# Lecture 5: Policy Gradient I

Emma Brunskill

CS234 Reinforcement Learning

Winter 2026

- With many slides from or derived from David Silver and John Schulman and Pieter Abbeel
- Additional reading: Sutton and Barto 2018 Chp. 13

# Refresh Your Knowledge. Polleverywhere Poll

- Which of the following equations express a TD update? (optional)
  1. $V(s_t) = r(s_t, a_t) + \gamma \sum_{s'} p(s'|s_t, a_t) V(s')$
  2. $V(s_t) = (1-\alpha)V(s_t) + \alpha(r(s_t, a_t) + \gamma V(s_{t+1}))$
  3. $V(s_t) = (1-\alpha)V(s_t) + \alpha \sum_{i=t}^{H} r(s_i, a_i)$
  4. $V(s_t) = (1-\alpha)V(s_t) + \alpha \max_a (r(s_t, a) + \gamma V(s_{t+1}))$
  5. Not sure
- Bootstrapping is (enter in poll)
  1. When samples of (s,a,s') transitions are used to approximate the true expectation over next states
  2. When an estimate of the next state value is used instead of the true next state value
  3. Used in Monte-Carlo policy evaluation
  4. Not sure

# Refresh Your Knowledge L4. Polleverywhere Poll

- Which of the following equations express a TD update?
  $V(s_t) = (1-\alpha)V(s_t) + \alpha(r(s_t, a_t) + \gamma V(s_{t+1}))$

- Bootstrapping is when:
  An estimate of the next state value is used instead of the true next state value

# Class Structure

- Last time: Learning to Control in Tabular MDPs to Deep RL / Generalization to scale RL
- **This time: DQN and Policy Search**
- Next time: Policy Search Cont.

# Recall: Incremental Model-Free Control Approaches

- Similar to policy evaluation, true state-action value function for a state is unknown and so substitute a target value for true $Q(s_t, a_t)$

$$\Delta \boldsymbol{w} = \alpha(Q(s_t, a_t) - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$$

- In Monte Carlo methods, use a return $G_t$ as a substitute target

$$\Delta \boldsymbol{w} = \alpha(G_t - \hat{Q}(s_t, a_t; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_t, a_t; \boldsymbol{w})$$

- SARSA: Use TD target $r + \gamma \hat{Q}(s', a'; \boldsymbol{w})$ which leverages the current function approximation value

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

- Q-learning: Uses related TD target $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$

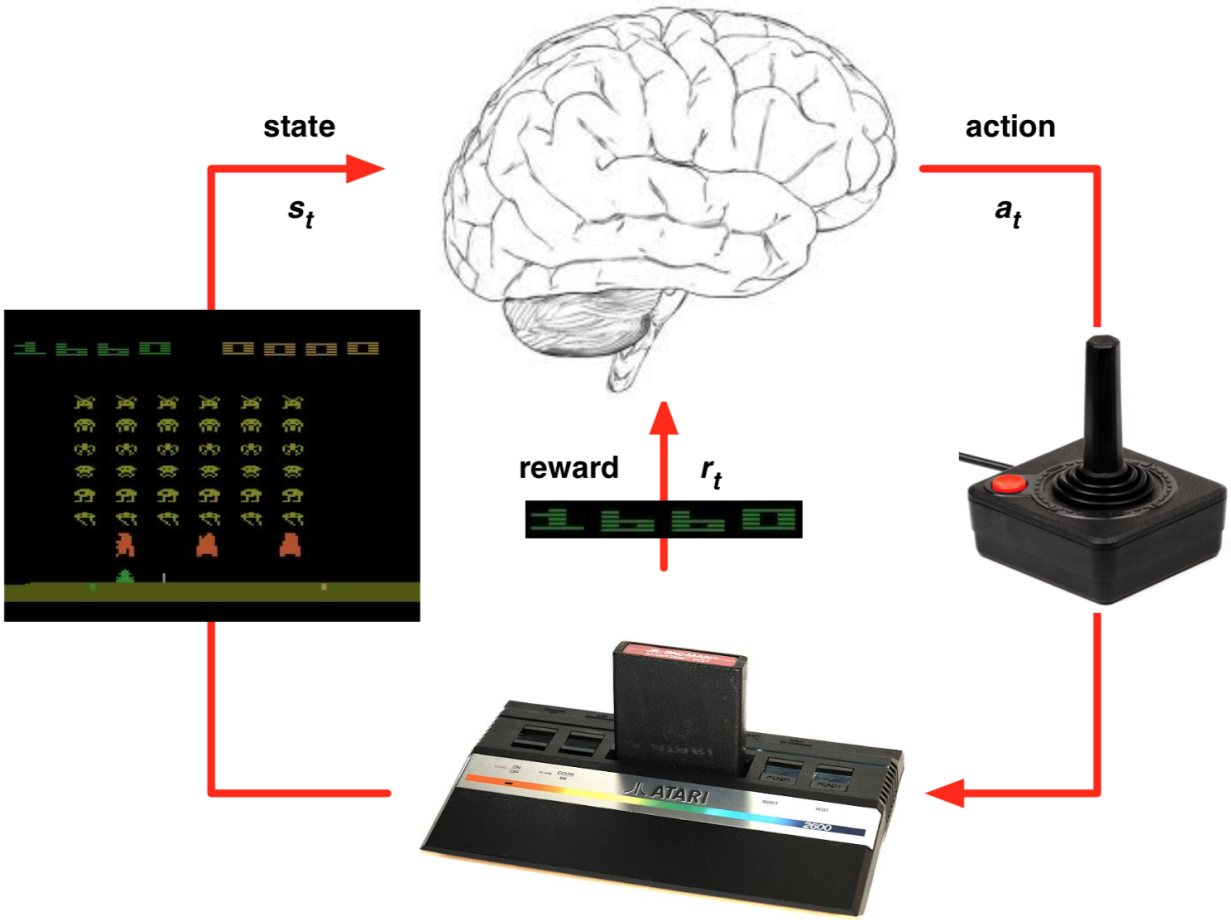
$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

