Policy Gradient Theorem, REINFORCE Algorithm

Reading

- Reinforcement Learning
  - http://www.incompleteideas.net/book/the-book-2nd.html
  - Chapters 13.1-13.3
- David Silver lecture on Policy Gradients
  - https://www.youtube.com/watch?v=KHZVXao4qXs&t=3s

Overview

- In Q-learning, we select actions based on their q values
- With policy gradient methods, the controller is just a function that has no notion of q values
  - Of course, during training, it will be trained to select actions maximize q values
- Policy gradient methods are more flexible than standard Q-learning for a number of reasons
  - Can handle partially observable MDPs
    - No need to estimate q values or even fully observe states
  - Can handle continuous control systems
- At the same time, training with policy gradient methods is very unstable

Policy Gradients vs Q-learning

- In Q-learning, we have one function (approximation)
  - We have an estimate $q(s, a)$ for each s, a pair
  - Those estimates define a deterministic policy
- In policy gradient methods, we have one function to estimate action values and a separate function for the policy
  - We update the two separately (using gradients)
  - Even if the Q approximations are (temporarily) wrong, the policy may not be affected much since it's slowly updated according to its learning rate

Finite-MDP Setup

- The finite MDP setup is the same as before
  - An MDP is the usual 5-tuple (S, A, P, R, η)
- The main difference is that now the policy does not depend on the q-values:
  - recall that $\pi(a|s; \boldsymbol{\theta})$ is the probability that action a is taken from state s
  - the parameters $\boldsymbol{\theta}$ are determined during training
  - looks the same as before except there is no explicit computation of q-values

Policy Example

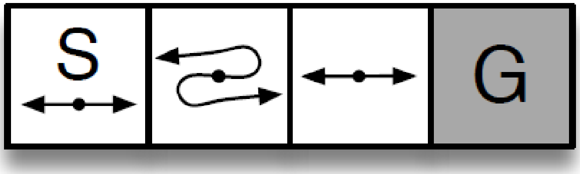
- Can encode a probabilistic policy, with parameters $\boldsymbol{\theta}$, using softmax
  - How?
    - Let the current state be s
    - To compute the probability of action a, we need to first encode the state and the action somehow
    - Let $\boldsymbol{x}(s, a)$ be a one-hot encoding of all states and actions
    - Then the probability of taking action a from state s is:

$$\pi(a|s; \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}^T \boldsymbol{x}(s,a)}}{\sum_{a'} e^{\boldsymbol{\theta}^T \boldsymbol{x}(s,a')}}$$

- The encoding $\boldsymbol{x}(s, a)$ can be any encoding, including non-linear functions of the states and actions
- Of course, π may also be arbitrarily complex (wink, wink)

