

Topics:

- Reinforcement Learning Part 2
  - Q-Learning
  - Deep Q-Learning

**CS 4803-DL / 7643-A**  
**ZSOLT KIRA**

Many slides by Pieter Abbeel

**RL:** Sequential decision making in an environment with evaluative feedback.

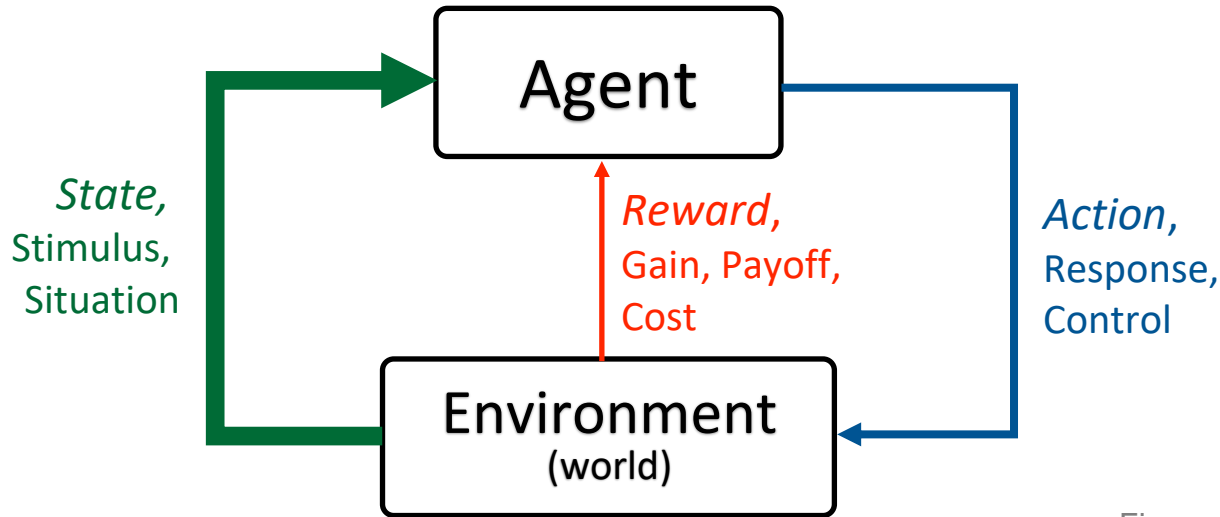


Figure Credit: Rich Sutton

- **Environment** may be unknown, non-linear, stochastic and complex.
- **Agent** learns a **policy** to map states of the environments to actions.
  - Seeking to maximize cumulative reward in the long run.

- **MDPs:** Theoretical framework underlying RL
- An MDP is defined as a tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$ 
  - $\mathcal{S}$  : Set of possible states
  - $\mathcal{A}$  : Set of possible actions
  - $\mathcal{R}(s, a, s')$  : Distribution of reward
  - $\mathbb{T}(s, a, s')$  : Transition probability distribution, also written as  $p(s'|s,a)$
  - $\gamma$  : Discount factor

- **MDPs:** Theoretical framework underlying RL
- An MDP is defined as a tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$ 
  - $\mathcal{S}$  : Set of possible states
  - $\mathcal{A}$  : Set of possible actions
  - $\mathcal{R}(s, a, s')$  : Distribution of reward
  - $\mathbb{T}(s, a, s')$  : Transition probability distribution, also written as  $p(s'|s,a)$
  - $\gamma$  : Discount factor
- **Interaction trajectory:**  $\dots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \dots$

## What we want

e.g.

State	Action
A	→ 2
B	→ 1

A policy  $\pi$

$$\pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid \pi \right]$$

Definition of **optimal policy**

## Some intermediate concepts and terms

A **Value function** (how good is a state?)

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

A **Q-Value function** (how good is a state-action pair?)

$$Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \quad Q^{\pi}(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right]$$

$$Q^*(s, a) = \mathbb{E}_{\sim p(s'|s, a)} [r(s, a) + \gamma V^*(s')] \quad (\text{Math in previous lecture})$$

## Equalities relating optimal quantities

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

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

## We can then derive the Bellman Equation

$$Q^*(s, a) = \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma \max_a Q^*(s', a') \right]$$

This must hold true for an optimal Q-Value!

