RL

model-based

learn $T(s,a,s')$ and $R(s,a,s')$

learn the actual model of the world

model-free

don't model the world

Q-learning

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

optimal reward at s and take action a

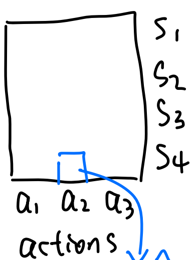
Q-learning with function approximation

policy gradient

directly predict best action a in state s

one policy gradient algorithm

Q-learning



$\hat{Q}(s_4, 2)$

= estimate of $Q_{opt}(s_4,2)$

we act with π_{act} upon observing (s,a,r,s')

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

≈ 0.1

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

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

↑ looks like gradient descent / ascent

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

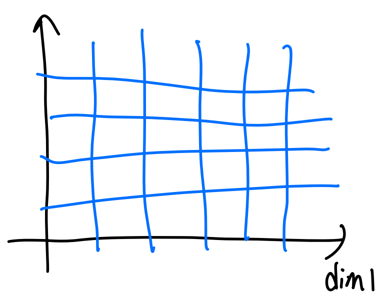
$$\nabla_{\hat{Q}(s,a)} = \frac{1}{2} \cdot 2(r + \gamma \hat{V}(s') - \hat{Q}(s,a)) \cdot -1$$

$\Rightarrow$ Q-learning is doing gradient descent on squared loss

In real world, states are not discrete.

discretize by dividing each continuous dimension into buckets.

$\#\text{ states} = (\#\text{ buckets})^{\text{dim}}$



Solution: don't use table, approximate $Q_{opt}(s,a)$ with a learned model.

first attempt linear model:

- constant feature function $\phi(s,a) \in \mathbb{R}^d$
- predict $\hat{Q}(s,a) = w^T \phi(s,a)$ for $w \in \mathbb{R}^d$

To do Q-learning, minimize squared error

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

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

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

Second attempt: neural network

Deep Q Network (DQN)

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

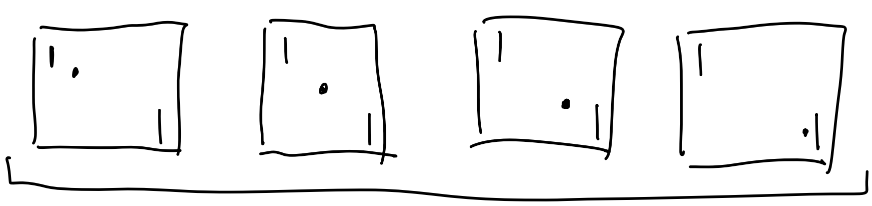
$$\nabla_\theta \text{Loss} = \underbrace{-\left(r + \gamma \hat{V}(s') - \hat{Q}_\theta(s,a)\right)}_{\text{easy to compute}} \cdot \underbrace{\nabla_\theta \hat{Q}(s,a)}_{\text{computable by backpropagation}}$$

parameters of NN

Example DQN architecture for video game

represent state: last k frames

- each frame is 84x84



feed to CNN ↓

$v(s)$: vector encoding of 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

$\pi_\theta(a|s)$ probability distribution over actions given current state

$\uparrow$ params

$$\begin{array}{cl} \begin{bmatrix} 0.1 \\ 0.2 \\ 0.7 \end{bmatrix} & \begin{array}{l} \text{up} \\ \text{down} \\ \text{stay} \end{array} \end{array}$$

How to train?

- normal classification: given correct y's for x's, maximize $p(y|x)$
- policy gradient: don't know best action in any state

Want to maximize value of policy π_θ :

$$\underbrace{V(\theta)}_{\text{expected total rewards when using } \pi_\theta} = \sum_{\substack{\text{trajectories} \\ z}} P(z;\theta) \cdot \underbrace{R(z)}_{\text{rewards for } z \,:\, \sum_{t=1}^{T} r_t}$$

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

$V(\theta)$ is for training,

maximize with gradient descent

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 compute expected value over trajectories

$$\nabla_\theta \log p(z;\theta) = \frac{1}{p(z;\theta)} \nabla p(z;\theta)$$

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

$$\nabla_\theta V(\theta) = \underbrace{\sum_z p(z;\theta)}_{\text{expected value}} \underbrace{\nabla \log p(z;\theta) \cdot R(z)}_{\text{of this quantity}}$$

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

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

$$+ \log \pi_\theta(a_2|s_2) + \underbrace{\log T(s_2, a_2, s_3)}_{\text{independent of } \theta}$$

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

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 $V(\theta)$