CS 4803 / 7643
Deep Learning, Fall 2020

Reinforcement Learning: Module 3/3

Presented by [Nirbhay Modhe]

Georgia Tech

Previous Lecture

- **MDPs:** Value function, optimal quantities, algorithms for solving MDPs
- **RL**: No rewards and transitions (RL), function approximation (deep RL).

Today

- **RL Algorithms**
  - Overview, types of RL algorithms
  - Deep-Q Learning: A value based RL algorithm
  - Policy gradients: A policy-based RL algorithm

Recap & Overview

- Recursive Bellman expansion (from definition of Q)

(Expected) return from t = 0

$$Q^*(s,a) = \mathbb{E}_{\substack{a_t \sim \pi^*(\cdot|s_t) \\ s_{t+1} \sim p(\cdot|s_t,a_t)}} \left[\sum_{t \geq 0} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right]$$

- **RL setting**:
  - $\mathbb{T}(s, a, s')$ unknown, how actions affect the environment
  - $\mathcal{R}(s, a, s')$ unknown, what/when are the good actions?
- **Deep RL setting**:
  - Large or continuous state space
  - Use deep neural networks to learn state representations

- **Value-based RL**
  - (Deep) Q-Learning, approximating $Q^*(s, a)$ with a deep Q-network (DQN)
- **Policy-based RL**
  - Directly approximate optimal policy π^* with a parametrized policy $\boxed{\pi_\theta^*}$
- **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!

Broad Types of RL Algorithms

Georgia Tech

Deep Q-Learning

Deep Q-Learning

- **Intuition**: Learn a **parametrized Q-function** from data $\{(s, a, s', r)_i\}_{i=1}^{N}$

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

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

- **Deep Q-Learning**: Fit a **deep Q-Network** $Q(s, a; \theta)$
  - Works well in practice
  - Q-Network can take RGB images

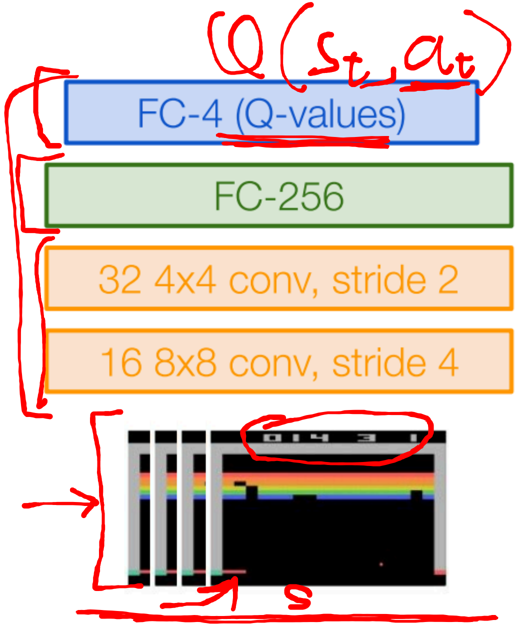
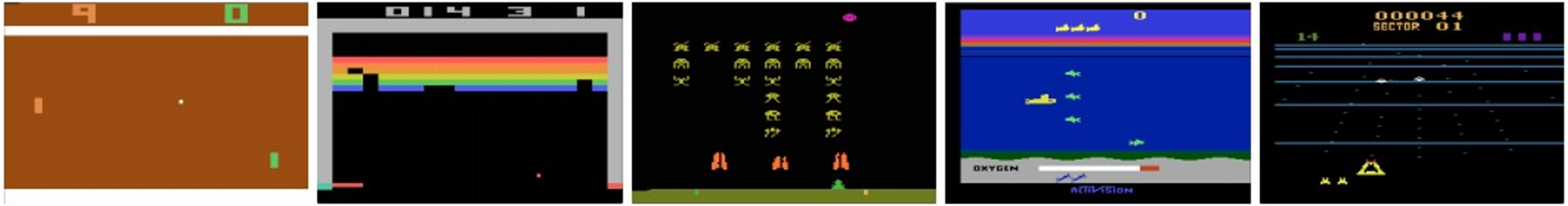