-> Leads to dynamic programming algorithm to find it

Summary of Last Time

- Equations relating optimal quantities

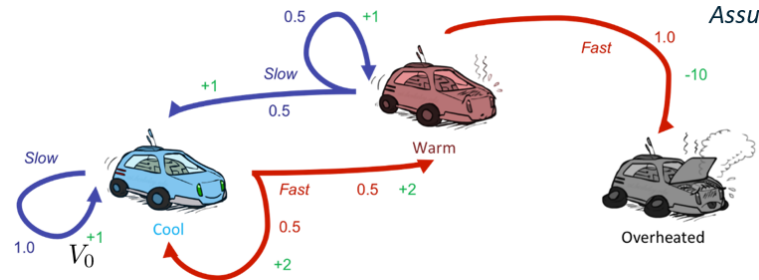
$$V^*(s) = \max_a Q^*(s, a)$$

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

- Recursive Bellman optimality equation

$$\begin{aligned} Q^*(s, a) &= \mathbb{E}_{s' \sim p(s'|s, a)} [r(s, a) + \gamma V^*(s')] \\ &= \sum_{s'} p(s'|s, a) [r(s, a) + \gamma V^*(s')] \\ &= \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma \max_a Q^*(s', a') \right] \end{aligned}$$

Assume no discount!



$V_2$

$V_1$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

$$V_1(s_0) = \max \left( \begin{array}{l} \overset{\text{slow}}{1.0 \cdot (+1 + V(s_0))} \\ \overset{\text{Fast}}{0.5 [+2 + V(s_1)] + 0.5 [+2 \cdot V(s_0)]} \end{array} \right) = 2$$

Slide Credit: <http://ai.berkeley.edu>

# Racing Search Tree



[https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld\\_dp.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html)

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00					0.00				0.00
0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00					0.00				0.00
0.00	0.00	0.00	-1.00		0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00		-0.10	-1.00	0.59	0.00	0.00
0.00	0.00	0.00	0.00		1.00	-0.10	0.66	-0.41	0.00
0.00	0.00	0.00	0.00		0.90	0.81	0.73	-0.34	0.00
0.00	0.00	0.00	-1.00		-0.19	-0.17	0.66	0.59	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.59	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

0.22	0.25	0.27	0.30	0.34	0.37	0.34	0.30	0.34	0.37
0.25	0.27	0.30	0.34	0.37	0.42	0.37	0.34	0.37	0.42
0.20					0.46				0.46
0.20	0.22	0.25	-0.78		0.51	0.57	0.63	0.57	0.51
0.22	0.25	0.27	0.25		0.08	-0.37	0.71	0.63	0.57
0.25	0.27	0.30	0.27		1.20	0.08	0.78	-0.29	0.51
0.27	0.30	0.34	0.30		1.00	0.97	0.87	-0.22	0.57
0.30	0.34	0.37	-0.58		-0.13	-0.13	0.71	0.71	0.63
0.34	0.37	0.42	0.46	0.51	0.57	0.63	0.71	0.63	0.57
0.30	0.34	0.37	0.42	0.46	0.51	0.57	0.63	0.57	0.51



## Value Iteration Update:

$$V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) [r(s, a) + \gamma V^i(s')]$$

## Q-Iteration Update:

$$Q^{i+1}(s, a) \leftarrow \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma \max_{a'} Q^i(s', a') \right]$$

The algorithm is same as value iteration, but it loops over actions as well as states

## For Value Iteration:

Theorem: will converge to unique optimal values

Basic idea: approximations get refined towards optimal values

Policy may converge long before values do

Time complexity per iteration  $O(|\mathcal{S}|^2|\mathcal{A}|)$

## Feasible for:

- ◆ 3x4 Grid world?
- ◆ Chess/Go?
- ◆ Atari Games with integer image pixel values [0, 255] of size 16x16 as state?