# "Deadly Triad" which Can Cause Instability

- Informally, updates involve doing an (approximate) Bellman backup followed by best trying to fit underlying value function to a particular feature representation
- Bellman operators are contractions, but value function approximation fitting can be an expansion
  - To learn more, see Baird example in Sutton and Barto 2018
- "Deadly Triad" can lead to oscillations or lack of convergence
  - Bootstrapping
  - Function Approximation
  - Off policy learning (e.g. Q-learning)

# Table of Contents

# Using these ideas to do Deep RL in Atari



# Q-Learning with Neural Networks

- Q-learning converges to optimal $Q^*(s, a)$ using tabular representation
- In value function approximation Q-learning minimizes MSE loss by stochastic gradient descent using a target $Q$ estimate instead of true $Q$
- But Q-learning with VFA can diverge
- Two of the issues causing problems:
  - Correlations between samples
  - Non-stationary targets
- Deep Q-learning (DQN) addresses these challenges by using
  - Experience replay
  - Fixed Q-targets

# DQNs: Experience Replay

- To help remove correlations, store dataset (called a **replay buffer**) $\mathcal{D}$ from prior experience

| |
|---|
| $s_1, a_1, r_2, s_2$ |
| $s_2, a_2, r_3, s_3$ |
| $s_3, a_3, r_4, s_4$ |
| ... |
| $s_t, a_t, r_{t+1}, s_{t+1}$ |

$\rightarrow \quad s, a, r, s'$

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled $s$: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

# DQNs: Experience Replay

- To help remove correlations, store dataset $\mathcal{D}$ from prior experience

| |
|---|
| $s_1, a_1, r_2, s_2$ |
| $s_2, a_2, r_3, s_3$ |
| $s_3, a_3, r_4, s_4$ |
| ... |
| $s_t, a_t, r_{t+1}, s_{t+1}$ |

$\rightarrow \quad s, a, r, s'$

- To perform experience replay, repeat the following:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled $s$: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w})$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

- **Uses target as a scalar, but function weights will get updated on the next round, changing the target value**

# DQNs: Fixed $Q$-Targets

- To help improve stability, fix the **target weights** used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let parameters $\boldsymbol{w}^-$ be the set of weights used in the target, and $\boldsymbol{w}$ be the weights that are being updated
- Slight change to computation of target value:
  - $(s, a, r, s') \sim \mathcal{D}$: sample an experience tuple from the dataset
  - Compute the target value for the sampled $s$: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$
  - Use stochastic gradient descent to update the network weights

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

# DQN Pseudocode

Input $C$, $\alpha$, $D = \{\}$, Initialize $\boldsymbol{w}$, $\boldsymbol{w}^- = \boldsymbol{w}$, $t = 0$
Get initial state $s_0$
**loop**
    Sample action $a_t$ given $\epsilon$-greedy policy for current $\hat{Q}(s_t, a; \boldsymbol{w})$
    Observe reward $r_t$ and next state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $D$
    Sample random minibatch of tuples $(s_i, a_i, r_i, s_{i+1})$ from $D$
    **for** $j$ in minibatch **do**
        **if** episode terminated at step $i + 1$ **then**
            $y_i = r_i$
        **else**
            $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \boldsymbol{w}^-)$
        **end if**
        Do gradient descent step on $(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))^2$ for parameters $\boldsymbol{w}$: $\Delta \boldsymbol{w} = \alpha(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_i, a_i; \boldsymbol{w})$
    **end for**
    $t = t + 1$
    **if** mod(t,C) == 0 **then**
        $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$
    **end if**
**end loop**

Note there are several hyperparameters and algorithm choices. One needs to choose the neural network architecture, the learning rate, and how often to update the target network. Often a fixed size replay buffer is used for experience replay, which introduces a parameter to control the size, and the need to decide how to populate it.

