

Lecture notes - Introduction to Reinforcement learning with David Silver

Lecture 7

By - Amal Sunny

Policy Gradient

- Normally, for control we'd take the approximated values from the action-value function and act greedily w.r.t to it, to get the optimal policy.
- But thats only one approach. In some sense its more natural/useful to just directly paremetrize the policy instead of the value function.
- Essentially, we define weights(u instead of w for value function last lecture) for the policy which we can directly adjust to affect the distribution - i.e the actions we would take in different states.
- Our problem to solve, is to adjust u to solve the RL problem, i.e maximize cumulative reward.
- This idea will still be focusing on model-free RL.

Advantages of Policy-Based RL

Advantages:

- Better convergence due to following gradient in policy - chatter is minimized
- Effective in high-dimensional/continuous action space
- Can learn stochastic policies (Why ?)
 - Imagine a game of rock paper scissor. A deterministic policy in that game will surely never win(opponent would start countering very easily)
 - Not only that, but environments in which two states would have the same information but require different actions - would end up taking the wrong decision on one of them atleast.

Disadvantages:

- Naive policy-based methods can be slower, high variance and less efficient (due to incremental steps here over large steps in value based methods - essentially trading some efficiency for stability)

- Typically converges to a local rather than global optimum

Policy Objective functions

- Our goal is to optimize the policy, but before that we need to define our objective function here.
- Goal: given a policy $\pi_\theta(s, a)$ with parameters θ , find best θ
- To measure the quality of different policies, we have three methods
 - In episodic environments, the start value is used - which is the total reward obtained from following current policy from given start state.

- $$J_1(\theta) = V^{\pi_\theta}(s_1) = E^{\pi_\theta}[v_1]$$

- In continuing environments, the average value of reward obtainable is used

- $$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or, the average reward per time-step

- $$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R_s^a$$

- where d^{π_θ} is a stationary distribution of the markov chain for π_θ - essentially probabilities of state-state transition

Policy optimization

- Policy based RL is an optimization problem (θ in the function)
- Find θ maximizing $J(\theta)$
- Some non gradient approaches:
 - Hill climbing
 - Genetic algorithm
 - Simplex/ amoeba
- However, if gradient methods exist they'd provide a greater efficiency in this case:
 - Gradient descent
 - Conjugate gradient
 - Quasi-newton
- We focus on the simplest case gradient descent, with further extensions as needed.

Policy Gradient

- Gradient works by adjusting the function parameters in the direction of the function gradient by a small step size, repeatedly.
 - Here gradient is just the vector partial derivative
- The incremental step is given by:

$$\Delta\theta = \alpha \nabla_{\theta} J(\theta)$$

- where α is step size, and $\nabla_{\theta} J(\theta)$ is policy gradient.

Computing gradient by finite difference

- To evaluate policy gradient
- For each dimension $k \in [1, n]$
 - Estimate the derivative as such

- $$\frac{\delta J(\theta)}{\delta \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- where u_k is unit vector with 1 in kth component, 0 otherwise

- Uses n evaluations for n dimensional gradient descent
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if not differentiable.

Monte-Carlo Policy Gradient

Score Function(Likelihood ratios)

- To compute policy gradient analytically
- We assume policy is differentiable whenever it is non-zero(actions are being picked)
 - And the gradient is known
- Likelihood ratios(which will show up quite a lot after) exploit the following identity:

$$\begin{aligned}\nabla_{\theta} \pi_{\theta}(s, a) &= \pi_{\theta}(s, a) \frac{\nabla_{\theta} \pi_{\theta}(s, a)}{\pi_{\theta}(s, a)} \\ &= \pi_{\theta}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a)\end{aligned}$$

- The score function is $\nabla_{\theta} \log \pi_{\theta}(s, a)$
- And the reason we convert to this form, is it makes expectation of this value a lot easier to calculate (as this is the policy which we are evaluating - we can sample from this to calc. expectation)

Softmax Policy (An example of Score function in action)

- We weight actions using a linear combination of features $\phi(s, a)^T \theta$
- Probability of an action is proportional to exponentiated weight
 - $\pi_{\theta}(s, a) \propto e^{\phi(s, a)^T \theta}$
- Thus, the score function here becomes:

$$\nabla_{\theta} \log \pi_{\theta}(s, a) = \phi(s, a) - E_{\pi_{\theta}}[\phi(s, .)]$$

- Here $\phi(s, a)$ represents the actions taken, and the E stands for expectation of all remaining possible actions at this state. So the gradient pushes the function towards actions that incur a greater value than the average

One-Step MDPs (contextual bandit)

- Consider a simple class of one-step MDPs
 - Which start in some state $s \sim d(s)$
 - Terminate after one time-step with reward $r = R_{s,a}$
- So essentially, this is like a one-armed bandit problem but with the state of the agent being factored in as well.
- We use likelihood ratios to compute policy gradient

$$\begin{aligned} J(\theta) &= E_{\pi_{\theta}}[r] \\ &= \sum_{s \in S} d(s) \sum_{a \in A} \pi_{\theta}(s, a) R_{s,a} \\ \nabla_{\theta} J(\theta) &= \sum_{s \in S} d(s) \sum_{a \in A} \pi_{\theta}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a) R_{s,a} \\ &= E_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(s, a) r] \end{aligned}$$

- The gradient is in terms of expectation of the score times reward, showing which way to move the function in given the reward.
 - Good +ve reward, function moves in that direction
 - -ve reward, goes the other way.

