Reinforcement Learning

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

Recap

- Markov Decision Process is the mathematical formulation of the Reinforcement Learning Problem defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$
- Each state satisfies the Markov Property i.e., future is independent of the past given the present.
- The Agent and the Environment act in a time sequenced loop. Policy π determines how the agent chooses actions
- Value function approximates how good a state is while Q-value function approximates the goodness of a state action pair.
- Bellman Equation is a recursive formula for the Q-value function
- Q-learning is an algorithm that repeatedly adjusts Q to minimize the Bellman error
- If the Q-value function approximater is Deep Neural Network, we get Deep Q-Learning
- SARSA is an on-policy variation of Q-learning

- **What is the problem with Q-learning?**
- The Q-function can be very complicated.
- For example, a robot grasping an object can have a very high dimensional state. It can be hard to learn exact Q-value for every (state, action) pair!

- **What is the problem with Q-learning?**
- The Q-function can be very complicated.
- For example, a robot grasping an object can have a very high dimensional state. It can be hard to learn exact Q-value for every (state, action) pair!
- But the policy can be much simpler: just close your hand
- Can we learn a policy directly, e.g. finding the best policy from a collection of policies?

Policy Gradients

- Formally, let's define a class of parameterized policies:

$$\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$$

- For each policy, we can define its return:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

Policy Gradients

- Formally, let's define a class of parameterized policies:

$$\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$$

- For each policy, we can define its return:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

- Now, we want to find the optimal policy $\theta^\star = \arg\max_\theta \mathcal{J}(\theta)$
- How can we do this?

Policy Gradients

- Formally, let's define a class of parameterized policies:

$$\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$$

- For each policy, we can define its return:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta\right]$$

- Now, we want to find the optimal policy $\theta^\star = \arg\max_\theta \mathcal{J}(\theta)$
- How can we do this?
- **Solution**: Gradient Ascent on Policy parameters!

- REINFORCE is an elegant algorithm for maximizing the expected return
- Intuition: trial and error
- Sample a trajectory τ . If you get a high reward, try to make it more likely. If you get a low reward, try to make it less likely.
- A trajectory is sequence of state, action and reward. $\tau = (s_0, a_0, r_0, s_1, a_1, \cdots)$

- Expected Reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau)p(\tau;\theta)d\tau$$

- Expected Reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) d\tau$$

- Now let's differentiate this:

$$\nabla_{\theta}\mathcal{J}(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau;\theta) d\tau$$

$$\text{where} \quad p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_{\theta}(a_t|s_t)$$

- Expected Reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right] = \int_{\tau} r(\tau)p(\tau;\theta)d\tau$$

- Now let's differentiate this:

$$\nabla_{\theta}\mathcal{J}(\theta) = \int_{\tau} r(\tau)\nabla_{\theta}p(\tau;\theta)d\tau$$

$$\text{where} \quad p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_{\theta}(a_t|s_t)$$

- But this is **intractable!**

- However, we can use a nice trick:

$$
\begin{aligned}
\nabla_\theta p(\tau;\theta) &= p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} \\
&= p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)
\end{aligned}
$$

- However, we can use a nice trick:

$$\begin{aligned}\nabla_\theta p(\tau;\theta) &= p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} \\ &= p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)\end{aligned}$$

- Now, if we inject this back:

$$\begin{aligned}\nabla_\theta \mathcal{J}(\theta) &= \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right)\ p(\tau;\theta)d\tau \\ &= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]\end{aligned}$$

- We can estimate with Monte Carlo sampling

REINFORCE

- Now, we had

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

REINFORCE

- Now, we had

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Thus

$$\log\ p(\tau;\theta) = \sum_{t\geq 0} \log\ p(s_{t+1}|s_t, a_t) + \log\ \pi_\theta(a_t|s_t)$$

- Now, we had

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Thus

$$\log\ p(\tau;\theta) = \sum_{t\geq 0} \log\ p(s_{t+1}|s_t, a_t) + \log\ \pi_\theta(a_t|s_t)$$

