ECE433/COS435 Spring 2024 Introduction to RL

Lecture 8: REINFORCE; Value Iteration

Course Logistics

Midterm:

- March 7 in this classroom, during class time
- No internet. 1 page cheatsheet, double sided, typical-size paper

My office hour:
Will announce and let you sign up for 1:1 Chat (if interested)

RL Meet & Eat!

- Join small-group lunch with the teaching team!
- Will announce how to sign up on Ed
- First lunch starting next week

1. The REINFORCE Algorithm

In the last lecture, we have derived the policy gradient theorem

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a \mid s)]$$

which allows us to directly update policy parameters in a direction to improve the cumulative return.

Now we introduced REINFORCE, a Monte Carlo policy gradient method that estimates the policy gradient by sampling, then uses the estimate to update the policy parameters.

Iteration of REINFORCE

- Collect data: Run the policy π_θ in the environment to collect a sequence of states, actions, and rewards, typically represented as $(s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T)$, where T is the time step when the episode ends.
- Calculate Returns: For each time step t in the episode, calculate the total discounted return from that time step onwards:

$$G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}.$$

- Estimate the policy gradient using the formula:

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

- Update the policy parameters using the rule:

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

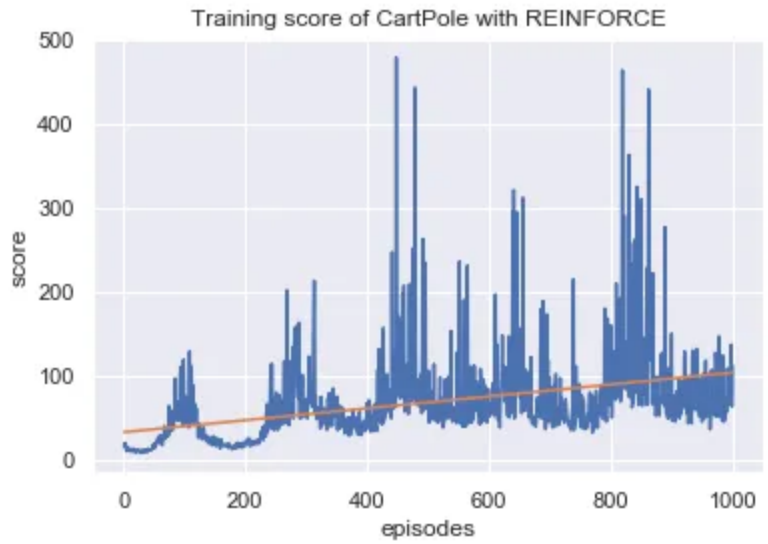
where α is the learning rate.

Discussion:

- why do you think the algorithm is called REINFORCE?

Numerical Simulation

You will implement REINFORCE in your homework. Here is a typical training curve of REINFORCE:



We can see that our agent slowly learns over time. However, the learning is slow and has high variance. Even with a long trajectory, samples drawn from this trajectory are highly dependent, leading to high variance.

Advantages

- REINFORCE is conceptually simple and easy to implement.
- It is an on-policy and act-in-the-world method.
- The method does not require knowledge of the environment dynamics or reward.
- It is value-free, i.e, not maintaining a value function estimate.
- It directly optimizes the policy and can work with arbitrary policy parametrization including neural networks.
- It is the pioneer of policy-based method, and give rise to many more efficient RL algorithms (we will discuss later in the course).

Disadvantages

- High variance in policy updates can lead to instability.
- Requires complete episodes for policy updates, making it inefficient in long or continuous tasks.
- Not using data efficiently: Each trajectory is only used once for estimating the current gradient, but data will be disregarded after one policy gradient update.
- This vanilla policy gradient update has no bias but high variance. Many following algorithms were proposed to reduce the variance while keeping the bias unchanged.

Discussion: How to make policy gradient method more efficient?

- Improve estimation of V, Q
- Improve the gradient ascent algorithm
- Does policy gradient even converge to the optimal policy?

2. Let's return to value function

To solve RL problems more efficiently, we have to go back to value function and Q functions, and we need more efficient ways to estimate values of state and state-action pairs, i.e., $V^\pi(s)$ and $Q^\pi(s,a)$.