# Check Your Understanding : Fixed Targets

- In DQN we compute the target value for the sampled $(s, a, r, s)$ using a separate set of target weights: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$
- Select all that are true
- This doubles the computation time compared to a method that does not have a separate set of weights
- This doubles the memory requirements compared to a method that does not have a separate set of weights
- Not sure

# Check Your Understanding : Fixed Targets. **Solutions**

- In DQN we compute the target value for the sampled $(s, a, r, s')$ using a separate set of target weights: $r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$
- Select all that are true
- This doubles the computation time compared to a method that does not have a separate set of weights
- This doubles the memory requirements compared to a method that does not have a separate set of weights
- Not sure

Answer: It doubles the memory requirements.

# DQNs Summary

- DQN uses experience replay and fixed Q-targets
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions $(s, a, r, s')$ from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters $\boldsymbol{w}^-$
- Optimizes MSE between Q-network and Q-learning targets
- Uses stochastic gradient descent

# DQNs in Atari

- End-to-end learning of values $Q(s, a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step
- Used a deep neural network with CNN
- Network architecture and hyperparameters fixed across all games

# DQN

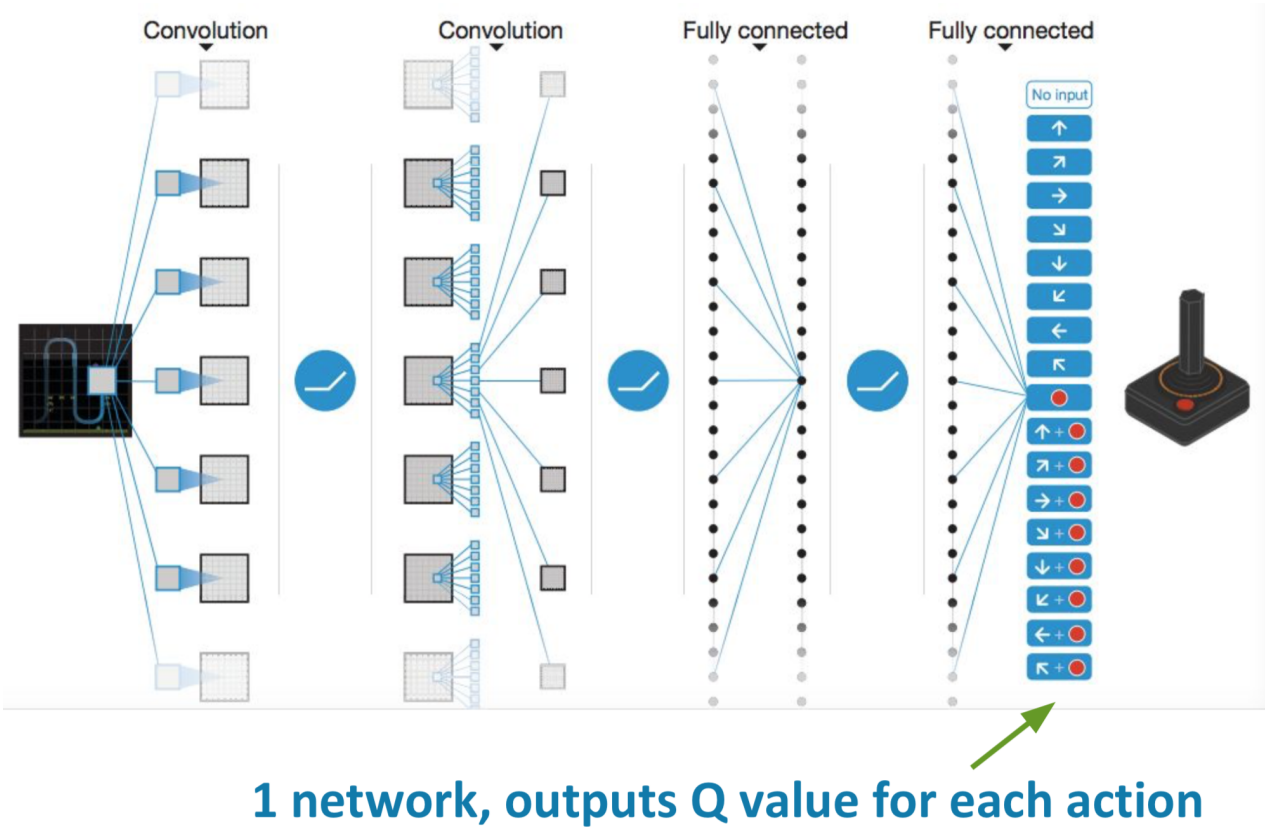



Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

# DQN Results in Atari

Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

# Which Aspects of DQN were Important for Success?

| Game | Linear | Deep Network | DQN w/ fixed Q | DQN w/ replay | DQN w/replay and fixed Q |
|---|---|---|---|---|---|
| Breakout | 3 | 3 | 10 | 241 | 317 |
| Enduro | 62 | 29 | 141 | 831 | 1006 |
| River Raid | 2345 | 1453 | 2868 | 4102 | 7447 |