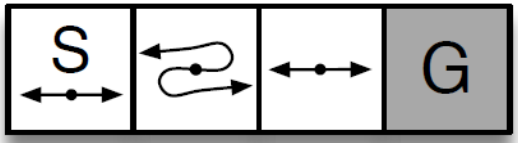
Partial Observability

- Consider this short corridor example

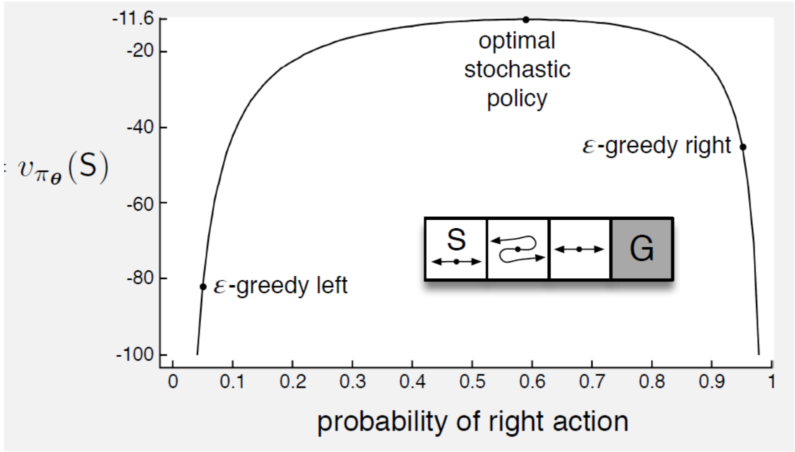


  - Actions are left/right, but their effect is reversed in state 2
  - Suppose features are $\boldsymbol{x}(s, right) = [1\ 0], \boldsymbol{x}(s, left) = [0\ 1]$
    - Same features regardless of the value of s
    - You don't see which state you're in effectively
  - Reward of -1 after each step
- What is the optimal policy (without knowing where you are)?
  - Need to make two rights and a left
    - So cannot be deterministic
  - Turns out a coin flip with a slight bias to the right is optimal
    - More next

Partial Observability, cont'd



- What would an action-value method do?
  - If $Q([1\ 0]) > Q([0\ 1])$, always go right
    - Or with ϵ-greedy probability
  - Cannot learn different policies per state
    - At best, take correct action w.p. ϵ
- Policy gradient method will learn a better probability than ϵ-greedy



Policy Gradient Setup

- Unlike Q-learning, we now have a policy that is learned separately from the q-values
- The policy π is defined in the same way as before:

$$\pi(a|s;\boldsymbol{\theta}) = \mathbb{P}_\pi[A_t = a|S_t = s]$$

- The state values, $v_\pi(s)$, and action values, $q_\pi(s,a)$, are defined in the same way as before
  - The main difference is that the policy is now trained separately from the value estimates
  - We are now directly training the policy to maximize the value of each state

Optimization Function

- What function should the policy optimize?
  - Maximize the value $v_\pi(s)$ for all s
  - What is an issue with this?
    - Don't know the real v_π
    - Also, v_π is policy-specific, so it changes every time we change π
  - For now, assume we know $v_\pi(s)$ for each state s
    - How do we train π?
- Suppose the policy π is parameterized by $\boldsymbol{\theta}$ (written $\pi_{\boldsymbol{\theta}}$)
  - The policy can be any function, as usual
    - E.g., a neural network's parameters
    - When clear from context, we'll just write π
    - Only requirement is that it's differentiable w.r.t $\boldsymbol{\theta}$

Optimization Function, cont'd

- For now, assume we know $v_\pi(s)$ for each state s
- Suppose the policy π is parameterized by $\boldsymbol{\theta}$ (written $\pi_{\boldsymbol{\theta}}$)
- We want to pick the $\boldsymbol{\theta}$ that maximize $v_{\pi_{\boldsymbol{\theta}}}(s)$ for all s
  - This would be the optimal policy within the family of functions we are considering
    - E.g., all NNs with some architecture, all softmax functions
- Even if we assume we know $v_{\pi_{\boldsymbol{\theta}}}(s)$, we can't just pick the optimal $\boldsymbol{\theta}$ usually (why?)
  - Function is non-convex in $\boldsymbol{\theta}$, especially if π is a neural net
  - What is our usual approach in this case?
    - Gradient descent!
    - In this case, we call it a policy gradient

Policy Gradients

Rensselaer

- Look at policy gradient (w.r.t. $\boldsymbol{\theta}$) in finite state case:

$$\nabla v_\pi(s) = \nabla\left[\sum_a \pi(a|s) q_\pi(s,a)\right]$$

$$= \sum_a \nabla\pi(a|s) q_\pi(s,a) + \pi(a|s)\nabla q_\pi(a,s)$$

$$= \sum_a \nabla\pi(a|s) q_\pi(s,a) +$$

$$+\pi(a|s)\nabla \sum_{s\prime} P(s,a,s')[R(s,a,s') + \gamma v_\pi(s')]$$

$$= \sum_a \nabla\pi(a|s) q_\pi(s,a) + \gamma\pi(a|s) \sum_{s\prime} P(s,a,s')\nabla v_\pi(s')$$

- Notice that ∇v_π appears recursively

Policy Gradients, cont'd

$$\nabla v_\pi(s) = \sum_a \nabla\pi(a|s) q_\pi(s,a) + \gamma\pi(a|s) \sum_{s'} P(s,a,s')$$
$$\left[\sum_{a'} \nabla\pi(a'|s') q_\pi(s',a') + \gamma\pi(a'|s') \sum_{s''} P(s',a',s'')\nabla v_\pi(s'')\right]$$

