Function approximation

Model-Free Reinforcement Learning

Model-Free Reinforcement Learning

Recap of the previous lecture

- **Model-based.** Dynamic programming and fixed-point methods to estimate the value function
- **Model-free.** Monte Carlo and Temporal Difference methods

For now, we simply have lookup tables to store the value and q-value functions, $V[s]$ and $Q[s, a]$.

Recap of the previous lecture

- **Model-based.** Dynamic programming and fixed-point methods to estimate the value function
- **Model-free.** Monte Carlo and Temporal Difference methods

For now, we simply have lookup tables to store the value and q-value functions, $V[s]$ and $Q[s, a]$.

We face the **curse of dimensionality** when the state space is large.

- Backgammon: 10^{20} states
- Go: 10^{170} states

Not only we would need a lot of memory, but we would also need a huge number of samples to estimate the value and q-value functions.

Key idea. We want to capture the symmetry in the value and q-value functions, in order to generalize from the states we have seen to the states we have not seen.

$$\begin{cases} V_\theta(s) \approx V_\pi(s) \\ Q_\theta(s,a) \approx Q_\pi(s,a) \end{cases}$$

where

- θ is a set of parameters to be learned.
- $\theta \ll |D|$.

We don't store the value and q-value values individually anymore, but we store a function that approximates them.

Approximators

We can use different types of approximators:

- Linear functions
  - $\phi : \mathcal{S} \to \mathbb{R}^k$ (k features, how we "represent" the state)
  - $\theta \in \mathbb{R}^k$ (to be learned)
  - $V_\theta(s) = \theta^T \phi(s)$
- Decision trees
- Fourier basis
- Neural networks (known as DeepRL)
- ...

Example of linear function approximation for tic-tac-toe

Blackboard

How to learn the parameters?

We use gradient descent to update θ, each time we observe a new state.

Goal. Define an optimization problem to find the best parameters θ.

$$\min_{\theta} L(\theta)$$

- $L(\theta)$ small $\iff$ $V_\theta(s)$ is close to $V_\pi(s)$
- L should be differentiable (we can use weaker forms of differentiability as well)
- L is called the loss function

How to learn the parameters?

We use gradient descent to update θ, each time we observe a new state.

Goal. Define an optimization problem to find the best parameters θ.

$$\min_{\theta} L(\theta)$$

- $L(\theta)$ small $\iff$ $V_\theta(s)$ is close to $V_\pi(s)$
- L should be differentiable (we can use weaker forms of differentiability as well)
- L is called the loss function

Update rule

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t)$$

Loss function

We would like to minimize the difference between the value function and the approximator, hence we can use the mean squared error:

$$L(\theta) = \mathbb{E}_{s \sim \pi} \left[\left(V_\pi(s) - V_\theta(s) \right)^2 \right]$$

where the expectation is taken over the states visited by the policy π.

Loss function

We would like to minimize the difference between the value function and the approximator, hence we can use the mean squared error:

$$L(\theta) = \mathbb{E}_{s \sim \pi} \left[\left(V_\pi(s) - V_\theta(s) \right)^2 \right]$$

where the expectation is taken over the states visited by the policy π.

$$L(\theta) = \sum_{s \in \mathcal{S}} d_\pi(s) \left(V_\pi(s) - V_\theta(s) \right)^2$$

where $d_\pi(s)$ is the expected time spent in state s under policy π. (We give less weight to states that are visited less often.)

Loss Function $L(\theta)$	Gradient of Loss Function $\nabla_\theta L(\theta)$
$\sum_{s \in \mathcal{S}} d_\pi(s) \left(V_\pi(s) - V_\theta(s)\right)^2$	$-2 \sum_{s \in \mathcal{S}} d_\pi(s) \left(V_\pi(s) - V_\theta(s)\right) \nabla_\theta V_\theta(s)$

Loss Function $L(\theta)$	Gradient of Loss Function $\nabla_\theta L(\theta)$
$\sum_{s \in \mathcal{S}} d_\pi(s) \left(V_\pi(s) - V_\theta(s)\right)^2$	$-2 \sum_{s \in \mathcal{S}} d_\pi(s) \left(V_\pi(s) - V_\theta(s)\right) \nabla_\theta V_\theta(s)$

Problem 1: We don't know the distribution $d_\pi(s)$.

Loss Function $L(\theta)$	Gradient of Loss Function $\nabla_\theta L(\theta)$
$\sum_{s\in\mathcal{S}} d_\pi(s)\left(V_\pi(s) - V_\theta(s)\right)^2$	$-2\sum_{s\in\mathcal{S}} d_\pi(s)\left(V_\pi(s) - V_\theta(s)\right)\nabla_\theta V_\theta(s)$

Problem 1: We don't know the distribution $d_\pi(s)$.

