Table of Contents

Greedy in the Limit of Infinite Exploration (GLIE)

Definition of GLIE

- All state-action pairs are visited an infinite number of times

$$\lim_{i \to \infty} N_i(s, a) \to \infty \quad \forall s, a$$

- Behavior policy (policy used to act in the world) converges to greedy policy
  $\lim_{i \to \infty} \pi(a|s) \to \arg\max_a Q(s, a)$ with probability 1

Greedy in the Limit of Infinite Exploration (GLIE)

Definition of GLIE

- All state-action pairs are visited an infinite number of times

$$\lim_{i \to \infty} N_i(s, a) \to \infty$$

- Behavior policy (policy used to act in the world) converges to greedy policy
  $\lim_{i \to \infty} \pi(a|s) \to \arg\max_a Q(s, a)$ with probability 1

- A simple GLIE strategy is ϵ-greedy where ϵ is reduced to 0 with the following rate: $\epsilon_i = 1/i$

and visit all states

GLIE Monte-Carlo Control using Tabular Representations

Theorem

GLIE Monte-Carlo control converges to the optimal state-action value function $Q(s, a) \rightarrow Q^*(s, a)$

Table of Contents

Model-free Policy Iteration with TD Methods

- Initialize policy π
- Repeat:
  - Policy evaluation: compute Q^π using temporal difference updating with ϵ-greedy policy
  - Policy improvement: Same as Monte carlo policy improvement, set π to ϵ-greedy (Q^π)
- Method 1: SARSA s,a,r,s',a'
- On policy: SARSA computes an estimate Q of policy used to act

General Form of SARSA Algorithm

1: Set initial ϵ-greedy policy π randomly, $t = 0$, initial state $s_t = s_0$
2: Take $a_t \sim \pi(s_t)$
3: Observe (r_t, s_{t+1})
4: **loop**
5: Take action $a_{t+1} \sim \pi(s_{t+1})$ // Sample action from policy
6: Observe (r_{t+1}, s_{t+2})
7: Update Q given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)$
8: Perform policy improvement: $\forall s \quad \pi(s) = \arg\max_a Q(s, a)$ w/prob $1-\epsilon$, otherwise random
9: $t = t + 1$, $\epsilon = 1/t$
10: **end loop**

target

for all

if s_{t+2} is terminal

vent episode sample S

General Form of SARSA Algorithm

1: Set initial ϵ-greedy policy π, $t = 0$, initial state $s_t = s_0$
2: Take $a_t \sim \pi(s_t)$ // Sample action from policy
3: Observe (r_t, s_{t+1})
4: **loop**
5: Take action $a_{t+1} \sim \pi(s_{t+1})$
6: Observe (r_{t+1}, s_{t+2})
7: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
8: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
9: $t = t + 1$ $\epsilon = 1/t$
10: **end loop**

- See worked example with Mars rover at end of slides

Properties of SARSA with ϵ-greedy policies

- Computational complexity?
- Converge to optimal Q^* function? Recall:
  - $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
  - $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
  - *Q is an estimate of the performance of a policy that may be changing at each time step*
- Empirical performance?

Convergence Properties of SARSA

Theorem

SARSA for finite-state and finite-action MDPs converges to the optimal action-value, $Q(s,a) \rightarrow Q^*(s,a)$, under the following conditions:

1. The policy sequence $\pi_t(a|s)$ satisfies the condition of GLIE
2. The step-sizes α_t satisfy the Robbins-Munro sequence such that

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

- For ex. $\alpha_t = \frac{1}{t}$ satisfies the above condition.

Properties of SARSA with ϵ-greedy policies

- Result builds on stochastic approximation
- Relies on step sizes decreasing at the right rate
- Relies on Bellman backup contraction property
- Relies on bounded rewards and value function

1992 1994 papers

On and Off-Policy Learning

- On-policy learning
  - Direct experience
  - Learn to estimate and evaluate a policy from experience obtained from following that policy
- Off-policy learning
  - Learn to estimate and evaluate a policy using experience gathered from following a different policy

Q-Learning: Learning the Optimal State-Action Value

- SARSA is an **on-policy** learning algorithm
- SARSA estimates the value of the current **behavior** policy (policy using to take actions in the world)
- And then updates that (behavior) policy
- Alternatively, can we directly estimate the value of π^* while acting with another behavior policy π_b?
- Yes! Q-learning, an **off-policy** RL algorithm

Q-Learning: Learning the Optimal State-Action Value

- SARSA is an **on-policy** learning algorithm
  - Estimates the value of **behavior** policy (policy using to take actions in the world)
  - And then updates the behavior policy
- Q-learning
  - estimate the Q value of π^* while acting with another behavior policy π_b
- Key idea: Maintain Q estimates and bootstrap for best future value
- Recall SARSA

actual action

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$$

- Q-learning:

$\sum_{s'} p(s'|s,a) V^*(s')$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} Q(s_{t+1}, a')) - Q(s_t, a_t))$$

Q-Learning with ϵ-greedy Exploration

1: Initialize $Q(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
2: Set π_b to be ϵ-greedy w.r.t. Q
3: **loop**
4: Take $a_t \sim \pi_b(s_t)$ // Sample action from policy
5: Observe (r_t, s_{t+1})
6: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
7: $\pi(s_t) = \arg\max_a Q(s_t, a)$ w.prob $1 - \epsilon$, else random
8: $t = t + 1$ $\epsilon = 1/t$
9: **end loop**

See optional worked example and optional understanding check at the end of the slides

Q-Learning with ϵ-greedy Exploration

*tabular

- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal Q^*?
  Visit all (s, a) pairs infinitely often, and the step-sizes α_t satisfy the Robbins-Munro sequence. Note: the algorithm does not have to be greedy in the limit of infinite exploration (GLIE) to satisfy this (could keep ϵ large).

- What conditions are sufficient to ensure that Q-learning with ϵ-greedy exploration converges to optimal π^*?
  The algorithm is GLIE, along with the above requirement to ensure the Q value estimates converge to the optimal Q.

Table of Contents

Table of Contents

Control using Value Function Approximation

- Use value function approximation to represent state-action values $\hat{Q}^{\pi}(s, a; \boldsymbol{w}) \approx Q^{\pi}$
- Interleave
  - Approximate policy evaluation using value function approximation
  - Perform ϵ-greedy policy improvement
- Can be unstable. Generally involves intersection of the following:
  - Function approximation
  - Bootstrapping
  - **Off-policy learning**

Action-Value Function Approximation with an Oracle

- $\hat{Q}^{\pi}(s, a; \boldsymbol{w}) \approx Q^{\pi}$
- Minimize the mean-squared error between the true action-value function $Q^{\pi}(s, a)$ and the approximate action-value function:

$$J(\boldsymbol{w}) = \mathbb{E}_{\pi}[(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a; \boldsymbol{w}))^2]$$

- Use stochastic gradient descent to find a local minimum

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = -2\mathbb{E}\left[(Q^{\pi}(s, a) - \hat{Q}^{\pi}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}^{\pi}(s, a; \boldsymbol{w})\right]$$

- Stochastic gradient descent (SGD) samples the gradient

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)

Geoff Gordon
1995

Table of Contents

Using these ideas to do Deep RL in Atari

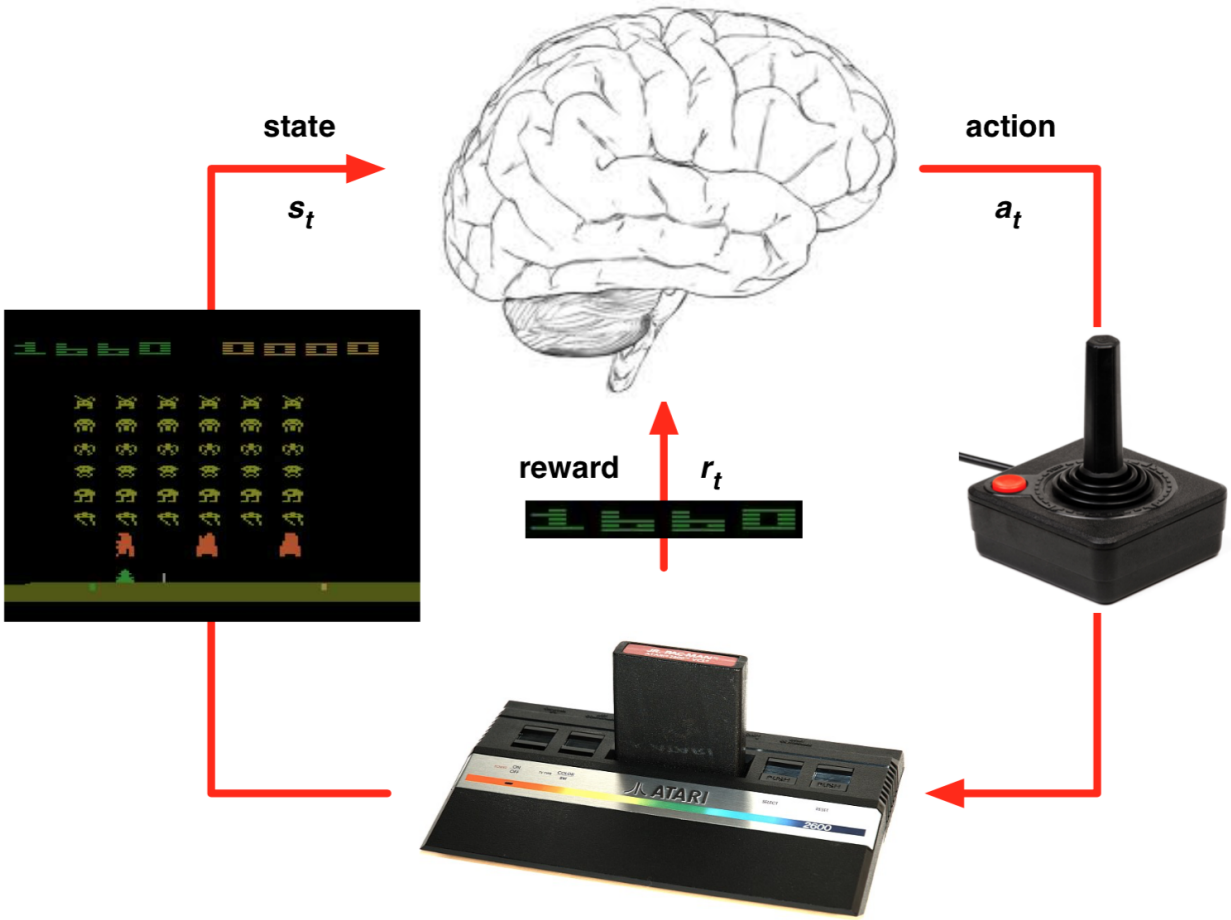


Image of brain: Wikimedia, CC BY 3.0
Image of game setup: Wikimedia, CC BY 2.0

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

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

- **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

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

target weights

DQN Pseudocode

Input $\boldsymbol{C}$, $\boldsymbol{\alpha}$, $D = \{\}$, Initialize $\boldsymbol{w}$, $\boldsymbol{w}^- = \boldsymbol{w}$, $t = 0$
Get initial state s_0
loop
    Sample action a_t given ϵ-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 L4N3: 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 L4N3: 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