## Summary: MDP Algorithms

### Value Iteration

- ◆ Bellman update to state value estimates

### Q-Value Iteration

- ◆ Bellman update to (state, action) value estimates

These algorithms just “solve” the known MDP by propagating through the equations – no experience!



# Reinforcement Learning, Deep RL

- Recall RL assumptions:
  - $\mathbb{T}(s, a, s')$  unknown, how actions affect the environment.
  - $\mathcal{R}(s, a, s')$  unknown, what/when are the good actions?
- But, we can learn by trial and error.
  - Gather experience (data) by performing actions.
    - Sampling from the transition function(!), have reward -> Everything you need!
  - Approximate unknown quantities from data.

# Reinforcement Learning

- Old Dynamic Programming Demo

- [https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld\\_dp.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html)

- RL Demo

- [https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld\\_td.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html)

Slide credit: Dhruv Batra

# Sample-Based Policy Evaluation?

- We want to improve our estimate of  $V$  by computing these averages:

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

- Idea: Take samples of outcomes  $s'$  (by doing the action!) and average

$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

$$\text{sample}_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$

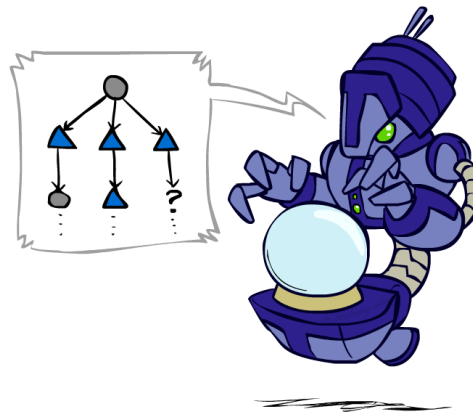
...

$$\text{sample}_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$

$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i \text{sample}_i$$

Why does this work  
without knowing  $T$ ?

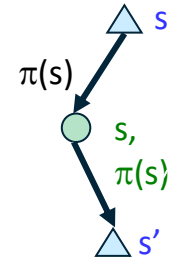
What is a practical  
difficulty?



Almost! But we can't  
rewind time to get  
sample after sample from  
state  $s$ .

# Temporal Difference Learning

- Big idea: learn from every experience!
  - Update  $V(s)$  each time we experience a transition  $(s, a, s', r)$
  - Likely outcomes  $s'$  will contribute updates more often
- Temporal difference learning of values
  - Policy can be fixed, just doing evaluation!
  - Move values toward value of whatever successor occurs: running average



Sample of  $V(s)$ :  $sample = R(s, \pi(s), s') + \gamma V^\pi(s')$

Update to  $V(s)$ :  $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$

Same update:  $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

# Q-Learning

- We'd like to do Q-value updates to each Q-state:

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

- But can't compute this update without knowing T, R

- Instead, compute average as we go

- Receive a sample transition (s,a,r,s')
- This sample suggests

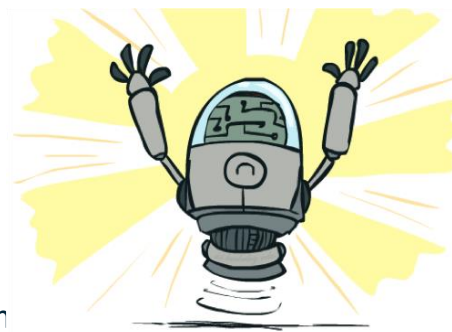
$$Q(s, a) \approx r + \gamma \max_{a'} Q(s', a')$$

- But we want to average over results from (s,a)
- So keep a running average

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) \left[ r + \gamma \max_{a'} Q(s', a') \right]$$

