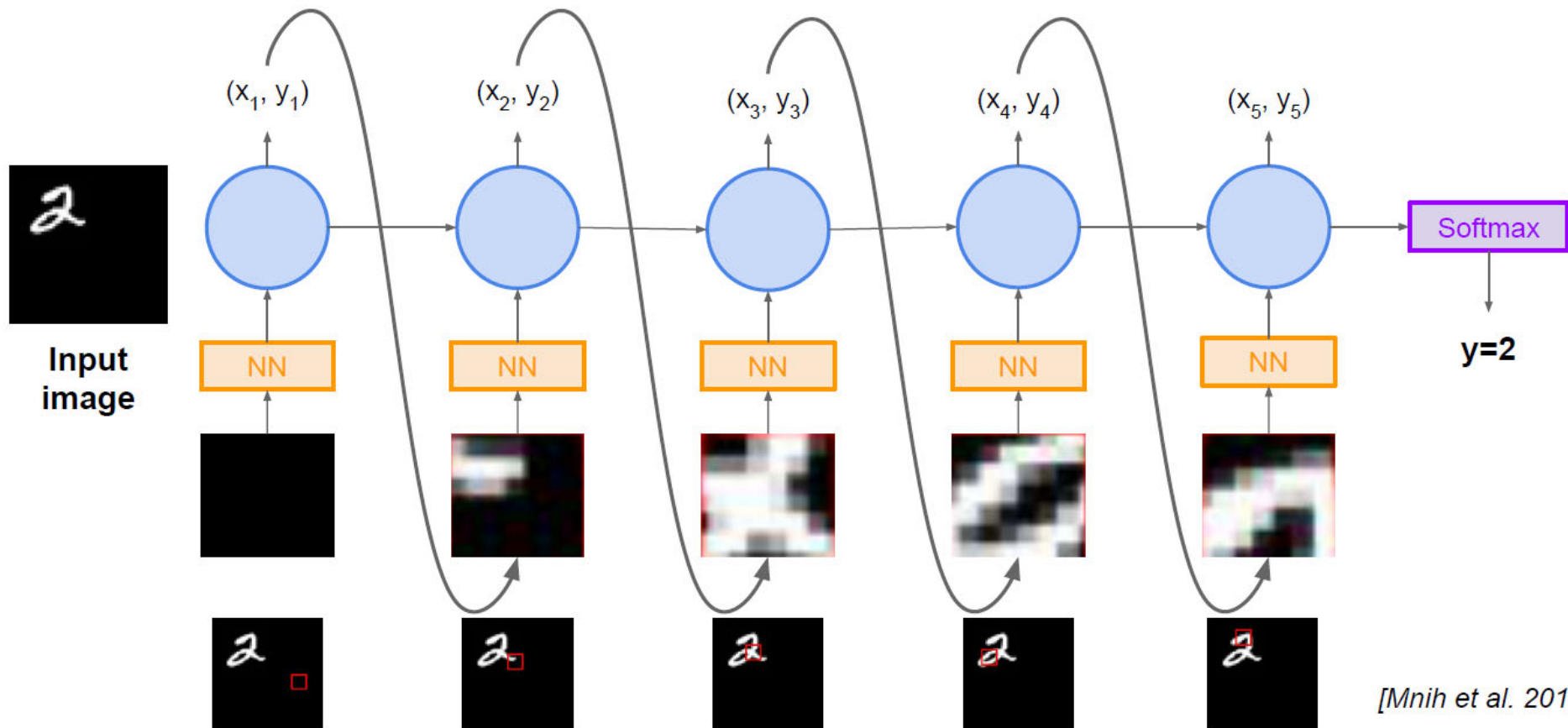
[Mnih et al. 2014]

## الگوریتم تقویت در کنش

مدل توجه بازگشتی

RECURRENT ATTENTION MODEL (RAM)

