Topics:

- Reinforcement Learning Part 2
  - **Q-Learning**
  - **Deep Q-Learning**
  - **Policy Gradient**

CS 4803-DL / 7643-A

ZSOLT KIRA

Admin

- HW4 – into the grace period!

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

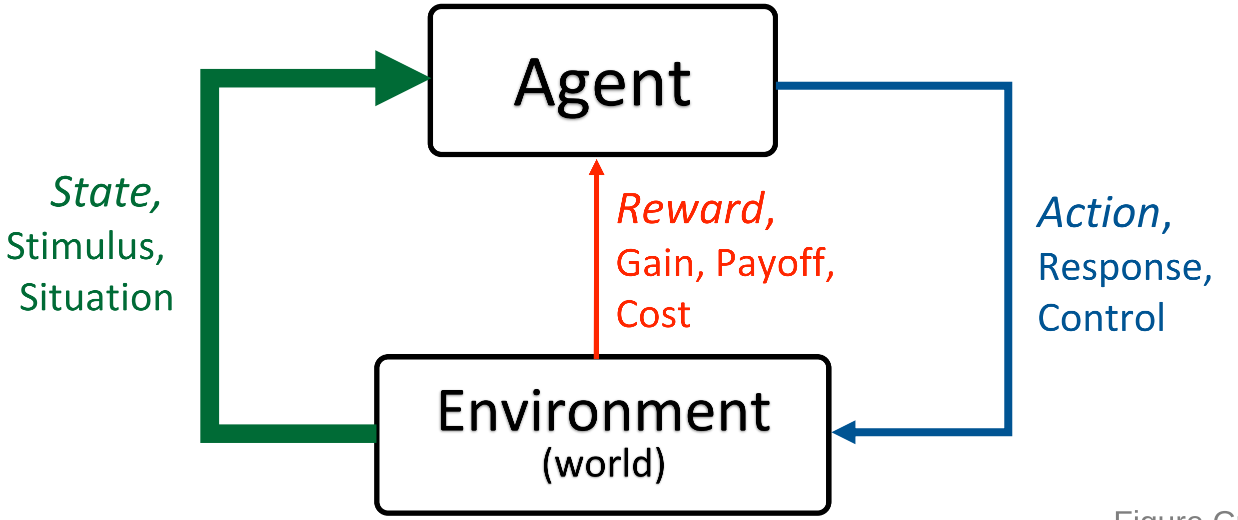


Figure Credit: Rich Sutton

- **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.

What is Reinforcement Learning?

- **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)

  γ : 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)
  - γ : Discount factor
- **Interaction trajectory**: $\ldots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \ldots$

Markov Decision Processes (MDPs)

Summary of Last Time

What we want

e.g.

State	Action
A $\longrightarrow$	2
B $\longrightarrow$	1

A policy π

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

Definition of **optimal policy**

Some intermediate concepts and terms

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

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

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

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

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

(Math in previous lecture)

Equalities relating optimal quantities

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

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

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

Georgia Tech

- Equations relating optimal quantities

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

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

- Recursive Bellman optimality equation

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

NOTE: In the lecture video for these slides, there was a typo having V(s) instead of V(s')

Bellman Optimality Equations

Based on the **bellman optimality equation**

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

Algorithm

- Initialize values of all states
- While not converged:
  - For each state: $\boxed{V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^i(s')\right]}$
- Repeat until convergence (no change in values)

$$V^0 \rightarrow V^1 \rightarrow V^2 \rightarrow \quad \cdots \rightarrow V^i \rightarrow \ldots \quad \rightarrow V^*$$

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

Value Iteration

Value Iteration Update:

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

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

Q-Iteration

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

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

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

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.
  - Approximate unknown quantities from data.

Reinforcement Learning

- Old Dynamic Programming Demo
  - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

- RL Demo
  - 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:

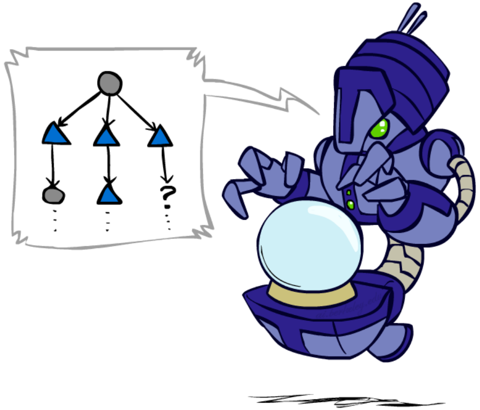
$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$$

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

$$sample_1 = R(s,\pi(s),s_1') + \gamma V_k^{\pi}(s_1')$$
$$sample_2 = R(s,\pi(s),s_2') + \gamma V_k^{\pi}(s_2')$$
$$\ldots$$
$$sample_n = R(s,\pi(s),s_n') + \gamma V_k^{\pi}(s_n')$$

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

What’s the difficulty of this algorithm?