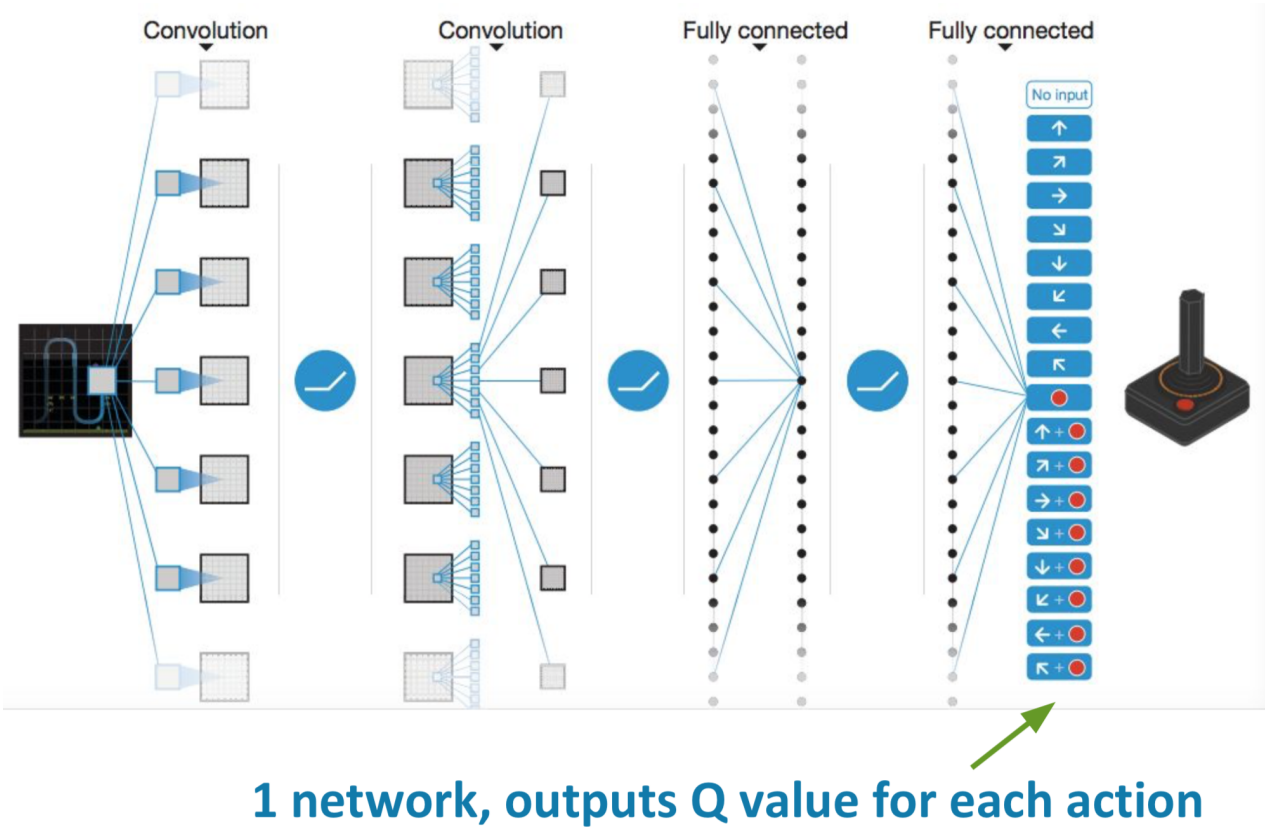


Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

DQN Results in Atari

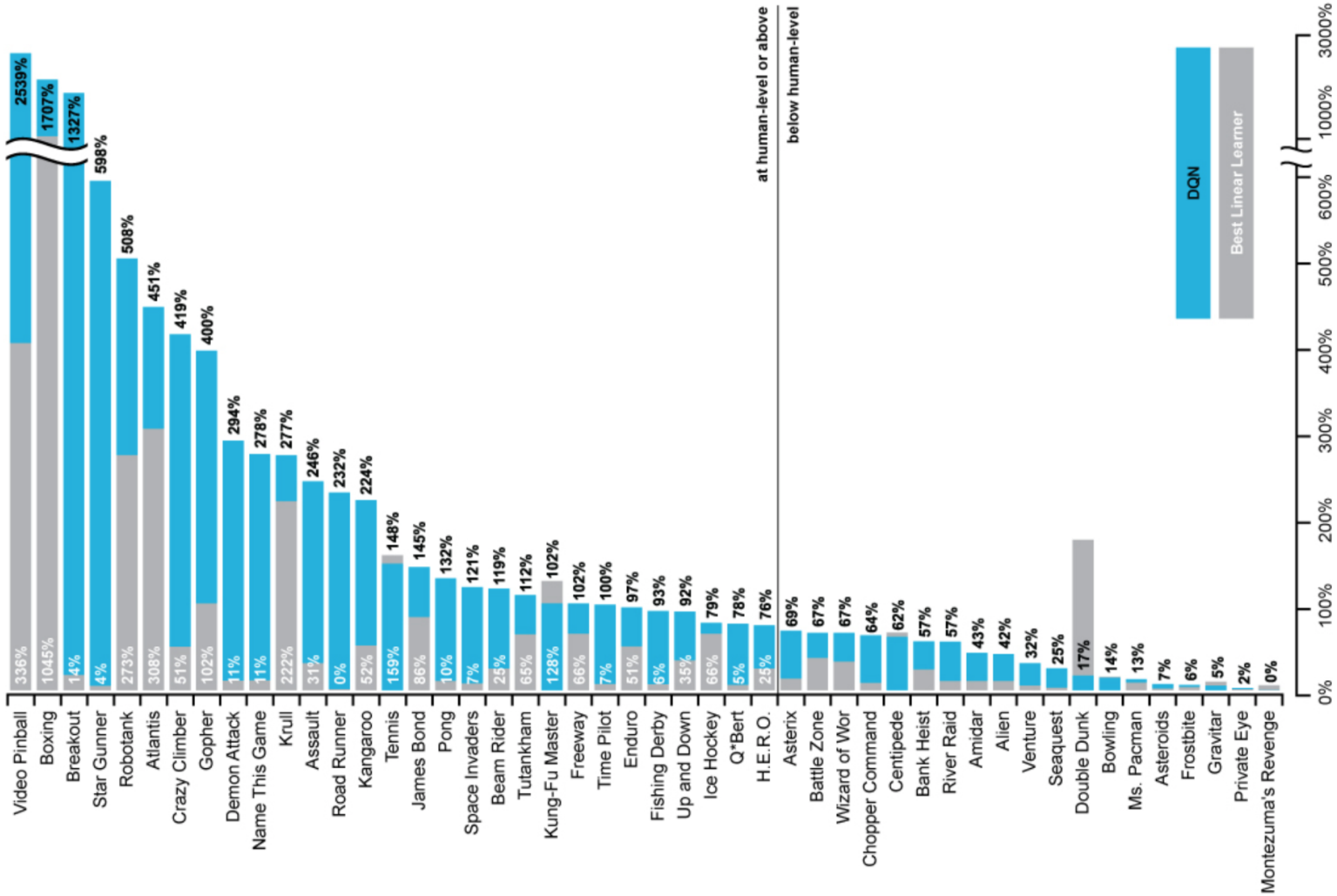


Figure: Human-level control through deep reinforcement learning, Mnih et al, 2015

Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network
Breakout	3	3
Enduro	62	29
River Raid	2345	1453
Seaquest	656	275
Space Invaders	301	302

Note: just using a deep NN actually hurt performance sometimes!

Which Aspects of DQN were Important for Success?

Game	Linear	Deep Network	DQN w/ fixed Q
Breakout	3	3	10
Enduro	62	29	141
River Raid	2345	1453	2868
Seaquest	656	275	1003
Space Invaders	301	302	373

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?

Deep RL

- Success in Atari has led to huge excitement in using deep neural networks to do value function approximation in RL
- Some immediate improvements (many others!)
  - **Double DQN** (Deep Reinforcement Learning with Double Q-Learning, Van Hasselt et al, AAAI 2016)
  - Prioritized Replay (Prioritized Experience Replay, Schaul et al, ICLR 2016)
  - Dueling DQN (best paper ICML 2016) (Dueling Network Architectures for Deep Reinforcement Learning, Wang et al, ICML 2016)

What You Should Understand

- Be able to implement TD(0) and MC on policy evaluation
- Be able to implement Q-learning and SARSA 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)

Class Structure

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

Lecture 5: Policy Gradient I

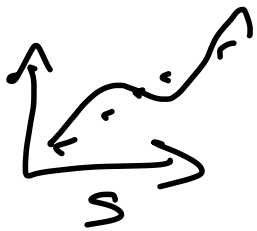
Emma Brunskill

CS234 Reinforcement Learning

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

L5N1 Refresh Your Knowledge. Comparing Policy Performance

- Consider doing experience replay over a finite, but extremely large, set of (s,a,r,s') tuples). Q-learning is initialized to 0 everywhere and all rewards are positive. Select all that are true
  1. Assume all tuples were gathered from a fixed, deterministic policy π. Then in the tabular setting, if each tuple is sampled at random and used to do a Q-learning update, and this is repeated an infinite number of times, then there exists a learning rate schedule so that the resulting estimate will converge to the true Q^{π}.
  2. In situation (1) (the first option above) the resulting Q estimate will be identical to if one computed an estimated dynamics model and reward model using maximum likelihood evaluation from the tuples, and performed policy evaluation using the estimated dynamics and reward models.
  3. If one uses DQN to populate the experience replay set of tuples, then doing experience replay with DQN is always guaranteed to converge to the optimal Q function.
  4. Not sure



L5N1 Refresh Your Knowledge. Comparing Policy Performance

- Consider doing experience replay over a finite, but extremely large, set of (s,a,r,s') tuples). Q-learning is initialized to 0 everywhere and all rewards are positive. Select all that are true
  1. Assume all tuples were gathered from a fixed, deterministic policy π. Then in the tabular setting, if each tuple is sampled at random and used to do a Q-learning update, and this is repeated an infinite number of times, then there exists a learning rate schedule so that the resulting estimate will converge to the true Q^{π}.
  2. In situation (1) (the first option above) the resulting Q estimate will be identical to if one computed an estimated dynamics model and reward model using maximum likelihood evaluation from the tuples, and performed policy evaluation using the estimated dynamics and reward models.
  3. If one uses DQN to populate the experience replay set of tuples, then doing experience replay with DQN is always guaranteed to converge to the optimal Q function.
  4. Not sure

Class Structure

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

$\pi(s) \rightarrow a$

$\pi(s,a) \rightarrow (0,1)$

RL Algorithms Involve

- Optimization
- Delayed consequences
- Exploration
- Generalization

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`
- In homework 2 you will be implementing Proximal Policy Optimization (PPO) which was used in training ChatGPT

Policy-Based Reinforcement Learning

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

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

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

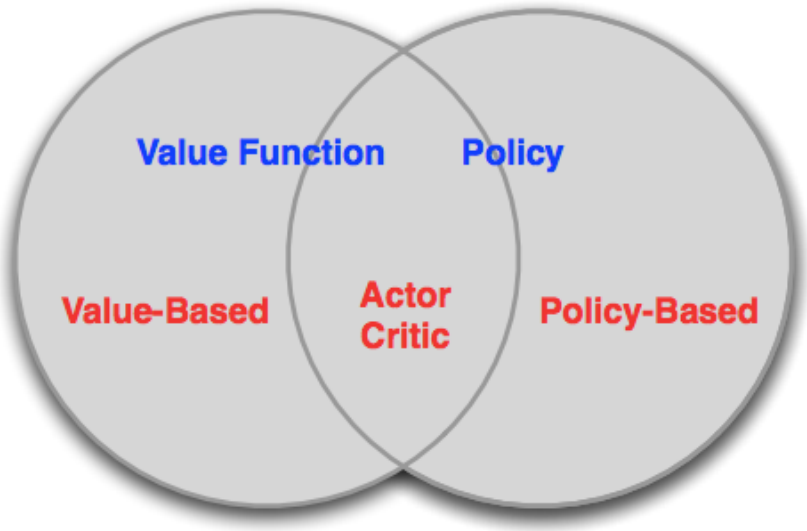
- A policy was generated directly from the value function
  - e.g. using ϵ-greedy
- In this lecture we will directly parametrize the policy, and will typically use θ to show parameterization:

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

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

Value-Based and Policy-Based RL

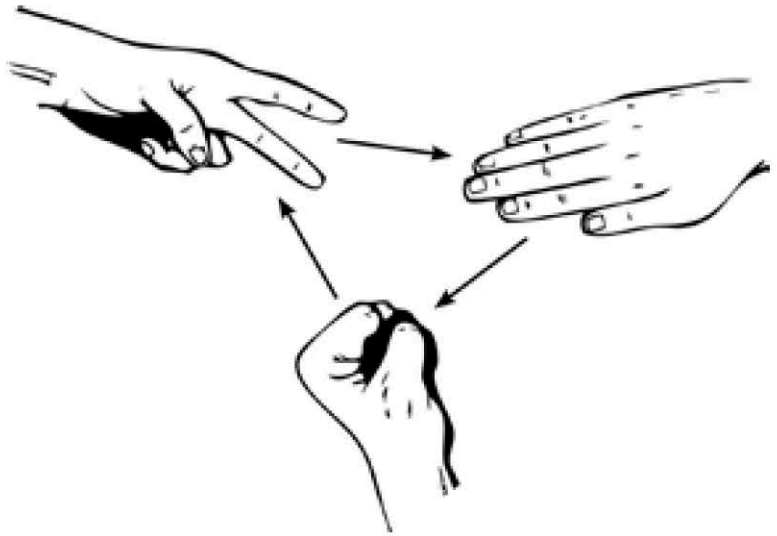
- Value Based
  - learned Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learned Policy
- Actor-Critic
  - Learned Value Function
  - Learned Policy



Types of Policies to Search Over

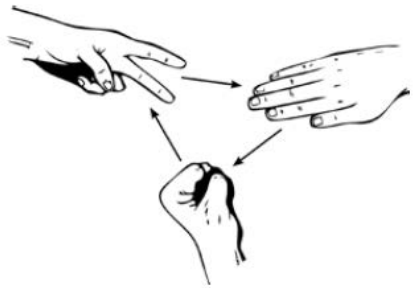
- So far have focused on deterministic policies or ϵ-greedy policies
- Now we are thinking about direct policy search in RL, will focus heavily on stochastic policies

Example: Rock–Paper–Scissors



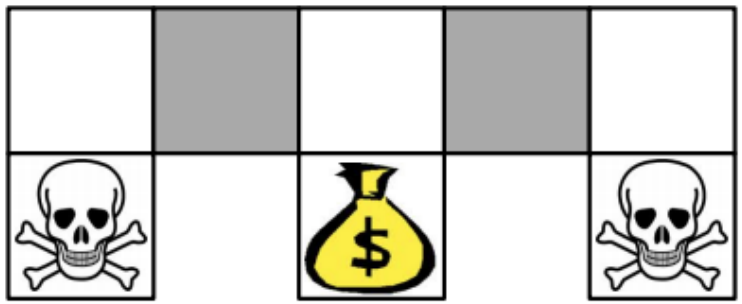
- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Let state be history of prior actions (rock, paper and scissors) and if won or lost
- Is deterministic policy optimal? Why or why not?

Example: Rock-Paper-Scissors, Vote



- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock
- Let state be history of prior actions (rock, paper and scissors) and if won or lost
  Deterministic policy is easily exploited by an adversary. System is not Markov. A uniform random policy is optimal (Nash equilibrium).