- Notation:
- $\mathbb{P}[s \to x, k, \pi]$ is the probability that state x is visited from state s after k steps (following policy π)
    - $\mathbb{P}[s \to x, 0, \pi] = 1$ if $s = x$ and 0, otherwise
    - $\mathbb{P}[s \to x, 1, \pi] = \sum_a \pi(a|s) P(s,a,x)$
    - $\mathbb{P}[s \to x, 2, \pi] = \sum_a \pi(a|s) \sum_{s'} P(s,a,s') \sum_{a'} \pi(a'|s') P(s',a',x)$

Policy Gradients, cont'd

$$\nabla v_\pi(s) = \sum_a \nabla\pi(a|s) q_\pi(s,a) + \gamma\pi(a|s) \sum_{s'} P(s,a,s') \left[\sum_{a'} \nabla\pi(a'|s') q_\pi(s',a') + \gamma\pi(a'|s') \sum_{s''} P(s',a',s'') \nabla v_\pi(s'') \right]$$

- Look at first term:

$$\sum_a \nabla\pi(a|s) q_\pi(s,a) =$$

$$= \sum_{x\in S} \mathbb{P}[s \to x, 0, \pi] \sum_a \nabla\pi(a|x) q_\pi(x,a)$$

  – since $\mathbb{P}[s \to x, 0, \pi] = 1$ only when $x = s$

Policy Gradients, cont'd

$$\nabla v_\pi(s) = \sum_a \nabla\pi(a|s)q_\pi(s,a) + \gamma\pi(a|s)\sum_{s'} P(s,a,s')$$
$$\left[\sum_{a'} \nabla\pi(a'|s')q_\pi(s',a') + \gamma\pi(a'|s')\sum_{s''} P(s',a',s'')\nabla v_\pi(s'')\right]$$

- Look at second term (rename s' to x):

$$\gamma\sum_a \pi(a|s)\sum_x P(s,a,x)\left[\sum_{a'} \nabla\pi(a'|x)q_\pi(x,a')\right] =$$
$$= \gamma\sum_x\left[\sum_{a'} \nabla\pi(a'|x)q_\pi(x,a')\right]\sum_a \pi(a|s)P(s,a,x)$$
$$= \sum_{x\in S}\mathbb{P}[s \to x, 1, \pi]\left[\sum_{a'} \nabla\pi(a'|x)q_\pi(x,a')\right]$$

Policy Gradients, cont'd

- Rewriting the policy gradient:

$$\nabla v_\pi(s) = \sum_{x \in S} \mathbb{P}[s \to x, 0, \pi] \sum_a \nabla \pi(a|x) q_\pi(x,a) +$$
$$+\gamma \sum_x \mathbb{P}[s \to x, 1, \pi] \sum_a \nabla \pi(a|x) q_\pi(x,a)$$
$$+ \sum_a \gamma \pi(a|s) \sum_{s\prime} P(s,a,s') \left[\sum_{a'} \gamma \pi(a'|s') \sum_{s''} P(s',a',s'') \nabla v_\pi(s'') \right]$$

- We can continue the expansion in the same fashion for future steps

Policy Gradients, cont'd

$$\nabla v_\pi(s_0) = \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}[s_0 \to s, k, \pi] \sum_a \nabla \pi(a|s) q_\pi(s, a)$$

- We can treat the sum of probabilities as the discounted aggregate state visitation "probability"
  - Call it d_π
  - Similar to the stationary distribution $\boldsymbol{\mu}_\pi$ but not the same

$$\boldsymbol{\mu}_\pi \boldsymbol{P} = \boldsymbol{\mu}_\pi$$

- What probability does $\boldsymbol{\mu}_\pi$ capture?

$$\lim_{k \to \infty} \mathbb{P}[s_0 \to s, k, \pi]$$

- If you want to treat d_π as a real probability distribution, need to normalize it so that it sums up to 1