- And when differentiating:

$$\nabla_\theta \log\ p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log\ \pi_\theta(a_t|s_t)$$

REINFORCE

- Now, we had

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

- Thus

$$\log\ p(\tau;\theta) = \sum_{t\geq 0} \log\ p(s_{t+1}|s_t, a_t) + \log\ \pi_\theta(a_t|s_t)$$

- And when differentiating:

$$\nabla_\theta \log\ p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log\ \pi_\theta(a_t|s_t)$$

- It doesn't depend on transition probabilities!

- Therefore when sampling a trajectory τ, we can estimate $\mathcal{J}(\theta)$ with:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

REINFORCE

- Therefore when sampling a trajectory τ, we can estimate $\mathcal{J}(\theta)$ with:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

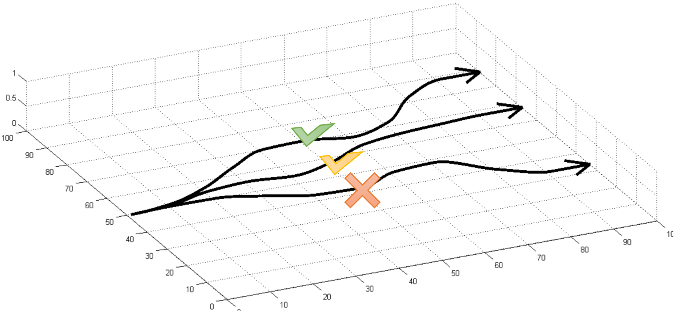
- Interpretation:
  - If $r(\tau)$ is high, push up the probabilities of the actions seen
  - If $r(\tau)$ is low, push down the probabilities of the actions seen

- Therefore when sampling a trajectory τ, we can estimate $\mathcal{J}(\theta)$ with:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Interpretation:
  - If $r(\tau)$ is high, push up the probabilities of the actions seen
  - If $r(\tau)$ is low, push down the probabilities of the actions seen
- Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

REINFORCE



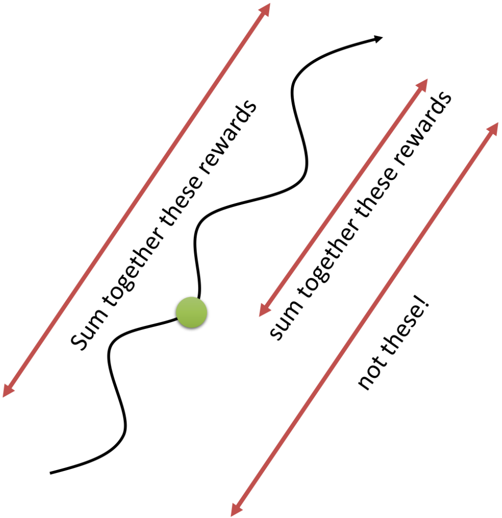
- However, there is a problem.
- This suffers from high variance because credit assignment is really hard.
- Can we help the estimator?

Variance Reduction

- However, there is a problem.
- This suffers from high variance because credit assignment is really hard.
- Can we help the estimator?
- **First idea:** Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t\geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance Reduction

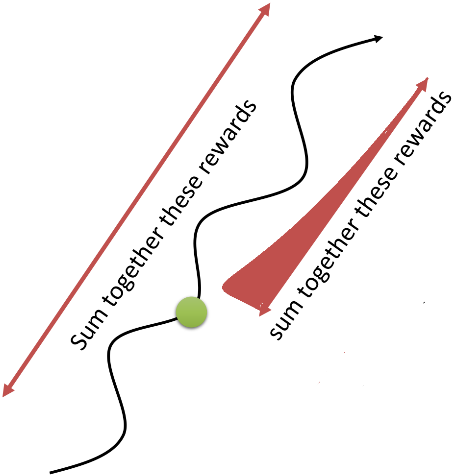


- But this still doesn't completely solve the problem of credit assignment.
- It can lead to bias due to delayed rewards.