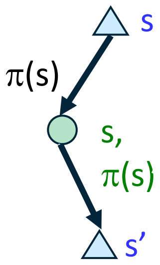


state s.

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 still fixed, still 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

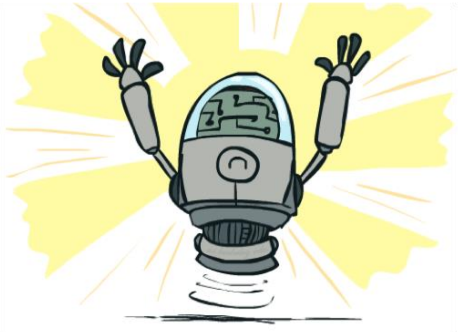
$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

  - 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

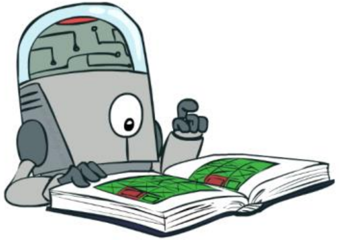
- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit,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

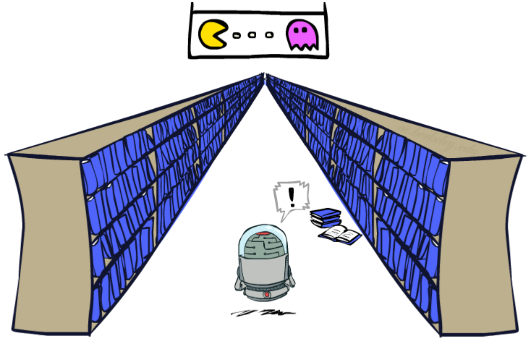


Deep Q-Learning

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!

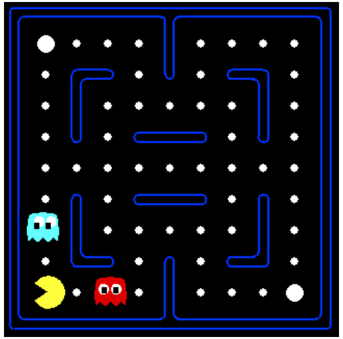




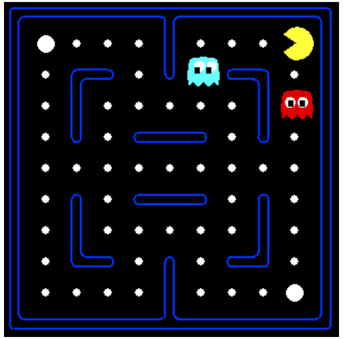
[demo – RL pacman]

Example: Pacman

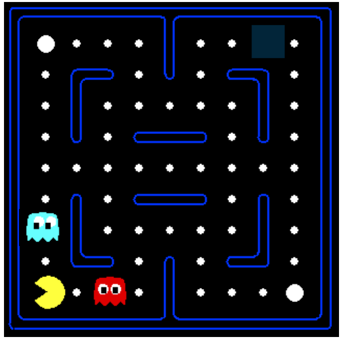
Let's say we discover through experience that this state is bad:



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

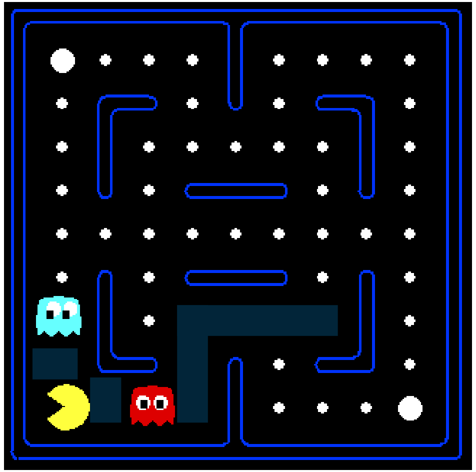


Or even this one!



Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers

- Disadvantage: states may share features but can actually be very different in value!

- 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 π^* with a parametrized policy π^*_θ
- **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!

Deep RL: Algorithm Categories

- **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

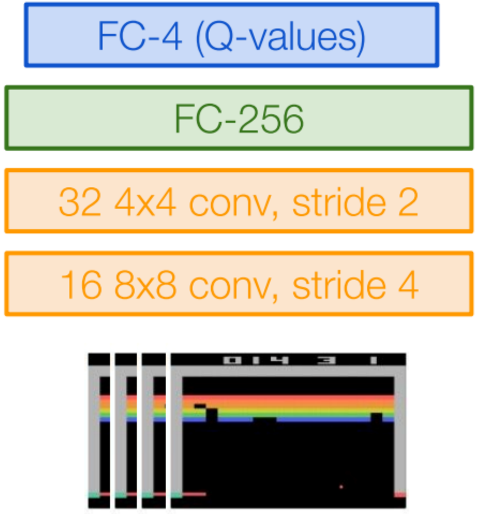


Image Credits: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

- Assume we have collected a dataset:

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

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

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

- Loss for a single data point:

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

- Minibatch of $\{(s, a, s', r)_i\}_{i=1}^{B}$
- Forward pass: State → Q-Network → Q-Values per action

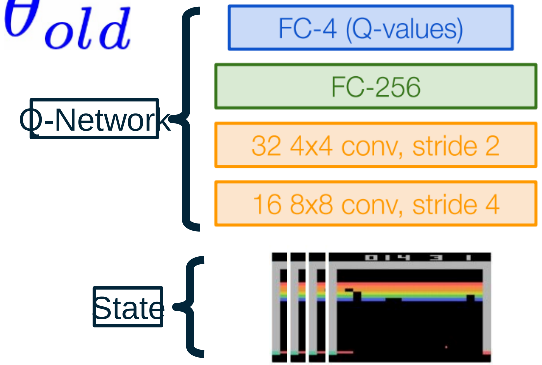
  State: $B \times D$; Q-Values per action: $B \times n_{actions}$

- Compute loss:

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

- Backward pass:

$$\frac{\partial Loss}{\partial \theta_{new}}$$



Deep Q-Learning

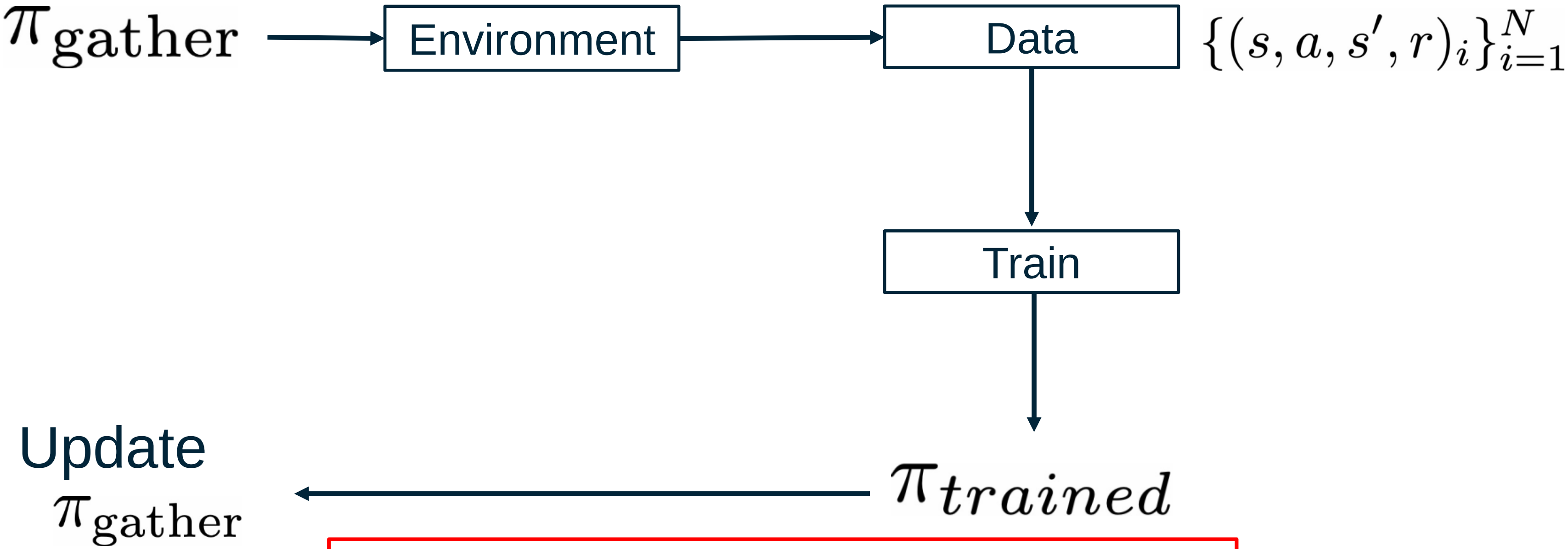
$$\text{MSE Loss} := \left(Q_{new}(s,a) - (r + \max_a Q_{old}(s', a)) \right)^2$$