# Q-Learning Properties

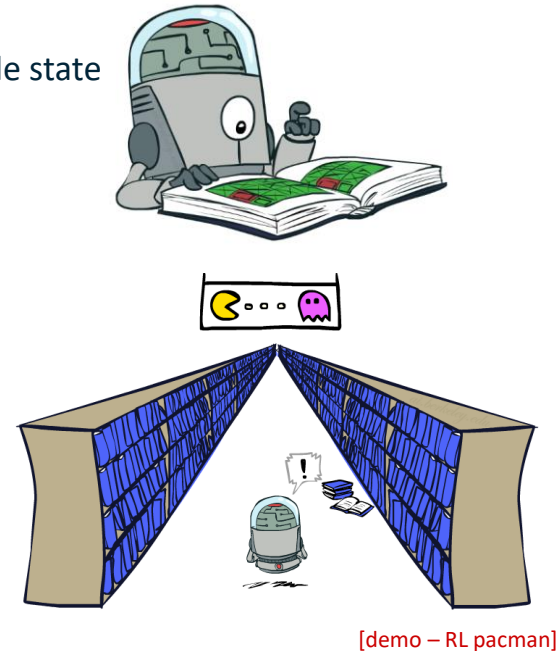
- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called **off-policy learning**
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - ... but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select action



# Deep Q-Learning

# Generalizing Across States

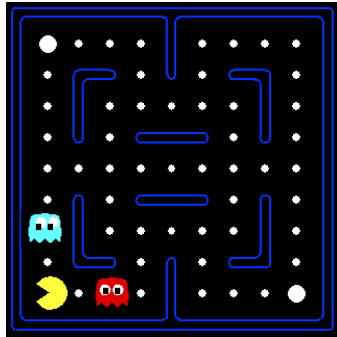
- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is the fundamental idea in machine learning!



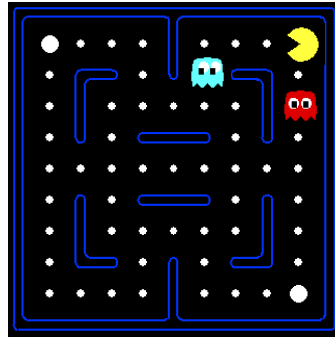
[demo – RL pacman]

# Example: Pacman

Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



- State space is too large and complicated for feature engineering though!
- Recall: Value iteration not scalable (chess, RGB images as state space, etc)
- Solution: Deep Learning! ... more precisely, function approximation.
  - Use deep neural networks to learn state representations
  - Useful for continuous action spaces as well

## Deep Reinforcement Learning

## Value-based RL

- (Deep) Q-Learning, approximating  $Q^*(s, a)$  with a deep Q-network

## Policy-based RL

- Directly approximate optimal policy  $\pi^*$  with a parametrized policy  $\pi_\theta^*$

## Model-based RL

- Approximate transition function  $T(s', a, s)$  and reward function  $\mathcal{R}(s, a)$
- Plan by looking ahead in the (approx.) future!

- **Q-Learning with linear function approximators**

$$Q(s, a; w, b) = w_a^\top s + b_a$$

- Has some theoretical guarantees

- **Deep Q-Learning: Fit a deep Q-Network**  $Q(s, a; \theta)$

- Works well in practice
- Q-Network can take RGB images

FC-4 (Q-values)

FC-256

32 4x4 conv, stride 2

16 8x8 conv, stride 4

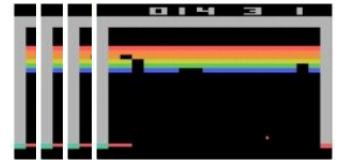


Image Credits: Fei-Fei Li, Justin Johnson,  
Serena Yeung, CS 231n

- Assume we have collected a dataset:

$$\left\{ (s, a, s', r)_i \right\}_{i=1}^N$$

- We want a Q-function that satisfies bellman optimality (Q-value)

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

- Loss for a single data point:

$$\text{MSE Loss} := \left( \underbrace{Q_{new}(s, a)}_{\text{Predicted Q-Value}} - \underbrace{\left( r + \gamma \max_a Q_{old}(s', a) \right)}_{\text{Target Q-Value}} \right)^2$$

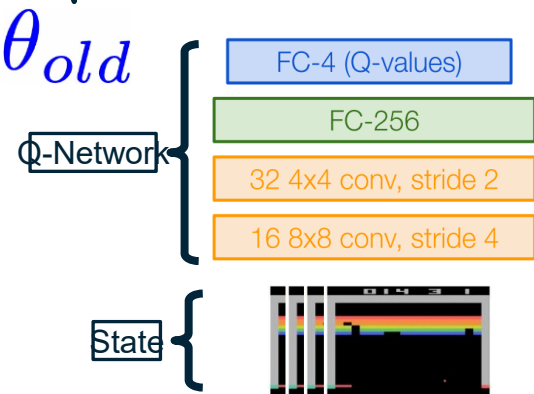
- Minibatch of  $\{(s, a, s', r)_i\}_{i=1}^B$

What is a challenge for using two different Q networks that change?