Policy Gradient Theorem

- Now, we move on to generalizing for multi-step MDPs
- We take the likelihood ratio approach, but replace instantaneous reward r with long-term value $Q^\pi(s, a)$
- Policy gradient theorem applies to all three cases of comparing objectives: start state objective, average reward and average value.

Theorem

For any differentiable policy $\pi_\theta(s, a)$.

For any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$, the policy gradient is

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

Algorithm in action: Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

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

function REINFORCE

Initialise θ arbitrarily

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

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

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

end for

end for

return θ

end function

Actor-Critic

Reducing variance using a critic

- MC policy gradient still has high variance.
- Now, here is where we introduce the actor-critic model to solve that.
- Instead of using the return to estimate action-value function, we use a critic to estimate it instead(using a value function approximator).

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Essentially, we have two components to this system who maintain their two respective sets of parameters:
 - Critic: Updates action-value function parameters w - i.e the function approximator
 - Actor: Updates policy parameters θ , in direction given by critic via action-value function - i.e policy gradient evaluator.
- Actor-critic algorithms follow an approx. policy gradient

$$\begin{aligned}\nabla_\theta J(\theta) &= E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)] \\ \Delta \theta &= \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)\end{aligned}$$

- We can apply the methods used to improve approximators in last lecture for the critic here too (MC policy evaluation, TD learning, TD(λ))

Reducing variance Using a Baseline

- We can subtract a baseline function $B(s)$ from policy gradient without affecting the expectation of it.
- This can reduce variance, but doesn't affect expectation.
- Now, a good baseline function to choose is the state value function $B(s) = V^{\pi_\theta}(s)$ - This ensures the value function is replaced by the **advantage function** $A^{\pi_\theta}(s, a)$

$$\circ \quad A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

- Advantage functions tell us how much better is a particular action from a given state.

$$\circ \quad \nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)]$$

Estimating the Advantage Function

- So, now that we know we can replace value function with advantage function **and** that it can significantly reduce variance - critic should use it for estimation as well.
- i.e both state and action value functions should be estimated with it.
- One way to do this, is by using two function approximators and parameter vectors, then subtracting to get the advantage function.
- However, there's an alternative. TD error is an unbiased estimator of the advantage function -

$$\begin{aligned}E_{\pi_\theta}[\delta^{\pi_\theta} | s, a] &= E_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s') | s, a] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s, a)\end{aligned}$$

- So we can use TD error instead to compute the policy gradient

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(s, a) \delta^{\pi_{\theta}}]$$

- This approach only requires one set of critic parameters v .

Actor Critic at Different Time-Scales

- Both actor and critic can substitute their target functions for the various policy evaluation methods we've covered before (MC, TD, TD(λ))
- For critic:

- For MC, the target is the return v_t

$$\Delta\theta = \alpha(v_t - V_{\theta}(s))\phi(s)$$

- For TD(0), the target is the TD target $r + \gamma V(s')$

$$\Delta\theta = \alpha(r + \gamma V(s') - V_{\theta}(s))\phi(s)$$

- For forward-view TD(λ), the target is the λ -return v_t^{λ}

$$\Delta\theta = \alpha(v_t^{\lambda} - V_{\theta}(s))\phi(s)$$

- For backward-view TD(λ), we use eligibility traces

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$e_t = \gamma \lambda e_{t-1} + \phi(s_t)$$

$$\Delta\theta = \alpha \delta_t e_t$$

- For actor, we only consider the more complicated backward-view TD(λ)

- Just like forward-view TD(λ), we can mix over time-scales

$$\Delta\theta = \alpha(v_t^{\lambda} - V_v(s_t))\nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$$

- where $v_t^{\lambda} - V_v(s_t)$ is a biased estimate of advantage fn

- Like backward-view TD(λ), we can also use eligibility traces

- By equivalence with TD(λ), substituting $\phi(s) = \nabla_{\theta} \log \pi_{\theta}(s, a)$

$$\delta = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

$$e_{t+1} = \lambda e_t + \nabla_{\theta} \log \pi_{\theta}(s, a)$$

$$\Delta\theta = \alpha \delta e_t$$

Alternative policy gradient directions

- We started off by defining policy gradient as moving in the direction of the actual return from the critic, but then we moved on to replace it with some approximation - assuming it works. But how

can we be sure it works ?