- 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 π_{gather} be?
  - Greedy? -> Local minimas, no exploration

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

- An exploration strategy:
  - ϵ-greedy

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

Exploration Problem

Correlated Data Problem

- 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

		start		
R=10				R=1

- 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

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 ϵ 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

Epsilon-greedy

Q Update

Deep Q-Learning Algorithm

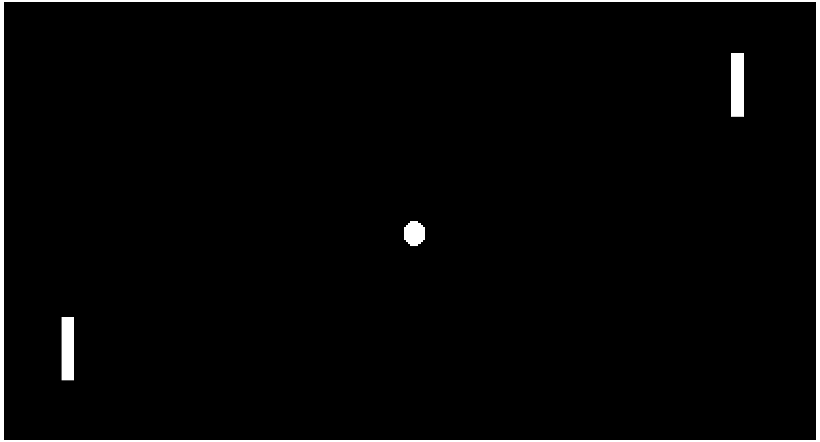
Georgia Tech

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

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

Atari Games



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

