4/13/2023 Reinforcement Learning II

RL → Model-based, Model-Free

Model-based

Learn $T(s, a, s')$ — old state, action, new state

and $\text{Reward}(s, a, s')$

Learning the actual model of the world

Model-Free

Does not model the whole world

Q-Learning

Learn $\hat{Q}(s,a)$ to approximate $Q_{opt}(s,a)$

optimal rewards starting at s and taking action a

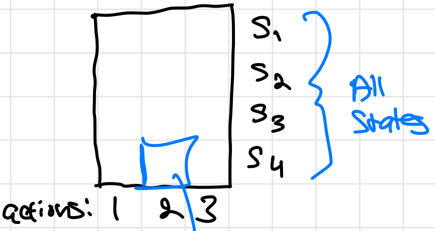
Part 1 today: Q-learning with function approximation "Regression"

Policy Gradient

Directly predict best action a in a state s

Part 2: one policy gradient algorithm "classification"

Q-Learning Recap



$\hat{Q}(s_4, 2)$ = our estimate of $Q_{opt}(s_4, 2)$

We act with policy π_{act} & upon observing (s, a, r, s'):

$$\hat{Q}(s,a) \leftarrow (1-\eta)\hat{Q}(s,a) + \eta\left(r + \gamma \hat{V}(s')\right)$$

$\eta \approx 0.1$

where $\hat{V}(s') = \max_{a \in \text{Actions}(s')} \hat{Q}(s', a)$

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \eta\,(\underbrace{r + \gamma \hat{V}(s')}_{\text{the target value}} - \underbrace{\hat{Q}(s,a)}_{\text{our prediction on input } (s,a)})$$

This looks like gradient descent/ascent

It actually is where we minimize

$$\text{Loss} = \frac{1}{2}\left(r + \gamma \hat{V}(s') - \hat{Q}(s,a)\right)^2$$

Linear regression:

$(y - w^T x)^2$

$-2 \cdot (y - w^T x) \cdot x$

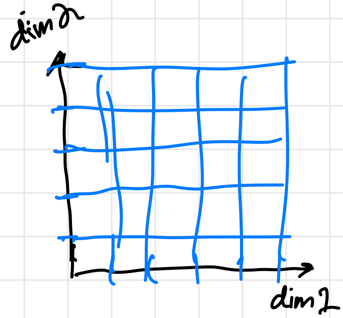
Proof: $\nabla_{\hat{Q}(s,a)} = \cancel{\frac{1}{2}} \cdot \cancel{2}\left(r + \gamma \hat{V}(s') - \hat{Q}(s,a)\right) \cdot \underline{-1}$

So: Q-Learning is just doing gradient descent on squared loss

Going beyond tabular Q-Learning

In real world, States are not discrete
- positions of joints in robot/objects

Can discretize by dividing each continuous dimension into buckets



$\#\text{States} = (\#\text{buckets})^{\text{dimensions}}$

Bad in high dimensions

Solution: Dont use a table, instead approximate $Q_{OPT}(s,a)$ with a learned model

First Attempt: Linear model
- Construct feature function $\phi(s,a) \in \mathbb{R}^d$
- Predict $\hat{Q}(s,a) = w^T \phi(s,a)$ for parameters $w \in \mathbb{R}^d$

To do Q-Learning, minimize squared error

$$\text{Loss on } (s,a,r,s') = \frac{1}{2}\left(r + \gamma \hat{V}(s') - w^T \phi(s,a)\right)^2$$

$$\nabla_w \text{loss} = \cancel{\frac{1}{2}} \cdot \cancel{2} \cdot \left(r + \gamma \hat{V}(s') - w^T \phi(s,a)\right) \cdot -\phi(s,a)$$

$$\hat{V}(s') \text{ is still } \max_{a \in Actions(s')} \underbrace{\hat{Q}(s',a)}_{= w^T \phi(s',a)}$$

Second Attempt: Neural network

Deep Q Network (DQN)

Idea: $\hat{Q}_\theta(s,a)$ will be a neural network that maps (s,a) to estimate of $Q_{opt}(s,a)$