Recall Dynamic Programming (DP) is a method for solving complex problems by breaking them down into simpler subproblems. It is applicable to problems exhibiting the properties of overlapping subproblems and optimal substructure.

Optimal Value Functions

The optimal value function V^* is defined as

$$V^*(s) := \max_\pi \mathbb{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) \mid s_0 = s \right]$$

The optimal Q function Q^* is defined as

$$V^*(s,a) := \max_\pi \mathbb{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) \mid s_0 = s, a_0 = a \right]$$

They are related by:

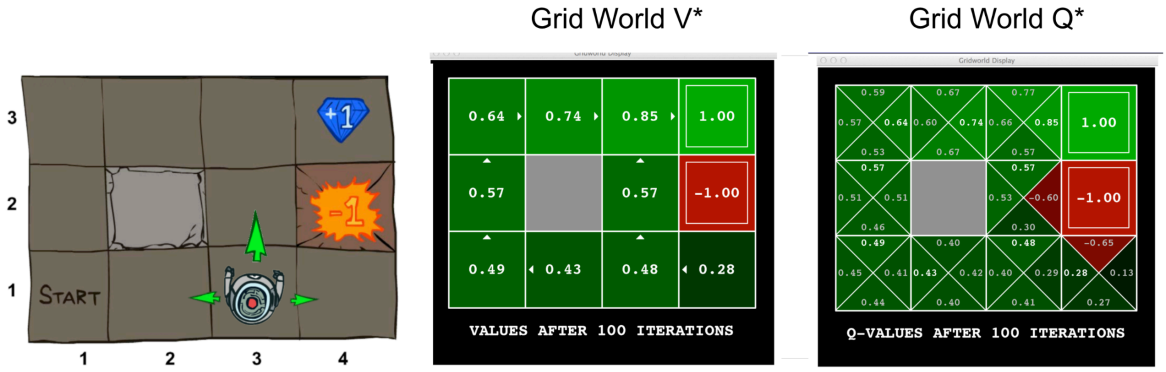
$$V^*(s) = \max_a Q^*(s,a).$$

Given Q^*, one can readout the optimal policy $\pi^*(s) = \text{argmax}_a Q^*(s,a)$.

Example: Grid World

A maze-like problem

- The agent lives in a grid
- Walls block the agent's path
- Noisy movement: actions do not always go as planned
  -- 80% of the time, the action North takes the agent North
  -- 10% of the time, North takes the agent West; 10% East
  -- If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
- Goal: maximize sum of (discounted) rewards



How do we find the optimal value function and Q function?

Let's characterize them first!

Theorem (Bellman Equation)

The function $V^* : \mathbb{R}^S \to \mathbb{R}$ is the optimal value function of the Markov decision process if and only if it satisfies the Bellman equation

$$V^*(s) = \max_a \mathbb{E}^\pi \left[r(s,a) + \gamma V^*(s') \right],$$

where $\mathbb{E}^\pi$ denotes the expectation over one state transition $s' \sim P(\cdot \mid s, a)$.

The Bellman equation can be viewed as the optimality condition for RL.

Let's discuss:

- Why does the optimal value function satisfy the Bellman equation? What would go wrong if the equation doesn't hold?
- Is the optimal value function the unique solution to the Bellman equation? Can there be more than one optimal value function?
- Can there be more than optimal policies?

The Bellman equation for a fixed policy π is

$$V^\pi(s) = \mathbb{E}^\pi \left[r(s,a) + \gamma V^*(s') \right]$$

2.1. Value Iteration (VI)

Next we want to solve the Bellman equation. We simply make it iterative, and we get Value Iteration!

Value iteration (VI) is an iterative algorithm used for computing the optimal value function in a Markov Decision Process (MDP). The key idea is to update the value of each state under the current policy and iteratively improve the policy based on these values.

One iteration of VI works by

- start with arbitrary initial value function, for example
$$V_0(s) = 0, \qquad \forall s$$
- The value iteration algorithm updates the value function $V(s)$ for each state s as follows:

$$V_{k+1}(s) = \max_{a \in A} \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V_k(s')]. \tag{1}$$

The above update is often called Bellman update or policy improvement.