Image Credits: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

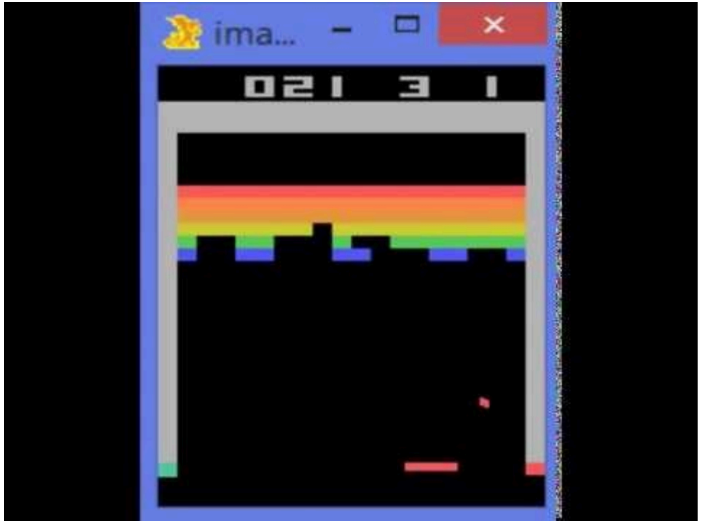
Atari Games

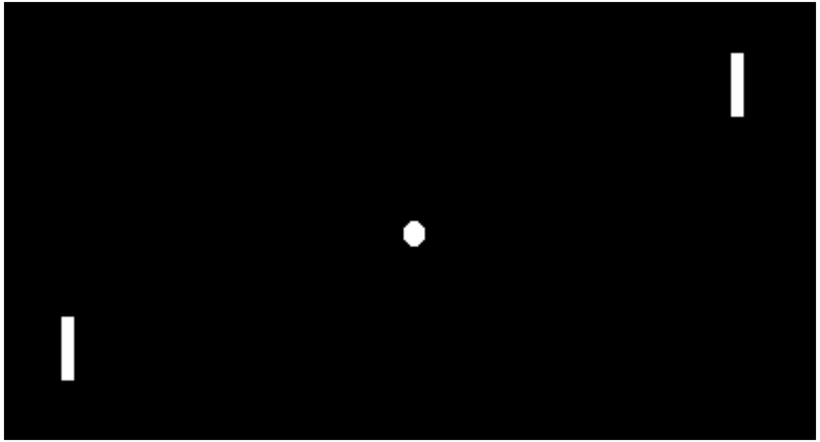


- **Objective**: Complete the game with the highest score
- **State**: Raw pixel inputs of the game state
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

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

- Assume we have collected a dataset:

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

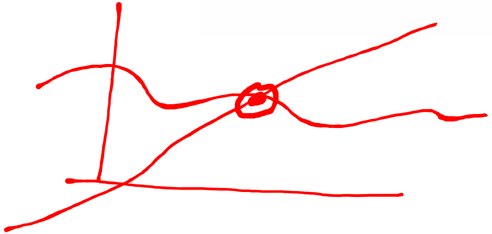
- We want a Q-function that satisfies bellman optimality (Q-value)

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

$Q_{i+1} =$ $+$ Q_i

- Loss for ~~a single data~~ point:

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



Deep Q-Learning

- 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: $B \times n_{actions}$

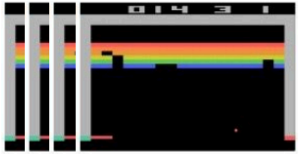
- Compute loss: $\Big(\underbrace{Q_{new}(s, a)}_{\theta_{new}} - (r + \gamma \max_{a}$

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

Q-Network:

- FC-4 (Q-values)
- FC-256
- 32 4x4 conv, stride 2
- 16 8x8 conv, stride 4

State:



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

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

- Assuming a fixed dataset, the MSE Loss can be optimized
  - This is known as the **Fitted Q-Iteration** algorithm
- However…

How to gather experience or “data”?