| Seaquest | 656 | 275 | 1003 | 823 | 2894 |
| Space Invaders | 301 | 302 | 373 | 826 | 1089 |

- Replay is **hugely** important
- Why? Beyond helping with correlation between samples, what does replaying do?

# What You Should Understand from Model-Free RL Lectures

- Be able to implement TD(0) and MC on policy evaluation
- Be able to implement Q-learning and MC control algorithms
- List the 3 issues that can cause instability and describe the problems qualitatively: function approximation, bootstrapping and off-policy learning
- Know some of the key features in DQN that were critical (experience replay, fixed targets)

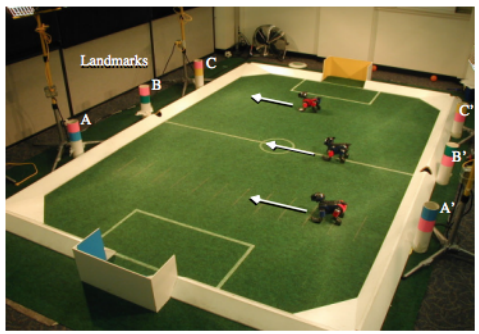
+ func approx

# Class Structure

- Last time and start of this time: Model-free reinforcement learning with function approximation
- Policy gradients

# Do We Need "RL" at All? Can we Just do Online Optimization?

- Policy gradient methods have been **very** influential
- In NLP (Sequence Level Training with Recurrent Neural Networks built on REINFORCE)
- End-to-End Training of Deep Visuomotor Policies `https://arxiv.org/abs/1504.00702`
- ChatGPT and beyond!
- In homework 2 you will be implementing Proximal Policy Optimization (PPO) which was used in training ChatGPT



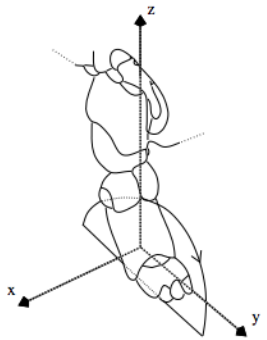




Figure: Early example of policy gradient methods: training a AIBO to have a faster walk. Paper: Kohl and Stone, ICRA 2004

# Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters $w$,

$$V_w(s) \approx V^\pi(s)$$

$$Q_w(s,a) \approx Q^\pi(s,a)$$

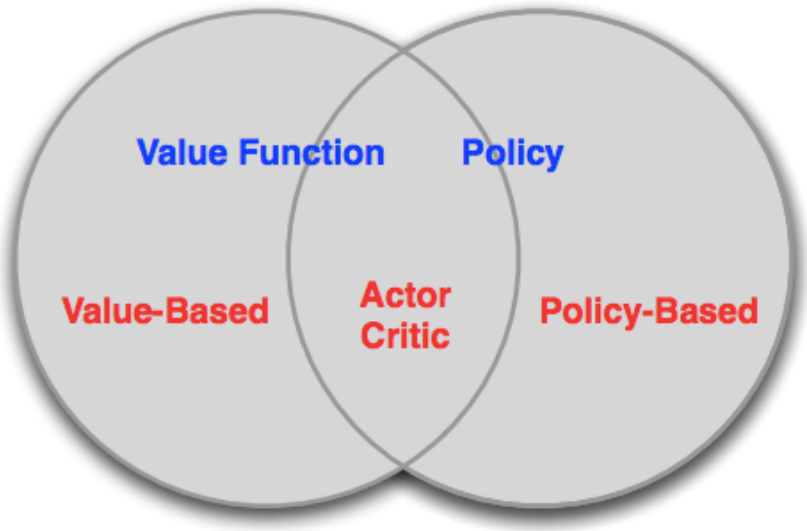
- A policy was generated directly from the value function
  - e.g. using $\epsilon$-greedy
- In this lecture we will directly parametrize the policy, and will typically use $\theta$ to show parameterization:

$$\pi_\theta(s,a) = \mathbb{P}[a|s;\theta]$$

- Goal is to find a policy $\pi$ with the highest value function $V^\pi$
- We will focus again on model-free reinforcement learning