- But this still doesn't completely solve the problem of credit assignment.
- It can lead to bias due to delayed rewards.
- **Second idea:** Use discount factor γ to ignore delayed effects

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Variance Reduction



Variance Reduction - Baselines

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

Variance Reduction - Baselines

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.
- **What is important then?** Whether a reward is better or worse than what you expect to get

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.
- **What is important then?** Whether a reward is better or worse than what you expect to get
- **Idea**: Introduce a baseline function dependent on the state

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Variance Reduction - Baselines

- **Problem:** The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.
- **What is important then?** Whether a reward is better or worse than what you expect to get
- **Idea**: Introduce a baseline function dependent on the state

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- A simple baseline: constant moving average of rewards experienced so far from all trajectories

How to choose the baseline?

- Can we choose a better baseline?
- Basically, we want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

How to choose the baseline?

- Can we choose a better baseline?
- Basically, we want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.
- What does this remind you of?

- Can we choose a better baseline?
- Basically, we want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.
- What does this remind you of?
- **Answer:** Q-function and value function!

How to choose the baseline?

- Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

How to choose the baseline?

- Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.
- The term $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called **Advantage** and is donated by $A^\pi(s_t, a_t)$

How to choose the baseline?

- Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.
- The term $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called **Advantage** and is donated by $A^\pi(s_t, a_t)$
- Using this, we get the estimator:

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t \geq 0} \left(Q^\pi(s_t, a_t) - V^\pi(s_t)\right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- But we don't know Q and V. Can we learn them?

- But we don't know Q and V. Can we learn them?
- Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-network).

Actor-Critic

- But we don't know Q and V. Can we learn them?
- Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-network).
- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- The two networks adapt to each other, much like GAN training

- But we don't know Q and V. Can we learn them?
- Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-network).
- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- The two networks adapt to each other, much like GAN training
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay

Actor-Critic

- But we don't know Q and V. Can we learn them?
- Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-network).
- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- The two networks adapt to each other, much like GAN training
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay
- **Remark:** we can define by the advantage function how much an action was better than expected

Initialize policy parameters θ, critic parameters ϕ

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, ... , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$$\theta \leftarrow \alpha\Delta\theta$$

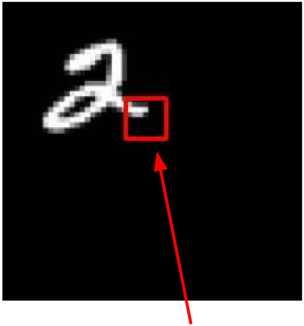
$$\phi \leftarrow \beta\Delta\phi$$

End for

REINFORCE in action: Recurrent Attention Model (RAM)

- **Objective:** Image Classification
- Take a sequence of "glimpses" selectively focusing on regions of the image, to predict class
  - Inspiration from human perception and eye movements
  - Saves computational resources ⇒ scalability
  - Able to ignore clutter / irrelevant parts of image
- **State:** Glimpses seen so far
- **Action:** (x,y) coordinates (center of glimpse) of where to look next in image
- **Reward:** 1 at the final timestep if image correctly classified, 0 otherwise

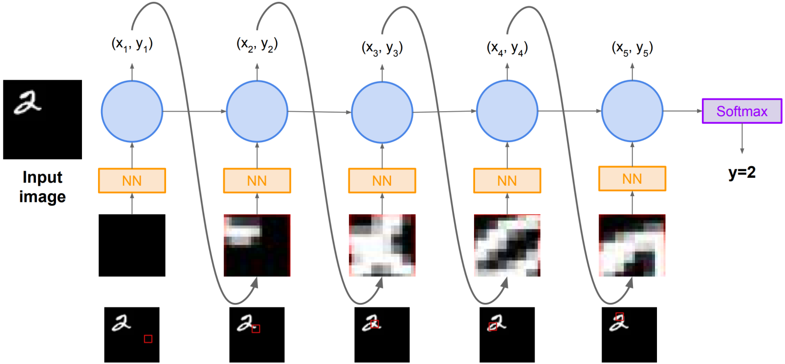
[0] [Mnih et al. 2014]