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

This is why RL is hard

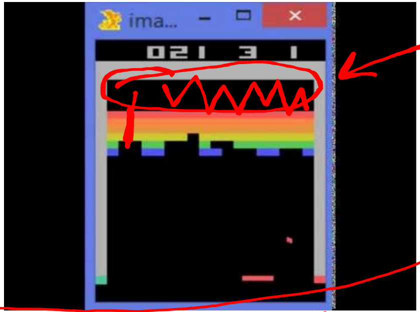
π_{gather} → Environment → Data $\{(s, a, s', r)_i\}_{i=1}^{N}$

$s a, s_2$

↓

Train

↓



Update π_{gather} ← $\pi_{trained}$

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

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

0.5 → 0.1

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

- 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

Georgia Tech

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

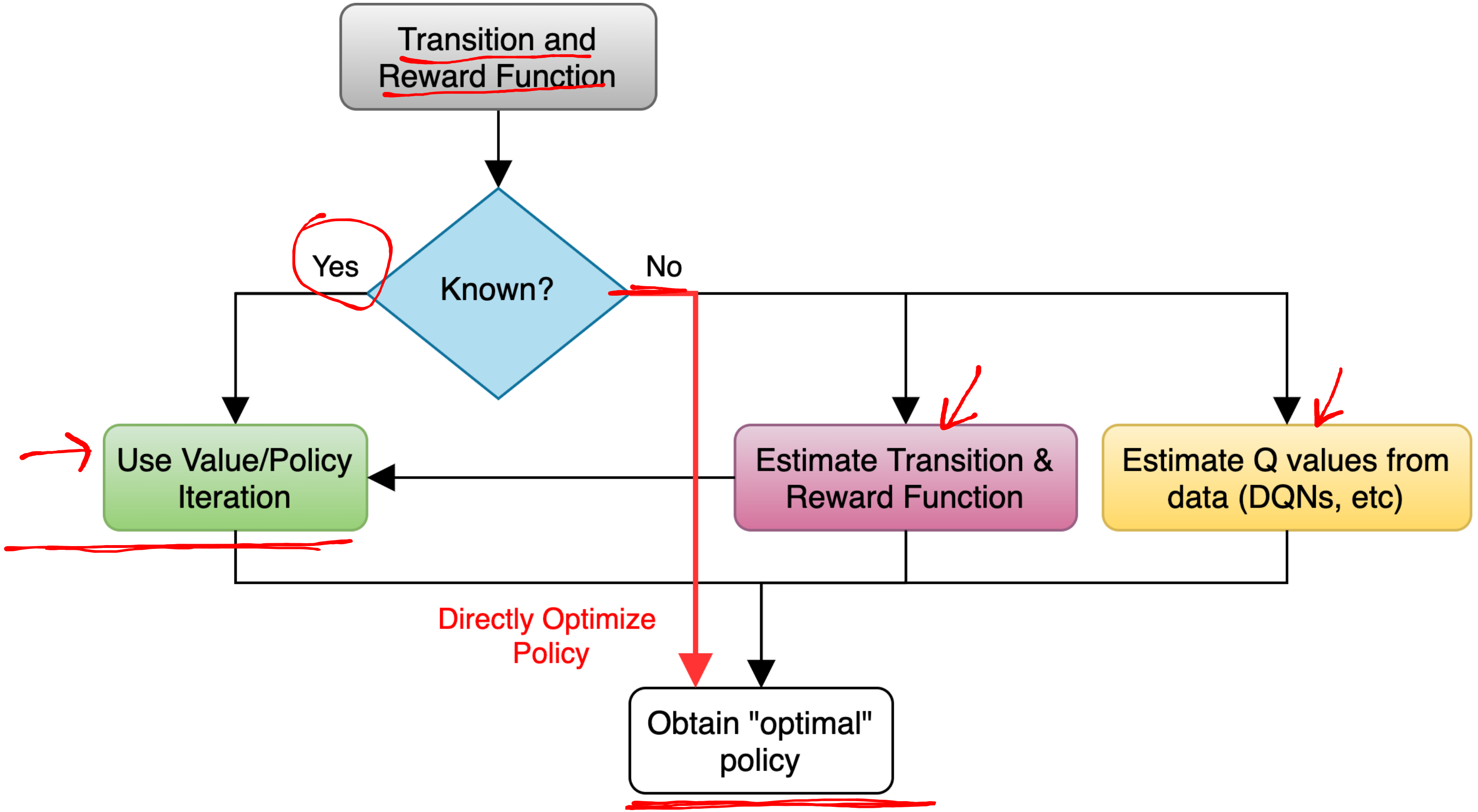
Epsilon-greedy

Q Update

Source: Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning."