# Value-Based and Policy-Based RL

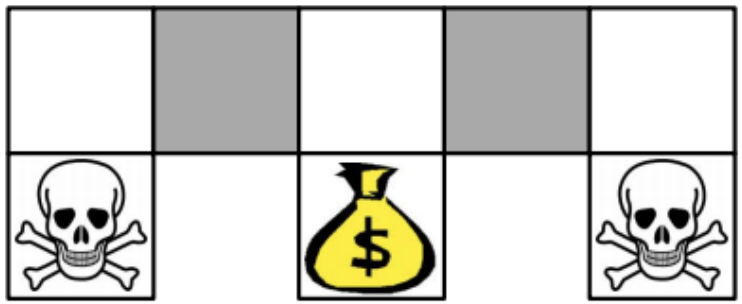
- Value Based
  - learned Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)
- Policy Based
  - No Value Function
  - Learned Policy
- Actor-Critic
  - Learned Value Function
  - Learned Policy



# Types of Policies to Search Over

- So far have focused on deterministic policies or $\epsilon$-greedy policies
- Now we are thinking about direct policy search in RL, will focus heavily on stochastic policies

# Example: Aliased Gridword (1)



- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\phi(s, a) = \mathbb{1}(\text{wall to N}, a = \text{move E})$$

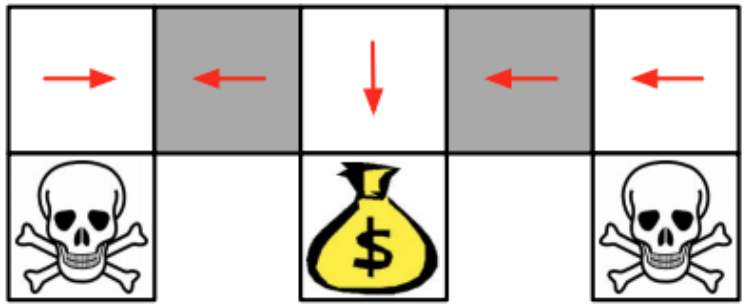
- Compare value-based RL, using an approximate value function

$$Q_\theta(s, a) = f(\phi(s, a); \theta)$$

- To policy-based RL, using a parametrized policy

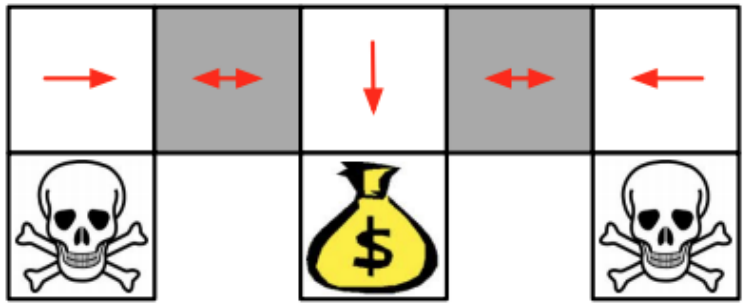
$$\pi_\theta(s, a) = g(\phi(s, a); \theta)$$

# Example: Aliased Gridworld (2)



- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - move E in both grey states
- Either way, it can get stuck and never reach the money
- Value-based RL learns a near-deterministic policy
  - e.g. greedy or $\epsilon$-greedy
- So it will traverse the corridor for a long time

# Example: Aliased Gridworld (3)



- An optimal stochastic policy will randomly move E or W in grey states

$$\pi_\theta(\text{wall to N and S, move E}) = 0.5$$

$$\pi_\theta(\text{wall to N and S, move W}) = 0.5$$

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

# Policy optimization

- Policy based reinforcement learning is an optimization problem
- Find policy parameters $\theta$ that maximize $V(s_0, \theta)$
- Can use gradient free optimization:
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

# Policy Gradient

- Define $V^{\pi_\theta} = V(s_0, \theta)$ to make explicit the dependence of the value on the policy parameters
- Assume episodic MDPs
- Policy gradient algorithms search for a *local* maximum in $V(s_0, \theta)$ by ascending the gradient of the policy, w.r.t parameters $\theta$

$$\Delta\theta = \alpha \nabla_\theta V(s_0, \theta)$$

- Where $\nabla_\theta V(s_0, \theta)$ is the policy gradient

$$\nabla_\theta V(s_0, \theta) = \begin{pmatrix} \frac{\partial V(s_0,\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial V(s_0,\theta)}{\partial \theta_n} \end{pmatrix}$$