Example: Aliased Gridword (1)



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

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

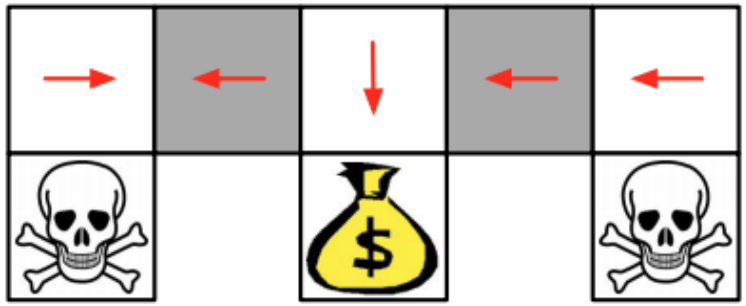
- Compare value-based RL, using an approximate value function

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

- To policy-based RL, using a parametrized policy

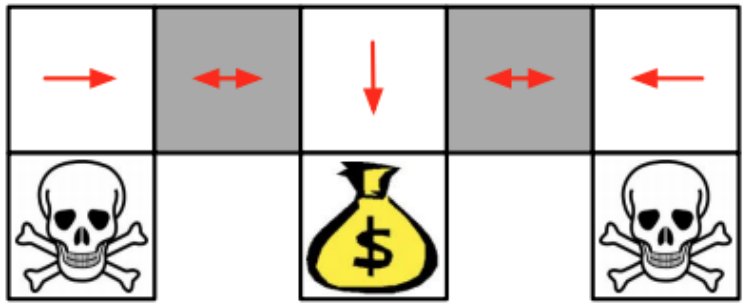
$$\pi_\theta(s, a) = g(\phi(s, a); \theta)$$

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 ϵ-greedy
- So it will traverse the corridor for a long time

Example: Aliased Gridworld (3)



not Markov in state features

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

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

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

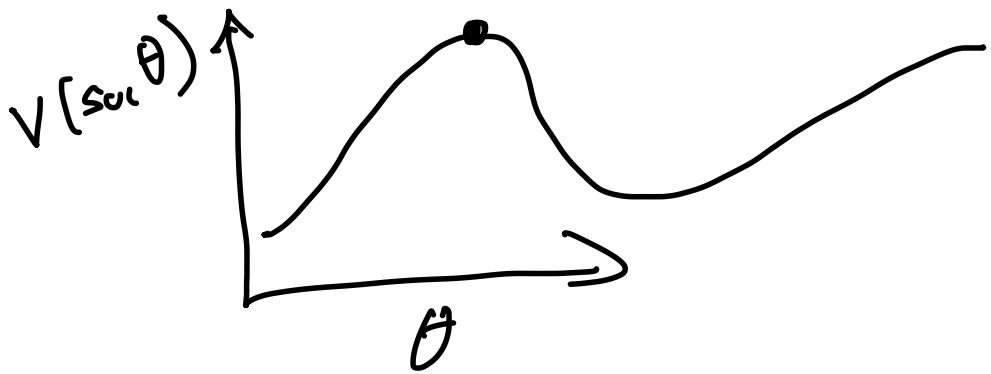
- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Objective Functions

- Goal: given a policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality for a policy π_θ?
- In episodic environments can use policy value at start state $V(s_0, \theta)$
- For simplicity, today will mostly discuss the episodic case, but can easily extend to the continuing / infinite horizon case

Policy optimization

- Policy based reinforcement learning is an optimization problem
- Find policy parameters θ that maximize $V(s_0, \theta)$



Policy optimization

- Policy based reinforcement learning is an optimization problem
- Find policy parameters θ that maximize $V(s_0, \theta)$
- Can use gradient free optimization
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
  - Cross-Entropy method (CEM)
  - Covariance Matrix Adaptation (CMA)

Human-in-the-Loop Exoskeleton Optimization (Zhang et al. Science 2017)

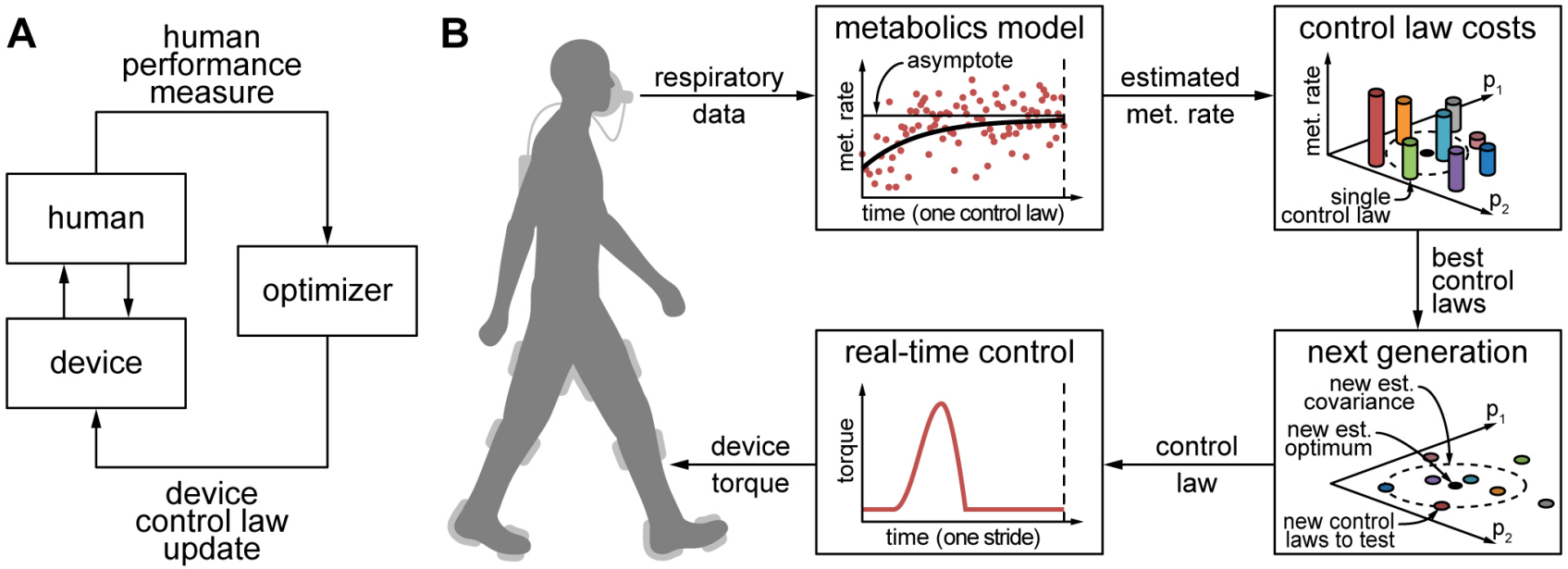


Figure: Zhang et al. Science 2017

- Optimization was done using CMA-ES, variation of covariance matrix evaluation

Gradient Free Policy Optimization

- Can often work embarrassingly well: "discovered that evolution strategies (ES), an optimization technique that's been known for decades, rivals the performance of standard reinforcement learning (RL) techniques on modern RL benchmarks (e.g. Atari/MuJoCo)" (https://blog.openai.com/evolution-strategies/)

Gradient Free Policy Optimization

- Often a great simple baseline to try
- Benefits
  - Can work with any policy parameterizations, including non-differentiable
  - Frequently very easy to parallelize
- Limitations
  - Often less sample efficient because it ignores temporal structure

Policy optimization

- Policy based reinforcement learning is an optimization problem
- Find policy parameters θ 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

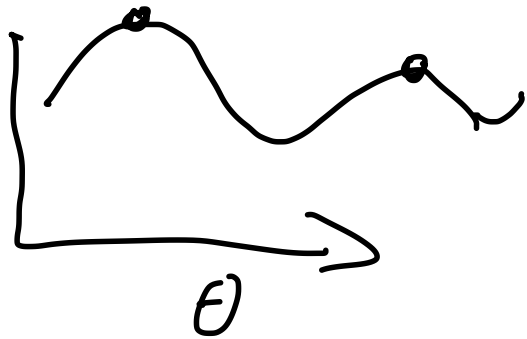
Policy Gradient

- Define $V(\theta) = V(s_0, \theta)$ to make explicit the dependence of the value on the policy parameters [but don't confuse with value function approximation, where parameterized value function]
- Assume episodic MDPs (easy to extend to related objectives, like average reward)

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 θ

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

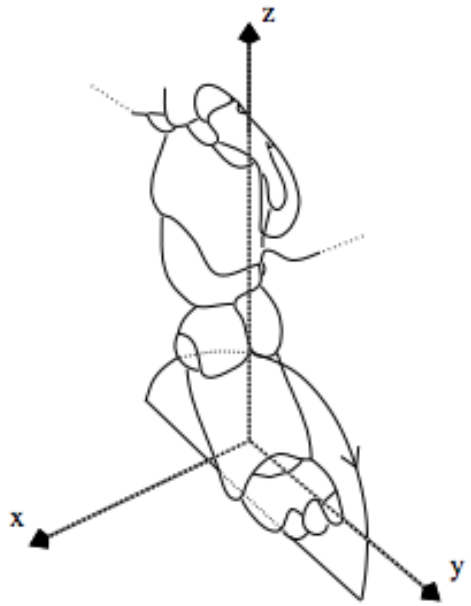


- Where $\nabla_\theta V(s_0, \theta)$ is the policy gradient

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

- and α is a step-size parameter

Example: Training AIBO to Walk by Finite Difference Policy Gradient[1]



- Goal: learn a fast AIBO walk (useful for Robocup)
- Adapt these parameters by finite difference policy gradient
- Evaluate performance of policy by field traversal time

[1] Kohl and Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. ICRA 2004. http://www.cs.utexas.edu/ ai-lab/pubs/icra04.pdf

Summary of Benefits of Policy-Based RL

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Shortly will see some ideas to help with this last limitation

Table of Contents

Computing the gradient analytically

- We now compute the policy gradient *analytically*
- Assume policy π_θ is differentiable whenever it is non-zero
- Assume we can calculate gradient $\nabla_\theta \pi_\theta(s, a)$ analytically
- What kinds of policy classes can we do this for?

Differentiable Policy Classes

$$a = \pi(a|s) \sim \mathcal{N}(0.5, 1)$$

- Many choices of differentiable policy classes including:
  - Softmax
  - Gaussian
  - Neural networks

Value of a Parameterized Policy

no discounting for now

- Now assume policy π_θ is differentiable whenever it is non-zero and we know the gradient $\nabla_\theta \pi_\theta(s, a)$ ← assume finite
- 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 π_θ
- We can re-express this in multiple ways
  - $V(s_0, \theta) = \sum_a \pi_\theta(a|s_0) Q(s_0, a, \theta)$ ← sometimes called this G
  - $= \sum_\tau P(\tau, \theta) R(\tau)$ ← reward for that whole traj

in general intractable but we can approx by sampling

$P(\tau, \theta)$: traj $(s_0, a, s_1, \ldots, s_T)$ sampled from π_θ

Value of a Parameterized Policy

- Now assume policy π_θ 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 π_θ
- 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 τ
- 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 τ
- 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 θ:

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

Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters θ:

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

- Take the gradient with respect to θ:

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

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

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

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

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

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

$$\nabla_\theta \log P(\tau;\theta) = \frac{1}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)$$

Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters θ:

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

- Take the gradient with respect to θ:

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

Likelihood Ratio Policy Gradient

- Goal is to find the policy parameters θ:

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

- Take the gradient with respect to θ:

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

expectation

- Approximate with empirical estimate for m sample trajectories under policy π_θ:

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

Decomposing the Trajectories Into States and Actions

- Approximate with empirical estimate for m sample paths under policy π_θ:

$\mu(s_0) = P(s_0)$

$\tau = (s_0, a_0, r_0, s_1, a_1, r_1 \ldots)$ ← Trajectory

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

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

$$= \nabla_\theta \left[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{T-1} \log P(s_{t+1}|s_{0:t}, a_{0:t}) \right]$$

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

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

Decomposing the Trajectories Into States and Actions

- Approximate with empirical estimate for m sample paths under policy π_θ:

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

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

Decomposing the Trajectories Into States and Actions

- Approximate with empirical estimate for m sample paths under policy π_θ:

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

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

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 θ
- 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 $\nabla_\theta \log \pi_\theta(s,a) =$

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

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

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

$$= \phi(s,a) - \sum_a \pi_\theta(s,a)\, \phi(s,a)$$

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 σ^2, or can also parametrised
- Policy is Gaussian $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

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

Likelihood Ratio / Score Function Policy Gradient

- Putting this together
- Goal is to find the policy parameters θ:

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

- Approximate with empirical estimate for m sample paths under policy π_θ 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

