

## Value function approximation

So far, we considered:

- finite set of states  $s : \mathcal{S}$
- finite set of actions  $a : \mathcal{A}$

We use tables to evaluate  $V(s)$ ,  $Q(s, a)$

$$V = \begin{bmatrix} v(s_0) \\ \vdots \\ v(s_5) \end{bmatrix} \quad Q = \begin{bmatrix} q(s=0, a=0) & \dots & q(s=0, a=A) \\ \vdots & & \vdots \\ q(s=5, a=0) & \dots & \end{bmatrix}$$

This is not possible for large problem  
( $G_0 : 10^{170}$  states).

This is not possible also if the set of  
states  $\mathcal{S}$  or actions  $\mathcal{A}$  is continuous.

## Solution

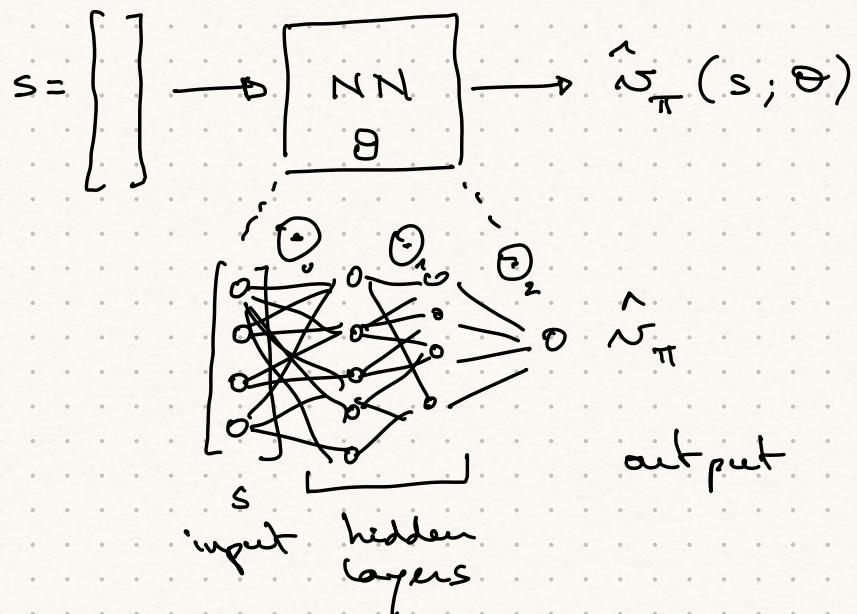
Estimate the value function with  
an approximation

$$\hat{v}_\pi(s; \theta) \approx v_\pi(s) \quad \text{parameters (weights for XIN)}$$

$$\hat{q}_\pi(s, a; \theta) \approx q_\pi(s, a)$$

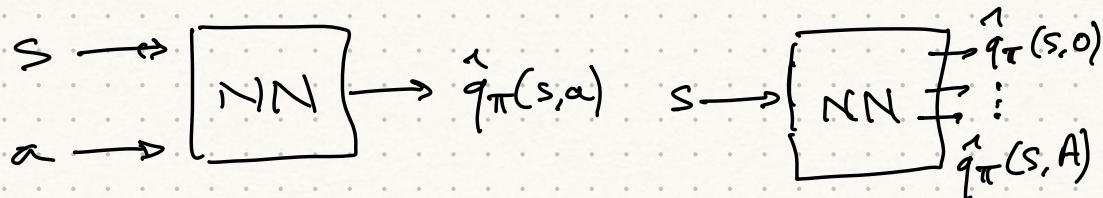
- Reason:
- Using NN allows to generalize  
(NN learn smooth functions)
  - We can use the same methods  
to learn  $\theta$  as the methods to learn the  
table (DP, MC, TD) ...

### Examples



### Design choices:

- Architecture of NN (fully connected, CNN)
- Number of hidden layers
- Number of neurons in hidden layers
- Nonlinear function



## Reur: Why NN?

- Approximate continuous functions (generally)
- Efficient way to calculate  $\nabla_{\theta} \hat{\pi}$  (backpropagation)
- It can approach any continuous function provided there are enough hidden neurons
- GPU compatible

## Function approximation in practice

Let's assume we want to learn  $\hat{\pi}_{\pi}(s)$  knowing  $\pi$  (prediction problem).

In supervised learning:

- we want to minimize the cost:

$$J(\theta) = \frac{1}{2} \left( \underbrace{\hat{\pi}_{\pi}(s)}_{\text{we know it}} - \underbrace{\hat{\pi}_{\pi}(s; \theta)}_{\substack{\text{output} \\ y}} \right)^2$$

- we then use gradient descent (iterative process)

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J$$

$$\theta \leftarrow \theta + \alpha \left( \hat{\pi}_{\pi}(s) - \hat{\pi}_{\pi}(s; \theta) \right) \nabla_{\theta} \hat{\pi}_{\pi}(s; \theta)$$

learning rate       $\uparrow$  error       $\downarrow$  output       $\uparrow$  calculated with backpropag.