Deep Q-Learning Algorithm

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.

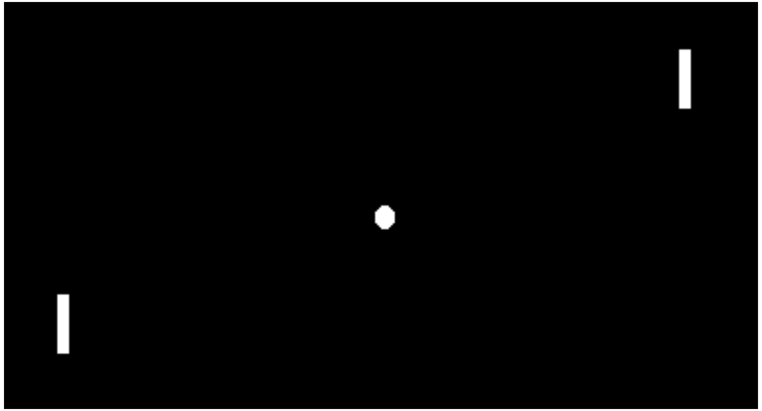
- Want to maximize:

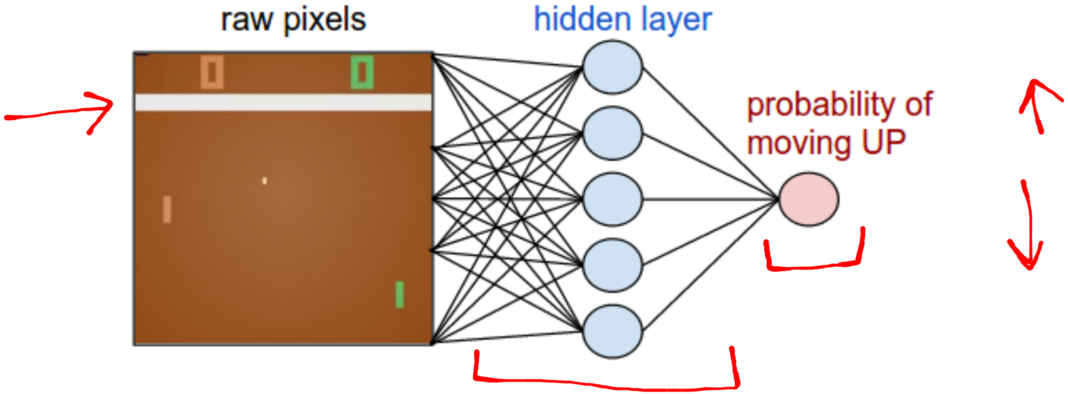
$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$