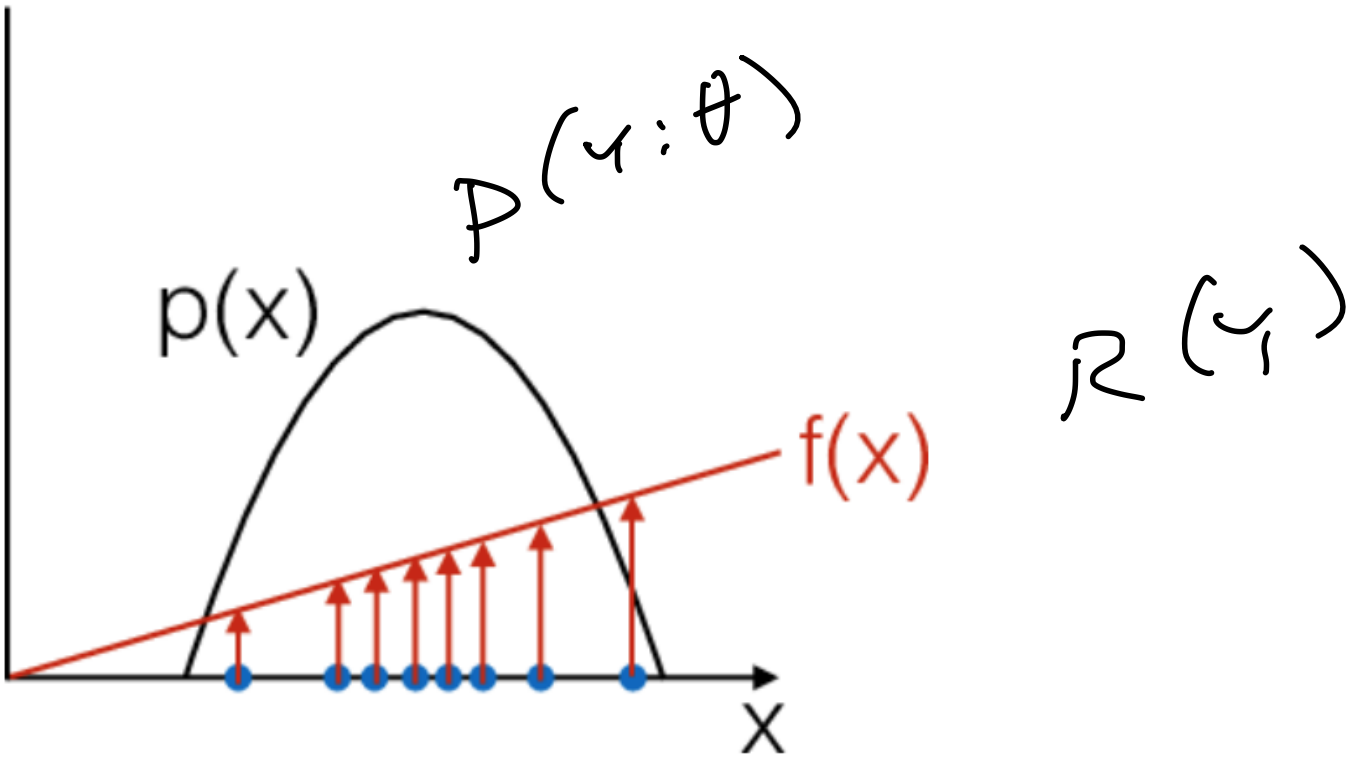
None of the above

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*

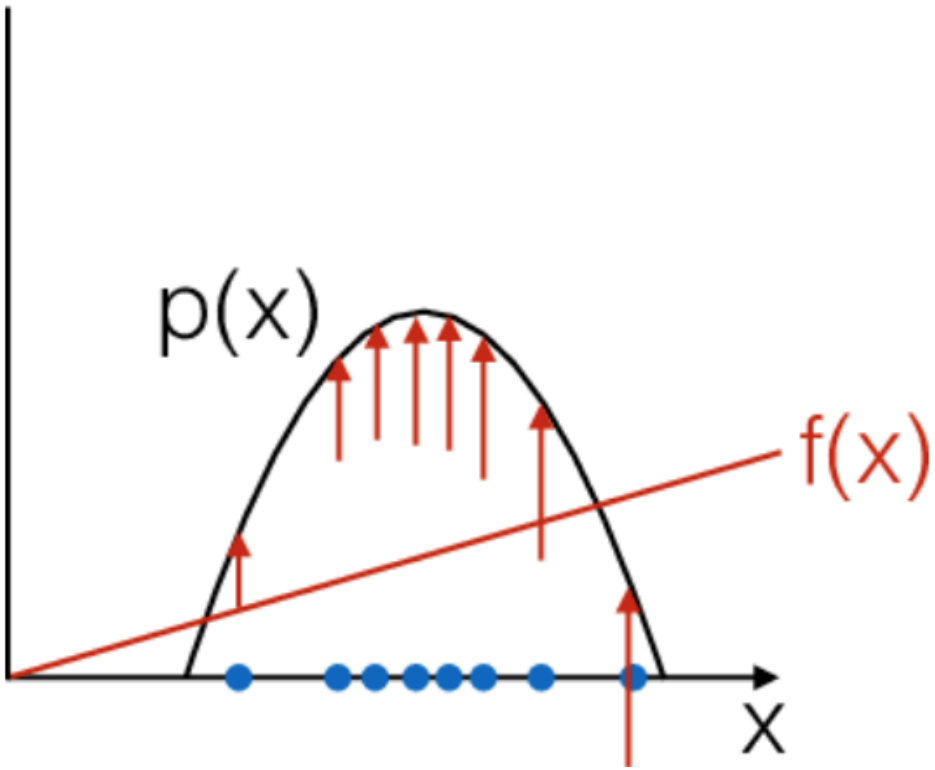
Score Function Gradient Estimator: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$$



Score Function Gradient Estimator: Intuition

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$$



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 π_θ

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

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

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

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

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

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)

~1992

- 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 θ 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 θ

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

Baseline $b(s)$ Does Not Introduce Bias–Derivation

$$\mathbb{E}_\tau[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]$$
$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]\right]$$

Lecture 6: Policy Gradient II. Advanced policy gradient section slides from Joshua Achiam (OpenAI)'s slides, with minor modifications

Emma Brunskill

CS234 Reinforcement Learning.

Refresh Your Knowledge L6N1

- Select all that are true about policy gradients:
  1. $\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$ T
  2. θ is always increased in the direction of $\nabla_\theta \ln(\pi(S_t, A_t, \theta)$. F
  3. State-action pairs with higher estimated Q values will increase in probability on average T
  4. Are guaranteed to converge to the global optima of the policy class F
  5. Not sure

Refresh Your Knowledge L6N1 Solutions

- Select all that are true about policy gradients:
  1. $\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$
  2. θ is always increased in the direction of $\nabla_\theta \ln(\pi(S_t, A_t, \theta)$.
  3. State-action pairs with higher estimated Q values will increase in probability on average
  4. Are guaranteed to converge to the global optima of the policy class
  5. Not sure

1 and 3 are true. The direction of θ also depends on the Q-values /returns. We are only guaranteed to reach a local optima

Class Structure

- Last time: Policy Search
- This time: Policy search continued.

Today

- Likelihood ratio / score function policy gradient
  - Baseline
  - Alternative targets
- Advanced policy gradient methods
  - Proximal policy optimization (PPO) (will implement in homework)

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
  - To obtain data that use to learn, have to make actual decisions which may be suboptimal
  - Aim: minimize number of iterations / time steps until reach a good policy

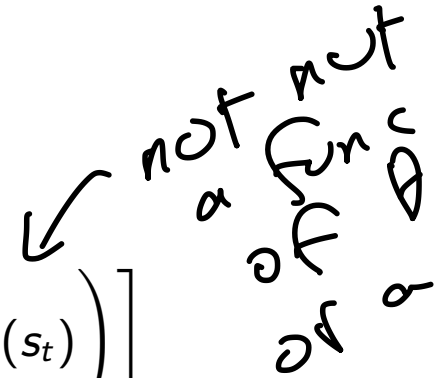
Policy Gradient Algorithms and Reducing Variance

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

Baseline $b(s)$ Does Not Introduce Bias–Derivation

goal is to show $= 0$

$Y = s_{0:T}\ a_{0:T-1}$

$$\mathbb{E}_\tau[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]\right]$$

$$= E_{s_{0:t},a_{0:t-1}}\left[b(s_t)\, E_{s_{t+1:T}\, a_{T:T-1}} \nabla_\theta \log \pi(a_t|s_t,\theta)\right]$$

$$= " \quad " \quad E_{a_t} \nabla_\theta \log \pi(a_t|s_t,\theta)\Big]$$

$$= E_{s_{0:t},a_{0:t-1}}\left[b(s_t) \sum_a \pi(a_t|s_t,\theta) \nabla \log \pi(a_t|s_t,\theta)\right]$$

$$= " \quad " \quad \left[b(s_t) \sum_a \pi(a_t|s_t,\theta) \frac{\nabla_\theta \pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)}\right]$$

$$= E_{s_{0:T},a_{0:t-1}}\ b(s_t) \sum_a \nabla_\theta \pi(a_t|s_t,\theta)$$

$$= " \quad " \quad b(s_t) \nabla_\theta \underbrace{\sum_a \pi(a_t|s_t,\theta)}_{=1}$$

$$= " \quad " \quad b(s_t) \nabla_\theta 1$$

$$= 0$$

Baseline $b(s)$ Does Not Introduce Bias–Derivation

$$\mathbb{E}_\tau[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta)b(s_t)]\right] \text{ (break up expectation)}$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t;\theta)]\right] \text{ (pull baseline term out)}$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[b(s_t)\mathbb{E}_{a_t}[\nabla_\theta \log \pi(a_t|s_t;\theta)]] \text{ (remove irrelevant variables)}$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\sum_a \pi_\theta(a_t|s_t)\frac{\nabla_\theta \pi(a_t|s_t;\theta)}{\pi_\theta(a_t|s_t)}\right] \text{ (likelihood ratio)}$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\sum_a \nabla_\theta \pi(a_t|s_t;\theta)\right]$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\nabla_\theta \sum_a \pi(a_t|s_t;\theta)\right]$$

$$= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[b(s_t)\nabla_\theta 1]$$

$$= \mathbb{E}_{s_0:t,a_0:(t-1)}[b(s_t)\cdot 0] = 0$$

"Vanilla" Policy Gradient Algorithm

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \cdots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep t in each trajectory τ^i, compute
    Return $G_t^i = \sum_{t'=t}^{T-1} r_{t'}^i$, and
    Advantage estimate $\hat{A}_t^i = G_t^i - b(s_t^i)$.
  Re-fit the baseline, by minimizing $\sum_i \sum_t |b(s_t^i) - G_t^i|^2$,
  Update the policy, using a policy gradient estimate $\hat{g}$,
    Which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t$.
    (Plug $\hat{g}$ into SGD or ADAM)
endfor

Other Choices for Baseline?

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \cdots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep t in each trajectory τ^i, compute
    Return $G_t^i = \sum_{t'=t}^{T-1} r_{t'}^i$, and
    Advantage estimate $\hat{A}_t^i = G_t^i - b(s_t^i)$.
  Re-fit the baseline, by minimizing $\sum_i \sum_t |b(s_t^i) - G_t^i|^2$,
  Update the policy, using a policy gradient estimate $\hat{g}$,
    Which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t$.
    (Plug $\hat{g}$ into SGD or ADAM)
endfor

Choosing the Baseline: Value Functions

(for now assume episodic so you can set $\gamma = 1$)

- Recall Q-function / state-action-value function:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 \cdots | s_0 = s, a_0 = a\right]$$

- State-value function can serve as a great baseline

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 \cdots | s_0 = s\right] \\ &= \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a)] \end{aligned}$$

Policy Gradient Algorithms and Reducing Variance

Table of Contents

Likelihood Ratio / Score Function Policy Gradient

- Policy gradient:

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

- Fixes that improve simplest estimator
  - Temporal structure (shown in above equation)
  - Baseline (shown in above equation)
  - **Alternatives to using Monte Carlo returns G_t^i as estimate of expected discounted sum of returns for the policy parameterized by θ?**

Choosing the Target

- G_t^i is an estimation of the value function at s_t from a single roll out
- Unbiased but high variance
- Reduce variance by introducing bias using bootstrapping and function approximation
  - Just like we saw for TD vs MC, and value function approximation

Actor-critic Methods

- Estimate of V/Q is done by a **critic**
- **Actor-critic** methods maintain an explicit representation of policy and the value function, and update both
- A3C (Mnih et al. ICML 2016) is a very popular actor-critic method

Policy Gradient Formulas with Value Functions

- Recall:

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

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t;\theta)\left(Q(s_t,a_t;\boldsymbol{w}) - b(s_t)\right)\right]$$

- Letting the baseline be an estimate of the value V, we can represent the gradient in terms of the state-action advantage function

$$\nabla_\theta \mathbb{E}_\tau[R] \approx \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t;\theta)\hat{A}^\pi(s_t,a_t)\right]$$

- where the advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

Advanced Policy Gradients

Outline

Theory:

1. Problems with Policy Gradient Methods
2. Policy Performance Bounds
3. Monotonic Improvement Theory] next week

Algorithms:

1. Proximal Policy Optimization

The Problems with Policy Gradients

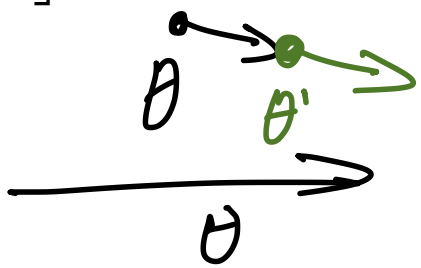
Policy Gradients Review

Policy gradient algorithms try to solve the optimization problem

$$\max_{\theta} J(\pi_\theta) \doteq \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

by taking stochastic gradient ascent on the policy parameters θ, using the *policy gradient*

$$g = \nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t) \right].$$



Limitations of policy gradients:

- Sample efficiency is poor
- Distance in parameter space $\neq$ distance in policy space!
  - What is policy space? For tabular case, set of matrices

$$\Pi = \left\{ \pi \ : \ \pi \in \mathbb{R}^{|S| \times |A|}, \ \sum_a \pi_{sa} = 1, \ \ \pi_{sa} \geq 0 \right\}$$

  - Policy gradients take steps in parameter space
  - Step size is hard to get right as a result

Sample Efficiency in Policy Gradients

- Sample efficiency for vanilla policy gradient methods is poor
- Discard each batch of data immediately after **just one gradient step**
- Why? PG is an **on-policy expectation**.
- Two main approaches to obtaining an unbiased estimate of the policy gradient
  - Collect sample trajectories from policy, then form sample estimate. (More stable)
  - Use trajectories from other policies (Less stable)
- Opportunity: use old data to take **multiple gradient steps** before using the resulting new policy to gather more data
- Challenge: even if this is possible to use old data to estimate multiple gradients, how many steps should be taken?