- Compute loss:
 
$$\left( \underbrace{Q_{new}(s, a)}_{\theta_{new}} - \left( r + \gamma \max_a \underbrace{Q_{old}(s', a)}_{\theta_{old}} \right) \right)^2$$

- Backward pass:  $\frac{\partial Loss}{\partial \theta_{new}}$



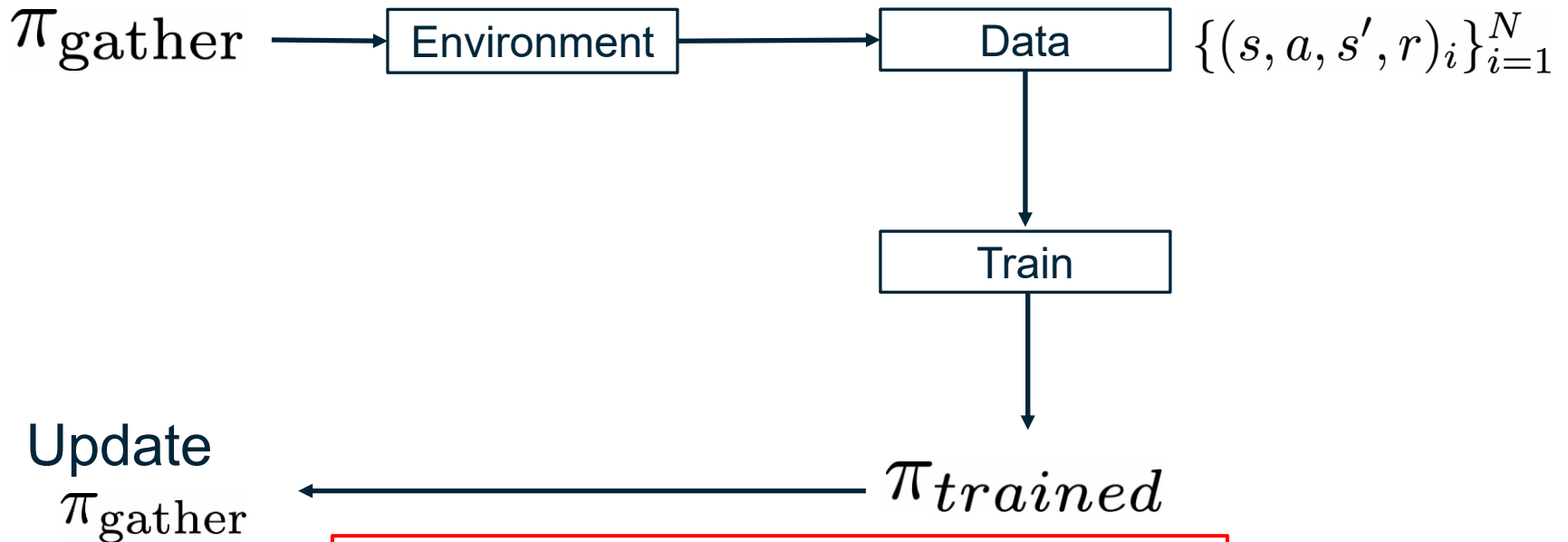
$$\text{MSE Loss} := \left( Q_{new}(s, a) - \left( r + \max_a Q_{old}(s', a) \right) \right)^2$$

- In practice, for stability:
  - Freeze  $Q_{old}$  and update  $Q_{new}$  parameters
  - Set  $Q_{old} \leftarrow Q_{new}$  at regular intervals

How to gather experience?

$$\{(s, a, s', r)_i\}_{i=1}^N$$

This is why RL is hard



Challenge 1: Exploration vs Exploitation

Challenge 2: Non iid, highly correlated data

How to gather experience?