## REINFORCE in action: Recurrent Attention Model (RAM)



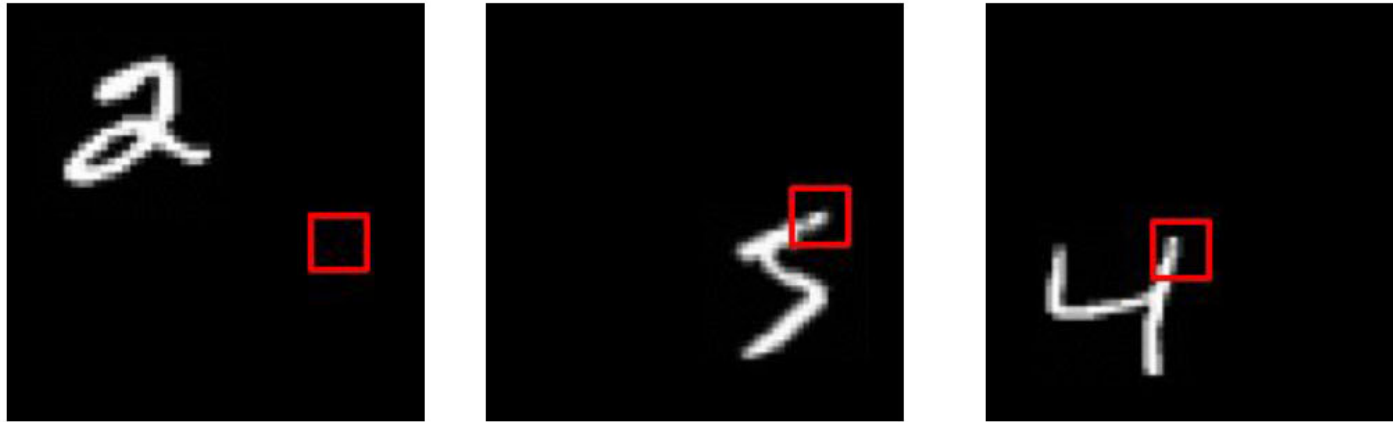
[Mnih et al. 2014]

## الگوریتم تقویت در کنش

مدل توجه بازگشتی

### RECURRENT ATTENTION MODEL (RAM)

## REINFORCE in action: Recurrent Attention Model (RAM)



Has also been used in many other tasks including fine-grained image recognition, image captioning, and visual question-answering!

همچنین استفاده شده در بسیاری از وظایف دیگر، شامل: بازشناسی تصویر دانه-ریز، عنوان‌گذاری تصاویر، و پاسخ به پرسش دیداری!

Figures copyright Daniel Levy, 2017. Reproduced with permission.

[Mnih et al. 2014]

## الگوریتم تقویت

بیشتر در مورد گرادیان‌های سیاست: AlphaGo

## More policy gradients: AlphaGo

### AlphaGo [Nature 2016]:

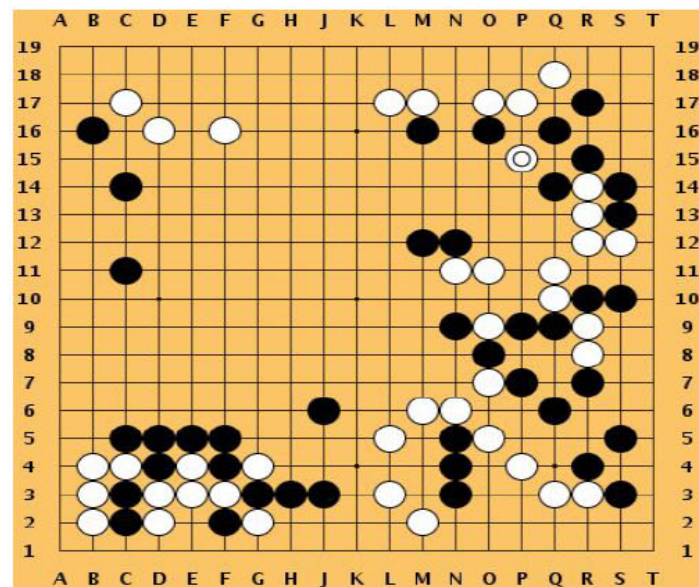
- Required many engineering tricks
- Bootstrapped from human play
- Beat 18-time world champion Lee Sedol