- and $\alpha$ is a step-size parameter

# Value of a Parameterized Policy

- Now assume policy $\pi_\theta$ is differentiable whenever it is non-zero and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Recall policy value is $V(s_0, \theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0\right]$ where the expectation is taken over the states & actions visited by $\pi_\theta$
- We can re-express this in multiple ways
  - $V(s_0, \theta) = \sum_a \pi_\theta(a|s_0) Q(s_0, a, \theta)$

$= \sum_\tau P(\tau|\theta) R(\tau)$ ← $\sum$ rewards in a traj also called $G_t$ in prior lectures

$\tau$ = traj / episode

approx w/ sampling

# Value of a Parameterized Policy

- Assume policy $\pi_\theta$ is differentiable whenever it is non-zero and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$
- Recall policy value is $V(s_0, \theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0\right]$ where the expectation is taken over the states & actions visited by $\pi_\theta$
- We can re-express this in multiple ways
  - $V(s_0, \theta) = \sum_a \pi_\theta(a|s_0) Q(s_0, a, \theta)$
  - $V(s_0, \theta) = \sum_\tau P(\tau; \theta) R(\tau)$
    - where $\tau = (s_0, a_0, r_0, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ is a state-action trajectory,
    - $P(\tau; \theta)$ is used to denote the probability over trajectories when executing policy $\pi(\theta)$ starting in state $s_0$, and
    - $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ the sum of rewards for a trajectory $\tau$
- To start will focus on this latter definition. See Chp 13.1-13.3 of SB for a nice discussion starting with the other definition

# Likelihood Ratio Policies

- Denote a state-action trajectory as $\tau = (s_0, a_0, r_0, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T)$
- Use $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ to be the sum of rewards for a trajectory $\tau$
- Policy value is

$$V(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta\right] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

- where $P(\tau; \theta)$ is used to denote the probability over trajectories when executing policy $\pi(\theta)$
- In this new notation, our goal is to find the policy parameters $\theta$:

$$\arg\max_{\theta} V(\theta) = \arg\max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

# Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters $\theta$:

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Take the gradient with respect to $\theta$:

$$\nabla_\theta V(\theta) = \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

$$= \sum_\tau R(\tau) \nabla_\theta P(\tau;\theta)$$

$$= \sum_\tau R(\tau) \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)$$

$$= \sum_\tau R(\tau) P(\tau;\theta) \nabla_\theta \log P(\tau;\theta)$$

# Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters $\theta$:

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Take the gradient with respect to $\theta$:

$$\begin{aligned}
\nabla_\theta V(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau P(\tau;\theta)R(\tau) \underbrace{\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}}_{\text{likelihood ratio}} \\
&= \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)
\end{aligned}$$

# Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters $\theta$:

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Take the gradient with respect to $\theta$:

$$\nabla_\theta V(\theta) = \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)$$

- Approximate using $m$ sample trajectories under policy $\pi_\theta$:

$$\nabla_\theta V(\theta) \approx \hat{g} = (1/m)\sum_{i=1}^{m} R(\tau^{(i)})\nabla_\theta \log P(\tau^{(i)};\theta)$$

# Decomposing the Trajectories Into States and Actions

- Approximate using $m$ sample paths under policy $\pi_\theta$:

$$\nabla_\theta V(\theta) \approx \hat{g} = (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log P(\tau^{(i)})$$

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[ \mu(s_0) \prod_{t=0}^{T-1} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t) \right]$$

$$= \nabla_\theta \log \mu(s_0) + \sum_{t=0}^{T-1} \nabla_\theta \log p(s_{t+1} | s_t, a_t) + \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t)$$

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t)$$

# Decomposing the Trajectories Into States and Actions

- Approximate using $m$ sample paths under policy $\pi_\theta$:

$$\nabla_\theta V(\theta) \approx \hat{g} = (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log P(\tau^{(i)})$$

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log \left[ \underbrace{\mu(s_0)}_{\text{Initial state distrib.}} \prod_{t=0}^{T-1} \underbrace{\pi_\theta(a_t|s_t)}_{\text{policy}} \underbrace{P(s_{t+1}|s_t, a_t)}_{\text{dynamics model}} \right] \\
&= \nabla_\theta \left[ \log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t, a_t) \right] \\
&= \sum_{t=0}^{T-1} \underbrace{\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{no dynamics model required!}}
\end{aligned}$$