Choosing a Step Size for Policy Gradients

Policy gradient algorithms are stochastic gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k$$

with step $\Delta_k = \alpha_k \hat{g}_k$.

- If the step is too large, **performance collapse** is possible (Why?)

Choosing a Step Size for Policy Gradients

Policy gradient algorithms are stochastic gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k$$

with step $\Delta_k = \alpha_k \hat{g}_k$.

- If the step is too large, **performance collapse** is possible (Why?)
- If the step is too small, progress is unacceptably slow
- "Right" step size changes based on θ

Automatic learning rate adjustment like advantage normalization, or Adam-style optimizers, can help. But does this solve the problem?

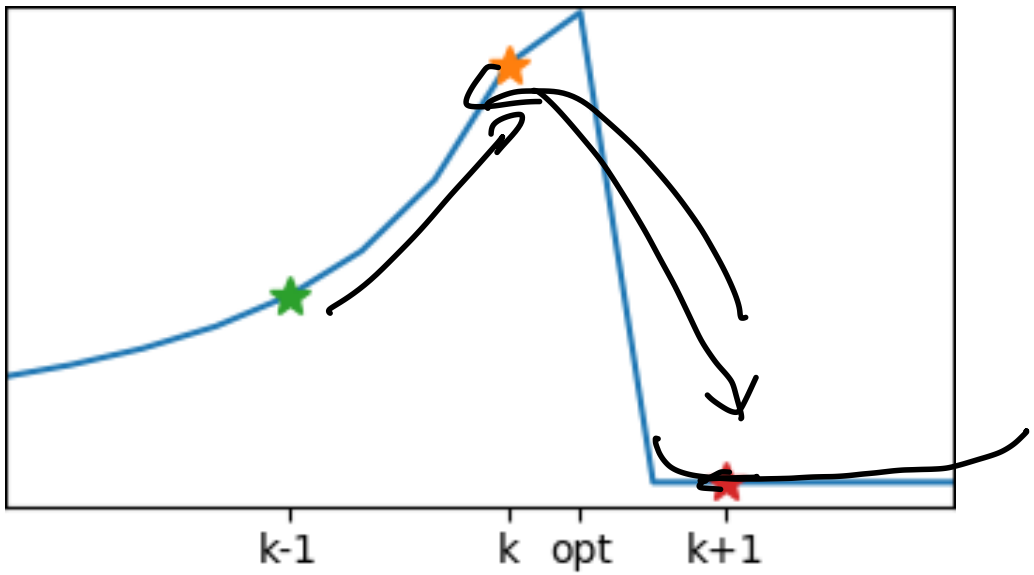


Figure: Policy parameters on x-axis and performance on y-axis. A bad step can lead to performance collapse, which may be hard to recover from.

The Problem is More Than Step Size

Consider a family of policies with parametrization:

$$\pi_\theta(a) = \begin{cases} \sigma(\theta) & a = 1 \\ 1 - \sigma(\theta) & a = 2 \end{cases}$$

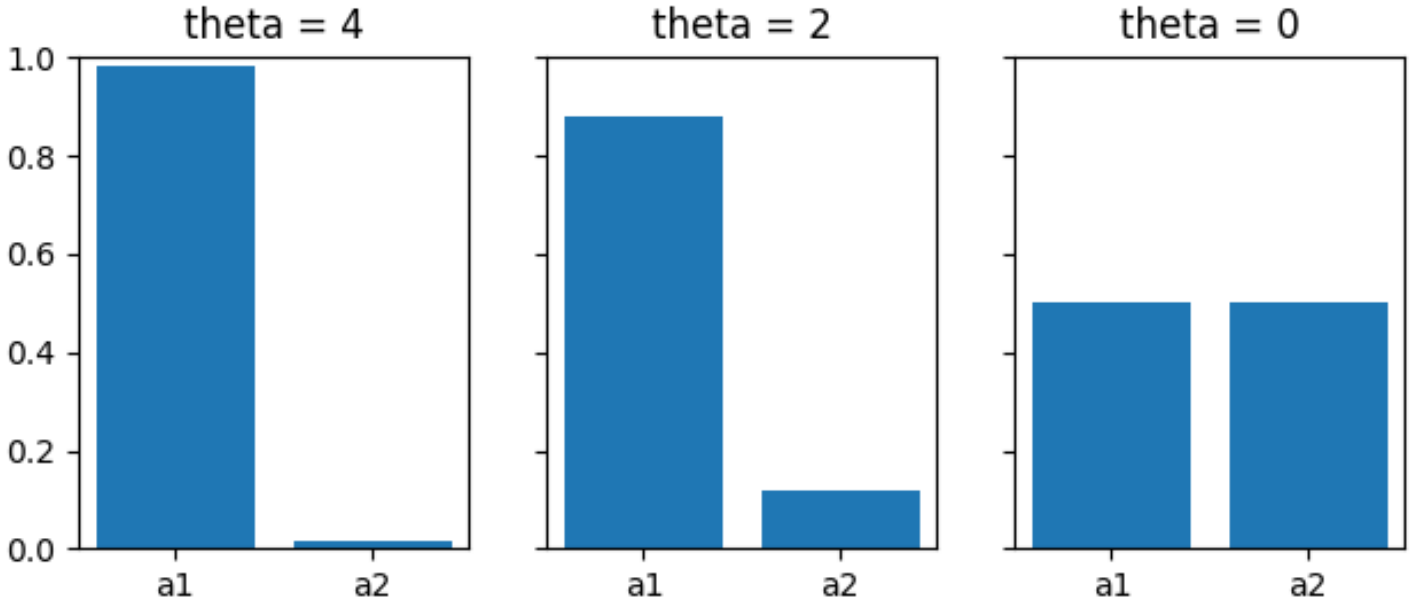


Figure: Small changes in the policy parameters can unexpectedly lead to **big** changes in the policy.

Big question: how do we come up with an update rule that doesn't ever change the policy more than we meant to?

Policy Performance Bounds

Relative Performance of Two Policies

In a policy optimization algorithm, we want an update step that

- uses rollouts collected from the most recent policy as efficiently as possible,
- and takes steps that respect **distance in policy space** as opposed to distance in parameter space.

To figure out the right update rule, we need to exploit relationships between the performance of two policies.

Performance difference lemma: In CS234 HW2 we ask you to prove that for any policies π, π'

$$J(\pi') - J(\pi) = \mathop{\mathrm{E}}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \right] \quad (1)$$

$$= \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s \sim d^{\pi'} \\ a \sim \pi'}} [A^{\pi}(s, a)] \quad (2)$$

where

$$d^{\pi}(s) = (1-\gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \pi)$$

What is it good for?

Can we use this for policy improvement, where π' represents the new policy and π represents the old one?

$$\max_{\pi'} J(\pi') = \max_{\pi'} J(\pi') - J(\pi)$$

$$= \max_{\pi'} \mathop{\mathrm{E}}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \right]$$

This is suggestive, but not useful yet.

Nice feature of this optimization problem: defines the performance of π' in terms of the advantages from π!

But, problematic feature: still requires trajectories sampled from π'...

goal: estimate $J(\pi')$ only from data from π

Looking at it from another angle...

In terms of the **discounted future state distribution** d^π, defined by

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi),$$

we can rewrite the relative policy performance identity:

$$J(\pi') - J(\pi) = \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t, a_t)\right]$$

time/traj notation

$$= \frac{1}{1-\gamma}\mathop{E}_{\substack{s\sim d^{\pi'}\\ a\sim\pi'}} A^\pi(s,a)$$

$$= \frac{1}{1-\gamma} E_{s\sim d^{\pi'}} \sum_a \pi'(a|s) A^\pi(s,a)$$

$$= \frac{1}{1-\gamma} E_{s\sim d^{\pi'}} \sum_a \frac{\pi'(a|s)}{\pi(a|s)} \pi(a|s) A^\pi(s,a)$$

$$= \frac{1}{1-\gamma} E_{s\sim d^{\pi'}} E_{a\sim\pi}\left(A^\pi(s,a)\frac{\pi'(a|s)}{\pi(a|s)}\right)$$

Note: Instance of Importance Sampling

In terms of the **discounted future state distribution** d^π, defined by

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi),$$

we can rewrite the relative policy performance identity:

$$\begin{aligned} J(\pi') - J(\pi) &= \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^\pi(s_t, a_t)\right] \\ &= \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi'}}[A^\pi(s,a)] \\ &= \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi}}\left[\frac{\pi'(a|s)}{\pi(a|s)}A^\pi(s,a)\right] \end{aligned}$$

Last step is an instance of **importance sampling** (more on this next time)

Problem: State Distribution

In terms of the **discounted future state distribution** d^{π}, defined by

$$d^{\pi}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}P(s_t = s|\pi),$$

we can rewrite the relative policy performance identity:

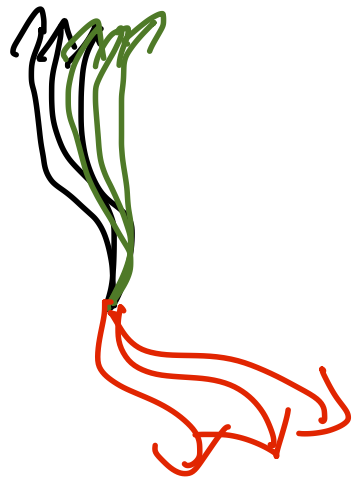
$$\begin{aligned} J(\pi') - J(\pi) &= \mathop{\mathrm{E}}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^{t}A^{\pi}(s_t, a_t)\right] \\ &= \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi'}}[A^{\pi}(s,a)] \\ &= \frac{1}{1-\gamma}\mathop{\mathrm{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi}}\left[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\right] \end{aligned}$$

...almost there! Only problem is $s \sim d^{\pi'}$.

A Useful Approximation

What if we just said $d^{\pi'} \approx d^{\pi}$ and didn't worry about it?

$$J(\pi') - J(\pi) \approx \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi} \\ a\sim\pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^{\pi}(s,a) \right]$$
$$\doteq \mathcal{L}_{\pi}(\pi')$$



Turns out: this approximation is pretty good when π' and π are close! But why, and how close do they have to be?

Relative policy performance bounds: [1]

constant

$$\left| J(\pi') - \left(J(\pi) + \mathcal{L}_{\pi}(\pi') \right) \right| \leq C \sqrt{\mathop{\mathrm{E}}_{s\sim d^{\pi}} \left[D_{KL}(\pi' || \pi)[s] \right]} \tag{3}$$

If policies are close in KL-divergence—the approximation is good!

[1] Achiam, Held, Tamar, Abbeel, 2017

What is KL-divergence?

For probability distributions P and Q over a discrete random variable,

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Properties:

- $D_{KL}(P||P) = 0$
- $D_{KL}(P||Q) \geq 0$
- $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ — Non-symmetric!

What is KL-divergence between policies?

$$D_{KL}(\pi'||\pi)[s] = \sum_{a \in \mathcal{A}} \pi'(a|s) \log \frac{\pi'(a|s)}{\pi(a|s)}$$

A Useful Approximation

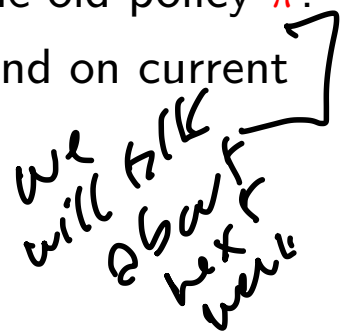
What did we gain from making that approximation?

$$J(\pi') - J(\pi) \approx \mathcal{L}_\pi(\pi')$$

$$\mathcal{L}_\pi(\pi') = \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^\pi \\ a\sim\pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s,a) \right]$$

$$= \mathop{\mathrm{E}}_{\tau\sim\pi} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)} A^\pi(s_t,a_t) \right]$$

- This is something we can optimize using trajectories sampled from the old policy π!
- Similar to using importance sampling, but because weights only depend on current timestep (and not preceding history), they don't vanish or explode.