- What should  $\pi_{\text{gather}}$  be?
- Greedy? -> Local minimas, no exploration

$$\arg \max_a Q(s, a; \theta)$$

- An exploration strategy:

- $\epsilon$ -greedy

$$a_t = \begin{cases} \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

- Samples are correlated => high variance gradients => **inefficient learning**
- Current Q-network parameters determines next training samples => can lead to **bad feedback loops**
  - e.g. if maximizing action is to move right, training samples will be dominated by samples going right, may fall into local minima



- Correlated data: addressed by using experience replay
  - A replay buffer stores transitions  $(s, a, s', r)$
  - Continually update replay buffer as game (experience) episodes are played, older samples discarded
  - Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples
- Larger the buffer, lower the correlation

---

## Algorithm 1 Deep Q-learning with Experience Replay

---

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

Initialize action-value function  $Q$  with random weights

Experience Replay

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

Epsilon-greedy

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

Q Update

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

**end for**

**end for**

---

# Atari Games



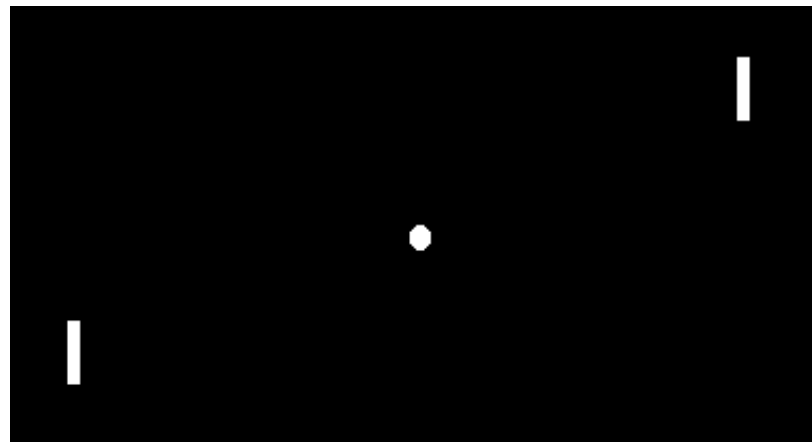
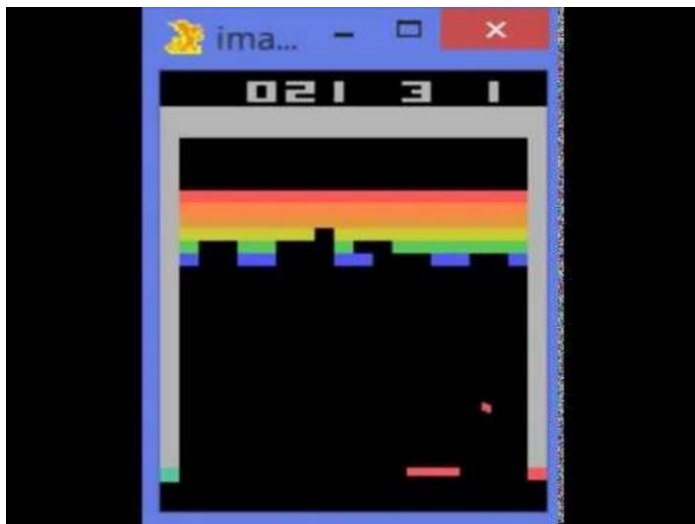
- ◆ **Objective:** Complete the game with the highest score
- ◆ **State:** Raw pixel inputs of the game state
- ◆ **Action:** Game controls e.g. Left, Right, Up, Down
- ◆ **Reward:** Score increase/decrease at each time step

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

## Case study: Playing Atari Games

# Atari Games



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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

**Case study: Playing Atari Games**

Thus far, we looked at

- ◆ **Dynamic Programming**
  - ◆ Value, Q-Value Iteration
  
- ◆ **Reinforcement Learning (RL)**
  - ◆ The challenges of (deep) learning based methods
  - ◆ Value-based RL algorithms
    - ◆ Deep Q-Learning

Next:

- ◆ **Policy-based RL algorithms** (policy gradients)