In reinforcement learning, we replace the target by:

- $G_t$  : Monte Carlo approach (MC)
- $R_{t+1} + \gamma \hat{q}(s_{t+1}; \theta)$  : Temporal diff. (TD)

### Example

Q-learning with function approximation

$$\hat{q}(s, a; \theta) \approx q(s, a)$$

$$\theta \leftarrow \theta + \alpha \underbrace{(r + \gamma \max_{a'} \hat{q}(s', a') - \hat{q}(s, a))}_{\text{target for Q-learning}} \nabla \hat{q}(s, a)$$

We need  $\{s, a, r, s'\}$

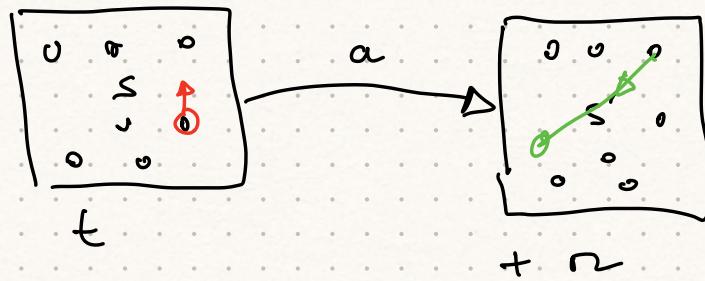
### Batch methods / experience replay

- "stochastic" gradient descent: you use experiences  $\{s, a, r, s'\}$  in the order they are experienced.
- Mini-batch learning:
  - Store experiences  $\{s, a, r, s'\}$  in a buffer
  - Sample a mini-batch of experiences for learning

used for policy  $\pi = \arg \max q$

$$\theta \leftarrow \theta + \alpha \left( r + \gamma \max_{a'} \hat{q}_\theta(s', a') - \hat{q}_\theta(s, a) \right) \nabla_\theta \hat{q}_\theta(s, a)$$

old value of  $\theta$  / current value of  $\theta$



## Policy gradient

Policy-based algorithm

vs. value-based algorithm (SARSA,  
Q-learning, ...).

the goal is to learn a stochastic policy

$$\pi(a|s) \approx \hat{\pi}(s, a; \theta)$$

Advantage:

- learn a stochastic policy (this is  
needed if state is only partially observable;  
e.g., poker)

## Score function

$$\nabla_{\theta} \hat{\pi} = \hat{\pi} \frac{\nabla_{\theta} \hat{\pi}}{\hat{\pi}} = \hat{\pi} \underbrace{\nabla_{\theta} \log \hat{\pi}}_{\text{score function}}$$

## Examples (not NN)

One continuous action to choose  
the policy is a soft-max of a  
linear combination of features.

$$s = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} \rightarrow \phi(s, a) = \begin{bmatrix} \phi_0 = 1 \\ \phi_1(s, a) \\ \vdots \\ \phi_n(s, a) \end{bmatrix} \rightarrow \underbrace{\phi(s, a)^T \cdot \theta}_{\text{one scalar product for each action}}$$

$$\phi^T(s, a) \cdot \theta \longrightarrow \hat{\pi}(s, a; \theta) = \frac{\exp[\phi^T(s, a) \cdot \theta]}{\sum_{a'} \exp[\phi^T(s, a') \cdot \theta]}$$

*soft-max*

Score function:

$$\begin{aligned} \nabla_\theta \log \hat{\pi} &= \nabla_\theta (\phi^T \cdot \theta) - \nabla_\theta \log \sum_{a'} e^{\phi^T \cdot \theta} \\ &= \phi - \frac{\sum_{a'} \nabla_\theta e^{\phi^T \cdot \theta}}{\sum_{a'} e^{\phi^T \cdot \theta}} \\ &= \phi - \frac{\sum_{a'} \phi(s, a') e^{\phi^T \cdot \theta}}{\sum_{a'} e^{\phi^T \cdot \theta}} \quad \hat{\pi}(s, a') \\ &= \phi - \mathbb{E}_{\hat{\pi}}(\phi(s, \cdot)) \end{aligned}$$

Example 2

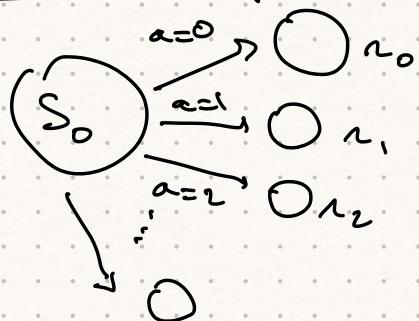
$$\hat{\pi}(s, a) = \frac{1}{Z} \exp \left[ - \frac{(a - \mu)^2}{2 \sigma^2} \right]$$

$\tau$  is fixed

$\mu = \phi^T(s) \cdot \theta$  (we want to learn the average action from state  $s$ , with action continuous).