Complexity of Bellman Update: For each state, the algorithm needs to consider each possible action and the resulting next states. This involves summing over all possible next states and rewards, which is proportional to the number of states and actions. Thus, the time complexity for each iteration is $O(S \times A \times S)$.

Value iteration for grid world: https://nowke.github.io/rlviz/

2.2. Convergence of value iteration for a fixed policy

Value iteration converges to the optimal value function V^* as $k \to \infty$. This convergence is guaranteed under the condition that the discount factor γ is between 0 and 1 ($0 \leq \gamma < 1$).

Consider the simple case of value iteration with a fixed policy:

$$V_{k+1} = R + \gamma P V_k,$$

where
- R denotes for short the reward corresponding to a fixed policy: $R(s) := \sum_a \pi(a|s) r(s,a)$
- P denotes for short the transition probability of the MDP following π: $P(s'|s) := \sum_a \pi(a|s) P(s'|s,a)$.

The value of policy π is $V^{\pi} $ and it satisfies the Bellman equation

$$V^{\pi} = R + \gamma P V^{\pi}$$

We subtract the Bellman equation from the value iteration for a fixed policy, and we get

$$V_{k+1} - V^{\pi} = \gamma P (V_k - V^{\pi}) = \gamma^{k+1} P^{k+1} (V_0 - V^{\pi})$$

Letting $k \to \infty$, we have $ \gamma^{k+1}$ converges to 0 exponentially fast, therefore
$$V_{k+1} - V^* \to 0.$$

Now we have proved the convergence of value iteration for a fixed policy.

2.3. Convergence of Value iteration for finding the optimal policy (Optional Reading)

Set up math notations

- **Bellman Operator** $\mathbb{B}$: For a given value function V, the Bellman operator $\mathbb{B}$ representing the Bellman update and transforms V to a new value function:
$$(\mathbb{B}V)(s) = \max_a \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V(s')].$$

  Thus the value iteration can be written equivalently as
$$V_{k+1} = \mathbb{B} V_k.$$
- **Metric Space**: Consider the space of value functions with the ∞-norm, defined for a function V as $||V||_{\infty} = \max_s |V(s)|$.

Contractiveness of the Bellman Operator

A function T is a contraction mapping on a metric space if there exists a constant β with $0 \leq \beta < 1$ such that for all x and y in the space,

$$||Tx - Ty|| \leq \beta ||x - y||.$$

For the Bellman operator $\mathbb{B}$, we show that it is a contraction mapping with $\beta = \gamma$ with respect to the $\|\cdot\|_\infty$ norm:

1. Take any two value functions V and W in the space of real-valued functions on the state space.
2. Apply the Bellman operator to both V and W and examine the difference:
$$|\mathbb{B}V(s) - \mathbb{B}W(s)| = |\max_a \sum_{s'} P(s'|s,a)[r(s,a) + \gamma V(s')] - \max_a \sum_{s'} P(s'|s,a)[r(s,a) + \gamma W(s')]|.$$
3. Using the property that $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$, we can bound the above expression:
$$|\mathbb{B}V(s) - \mathbb{B}W(s)| \leq \max_a \sum_{s'} P(s'|s,a)|\gamma V(s') - \gamma W(s')|.$$
4. Factor out γ and apply the definition of the ∞-norm:
$$|\mathbb{B}V(s) - \mathbb{B}W(s)| \leq \gamma \max_a \sum_{s'} P(s'|s,a)|V(s') - W(s')| \leq \gamma ||V - W||_\infty.$$
5. Since $\gamma < 1$ and the inequality holds for every state s, we have:
$$||\mathbb{B}V - \mathbb{B}W||_\infty \leq \gamma ||V - W||_\infty.$$
6. This shows that $\mathbb{B}$ is a contraction mapping with contraction constant γ.

Since $\mathbb{B}$ is a contraction mapping in the ∞-norm, by the Banach Fixed-Point Theorem, it has a unique fixed point. This fixed point is the optimal value function V^*. The Value Iteration algorithm, which iteratively applies $\mathbb{B}$, therefore converges to V^*. The rate of convergence is governed by the discount factor γ, which determines the 'tightness' of the contraction.