- In other words,

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

Parametrized Policy





Pong from Pixels

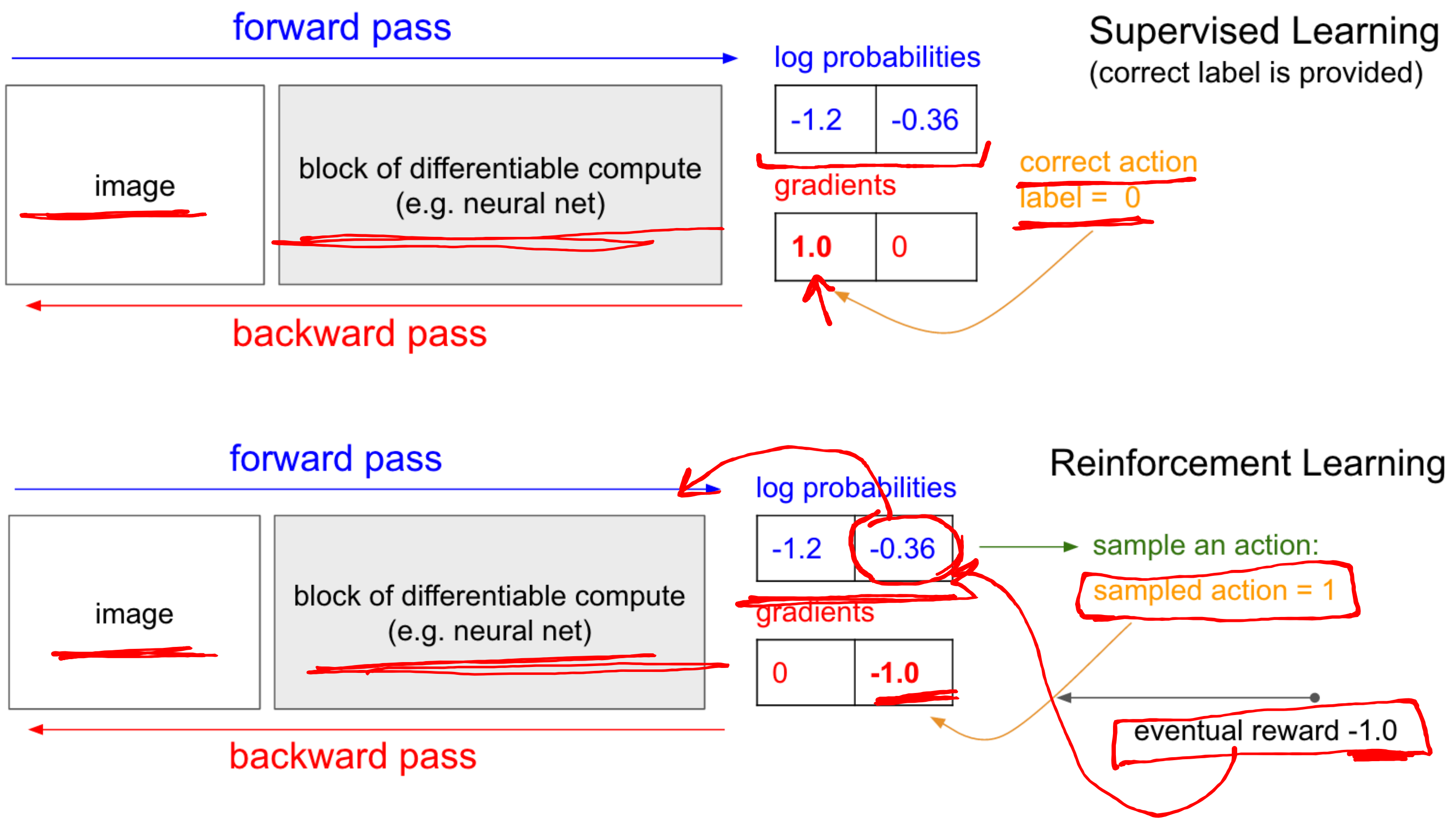


Image Source: http://karpathy.github.io/2016/05/31/rl/

Policy Gradient: Loss Function

Georgia Tech

- Slightly re-writing the notation

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

$$\pi_\theta(\tau) = p_\theta(\tau) = p_\theta\left(s_0, a_0, \ldots s_T, a_T\right)$$

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

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

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

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

- 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

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

The REINFORCE Algorithm

- Sample trajectories $\tau_i = \{s_1, a_1, \dots s_T, a_T\}_i$ 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: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Slide credit: Sergey Levine

Georgia Tech

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

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

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

Expectation as integral

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

Exchange integral and gradient

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

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

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

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

Deriving The Policy Gradient

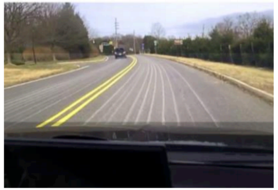
Georgia Tech

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

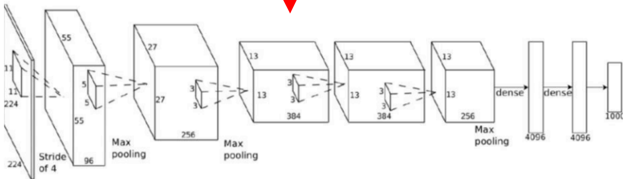
$$\nabla_\theta\left[\cancel{\log p(s_0)} + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \cancel{\log p(s_{t+1}\,|\,s_t, a_t)}\right]$$

Doesn't depend on Transition probabilities!