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

In today's class, we looked at

- **Dynamic Programming**
  - Value, Q-Value Iteration

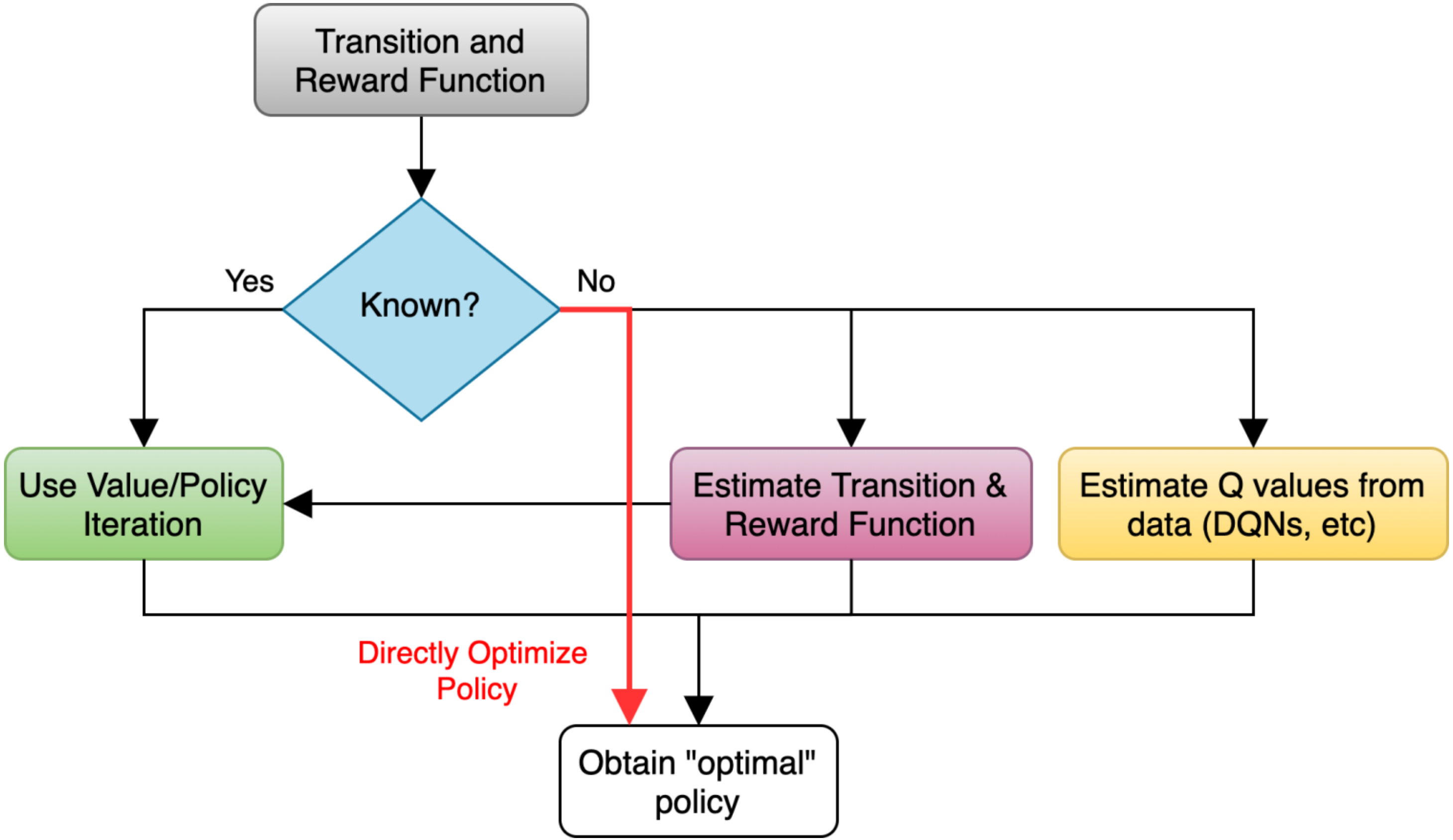
- **Reinforcement Learning (RL)**
  - The challenges of (deep) learning based methods
  - Value-based RL algorithms
    - Deep Q-Learning

Now:

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

Summary

Policy Gradients, Actor-Critic


Transition and Reward Function
Known?
Yes
No
Use Value/Policy Iteration
Estimate Transition & Reward Function
Estimate Q values from data (DQNs, etc)
Directly Optimize Policy
Obtain "optimal" policy

Overview

- Class of policies defined by parameters θ

$$\boxed{\pi_\theta(a|s) : \mathcal{S} \rightarrow \mathcal{A}}$$

  - Eg: θ can be parameters of linear transformation, deep network, etc.

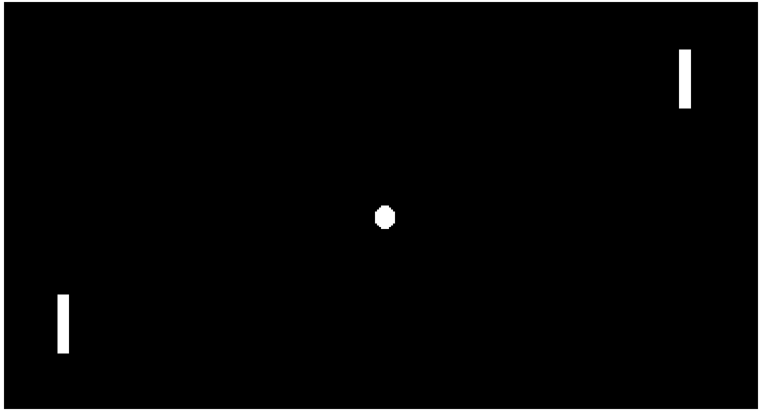
- Want to maximize:

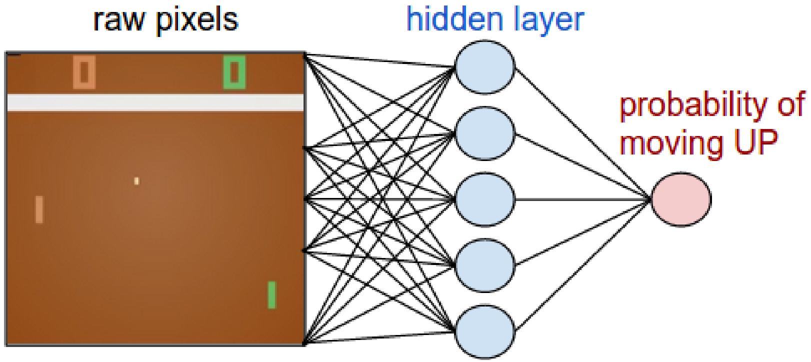
$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

- In other words,

$$\pi^* = \arg \max_{\pi : \mathcal{S} \rightarrow \mathcal{A}} \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right] \quad \Longrightarrow \quad \theta^* = \arg \max_{\theta} \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

Parametrized Policy





Pong from Pixels

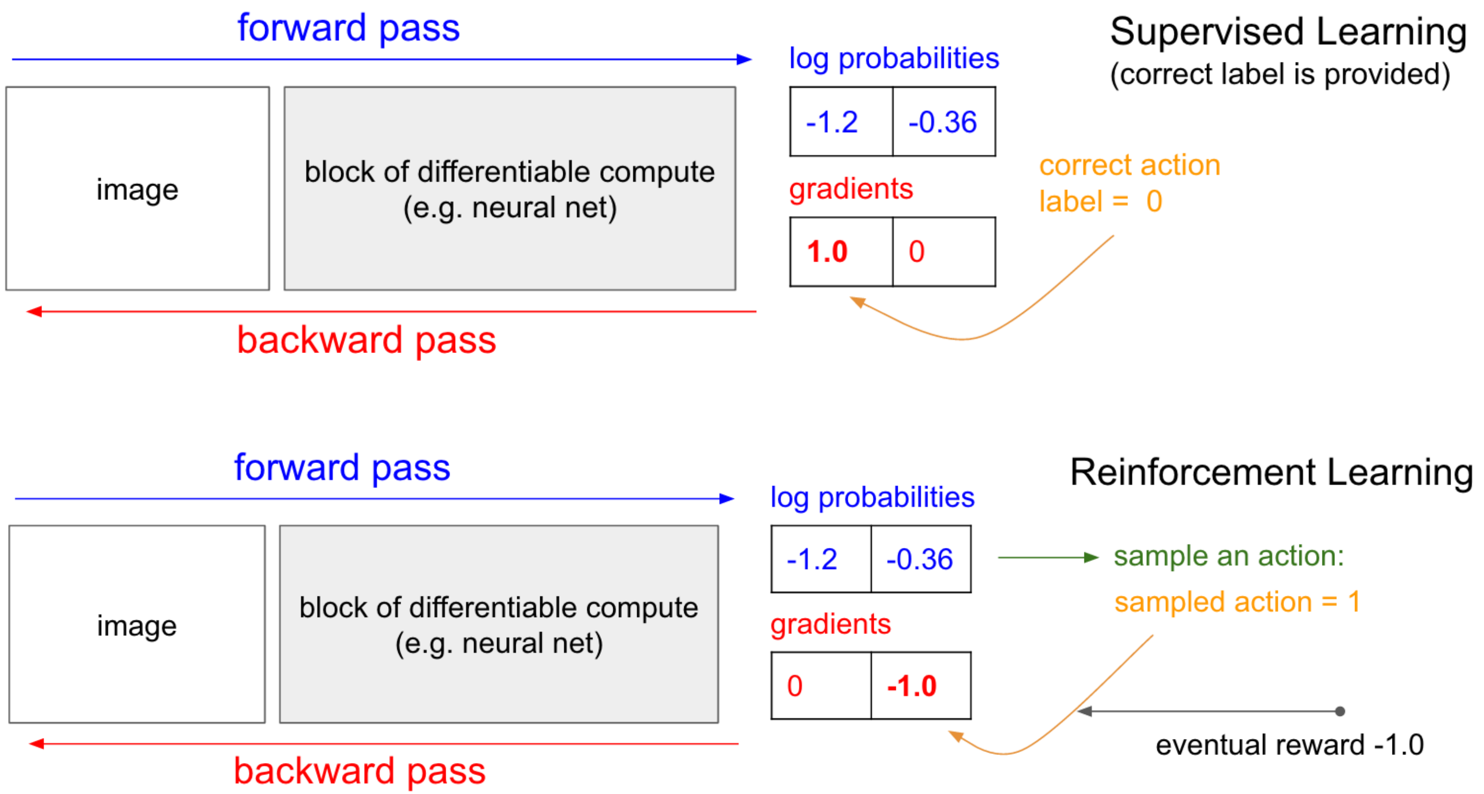


Image Source: http://karpathy.github.io/2016/05/31/rl/

Policy Gradient: Loss Function

- Slightly re-writing the notation

Let $\tau = (s_0, a_0, \ldots s_T, a_T)$ denote a trajectory

$$\pi_\theta(\tau) = p_\theta(\tau) = p_\theta\left(s_0, a_0, \ldots s_T, a_T\right)$$
$$= p(s_0) \prod_{t=0}^{T} p_\theta\left(a_t \mid s_t\right) \cdot p\left(s_{t+1} \mid s_t, a_t\right)$$

$$\boxed{\arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\mathcal{R}(\tau)\right]}$$

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\mathcal{R}(\tau) \right]$$

$$= \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)} \left[\sum_{t=0}^{T} \mathcal{R}(s_t, a_t) \right]$$

- How to gather data?
  - We already have a policy: π_θ
  - Sample N trajectories $\{\tau_i\}_{i=1}^{N}$ by acting according to π_θ

$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} r(s_t^i, a_t^i)$$

Gathering Data/Experience

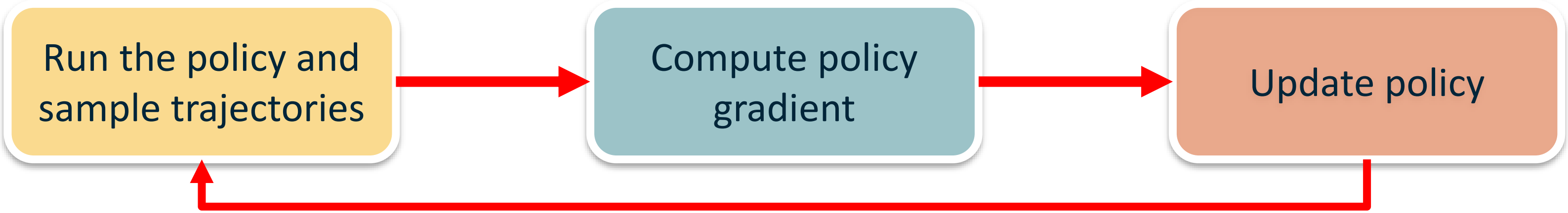
The REINFORCE Algorithm

- Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to π_θ

- Compute policy gradient as

$$\nabla_\theta J(\theta) \approx \ ?$$

- Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Slide credit: Sergey Levine

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}[\mathcal{R}(\tau)]$$

$$= \nabla_\theta \int \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

Expectation as integral

$$= \int \nabla_\theta \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

Exchange integral and gradient

$$= \int \nabla_\theta \pi_\theta(\tau) \cdot \frac{\pi_\theta(\tau)}{\pi_\theta(\tau)} \cdot \mathcal{R}(\tau)d\tau$$

$$= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau)d\tau$$

$$\nabla_\theta \log \pi(\tau) = \frac{\nabla_\theta \pi(\tau)}{\pi(\tau)}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\mathcal{R}(\tau)]$$

Deriving The Policy Gradient

Georgia Tech

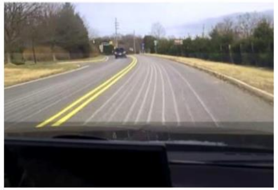
$$\pi_\theta(\tau) = p(s_0) \prod_{t=0}^{T} p_\theta\left(a_t \mid s_t\right) \cdot p\left(s_{t+1} \mid s_t, a_t\right)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\underbrace{\nabla_\theta \log \pi_\theta(\tau)}\mathcal{R}(\tau)\right]$$

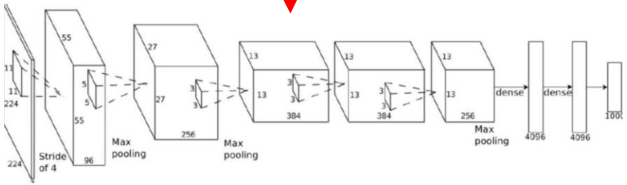
$$\nabla_\theta \left[\cancel{\log p(s_0)} + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \cancel{\log p(s_{t+1} \mid s_t, a_t)}\right]$$

Doesn't depend on Transition probabilities!

