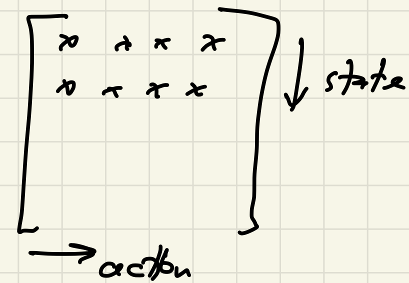
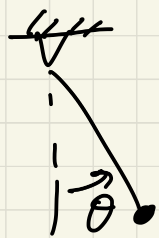
Q-learning with function approximation

We have been doing tabular methods

← table of numbers (= array)

$Q(s,a)\ \ \forall s,\ \forall a \longrightarrow$





discretize →

tabular methods cannot capture the sense of "nearby" states

→ unable to generalize

$$Q(s,a) \longrightarrow Q(s,a;\theta)$$

parameters

e.g. basis functions: $f_1(s,a),\ f_2(s,a),\ \ldots,\ f_n(s,a)$

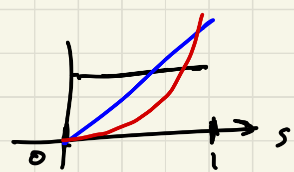
$n \ll$ number of states

"much less than"

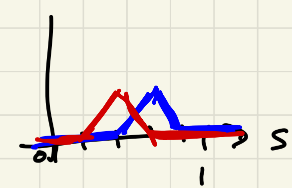
$$Q(s,a;\theta) = \sum_i \theta_i f_i(s,a)$$

$$f_1(s,a) = 1$$
$$f_2(s,a) = s$$
$$f_3(s,a) = a$$
$$f_4(s,a) = s^2$$
$$f_5(s,a) = a^2$$
$$f_6(s,a) = sa$$
$$\vdots$$

global basis functions:



local basis:



e.g. in practice, use a deep neural network (DNN)

$Q(s,a;\theta)$ weights in a DNN

$Q_\theta(a,b)$

$p(s_{t+1} \mid s_t, a_t)$

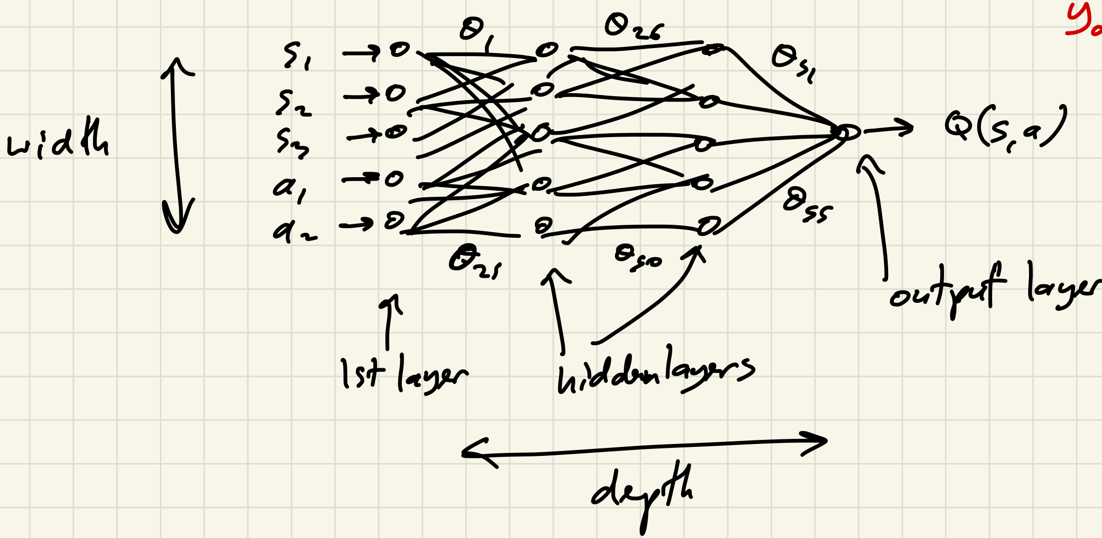
$s=(s_1,s_2,s_3)$ $\quad a=(a_1,a_2)$

$s\in\mathbb{R}^3$ $\quad a\in\mathbb{R}^2$

$y=ax+b$

$y(x;a,b)$

$y_{a,b}(x)$ parameters



e.g. tabular Q

$$Q = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 \\ \theta_5 & \cdot & \cdot & - \\ - & - & - & \\ & & & \end{bmatrix}$$

parameters are entries in Q matrix

tabular update rule :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

transform to a optimization problem

loss function $\quad L(\theta) = \frac{1}{2}\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_{old}) - Q(s_t, a_t; \theta)\right)^2$

old value

gradient descent :

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta_{old})$$

$$\frac{\partial L}{\partial \theta}$$

$\theta_k \rightarrow \theta_{k+1}$

$$L(\theta) = \frac{1}{2}\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_k) - Q(s_t, a_t; \theta)\right)^2$$