Score function:  $\nabla_\theta \log \hat{\pi} = \frac{a - \mu}{\sigma^2} \nabla_\theta \mu = \frac{a - \mu}{\sigma^2} \phi$

## One-step NDP



We want to maximize  
the expected reward  
 $J(\theta)$

$$J(\theta) = \mathbb{E}_{\hat{\pi}_\theta} (r) = \sum_a \hat{\pi}(s_0, a) \underbrace{r_a}_{Q_{s_0}^a}$$

We look for  $\max_\theta J(\theta)$

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_a \nabla_\theta \hat{\pi}(s_0, a) r_a \\ &= \sum_a \hat{\pi}(s_0, a) \nabla_\theta \log \hat{\pi}(s_0, a) r_a \\ &= \mathbb{E}_{\hat{\pi}_\theta} \left[ \underbrace{\nabla_\theta \log \hat{\pi}}_{\text{score function}} \underbrace{r}_\text{reward} \right] \end{aligned}$$

In general

$$\nabla_\theta J(\theta) = \mathbb{E}_{\hat{\pi}_\theta} \left[ \nabla_\theta \log \hat{\pi} \underbrace{\hat{q}_{\hat{\pi}_\theta}(s, a)}_{\text{action value}} \right]$$

can be replaced  
by an approximation  
of  $\mathbb{E}(G_t | s, a)$ .

## REINFORCE (PG policy gradient)

- Compute a complete episode with policy  $\hat{\pi}$
  - Compute for each  $(S_t, A_t)$ :  $G_t$
  - For each  $(S_t, A_t)$ , update the weights  $\Theta$  of policy approximate
- $$\Theta \leftarrow \Theta + \alpha \left[ \nabla_{\Theta} \log \hat{\pi}(S_t, A_t; \Theta) G_t \right]$$

## ACTOR-CRITIC algorithm

Approximation for  $q_{\pi}(s, a)$

$$\tilde{q}(s, a; \tilde{\Theta}) \quad (\text{CRITIC})$$

Approximation for  $\pi(s | a)$

$$\hat{\pi}(s, a; \hat{\Theta}) \quad (\text{ACTOR})$$

TD algorithm for  $\tilde{q}$  (SARSA)

$$\tilde{\Theta} \leftarrow \tilde{\Theta} + \tilde{\alpha} \left( r + \tilde{\delta} \tilde{q}(s', a'; \tilde{\Theta}) - \tilde{q}(s, a; \tilde{\Theta}) \right) \nabla_{\tilde{\Theta}} \tilde{q}$$

Policy-gradient for  $\hat{\pi}$

$$\hat{\Theta} \leftarrow \hat{\Theta} + \hat{\alpha} \nabla_{\hat{\Theta}} \log \hat{\pi}(s, a; \hat{\Theta}) \tilde{q}(s, a; \tilde{\Theta})$$

$\hat{\alpha} < \tilde{\alpha}$  (critic needs to learn faster than the actor)

## Advantage actor-critic algorithm (A2C)

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\hat{\pi}} \left[ \nabla_{\theta} \log \hat{\pi} \mid Q_{\hat{\pi}}(s, a) \right]$$

$$= \mathbb{E}_{\hat{\pi}} \left[ \nabla_{\theta} \log \hat{\pi} \left[ \underbrace{Q_{\hat{\pi}}(s, a) - V_{\hat{\pi}}(s)}_{A_{\hat{\pi}}(s, a)} \right] \right]$$

$A_{\hat{\pi}}(s, a)$ : advantage function

Two approximations

$$\tilde{v}(s; \tilde{\theta}) \approx v_{\pi}(s)$$

$$\hat{\pi}(s, a; \hat{\theta}) \approx \pi(s|a)$$

Temporal difference algorithm for  $\tilde{v}$

$$\tilde{\theta} \leftarrow \tilde{\theta} + \alpha \underbrace{\left( r + \gamma \tilde{v}(s') - \tilde{v}(s) \right)}_{\delta(s, r, s')} \nabla_{\tilde{\theta}} \tilde{v}$$

$\delta(s, r, s') \approx A_{\hat{\pi}}(s, a)$

Policy-gradient for  $\hat{\pi}$

$$\hat{\theta} \leftarrow \hat{\theta} + \hat{\alpha} \left( \nabla_{\hat{\theta}} \log \hat{\pi} \mid \delta \right)$$

→ A3C : asynchronous advantage actor-critic

→ ACER : Actor-critic w/ experience replay