# Decomposing the Trajectories Into States and Actions

- Approximate using $m$ sample paths under policy $\pi_\theta$:

$$\nabla_\theta V(\theta) \approx \hat{g} = (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log P(\tau^{(i)})$$

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log \left[ \underbrace{\mu(s_0)}_{\text{Initial state distrib.}} \prod_{t=0}^{T-1} \underbrace{\pi_\theta(a_t|s_t)}_{\text{policy}} \underbrace{P(s_{t+1}|s_t, a_t)}_{\text{dynamics model}} \right] \\
&= \nabla_\theta \left[ \log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t, a_t) \right] \\
&= \sum_{t=0}^{T-1} \underbrace{\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\textbf{score function}}
\end{aligned}$$

# Score Function

- A score function is the derivative of the log of a parameterized probability / likelihood
- Example: let $\pi(s;\theta)$ be the probability of state $s$ under parameter $\theta$
- Then the score function would be

$$\nabla_\theta \log \pi(s;\theta) \tag{1}$$

- For many policy classes, it is not hard to compute the score function

# Softmax Policy

- Weight actions using linear combination of features $\phi(s,a)^T\theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) = e^{\phi(s,a)^T\theta} / (\sum_a e^{\phi(s,a)^T\theta})$$

- The score function is $\nabla_\theta \log \pi_\theta(s,a) = \nabla_\theta \log \left[ \frac{e^{\phi(s,a)^T\theta}}{\sum_a e^{\phi(s,a)^T\theta}} \right]$

$$= \nabla_\theta \left[ \phi(s,a)^T\theta - \log\left(\sum_a e^{\phi(s,a)^T\theta}\right) \right]$$

$$= \phi(s,a) - \frac{1}{\sum_a e^{\phi(s,a)^T\theta}} \sum_a \nabla_\theta e^{\phi(s,a)^T\theta}$$

$$= \phi(s,a) - \sum_a \phi(s,a) e^{\phi(s,a)^T\theta}$$

# Softmax Policy

- Weight actions using linear combination of features $\phi(s,a)^T\theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) = e^{\phi(s,a)^T\theta}/(\sum_a e^{\phi(s,a)^T\theta})$$

- The score function is

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(s,a) &= \nabla_\theta \left[\log \left[\frac{e^{\phi(s,a)^T\theta}}{\sum_a e^{\phi(s,a)^T\theta}}\right]\right] && (2)\\
&= \nabla_\theta \left[\phi(s,a)\theta - \log \left[\sum_a e^{\phi(s,a)^T\theta}\right]\right] && (3)\\
&= \phi(s,a) - \frac{\sum_a \nabla_\theta e^{\phi(s,a)^T\theta}}{\sum_a e^{\phi(s,a)^T\theta}} && (4)\\
&= \phi(s,a) - \frac{\sum_a \phi(s,a) e^{\phi(s,a)^T\theta}}{\sum_a e^{\phi(s,a)^T\theta}} && (5)\\
&= \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)] && (6)
\end{aligned}$$

# Softmax Policy

- Weight actions using linear combination of features $\phi(s,a)^T\theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) = e^{\phi(s,a)^T\theta} / (\sum_a e^{\phi(s,a)^T\theta})$$

- The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$$

# Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu(s) = \phi(s)^T \theta$
- Variance may be fixed $\sigma^2$, or can also parametrised
- Policy is Gaussian $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

- Deep neural networks (and other models where can compute the gradient) can also be used to represent the policy

# Likelihood Ratio / Score Function Policy Gradient

- Putting this together
- Goal is to find the policy parameters $\theta$:

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Approximate with empirical estimate for $m$ sample paths under policy $\pi_\theta$ using score function:

$$\begin{aligned}\nabla_\theta V(\theta) &\approx \hat{g} = (1/m)\sum_{i=1}^{m} R(\tau^{(i)})\nabla_\theta \log P(\tau^{(i)};\theta) \\ &= (1/m)\sum_{i=1}^{m} R(\tau^{(i)})\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\end{aligned}$$

- Do not need to know dynamics model

# L5N2 Check Your Understanding L5: Score functions

$$\nabla_\theta V(\theta) \quad = \quad (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

The likelihood ratio / score function policy gradient (select one):

- (a) requires reward functions that are differentiable
- (b) can only be used with Markov decision processes
- (c) Is useful mostly for infinite horizon tasks
- (a) and (b)
- a,b and c
- None of the above
- Not sure

# L5N2 Check Your Understanding L5: Score functions Solution

$$\nabla_\theta V(\theta) \quad = \quad (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

The likelihood ratio / score function policy gradient (select one):

- (a) requires reward functions that are differentiable
- (b) can only be used with Markov decision processes
- (c) Is useful mostly for infinite horizon tasks
- (a) and (b)
- a,b and c
- None of the above
- Not sure