Recommended Reading

- "Approximately Optimal Approximate Reinforcement Learning," Kakade and Langford, 2002 [2]
- "Trust Region Policy Optimization," Schulman et al. 2015 [3]
- "Constrained Policy Optimization," Achiam et al. 2017 [4]

[2] https://people.eecs.berkeley.edu/ pabbeel/cs287-fa09/readings/KakadeLangford-icml2002.pdf
[3] https://arxiv.org/pdf/1502.05477.pdf
[4] https://arxiv.org/pdf/1705.10528.pdf

Algorithms

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a family of methods that approximately penalize policies from changing too much between steps. Two variants:

- Adaptive KL Penalty
  - Policy update solves unconstrained optimization problem

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k) \quad (4)$$

$$\bar{D}_{KL}(\theta||\theta_k) = E_{s\sim d^{\pi_k}} D_{KL}(\theta_k(\cdot|s), \pi_\theta(\cdot|s)) \quad (5)$$

  - Penalty coefficient β_k changes between iterations to approximately enforce KL-divergence constraint

Proximal Policy Optimization with Adaptive KL Penalty

Algorithm PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \leq \delta / 1.5$ **then**
    $\beta_{k+1} = \beta_k / 2$
  end if
end for

- Initial KL penalty not that important—it adapts quickly
- Some iterations may violate KL constraint, but most don't

PPO with Adaptive KL Penalty: **Multiple Gradient Steps**

Algorithm PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1} || \theta_k) \leq \delta / 1.5$ **then**
    $\beta_{k+1} = \beta_k / 2$
  end if
end for

- Initial KL penalty not that important—it adapts quickly
- Some iterations may violate KL constraint, but most don't

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a family of methods that approximately enforce KL constraint **without computing natural gradients**. Two variants:

- Adaptive KL Penalty
  - Policy update solves unconstrained optimization problem

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  - Penalty coefficient β_k changes between iterations to approximately enforce KL-divergence constraint
- Clipped Objective
  - New objective function: let $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_k}_t)\right]\right]$$

    where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)
  - Policy update is $\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$

L6N3 Check Your Understanding: Proximal Policy Optimization

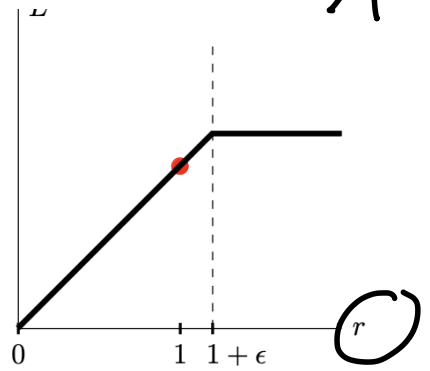
- Clipped Objective function: let $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then

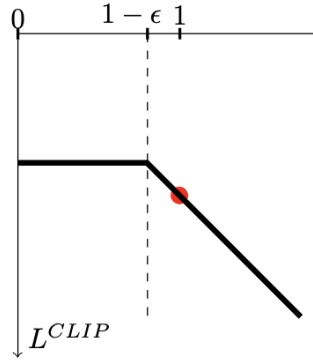
$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_k}_t)\right]\right]$$

- where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)
- Policy update is $\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$.

Consider the figure[5]. Select all that are true. $\epsilon \in (0, 1)$.

1. The left graph shows the L^{CLIP} objective when the advantage function $A > 0$ and the right graph shows when $A < 0$
2. The right graph shows the L^{CLIP} objective when the advantage function $A > 0$ and the left graph shows when $A < 0$
3. It depends on the value of ϵ
4. Not sure





[5] Schulman, Wolski, Dhariwal, Radford, Klimov, 2017

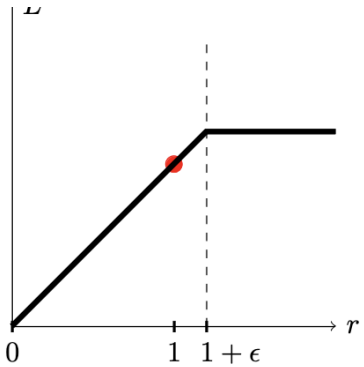
L6N3: Check Your Understanding L6N2 Proximal Policy Optimization Solutions

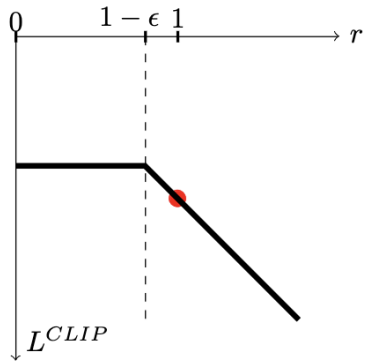
- Clipped Objective function: let $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_k}_t)\right]\right]$$

- where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)
- Policy update is $\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$.

Consider the figure[6]. Select all that are true. $\epsilon \in (0, 1)$.
The left graph shows the L^{CLIP} objective when the advantage function $A > 0$ and the right graph shows when $A < 0$





[6] Schulman, Wolski, Dhariwal, Radford, Klimov, 2017

Proximal Policy Optimization with Clipped Objective

But *how* does clipping keep policy close? By making objective as pessimistic as possible about performance far away from θ_k:

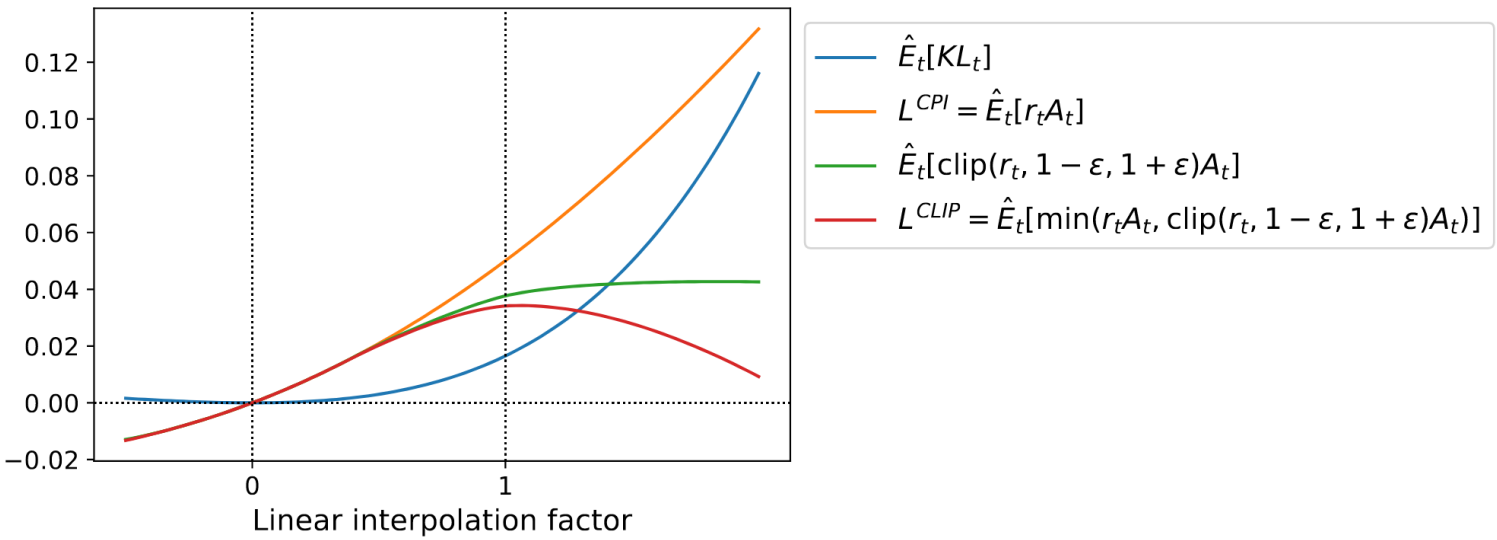


Figure: Various objectives as a function of interpolation factor α between θ_{k+1} and θ_k after one update of PPO-Clip [7]

[7] Schulman, Wolski, Dhariwal, Radford, Klimov, 2017

Proximal Policy Optimization with Clipped Objective

Algorithm PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

next time will discuss

end for

- Clipping prevents policy from having incentive to go far away from θ_{k+1}
- Clipping seems to work at least as well as PPO with KL penalty, but is simpler to implement

Empirical Performance of PPO

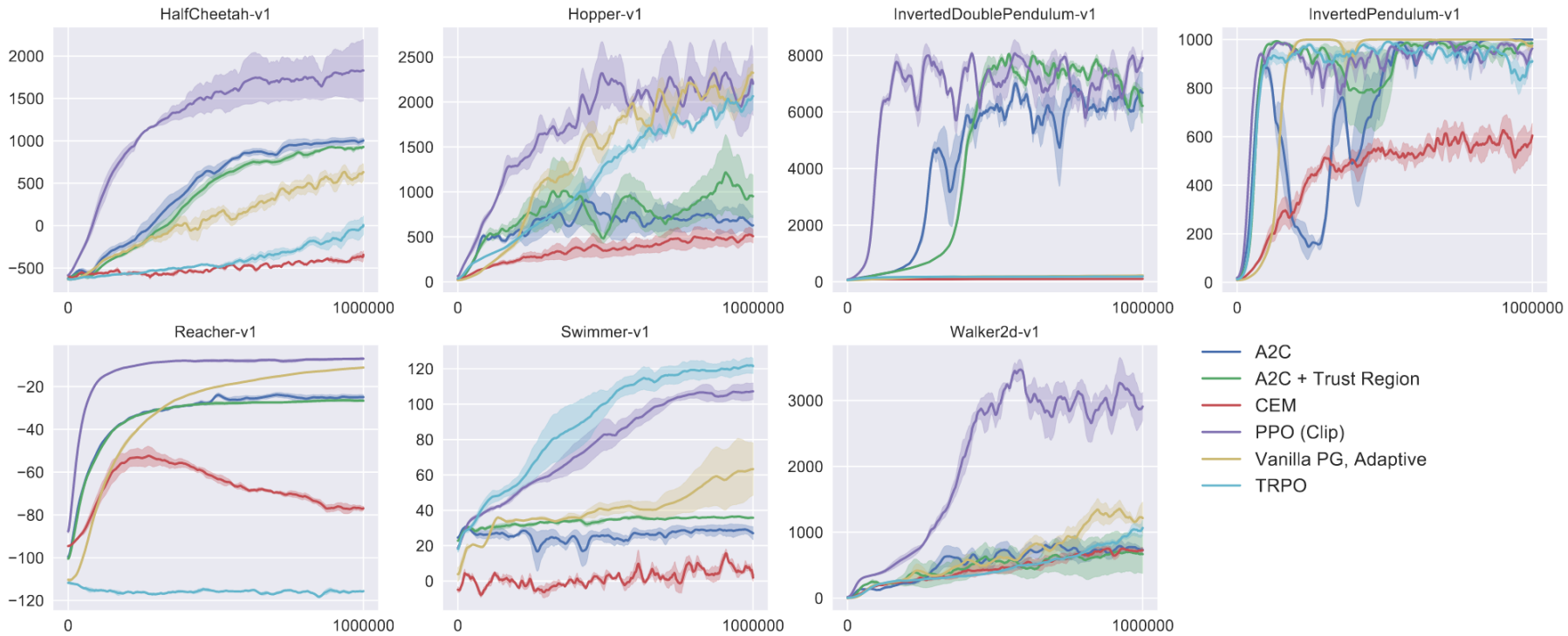


Figure: Performance comparison between PPO with clipped objective and various other deep RL methods on a slate of MuJoCo tasks. [8]

- Wildly popular, and key component of ChatGPT

[8] Schulman, Wolski, Dhariwal, Radford, Klimov, 2017

Recommended Reading

PPO

- "Proximal Policy Optimization Algorithms," Schulman et al. 2017 [9]
- OpenAI blog post on PPO, 2017 [10]

[9] https://arxiv.org/pdf/1707.06347.pdf
[10] https://blog.openai.com/openai-baselines-ppo/

PPO: Algorithm and Code Implementation Details

- Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation Matters in Deep RL: A Case Study on PPO and TRPO. ICLR 2020
  `https://openreview.net/forum?id=r1etN1rtPB`
- Reward scaling, learning rate annealing, etc. can make a significant difference

Lecture 7: Policy Gradients and Imitation learning

Emma Brunskill

CS234 Reinforcement Learning.

- Monotonic improvement slides and several PPO slides from Joshua Achiam

Refresh Your Understanding L7N1

Which of the following are true about REINFORCE? In the following options, PG stands for policy gradient.

a. Adding a baseline term can help to reduce the variance of the PG updates true
b. It will converge to a global optima false
c. It can be initialized with a sub-optimal, deterministic policy and still converge to a local optima, given the appropriate step sizes false
d. If we take one step of PG, it is possible that the resulting policy gets worse (in terms of achieved returns) than our initial policy true

Refresh Your Understanding L7N1

Which of the following are true about REINFORCE? In the following options, PG stands for policy gradient.