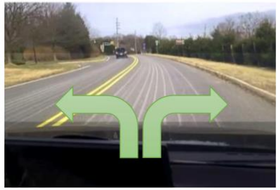
$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$



$\mathbf{s}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$



$\mathbf{a}_t$

Continuous Action Space?

Deriving The Policy Gradient

The REINFORCE Algorithm

- Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to π_θ

- Compute policy gradient as

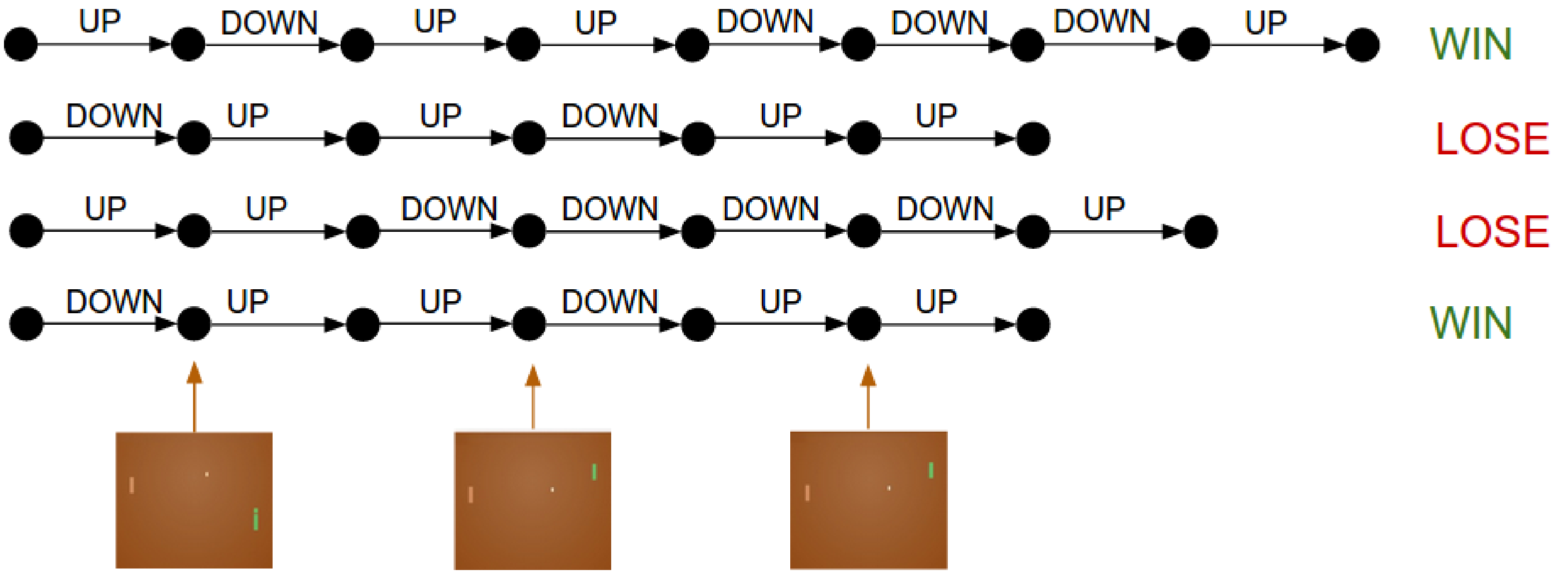
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i^N \left[\sum_{t=1}^T \nabla_\theta \log \pi_\theta \left(a_t^i \mid s_t^i \right) \cdot \sum_{t=1}^T \mathcal{R} \left(s_t^i \mid a_t^i \right) \right]$$

- Update policy parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Run the policy and sample trajectories → Compute policy gradient → Update policy

Slide credit: Sergey Levine

Georgia Tech



Slide credit: Dhruv Batra

Issues with Policy Gradients

- Credit assignment is hard!
  - Which specific action led to increase in reward
  - Suffers from high variance → leading to unstable training