None of the above

# Score Function Gradient Estimator: Intuition

- Consider generic form of $R(\tau^{(i)})\nabla_\theta \log P(\tau^{(i)};\theta)$:
  $\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$
- $f(x)$ measures how good the sample $x$ is.
- Moving in the direction $\hat{g}_i$ pushes up the logprob of the sample, in proportion to how good it is
- *Valid even if $f(x)$ is discontinuous, and unknown, or sample space (containing $x$) is a discrete set*

# Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach

**Theorem**

*For any differentiable policy* $\pi_\theta(s, a)$,
*for any of the policy objective function* $J = J_1$, *(episodic reward),* $J_{avR}$ *(average reward per time step), or* $\frac{1}{1-\gamma} J_{avV}$ *(average value),*
*the policy gradient is*

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

- Chapter 13.2 in SB has a nice derivation of the policy gradient theorem for episodic tasks and discrete states

# Table of Contents

# Likelihood Ratio / Score Function Policy Gradient

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- Unbiased but very noisy
- Fixes that can make it practical
  - Temporal structure
  - Baseline

# Policy Gradient: Use Temporal Structure

- Previously:

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau \left[ \left( \sum_{t=0}^{T-1} r_t \right) \left( \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \right]$$

- We can repeat the same argument to derive the gradient estimator for a single reward term $r_{t'}$.

$$\nabla_\theta \mathbb{E}[r_{t'}] = \mathbb{E} \left[ r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

- To see this, recall $V(s_0, \theta) = \mathbb{E}_{\pi_\theta} \left[ \sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0 \right]$ where the expectation is taken over the states & actions visited by $\pi_\theta$

# Policy Gradient: Use Temporal Structure

$t=3 \quad P(\tau_{0:3}|\theta) r_3$

- Previously:

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau \left[ \left( \sum_{t=0}^{T-1} r_t \right) \left( \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \right]$$

- We can repeat the same argument to derive the gradient estimator for a single reward term $r_{t'}$.

$$\nabla_\theta \mathbb{E}[r_{t'}] = \mathbb{E} \left[ r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

- Summing this formula over t, we obtain

$$\nabla_\theta V(\theta) = \nabla_\theta \mathbb{E}[R] = \mathbb{E} \left[ \sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

# Policy Gradient: Use Temporal Structure

- Previously:

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau \left[ \left( \sum_{t=0}^{T-1} r_t \right) \left( \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \right]$$

- We can repeat the same argument to derive the gradient estimator for a single reward term $r_{t'}$.

$$\nabla_\theta \mathbb{E}[r_{t'}] = \mathbb{E} \left[ r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

- Summing this formula over t, we obtain

$$\nabla_\theta V(\theta) = \nabla_\theta \mathbb{E}[R] = \mathbb{E} \left[ \sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

$$= \mathbb{E} \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) \sum_{t'=t}^{T-1} r_{t'} \right]$$

# Policy Gradient: Use Temporal Structure

- Recall for a particular trajectory $\tau^{(i)}$, $\sum_{t'=t}^{T-1} r_{t'}^{(i)}$ is the return $G_t^{(i)}$

$$\nabla_\theta \mathbb{E}[R] \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) G_t^{(i)}$$

# Monte-Carlo Policy Gradient (REINFORCE)

- Leverages likelihood ratio / score function and temporal structure

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$$

**REINFORCE:**

Initialize policy parameters $\theta$ arbitrarily

**for** each episode $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

  **for** $t = 1$ to $T - 1$ **do**

    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$

  **endfor**

**endfor**

**return** $\theta$

# Likelihood Ratio / Score Function Policy Gradient

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- Unbiased but very noisy
- Fixes that can make it practical
  - Temporal structure
  - **Baseline**
  - Alternatives to using Monte Carlo returns $R(\tau^{(i)})$ as targets

# Desired Properties of a Policy Gradient RL Algorithm

- Goal: Converge as quickly as possible to a local optima
  - Incurring reward / cost as execute policy, so want to minimize number of iterations / time steps until reach a good policy

# Table of Contents

# Policy Gradient: Introduce Baseline

- Reduce variance by introducing a *baseline* $b(s)$

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t;\theta)\left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t)\right)\right]$$

- For any choice of $b$, gradient estimator is unbiased.
- Near optimal choice is the expected return,

$$b(s_t) \approx \mathbb{E}[r_t + r_{t+1} + \cdots + r_{T-1}]$$

- Interpretation: increase logprob of action $a_t$ proportionally to how much returns $\sum_{t'=t}^{T-1} r_{t'}$ are better than expected