### AlphaGo Zero [Nature 2017]:

- Simplified and elegant version of AlphaGo
- No longer bootstrapped from human play
- Beat (at the time) #1 world ranked Ke Jie

### Alpha Zero: Dec. 2017

- Generalized to beat world champion programs on chess and shogi as well



[This image is CC0 public domain](#)



## یادگیری تقویتی عمیق

## خلاصه

## Summary


- **Policy gradients:** very general but suffer from high variance so requires a lot of samples. **Challenge:** sample-efficiency
- **Q-learning:** does not always work but when it works, usually more sample-efficient. **Challenge:** exploration
- Guarantees:
  - **Policy Gradients:** Converges to a local minima of  $J(\theta)$ , often good enough!
  - **Q-learning:** Zero guarantees since you are approximating Bellman equation with a complicated function approximator

یادگیری تقویتی عمیق

۵

منابع


## منبع اصلی

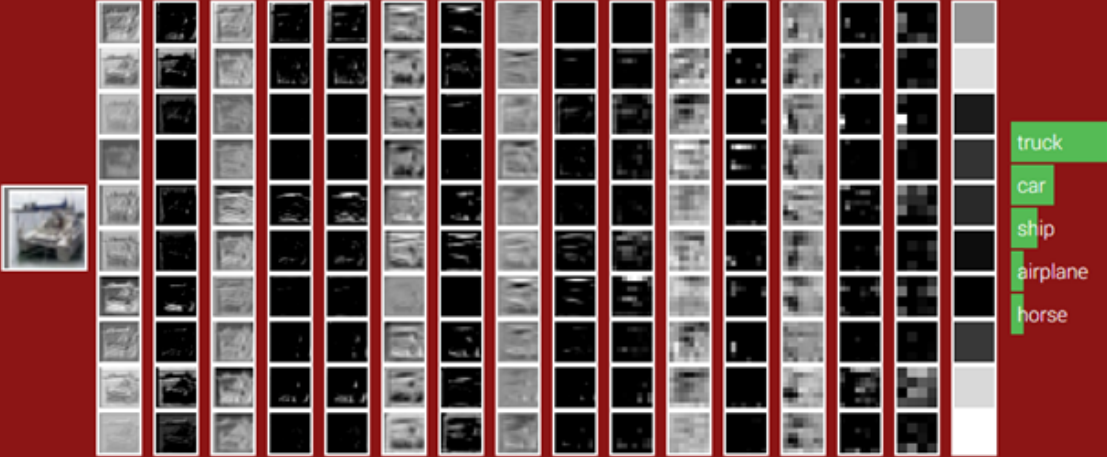


## CS231n: Convolutional Neural Networks for Visual Recognition

### Spring 2019

Previous Years: [\[Winter 2015\]](#) [\[Winter 2016\]](#) [\[Spring 2017\]](#) [\[Spring 2018\]](#)





\*This network is running live in your browser

### Course Description

Computer Vision has become ubiquitous in our society, with applications in search, image understanding, apps, mapping, medicine, drones, and self-driving cars. Core to many of these applications are visual recognition tasks such as image classification, localization and detection. Recent developments in neural network (aka "deep learning") approaches have greatly advanced the performance of these state-of-the-art visual recognition systems. This course is a deep dive into details of the deep learning architectures with a focus on learning end-to-end models for these tasks, particularly image classification. During the 10-week course, students will learn to implement, train and debug their own neural networks and gain a detailed understanding of cutting-edge research in computer vision. The final assignment will involve training a multi-million parameter convolutional neural network and applying it on the largest image classification dataset (ImageNet). We will focus on teaching how to set up the problem of image recognition, the learning algorithms (e.g. backpropagation), practical engineering tricks for training and fine-tuning the networks and guide the students through hands-on assignments and a final course project. Much of the background and materials of this course will be drawn from the [ImageNet Challenge](#).

<http://cs231n.stanford.edu>