Again, just train w/ gradient descent

$$\nabla_\theta \text{Loss} = -\underbrace{\left(r + \gamma \hat{V}(s') - \hat{Q}_\theta(s,a)\right)}_{\text{Easy to compute}} \cdot \underbrace{\nabla_\theta \hat{Q}(s,a)}_{\text{Computable by backpropagation}}$$

All parameters of network

Example DQN architecture for video games

Represent state: Last k frames

- Each frame is 84×84 (for Atari)
- Use last 4 frames $\rightarrow$ $84 \times 84 \times 4$ state representation

Feed to CNN

$u(s)$: vector encoding of the state

3 actions: 1 vector per action

up ▭ w_{up}
down ▭ w_{down}
stay ▭ w_{stay}

Predict:

$\hat{Q}(s, up) = w_{up}^T u(s)$

$\hat{Q}(s, down) = w_{down}^T u(s)$

$\hat{Q}(s, stay) = w_{stay}^T u(s)$

Policy Gradient

$$\pi_\theta(a \mid s)$$

parameters

probability distribution over actions given current state

$$\begin{array}{cl} \begin{bmatrix} .7 \\ .1 \\ .2 \end{bmatrix} & \begin{array}{l} \text{up} \\ \text{down} \\ \text{stay} \end{array} \end{array}$$

How do we train?

- Normal Classification: Given correct y's for a bunch of x's maximize $P(y \mid x)$
- Policy gradient: Nobody tells you best action in any state what training algorithm to use?

Want to maximize value of policy π_θ:

$$\underbrace{V(\theta)}_{\text{Expected total rewards when using policy } \pi_\theta} = \sum_{\substack{\text{trajectories} \\ z}} \underbrace{P(z;\theta)}_{\text{Prob. of } z \text{ happening when using } \pi_\theta} \cdot \underbrace{R(z)}_{= \sum_{t=1} r_t}$$

Reward for z

$z = [s_1, a_1, \underline{r_1}, s_2, a_2, \underline{r_2}, s_3, \ldots]$

$V(\theta)$ is our training objective
maximize with gradient ascent

What is $\nabla_\theta V(\theta)$?

$$\nabla_\theta V(\theta) = \sum_{z} \nabla P(z;\theta) \cdot R(z)$$

Sum over exponentially many z's — infeasible

Hope to instead compute an expected value over trajectories

Key trick: $\nabla_\theta \log P(z;\theta) = \frac{1}{P(z;\theta)} \nabla P(z;\theta)$

$$\nabla P(z;\theta) = P(z;\theta) \cdot \nabla \log P(z;\theta)$$

$$\nabla_\theta V(\theta) = \underbrace{\sum_z P(z;\theta)}_{\text{Expected Value of...}} \underbrace{\nabla \log P(z;\theta) \cdot R(z)}_{\text{this quantity}}$$

$$z = [s_1, a_1, r_1, s_2, a_2, r_2, s_3, \ldots]$$

$$\log P(z;\theta) = \underbrace{\log p(s_1)}_{\substack{\text{Start state}\\ \text{prob.}\\ \text{in MDP}}} + \underbrace{\log \pi_\theta(a_1 | s_1)}_{\substack{\text{Prob of}\\ \text{taking } a_1\\ \text{in}\\ \text{state } s_1}} + \underbrace{\log T(s_1, a_1, s_2)}_{\substack{\text{transition prob}\\ \text{of MDP}}}$$

$$+ \log \pi_\theta(a_2 | s_2) + \underbrace{\log T(s_2, a_2, s_3)}_{\text{Do not depend on } \theta} + \cdots$$

$$\boxed{\nabla_\theta \log P(z;\theta) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t)}$$

Final policy gradient algorithm:

Initialize θ randomly

For each episode:

Sample trajectory z using $\pi_\theta(a|s)$

$$\theta \leftarrow \theta + \eta \, R(z) \cdot \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t | s_t)$$

Gradient ascent on $V(\theta)$