Policy Gradients, cont'd

$$\nabla v_\pi(s_0) = \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \mathbb{P}[s_0 \to s, k, \pi] \sum_a \nabla \pi(a|s) q_\pi(s,a)$$

- We can treat the sum of probabilities as the discounted aggregate state visitation probability
  - Call it d_π
- So, finally

$$\nabla v_\pi(s_0) = \sum_s d_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)$$

- This is the **policy gradient theorem**!
  - Note that we need to know q_π for each (s,a) pair

Using the policy gradient theorem

- To improve a given a policy $\pi_{\boldsymbol{\theta}}$, we observe the next state-action-reward pair, and compute the gradient
- Note that we can think of the gradient as an expectation

$$\nabla v_\pi(s_0) = \sum_s d_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)$$
$$= \mathbb{E}_{d_\pi}\left[\sum_a \nabla \pi(a|S_t) q_\pi(S_t,a)\right]$$

  – Technically need to normalize d_π
    - That's just a constant which would be multiplied by the learning rate anyway
- How do we approximate the expectation using real data?
  – Average over real data

Using the policy gradient theorem, cont'd

- Note that we can think of the gradient as an expectation

$$\nabla v_\pi(s_0) = \mathbb{E}_{d_\pi}\left[\sum_a \nabla\pi(a|S_t)q_\pi(S_t,a)\right]$$

- How do we approximate the expectation using real data?
  - Average over real data
  - For each state s, compute gradient over all actions:

$$\sum_a \nabla\pi(a|s)q_\pi(s,a)$$

  - Any issues with this?
  - Need to know all $q_\pi(s,a)$

REINFORCE algorithm

- The benefit of the policy gradient theorem is that we can compute gradients w.r.t. $\boldsymbol{\theta}$ and improve the policy
  – As long as we have good estimates $\hat{q}$ of the real q function
  – We'll discuss several ways to get $\hat{q}$
- We could directly instantiate a gradient-descent algorithm:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \sum_a \hat{q}(S_t, a) \nabla \pi(a|S_t; \boldsymbol{\theta})$$

- What is the issue with this approach?
  – Updates the policy for all actions simultaneously
    - Each data point is for one action only
  – Requires good estimate of all action-values
  – May require a lot of data to converge

REINFORCE algorithm, cont'd

- To avoid needing an estimate for each action value, one could modify the policy gradient theorem
  - Could use the return G_t directly
  - To simplify the math, focus on finite-horizon case

$$\nabla v_\pi(s) = \nabla \mathbb{E}_\pi[G_1 | S_1 = s]$$

$$= \sum_{tr=(S_1,A_1,R_1\ldots)} \nabla \mathbb{P}_\pi[tr | S_1 = s] \mathbb{E}_\pi[G_1 | tr]$$

(log trick)

$$= \sum_{tr=(S_1,A_1,R_1\ldots)} \mathbb{P}_\pi[tr | S_1 = s] \nabla \log(\mathbb{P}_\pi[tr | S_1 = s]) \mathbb{E}_\pi[G_1 | tr]$$

$$= \sum_{tr=(S_1,A_1,R_1\ldots)} \mathbb{P}_\pi[tr | S_1 = s] \nabla \log\left(\prod_{t=1}^{T} \pi(A_t | S_t) P(S_t, A_t, S_{t+1})\right) \mathbb{E}_\pi[G_1 | tr]$$

$$= \sum_{tr} \mathbb{P}_\pi[tr | S_1 = s] \left(\sum_t \nabla \log\big(\pi(A_t | S_t)\big) + \nabla \log\big(P(S_t, A_t, S_{t+1})\big)\right) \mathbb{E}_\pi[G_1 | tr]$$

$$= \mathbb{E}_\pi\left[\sum_{t=1}^{T} \nabla \log\big(\pi(A_t | S_t)\big)\, G_1 \middle| S_1 = s\right]$$

REINFORCE algorithm, cont'd

- Final form for the gradient is

$$\nabla v_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=k}^{T} \nabla \log\big(\pi(A_t|S_t)\big)\, G_k \middle| S_k = s\right]$$