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k)$$

$$\theta_k \rightarrow \theta_{k+1} \qquad L(\theta) = \frac{1}{2}\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_k) - Q(s_t, a_t; \theta)\right)^2$$

scalar

vector

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k)$$

k is iteration number

$$\nabla_\theta L(\theta) = \frac{\partial L}{\partial \theta}^T(\theta) = \begin{bmatrix} \partial L/\partial\theta_1 \\ \partial L/\partial\theta_2 \\ \vdots \\ \partial L/\partial\theta_n \end{bmatrix}$$

$$\frac{\partial L}{\partial \theta_i}(\theta) = 2 \cdot \frac{1}{2}\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_k) - Q(s_t, a_t; \theta)\right)\left(-\frac{\partial Q}{\partial \theta_i}(s_t, a_t; \theta)\right)$$

vector index

$$= -\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_k) - Q(s_t, a_t; \theta)\right) \frac{\partial Q}{\partial \theta_i}(s_t, a_t; \theta)$$

$$\boxed{\theta_{k+1} = \theta_k + \alpha\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_k) - Q(s_t, a_t; \theta_k)\right) \nabla_\theta Q(s_t, a_t; \theta_k)}$$

Compare tabular:

special case: tabular

$$Q = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 \\ \theta_5 & \cdots & - & - \\ - & - & - & \end{bmatrix}$$

$$\frac{\partial Q_i}{\partial \theta} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{ith}$$

e_i

$$Q_{k+1} = Q_k + \alpha(\_\_\_)e_i$$

only update i-th entry of Q

i is the index for (s_t, a_t)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(\_\_\_)$$

$$Q_i \leftarrow Q_i + \alpha(\_\_\_)$$

$\longrightarrow$ are back at exactly tabular Q-learning.

Intuition

$$L(\theta) = \frac{1}{2}\Big(\underbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_T)}_{\text{target}} - Q(s_t, a_t; \theta)\Big)^2$$

find θ that minimizes $L(\theta)$

$\rightarrow$ choose θ to make

$Q(s_t, a_t; \theta)$ close to target

$$L(x) = \frac{1}{2}(3 - x)^2$$

pick $x = 3$ to minimize

Gradient descent to minimize $L(\theta)$, $\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k)$

Stochastic gradient descent (for supervised learning)

$$L(\theta) = \sum_j L_j(\theta) = \sum_j \| y_j - f(x_j;\theta) \|^2$$

where (x_j, y_j) is training data

and we want to fit

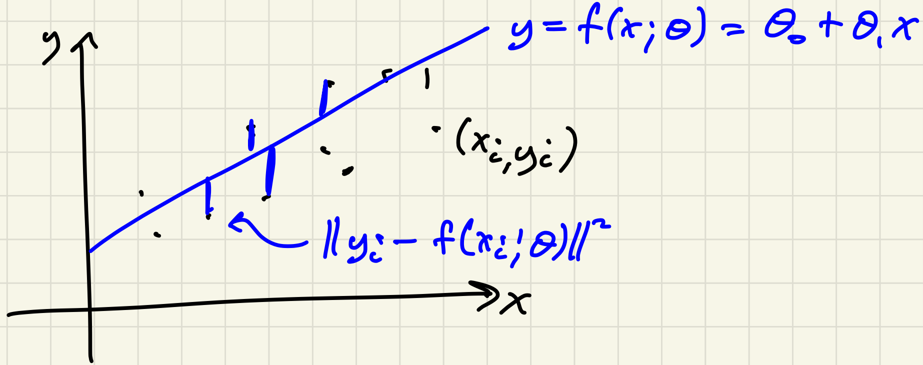
$y = f(x;\theta)$ $(= \theta_0 + \theta_1 x)$

grad. descent: $\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k)$

$$= \theta_k - \alpha \sum_j \nabla_\theta L_j(\theta_k)$$

stoc. grad. desc. (SGD): $\theta_{k+1} = \theta_k - \alpha \nabla_\theta L_j(\theta_k)$

j is chosen randomly at each step



RL samples $s_t, s_{t+1}, s_{t+2}, \ldots$ are highly correlated because the come from a trajectory

Instead, we want to include many diverse sample values

→ "experience replay"

Accumulate a history of states we have visited, constantly resample from this history.

DQN algorithm (Mnih 2015)

Simulate $t = 0, \ldots$

take ε-greedy action a_t

sytem simulator $\longrightarrow r_{t+1}, s_{t+1}$

store $(s_t, a_t, r_{t+1}, s_{t+1}) \longrightarrow$ history D ← experience replay buffer

choose m random history states from $D \longrightarrow (s_j, a_j, r_j, s_j')$

$$\delta = \frac{1}{m} \sum_{j=1}^{m} \left(r_j + \gamma \max_a Q_\theta(s_j', a) - Q_\theta(s_j, a_j) \right) \nabla_\theta Q_\theta(s_j, a_j)$$

$$\theta \leftarrow \theta - \alpha \delta$$

note: history samples were generated from very old policies
$\longrightarrow$ off policy, so need Q-learning because it's off policy