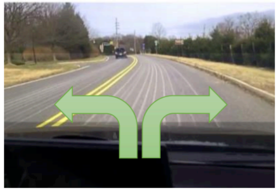
$$= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$$



$\mathbf{s}_t$



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



$\mathbf{a}_t$

Deriving The Policy Gradient

Georgia Tech

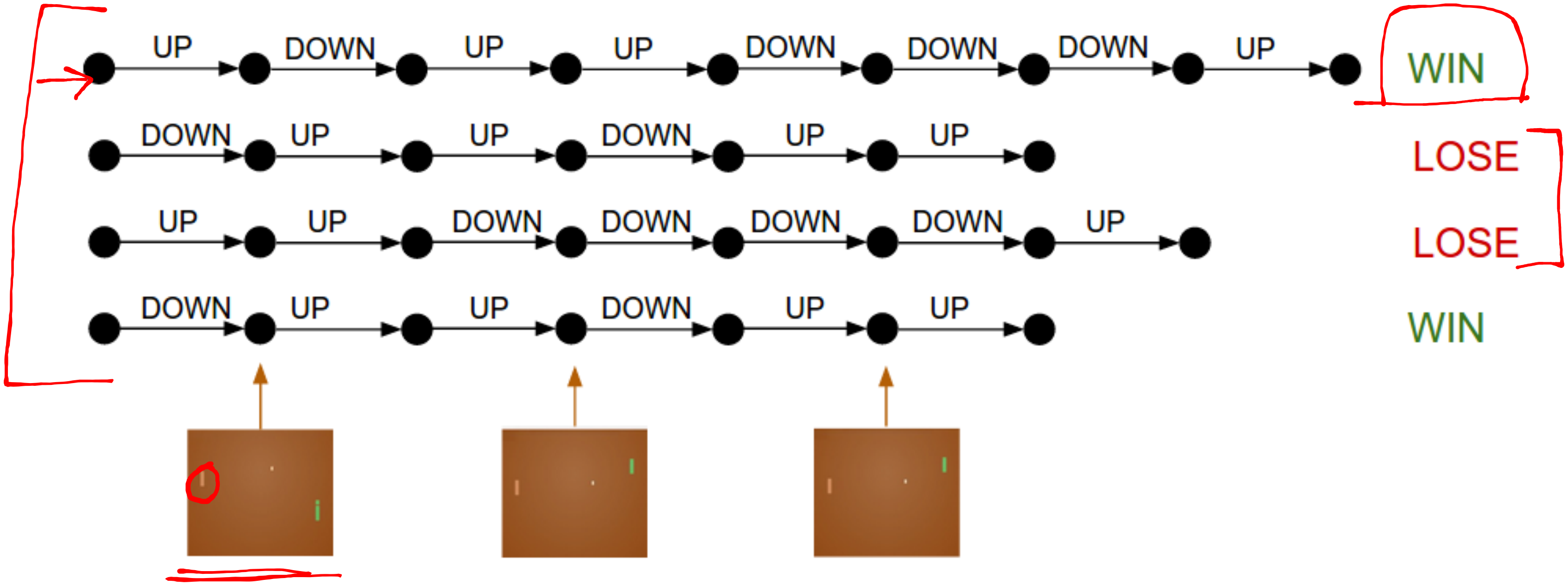
- Sample trajectories $\tau_i = \{s_1, a_1, \ldots s_T, a_T\}_i$ by acting according to π_θ
- Compute policy gradient as

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

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

Slide credit: Sergey Levine

The REINFORCE Algorithm



Slide credit: Dhruv Batra

Drawbacks of Policy Gradients

Georgia Tech

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

- How to reduce the variance?
  - Subtract an action independent baseline from the reward

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

- Why does it work?
- What is the best choice of b?

Homework

Drawbacks of Policy Gradients

Georgia Tech

- REINFORCE, use raw reward values

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\mathcal{R}(s,a)\right]$$

- Actor-critic, use Q-values (learnt from data)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)Q^{\pi_\theta}(s,a)\right]$$

- Advantage actor-critic, use Q minus V values (i.e. Advantage)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s)\left(Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)\right)\right]$$

Policy Gradient Variants