Solution 1 we proceed sample by sample since asymptotically we will see all the states in proportion to their visitation frequency (it is called Stochastic Gradient Descent).

For each observed state s:

$$\theta_{t+1} = \theta_t + \alpha\left(V_\pi(s) - V_\theta(s)\right)\nabla_\theta V_\theta(s)$$

Loss Function $L(\theta)$	Gradient of Loss Function $\nabla_\theta L(\theta)$
$\sum_{s \in \mathcal{S}} d_\pi(s) \left(V_\pi(s) - V_\theta(s)\right)^2$	$-2 \sum_{s \in \mathcal{S}} d_\pi(s) \left(V_\pi(s) - V_\theta(s)\right) \nabla_\theta V_\theta(s)$

Problem 1: We don't know the distribution $d_\pi(s)$.

Solution 1 we proceed sample by sample since asymptotically we will see all the states in proportion to their visitation frequency (it is called Stochastic Gradient Descent).

For each observed state s:

$$\theta_{t+1} = \theta_t + \alpha \left(V_\pi(s) - V_\theta(s)\right) \nabla_\theta V_\theta(s)$$

Problem 2: We don't know the value function $V_\pi(s)$.

Loss Function $L(\theta)$	Gradient of Loss Function $\nabla_\theta L(\theta)$
$\sum_{s\in\mathcal{S}} d_\pi(s)\left(V_\pi(s) - V_\theta(s)\right)^2$	$-2\sum_{s\in\mathcal{S}} d_\pi(s)\left(V_\pi(s) - V_\theta(s)\right)\nabla_\theta V_\theta(s)$

Problem 1: We don't know the distribution $d_\pi(s)$.

Solution 1 we proceed sample by sample since asymptotically we will see all the states in proportion to their visitation frequency (it is called Stochastic Gradient Descent).

For each observed state s:

$$\theta_{t+1} = \theta_t + \alpha\left(V_\pi(s) - V_\theta(s)\right)\nabla_\theta V_\theta(s)$$

Problem 2: We don't know the value function $V_\pi(s)$.

Solution 2 Use Monte-Carlo or TD error:

- Monte-Carlo: $V_\pi(s) \approx G_t$
- TD error: $V_\pi(s) \approx r + \gamma V_\theta(s')$

Monte-Carlo Control using function approximation

1. Initialize θ randomly
2. Repeat
   - 2.1 Generate a trajectory $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T$, using $\pi^{\epsilon}_{Q_\theta}$
   - 2.2 For all $t = 0, \ldots, T-1$
     - 2.2.1 $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$
     - 2.2.2 $\delta = G_t - Q_\theta(s_t, a_t)$ (Monte-Carlo error)
     - 2.2.3 $\theta = \theta + \alpha \delta \nabla_\theta Q_\theta(s_t, a_t)$ (update rule, a gradient descent step)

SARSA and Q-learning using function approximation

1. Initialize θ randomly
2. $s = s_0$
3. Repeat while trajectory is not finished
   - 3.1 $a = \pi^{\epsilon}_{Q_\theta}(s)$
   - 3.2 Take action a, observe r, s'
   - 3.3 if **SARSA**.
     - 3.3.1 $\delta = r + \gamma Q_\theta(s', \pi^{\epsilon}_{Q_\theta}(s')) - Q_\theta(s, a)$
   - 3.4 if **Q-learning**.
     - 3.4.1 $\delta = r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a)$
   - 3.5 $\theta = \theta + \alpha \delta \nabla_\theta Q_\theta(s, a)$
   - 3.6 $s = s'$

Example

Update for tic-tac-toe when using linear function approximation

DQN

- DQN = Q-learning + Function approximation + Experience replay

Playing Atari with Deep Reinforcement Learning (2013)

Breakthrough paper by DeepMind by Mnih et al.

In practice

- You don't need to compute the gradient by hand, you can use automatic differentiation libraries (e.g., TensorFlow, PyTorch, JAX).
- You can use more advanced optimization algorithms than simple gradient descent (e.g., Adam, RMSprop, etc.). This is automatically handled by the libraries.
- What can be tricky is to define the features $\phi(s)$ and the architecture of the approximator. For neural networks, how many layers, how many neurons, which activation functions, etc.

Summary

- We can use function approximation to generalize from the states we have seen to the states we have not seen.
- We can use different types of approximators: linear functions, decision trees, Fourier basis, neural networks, etc.
- We define an optimization problem to find the parameters θ:
  - solving $\min_\theta L(\theta)$ means that $V_\theta(s)$ is close to $V_\pi(s)$.
  - we use gradient descent solve this optimization problem.
  - update rule: $\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta_t)$
- We use methods similar model-free methods for control to find the best parameters.