  – Could apply the gradient after any step k

- Once we have the gradient, update weights as usual

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} v_{\pi_{\boldsymbol{\theta}}}(s)$$

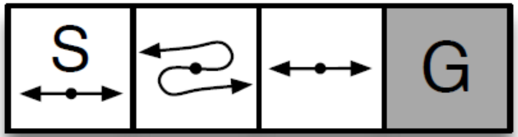
  – After visiting state s
  – This is similar to the Monte Carlo learning method where we wait until the end of the episode to observe G_t

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

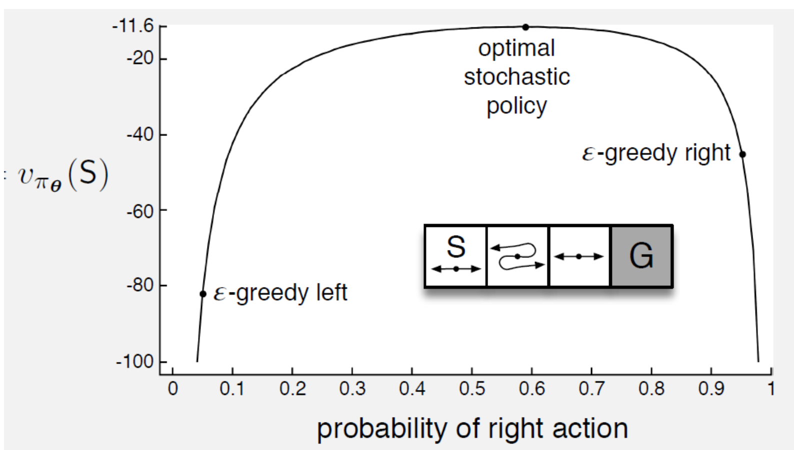
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

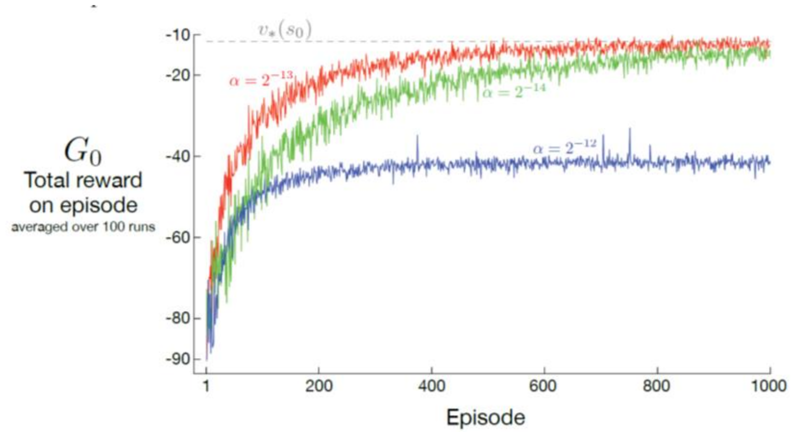
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Partial Observability, cont'd



- What would an action-value method do?
  - If $Q([1\ 0]) > Q([0\ 1])$, always go right
    - Or with ϵ-greedy probability
  - Cannot learn different policies per state
    - At best, take correct action w.p. ϵ
- Policy gradient method learns a better policy than ϵ-greedy
  - Suppose we use softmax policy $\pi([1\ 0]) = \dfrac{e^{\theta_1 \cdot 1 + \theta_2 \cdot 0}}{e^{\theta_1} + e^{\theta_2}}$





Issues with REINFORCE

- Can you spot any issues with this iteration?

$$\nabla v_\pi(s) = \mathbb{E}_\pi\left[\sum_{t=k}^{T} \nabla \log\big(\pi(A_t|S_t)\big)\, G_k \middle| S_k = s\right]$$

  – How important is the magnitude of G_k?

  – Turns out quite a bit – tasks have greatly varying returns

  – Especially problematic if *good* runs have zero returns

    - Gradient is 0!

- Vanilla REINFORCE has very large variance depending on G_k
- Next time we'll discuss how to address this issue