(a) Adding a baseline term can help to reduce the variance of the PG updates

(b) It will converge to a global optima

(c) It can be initialized with a sub-optimal, deterministic policy and still converge to a local optima, given the appropriate step sizes

(d) If we take one step of PG, it is possible that the resulting policy gets worse (in terms of achieved returns) than our initial policy

Class Structure

- Last time: Advanced Policy Search
- This time: Policy search continued and Imitation Learning

Today

- Proximal policy optimization (PPO) (will implement in homework)
  - Generalized Advantage Estimation (GAE)
  - Theory: Monotonic Improvement Theory
- Imitation Learning
  - Behavior cloning
  - DAGGER
  - Max entropy inverse RL

Recall Problems with Policy Gradients

Policy gradient algorithms try to solve the optimization problem

$$\max_{\theta} J(\pi_\theta) \doteq \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

by taking stochastic gradient ascent on the policy parameters θ, using the *policy gradient*

$$g = \nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t) \right].$$

Limitations of policy gradients:

- Sample efficiency is poor
- Distance in parameter space $\neq$ distance in policy space!
  - What is policy space? For tabular case, set of matrices

$$\Pi = \left\{ \pi \ : \ \pi \in \mathbb{R}^{|S| \times |A|}, \ \sum_a \pi_{sa} = 1, \ \ \pi_{sa} \geq 0 \right\}$$

  - Policy gradients take steps in parameter space
  - Step size is hard to get right as a result

Recall Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a family of methods that approximately enforce KL constraint Two variants:

- Adaptive KL Penalty
  - Policy update solves unconstrained optimization problem

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  - Penalty coefficient β_k changes between iterations to approximately enforce KL-divergence constraint
- Clipped Objective
  - New objective function: let $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_k}_t)\right]\right]$$

    where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)
  - Policy update is $\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$

Recall Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a family of methods that approximately enforce KL constraint **without computing natural gradients**. Two variants:

- Adaptive KL Penalty
  - Policy update solves unconstrained optimization problem

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

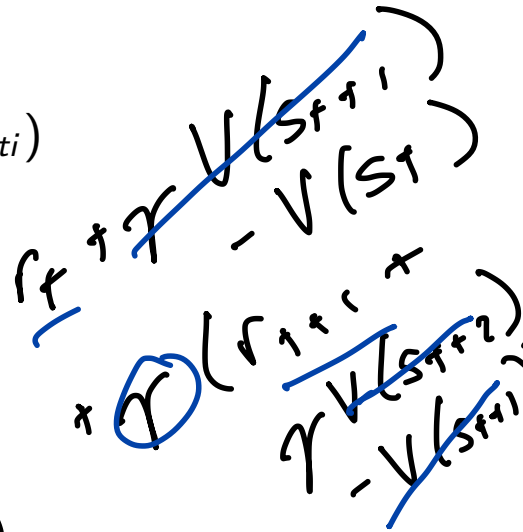
  - Penalty coefficient β_k changes between iterations to approximately enforce KL-divergence constraint
- Clipped Objective
  - New objective function: let $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_k}_t)\right]\right]$$

    where ϵ is a hyperparameter (maybe $\epsilon = 0.2$)
  - Policy update is $\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$
- **How do we estimate the advantage function inside the policy update?**

Recall N-step estimators

$$\nabla_\theta V(\theta) \approx (1/m) \sum_{i=1}^{m} \sum_{t=0}^{T-1} A_{ti} \nabla_\theta \log \pi_\theta(a_{ti}|s_{ti})$$



- Recall the N-step advantage estimators

note typo

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma V(s_{t+2}) - V(s_t)$$
$$\hat{A}_t^{(\text{inf})} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \cdots - V(s_t)$$

- Define $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$. Then

$$\hat{A}_t^{(1)} = \delta_t^V \qquad = r_t + \gamma V(s_{t+1}) - V(s_t)$$
$$\hat{A}_t^{(2)} = \delta_t^V + \gamma \delta_{t+1}^V \qquad = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$$
$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V \qquad = \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t)$$

- Note the above is an instance of a **telescoping sum**

Generalized Advantage Estimator (GAE)

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \tag{1}$$

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned} \hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots) \\ &= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \ldots) \end{aligned}$$

$$= (1-\lambda)\left(\delta_t^V(1 + \lambda + \lambda^2 + \lambda^3 \ldots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \lambda^3 + \ldots) + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \ldots\right.$$

$$= (1-\lambda)\left(\frac{\delta_t^V}{1-\lambda} + \right.$$

Generalized Advantage Estimator (GAE)

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \tag{2}$$

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots) \\
&= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \ldots) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \ldots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \ldots) \\
&\quad + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \ldots) + \ldots) \\
&= (1-\gamma)(\delta_t^V \frac{1}{1-\lambda} + \gamma\lambda\delta_{t+1}^V \frac{1}{1-\lambda} + \gamma^2\lambda^2\delta_{t+2}^V \frac{1}{1-\lambda} + \ldots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}$$

geometric

- Introduced in "High-Dimensional Continuous Control Using Generalized Advantage Estimation" ICLR 2016 by Schulman et al.
- Our derivation follows the derivation presented in the paper

Check Your Understanding L7N2: Generalized Advantage Estimator (GAE)

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \tag{3}$$

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots) \\
&= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \ldots) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \ldots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \ldots) \\
&\quad + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \ldots) + \ldots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}$$

$\lambda = 0$ look at 1st line

γ, λ

- What are the properties of GAE(γ,0) and GAE(γ,1)? (select all)
- (a) GAE(γ,1) is the advantage function using a TD(0) return
- (b) GAE(γ,0) is the advantage function using a TD(0) return
- (c) The variance of GAE(γ,0) is likely to be larger than GAE(γ,1)
- (d) The bias of GAE(γ,0) is likely to be larger than GAE(γ,1)
- (e) Not sure

Check Your Understanding L7N2: GAE Solution

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \tag{4}$$

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots) \\
&= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \ldots) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \ldots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \ldots) \\
&\quad + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \ldots) + \ldots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}$$

- What are the properties of GAE(γ,0) and GAE(γ,1)? (select all)
- (a) GAE(γ,1) is the advantage function using a TD(0) return
- (b) GAE(γ,0) is the advantage function using a TD(0) return
- (c) The variance of GAE(γ,0) is likely to be larger than GAE(γ,1)
- (d) The bias of GAE(γ,0) is likely to be larger than GAE(γ,1)
- (e) Not sure

b and d are true

Generalized Advantage Estimator (GAE) Balance

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \tag{5}$$

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots) \\
&= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \ldots) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \ldots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \ldots) \\
&\quad + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \ldots) + \ldots) \\
&= (1-\gamma)(\delta_t^V \frac{1}{1-\lambda} + \gamma\lambda\delta_{t+1}^V \frac{1}{1-\lambda} + \gamma^2\lambda^2\delta_{t+2}^V \frac{1}{1-\lambda} + \ldots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}$$

- Introduced in "High-Dimensional Continuous Control Using Generalized Advantage Estimation" ICLR 2016 by Schulman et al.
- In general will prefer $\lambda \in (0, 1)$ to balance bias and variance

Generalized Advantage Estimator (GAE) in PPO

- GAE is an exponentially-weighted average of k-step estimators

$$\begin{aligned}
\hat{A}_t^{(k)} &= \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t) \\
\delta_t^V &= r_t + \gamma V(s_{t+1}) - V(s_t) \\
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \ldots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned}$$

- PPO uses a truncated version of a GAE

$$\hat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}^V$$

- Benefits: Only have to run policy in environment for T timesteps before updating, improved estimate of gradient

Monotonic Improvement Theory

Return to Approximation Bound for Difference Between Two Policies

In last lecture used $d^{\pi'}$ as approximation of d^{π} (Why?)

$$J(\pi') - J(\pi) \approx \frac{1}{1-\gamma} \mathop{\mathrm{E}}_{\substack{s\sim d^{\pi} \\ a\sim\pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^{\pi}(s,a) \right]$$
$$\doteq \mathcal{L}_{\pi}(\pi')$$

This approximation is good when π' and π are close in KL-divergence

Relative policy performance bounds: [1]

$$\left| J(\pi') - \left(J(\pi) + \mathcal{L}_{\pi}(\pi') \right) \right| \leq C \sqrt{\mathop{\mathrm{E}}_{s\sim d^{\pi}} \left[D_{KL}(\pi' || \pi)[s] \right]} \tag{6}$$

[1] Achiam, Held, Tamar, Abbeel, 2017

$J(\pi) = V(\pi)$

From the bound on the previous slide, we get

$$J(\pi') - J(\pi) \geq \underbrace{\mathcal{L}_\pi(\pi')} - C\sqrt{\mathop{\mathrm{E}}_{s\sim d^\pi}[D_{KL}(\pi'||\pi)[s]]}.$$

- If we maximize the right hand side (RHS) with respect to π', we are **guaranteed to improve over π**.
  - This is a *majorize-maximize* algorithm w.r.t. the true objective, the LHS.
- And $\mathcal{L}_\pi(\pi')$ & the KL-divergence term *can both be estimated with samples from* π!

Monotonic Improvement Theory

Proof of improvement guarantee: Suppose π_{k+1} and π_k are related by

$$\pi_{k+1} = \arg\max_{\pi'} \mathcal{L}_{\pi_k}(\pi') - C\sqrt{\mathop{\mathrm{E}}_{s\sim d^{\pi_k}}[D_{KL}(\pi'||\pi_k)[s]]}.$$

(1) (2)

π_k feasible

$$\mathcal{L}_{\pi_k}(\pi_k) = \frac{1}{1-\gamma} \mathop{E}_{\substack{s\sim d^{\pi_k} \\ a\sim\pi_k}} \left[\frac{\pi_k(a|s)}{\pi_k(a|s)} A^{\pi_k}(s,a)\right]$$

$$\left[Q^{\pi_k}(s,a) - V^{\pi_k}(s)\right]$$

$$\sum_a \pi_k(a|s) Q^{\pi_k}(s,a) = V^{\pi_k}(s)$$

(2) $= 0$

$D_{KL}(\pi_k/\pi_k) = 0$

(1) - (2) ≥ 0 because argmax is at least as good as π_k

$J(\pi_{k+1}) - J(\pi_k) \geq$ (1) - (2) ≥ 0

Proof of improvement guarantee: Suppose π_{k+1} and π_k are related by

$$\pi_{k+1} = \arg\max_{\pi'} \mathcal{L}_{\pi_k}(\pi') - C\sqrt{\mathop{\mathrm{E}}_{s\sim d^{\pi_k}}[D_{KL}(\pi'||\pi_k)[s]]}.$$

- π_k is a feasible point, and the objective at π_k is equal to 0.
  - $\mathcal{L}_{\pi_k}(\pi_k) \propto \mathop{\mathrm{E}}_{s,a\sim d^{\pi_k},\pi_k}[A^{\pi_k}(s,a)] = 0$
  - $D_{KL}(\pi_k||\pi_k)[s] = 0$
- $\Longrightarrow$ optimal value ≥ 0
- $\Longrightarrow$ by the performance bound, $J(\pi_{k+1}) - J(\pi_k) \geq 0$

This proof works even if we restrict the domain of optimization to an arbitrary class of parametrized policies Π_θ, as long as $\pi_k \in \Pi_\theta$.

Approximate Monotonic Improvement

$$\pi_{k+1} = \arg\max_{\pi'} \mathcal{L}_{\pi_k}(\pi') - C\sqrt{\mathop{\mathrm{E}}_{s\sim d^{\pi_k}}[D_{KL}(\pi'||\pi_k)[s]]}. \tag{7}$$

Problem:

- C provided by theory is quite high when γ is near 1
- $\Longrightarrow$ steps from Equation (7) are too small.

Potential Solution:

- Tune the KL penalty ($\Longrightarrow$ PPO)
- Use KL constraint (called **trust region**).

Today

- Proximal policy optimization (PPO) (will implement in homework)
  - Generalized Advantage Estimation (GAE)
  - Theory: Monotonic Improvement Theory
- **Imitation Learning**[2]
  - Behavior cloning
  - DAGGER
  - Max entropy inverse RL

[2] With slides from Katerina Fragkiadaki and slides from Pieter Abbeel

Learning from Past Decisions and Outcomes

In some settings there exist very good decision policies and we would like to automate them

- One idea: humans provide reward signal when RL algorithm makes decisions
- Good: simple, cheap form of supervision
- Bad: High sample complexity

Alternative: imitation learning

Reward Shaping

Rewards that are **dense in time** closely guide the agent. How can we supply these rewards?

- **Manually design them**: often brittle
- **Implicitly specify them through demonstrations**

Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010

Examples

- Simulated highway driving [Abbeel and Ng, ICML 2004; Syed and Schapire, NIPS 2007; Majumdar et al., RSS 2017]
- Parking lot navigation [Abbeel, Dolgov, Ng, and Thrun, IROS 2008]

Learning from Demonstrations

$s, a, s', a', \ldots)$

- Expert provides a set of **demonstration trajectories**: sequences of states and actions
- Imitation learning is useful when it is easier for the expert to demonstrate the desired behavior rather than:
  - Specifying a reward that would generate such behavior,
  - Specifying the desired policy directly

Problem Setup

- Input:
  - State space, action space
  - Transition model $P(s' \mid s, a)$
  - No reward function R
  - Set of one or more teacher's demonstrations $(s_0, a_0, s_1, s_0, \ldots)$ (actions drawn from teacher's policy π^*)
- Behavioral Cloning:
  - Can we directly learn the teacher's policy using supervised learning?
- Inverse RL:
  - Can we recover R?
- Apprenticeship learning via Inverse RL:
  - Can we use R to generate a good policy?