- Surprisingly, if we pick our value function approximator carefully, its possible to not introduce any bias into it at all.
- Essentially, even without the true value function we can guarantee we'll end up following the true gradient. This approach is called **compatible function approximation**.
- We build up the features for the critic in such a way, the features themselves are the scores of our policy.
- Using linear combination of these features, we can guarantee that we will end up following the true gradient direction.

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

1. Value function approximator is compatible to the policy

$$\nabla_w Q_w(s, a) = \nabla_{\theta} \log \pi_{\theta}(s, a)$$

2. Value function parameters w minimise the mean-squared error

$$\epsilon = E_{\pi_{\theta}}[(Q^{\pi_{\theta}}(s, a) - Q_w(s, a))^2]$$

Then the policy gradient is exact,

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(s, a) Q_w(s, a)]$$

Natural Policy Gradient

- As our policy gets better over time, the variance of our estimates starts blowing up to infinity.
- This is an unfortunate property of the policy gradient algorithms seen so far.
- There is an alternative to fix this(recent discovery)
 - We start off with a deterministic policy, and adjust the parameters to get it closer to the objective - following same objective functions as before.
 - There, if we take the limiting case of the policy gradient theorem, there's this very simple update we can make there - its just rewriting the function to always pick the mean there, no noise added there.
 - The given info from the critic already contains info to make better decisions, from the gradient of the given value function(i.e what the critic gave, we take the gradient)
 - We adjust the parameters accordingly to get better reward as the actor.

Summary

- The **policy gradient** has many equivalent forms

$$\begin{aligned}
 \nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{v}_t] && \text{REINFORCE} \\
 &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) Q^w(s, a)] && \text{Q Actor-Critic} \\
 &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) A^w(s, a)] && \text{Advantage Actor-Critic} \\
 &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta] && \text{TD Actor-Critic} \\
 &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta e] && \text{TD}(\lambda) \text{ Actor-Critic} \\
 G_{\theta}^{-1} \nabla_{\theta} J(\theta) &= w && \text{Natural Actor-Critic}
 \end{aligned}$$

- Each leads a stochastic gradient ascent algorithm

Problems of the week

Problem 1 - Bayes Expression (Taken from CS234)

Write an expression for the probability that the state at time 0 is s given that the state at time 1 is s' and the action at time 0 is a . Let us define $d_0(s) = \Pr(S_0 = s)$. Please write your answer in terms of d, π , and the transition probabilities $P(s, a, s')$. Recall Bayes' Theorem:

Ans:

$$\begin{aligned}
 P(S_0 = s | A_0 = a, S_1 = s') &= \frac{P(S_0 = s, A_0 = a, S_1 = s')}{P(A_0 = a, S_1 = s')} \\
 &= \frac{d_0(s) \pi(s, a) P(s, a, s')}{\sum_{s_0} P(S_0 = s_0) P(S_1 = s', A_0 = a | S_0 = s_0)} \\
 &= \frac{d_0(s) \pi(s, a) P(s, a, s')}{\sum_{s_0} \pi(s_0, a) P(s_0, a, s')}
 \end{aligned}$$


s_1	s_2	s_3	s_4	s_5	s_6	s_7
+1	+0	-1		-1	+0	+10



Figure 1: Mars Rover MDP

Let us consider the Mars Rover MDP seen in Figure 1. Similar to the in class example, s_1 and s_7 are terminal states. The rewards are received when you enter a state (the reward for entering state s_4 is 0). There are two actions, TryLeft and TryRight. TryLeft transitions from state s_i to s_{i-1} with 0.5 probability and stays in state s_i with 0.5 probability. Similarly, TryRight transitions from state s_i to s_{i+1} with 0.5 probability and stays in state s_i with 0.5 probability. Let $\gamma = 1$.

We want to apply REINFORCE to learn a policy in this Mars Rover setting. Let our feature representation be a one-hot encoding using the state, action pair. More concretely, let us denote $a_1 = \text{TryLeft}$ and $a_2 = \text{TryRight}$. Then our feature representation is $\phi(s_i, a_j)_k = 1$ if $((j - 1) * 7) + (i - 1) = k$ and 0 otherwise (assuming the vector is 0-indexed). Let us use a softmax policy parameterized by θ :

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

a)What is the score function for this softmax policy?

Ans:

$$\text{Score func} = \nabla_{\theta} \log \pi_{\theta}(s, a) = \phi(s, a) - E_{\pi_{\theta}}[\phi(s, .)]$$

b)Using REINFORCE, what is the update equation for θ ?

Ans:

$$\theta = \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) G_t = \theta + \alpha [\varphi(s, a) - \sum_b \pi_{\theta}(s, b) \cdot \varphi(s, b)] G_t$$

Program of the week

Terrain navigating bot