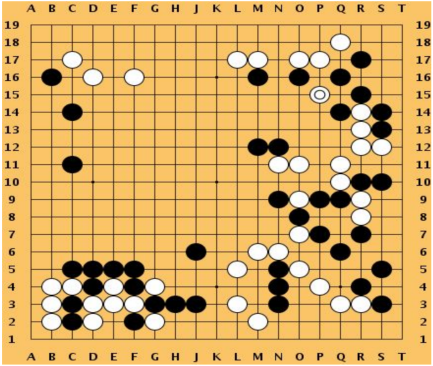
- Glimpsing is a non-differentiable operation
- Learn policy for how to take glimpse actions using REINFORCE
- Given state of glimpses seen so far, use RNN to model the state and output next action

[0] [Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



[0][Mnih et al. 2014]



- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

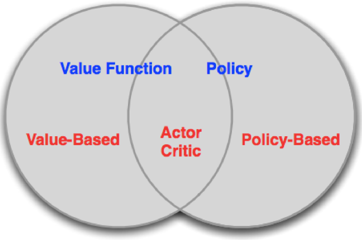
[0][Silver et al., Nature 2016]

How to beat the Go world champion - AlphaGo

- Featurize the board (stone color, move legality, bias, ...)
- Initialize policy network with supervised training from professional go games, then continue training using policy gradient (play against itself from random previous iterations, +1 / -1 reward for winning / losing)
- Also learn value network (critic)
- Finally, combine combine policy and value networks in a Monte Carlo Tree Search algorithm to select actions by lookahead search

[0] [Silver et al., Nature 2016]

- 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



Policy Gradient vs. Q-Learning

- Policy gradient and Q-learning use two very different choices of representation: policies and value functions
- Advantage of both methods: don't need to model the environment

Policy Gradient vs. Q-Learning

- Policy gradient and Q-learning use two very different choices of representation: policies and value functions
- Advantage of both methods: don't need to model the environment
- Pros/cons of policy gradient
  - Pro: unbiased estimate of gradient of expected return
  - Pro: can handle a large space of actions (since you only need to sample one)
  - Con: high variance updates (implies poor sample efficiency)
  - Con: doesn't do credit assignment

Policy Gradient vs. Q-Learning

- Policy gradient and Q-learning use two very different choices of representation: policies and value functions
- Advantage of both methods: don't need to model the environment
- Pros/cons of policy gradient
  - Pro: unbiased estimate of gradient of expected return
  - Pro: can handle a large space of actions (since you only need to sample one)
  - Con: high variance updates (implies poor sample efficiency)
  - Con: doesn't do credit assignment
- Pros/cons of Q-learning
  - Pro: lower variance updates, more sample efficient
  - Pro: does credit assignment
  - Con: biased updates since Q function is approximate
  - Con: hard to handle many actions (since you need to take the max)

Summary

- It can be hard to learn exact Q-value for every (state, action) pair for high dimensional states and actions
- But, we can just learn a policy that maximizes the reward
- We can use gradient ascent on policy parameters
- But this can suffer from high variance. Various strategies to tackle this.
- Actor-Critic methods combine Policy Gradients and Q-learning by training both an actor (the policy) and a critic (the Q-network)
- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust

These slides have been adapted from

- Chelsea Finn & Karol Hausman, Stanford CS224R: Deep Reinforcement Learning
- Fei-Fei Li, Yunzhu Li& Ruohan Gao, Stanford CS231n: Deep Learning for Computer Vision
- Jimmy Ba & Bo Wang, UofT CSC413/2516: Neural Networks and Deep Learning
- Sergey Levine, Berkeley CS285 Deep Reinforcement Learning