Behavioral Cloning

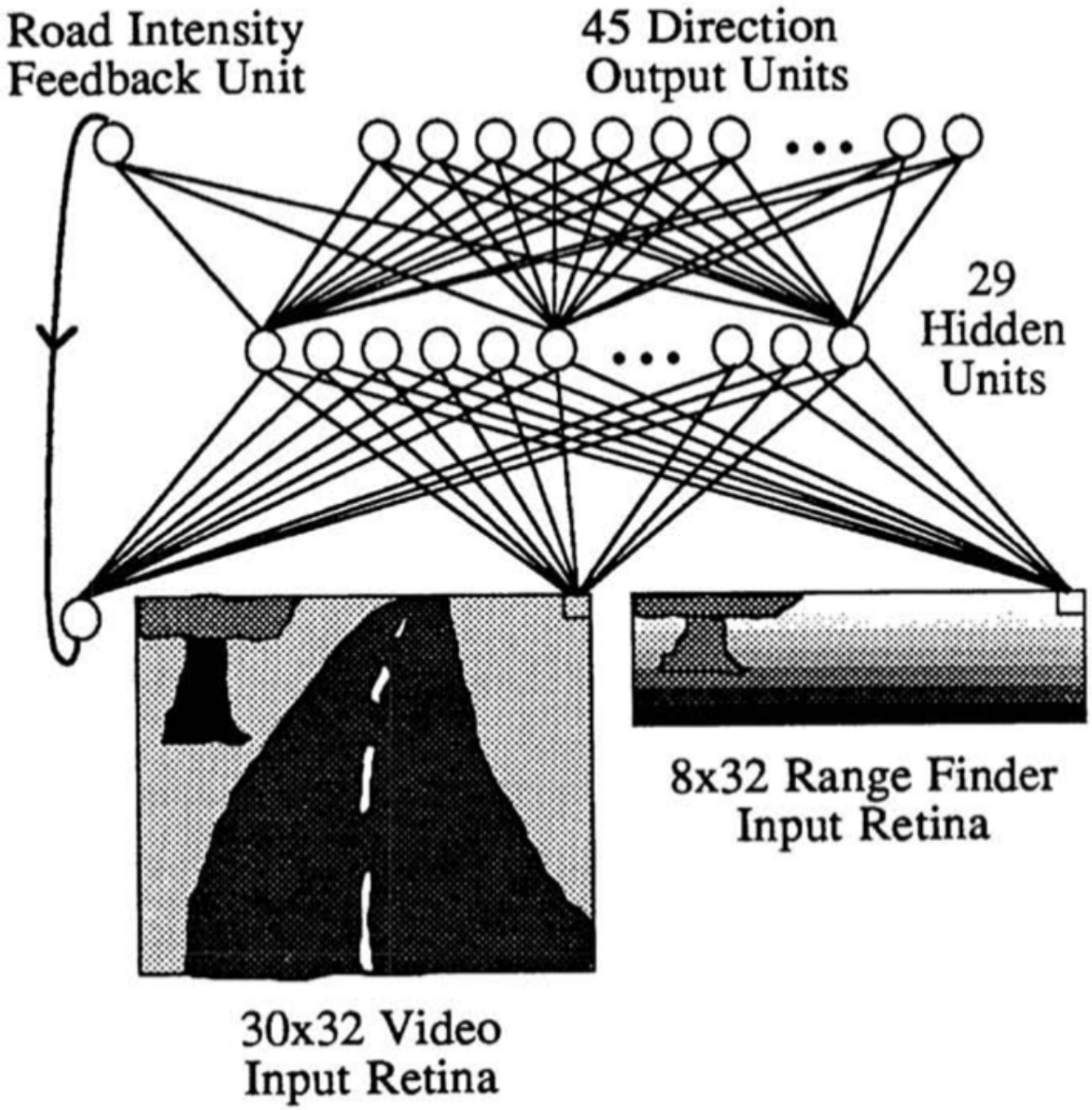
Behavioral Cloning

$s_0, a_0, s_1 \rightarrow a_1$

$s_0, a_0, s_1, a_1, s_2 \ldots \rightarrow a$

- Reduce problem to a standard supervised machine learning problem:
  - Fix a policy class (e.g. neural network, decision tree, etc.)
  - Estimate a policy from training examples $(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots$
- Two early notable success stories:
  - Pomerleau, NIPS 1989: ALVINN
  - Summut et al., ICML 1992: Learning to fly in flight simulator

ALVINN


Road Intensity
Feedback Unit
45 Direction
Output Units
29
Hidden
Units
8x32 Range Finder
Input Retina
30x32 Video
Input Retina

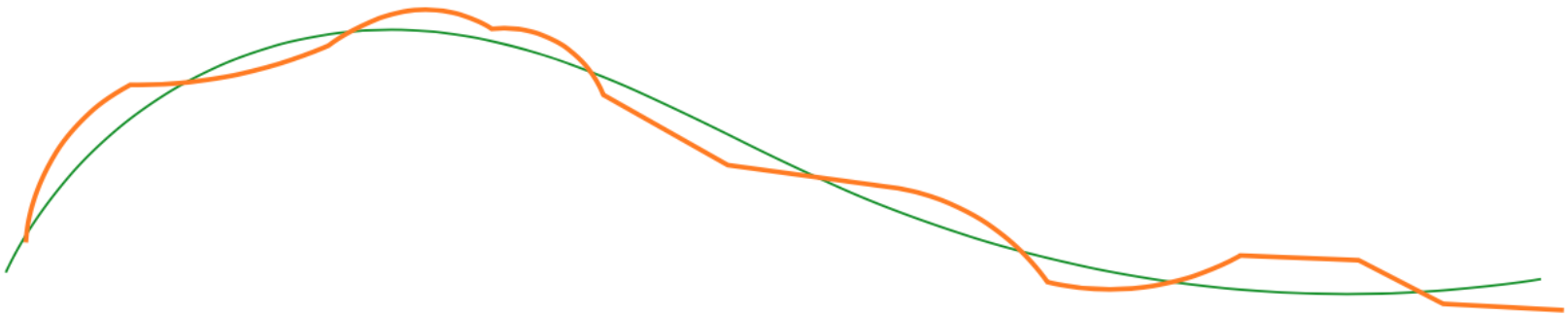
Behavioral cloning

- Often behavior cloning in practice can work very well, especially if use BCRNN
- See What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. Mandlekar et al. CORL 2021
- Extensively used in practice

DAGGER

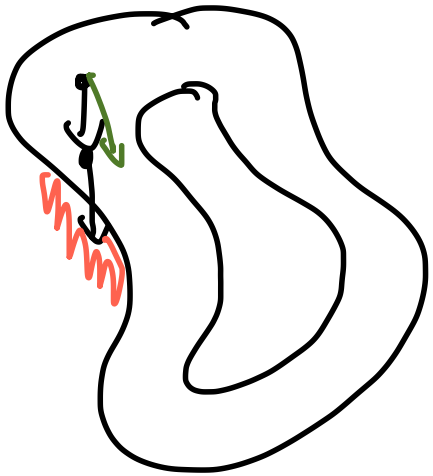
Potential Problem with Behavior Cloning: Compounding Errors

Supervised learning assumes iid. (s, a) pairs and ignores temporal structure
Independent in time errors:



Error at time t with probability $\leq \epsilon$
$\mathbb{E}[\text{Total errors}] \leq \epsilon T$ T decisions

Problem: Compounding Errors



Data distribution mismatch!

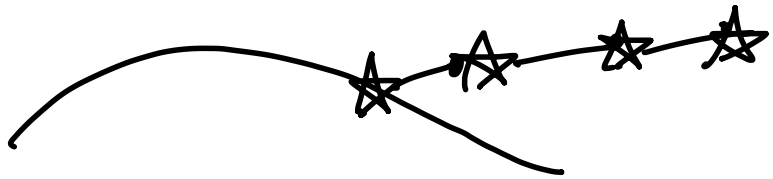
In supervised learning, $(x, y) \sim D$ during train and test. In MDPs:

- Train: $s_t \sim D_{\pi^*}$
- Test: $s_t \sim D_{\pi_\theta}$

Next

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011

Problem: Compounding Errors

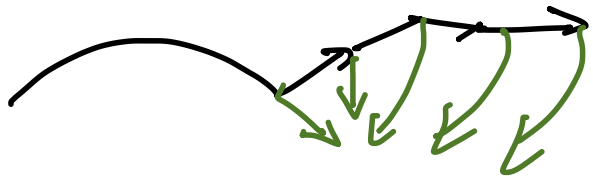


- Error at time t with probability ϵ
- Approximate intuition: $\mathbb{E}[\text{Total errors}] \leq \epsilon(T + (T-1) + (T-2)\ldots + 1) \propto \epsilon T^2$
- Real result requires more formality. See Theorem 2.1 in `http://www.cs.cmu.edu/~sross1/publications/Ross-AIStats10-paper.pdf` with proof in supplement: `http://www.cs.cmu.edu/~sross1/publications/Ross-AIStats10-sup.pdf`

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011

DAGGER: Dataset Aggregation

Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.



- Idea: Get more labels of the expert action along the path taken by the policy computed by behavior cloning
- Obtains a stationary deterministic policy with good performance under its induced state distribution
- Key limitation? human has to supervise constantly

Reward Learning

Feature Based Reward Function

- Given state space, action space, transition model $P(s' \mid s, a)$
- No reward function R
- Set of one or more expert's demonstrations $(s_0, a_0, s_1, s_0, \ldots)$ (actions drawn from teacher's policy π^*)
- Goal: infer the reward function R
- Assume that the ~~teacher~~'s policy is optimal. What can be inferred about R?

expert's

Check Your Understanding L7N3: Feature Based Reward Function

- Given state space, action space, transition model $P(s' \mid s, a)$
- No reward function R
- Set of one or more teacher's demonstrations $(s_0, a_0, s_1, s_0, \ldots)$ (actions drawn from teacher's policy π^*)
- Goal: infer the reward function R
- Assume that the teacher's policy is optimal.

teacher = expert

1. There is a single unique R that makes teacher's policy optimal
2. There are many possible R that makes teacher's policy optimal
3. It depends on the MDP
4. Not sure

Check Your Understanding L7N3: Feature Based Reward Function

- Given state space, action space, transition model $P(s' \mid s, a)$
- No reward function R
- Set of one or more teacher's demonstrations $(s_0, a_0, s_1, s_0, \ldots)$ (actions drawn from teacher's policy π^*)
- Goal: infer the reward function R
- Assume that the teacher's policy is optimal.

1. There is a single unique R that makes teacher's policy optimal
2. There are many possible R that makes teacher's policy optimal
3. It depends on the MDP
4. Not sure

Answer: There are an infinite set of R .

Linear Feature Reward Inverse RL

- Recall linear value function approximation
- Similarly, here consider when reward is linear over features
  - $R(s) = \boldsymbol{w}^T x(s)$ where $\boldsymbol{w} \in \mathbb{R}^n, x : S \rightarrow \mathbb{R}^n$ features on $x(s)$
- Goal: identify the weight vector $\boldsymbol{w}$ given a set of demonstrations
- The resulting value function for a policy π can be expressed as

$$V^{\pi}(s_0) = \mathbb{E}_{s \sim \pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0]$$

$$= E_{s \sim \pi} \sum_{t=0}^{\infty} \gamma^t w^T x(s_t) \, | s_0$$

$$= w^T E_{s \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t x(s_t) \, | s_0 \right]$$

$$= w^T \mu(\pi)$$ ← state distrib under π discounted

Linear Feature Reward Inverse RL

- Recall linear value function approximation
- Similarly, here consider when reward is linear over features
  - $R(s) = \boldsymbol{w}^T x(s)$ where $w \in \mathbb{R}^n, x : S \rightarrow \mathbb{R}^n$
- Goal: identify the weight vector $\boldsymbol{w}$ given a set of demonstrations
- The resulting value function for a policy π can be expressed as

$$V^\pi(s_0) = \mathbb{E}_{s\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0] \quad = \mathbb{E}_{s\sim\pi}[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{w}^T x(s_t) \mid s_0]$$

$$= w^T \mathbb{E}_{s\sim\pi}[\sum_{t=0}^{\infty} \gamma^t x(s_t) \mid s_0]$$

$$= \boldsymbol{w}^T \mu(\pi)$$

- where $\mu(\pi)(s)$ is defined as the discounted weighted frequency of state features under policy π, starting in state s_0.

Relating Frequencies to Optimality

- Assume $R(s) = \boldsymbol{w}^T x(s)$ where $w \in \mathbb{R}^n, x : S \rightarrow \mathbb{R}^n$
- Goal: identify the weight vector $\boldsymbol{w}$ given a set of demonstrations
- $V^\pi = \mathbb{E}_{s\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi] = \boldsymbol{w}^T \mu(\pi)$ where
  $\mu(\pi)(s) =$ discounted weighted frequency of state s under policy π.

$V^* \geq V^\pi$

$w^T \mu(\pi^*) \geq w^T \mu(\pi) \quad \forall \pi$

experts / observed

Relating Frequencies to Optimality

- Recall linear value function approximation
- Similarly, here consider when reward is linear over features
  - $R(s) = \boldsymbol{w}^T x(s)$ where $w \in \mathbb{R}^n, x : S \rightarrow \mathbb{R}^n$
- Goal: identify the weight vector $\boldsymbol{w}$ given a set of demonstrations
- The resulting value function for a policy π can be expressed as

$$V^{\pi} = \boldsymbol{w}^T \mu(\pi)$$

- $\mu(\pi)(s) =$ discounted weighted frequency of state s under policy π.

$$\mathbb{E}_{s \sim \pi^*}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi^*] = V^* \geq V^{\pi} = \mathbb{E}_{s \sim \pi}[\sum_{t=0}^{\infty} \gamma^t R^*(s_t) \mid \pi] \quad \forall \pi$$

- Therefore if the expert's demonstrations are from the optimal policy, to identify $\boldsymbol{w}$ it is sufficient to find w^* such that

$$w^{*T} \mu(\pi^*) \geq w^{*T} \mu(\pi), \forall \pi \neq \pi^*$$

Feature Matching

- Want to find a reward function such that the expert policy outperforms other policies.
- For a policy π to be guaranteed to perform as well as the expert policy π^*, sufficient if its discounted summed feature expectations match the expert's policy [Abbeel & Ng, 2004].
- More precisely, if

$$\|\mu(\pi) - \mu(\pi^*)\|_1 \leq \epsilon$$

then for all w with $\|w\|_\infty \leq 1$ (uses Holder's inequality):

$$|w^T \mu(\pi) - w^T \mu(\pi^*)| \leq \epsilon$$

Ambiguity

- There is an infinite number of reward functions with the same optimal policy.
- There are infinitely many stochastic policies that can match feature counts
- Which one should be chosen?

Learning from Demonstration / Imitation Learning Pointers

- Many different approaches
- Two of the key papers are:
  - Maximumum Entropy Inverse Reinforcement Learning (Ziebart et al. AAAI 2008)
  - Generative adversarial imitation learning (Ho and Ermon, NeurIPS 2016)