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 $V(s)$, $Q(s,a)$

$$V = \begin{bmatrix} v(s=0) \\ \vdots \\ v(s=S) \end{bmatrix} \qquad Q = \begin{bmatrix} q(s=0, a=0) & \cdots & q(s=0, a=A) \\ \vdots & & \\ q(s=S, a=0) & \cdots & \end{bmatrix}$$

This is not possible for large problem (Go: 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

$$\hat{v}_\pi(s;\theta) \simeq v_\pi(s)$$

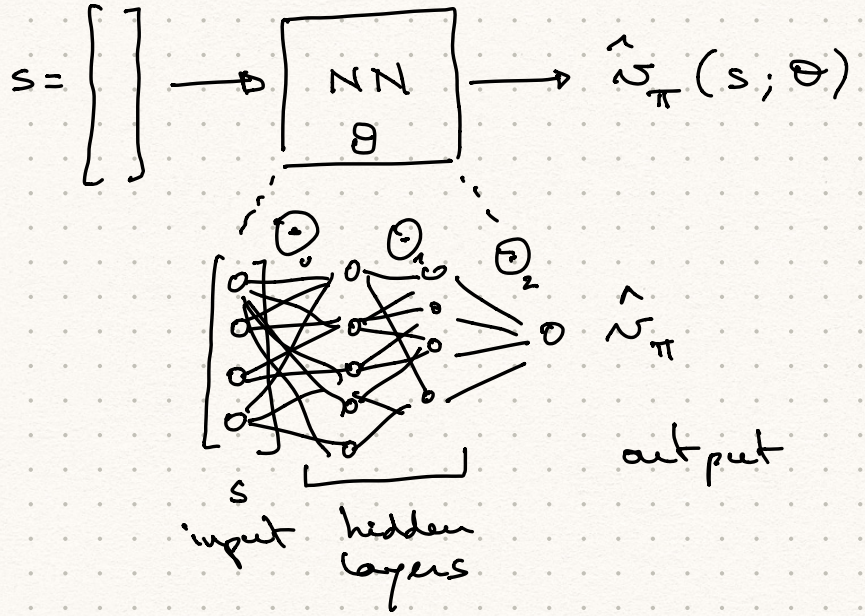
$$\hat{q}_\pi(s,a;\theta) \simeq q_\pi(s,a)$$

parameters (weights for NN)

Rem: • Using NN allows to generalize (NN learn smooth functions)

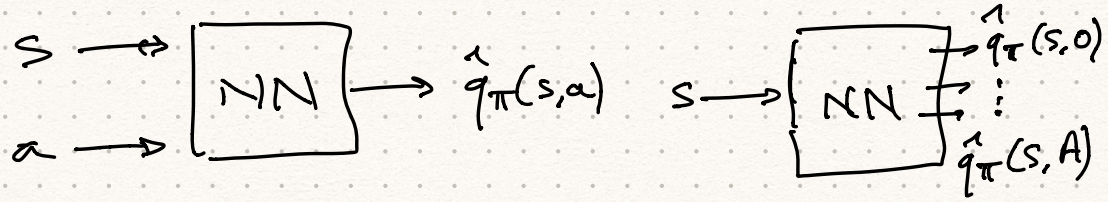
• We can use the same methods to learn θ as the methods to learn the table (DP, MC, TD) ...

Examples



Design choices:

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



Rem: Why NN?

- Approximate continuous functions (generally)
- Efficient way to calculate $\nabla_\theta \hat{v}_\pi$ (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 $v_\pi(s)$ knowing π (prediction problem).

In supervised learning:

- we want to minimize the cost:

$$J(\theta) = \frac{1}{2}\left(v_\pi(s) - \hat{v}_\pi(s;\theta)\right)^2$$

$v_\pi(s)$: we know it; $\hat{v}_\pi(s;\theta)$: output; y

- we then use gradient descent (iterative process)

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

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

α: learning rate; $v_\pi(s)$: target; $\hat{v}_\pi(s;\theta)$: output; $v_\pi(s) - \hat{v}_\pi(s;\theta)$: error; $\nabla_\theta \hat{v}_\pi(s;\theta)$: calculated with backpropagation

In reinforcement learning, we replace the target by:

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

Example

Q-learning with function approximation

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

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

target for Q-learning

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

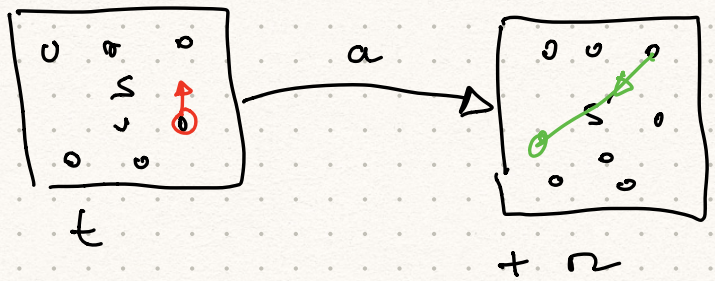
Batch methods / experience replay

- "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 (used for policy $\pi = \text{argmax}_a q$)
  - Sample a mini-batch of experiences for learning

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

old value of θ

current value of θ



Policy gradient

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

the goal is to learn a stochastic policy
$\pi(a|s) \simeq \hat{\pi}(s,a;\theta)$

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.

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

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

soft-max

Score function:

$$\nabla_\theta \log\hat{\pi} = \nabla_\theta(\phi^T\cdot\theta) - \nabla_\theta \log\sum_{a'} e^{\phi^T\cdot\theta}$$

$$= \phi - \frac{\sum_{a'}\nabla_\theta e^{\phi^T\cdot\theta}}{\sum_{a'} e^{\phi^T\cdot\theta}}$$

$$= \phi - \frac{\sum_{a'}\phi(s,a')e^{\phi^T\cdot\theta}}{\sum_{a'} e^{\phi^T\cdot\theta}} \qquad \hat{\pi}(s,a')$$

$$= \phi - \mathbb{E}_{\hat{\pi}}(\phi(s,\cdot))$$

Example 2

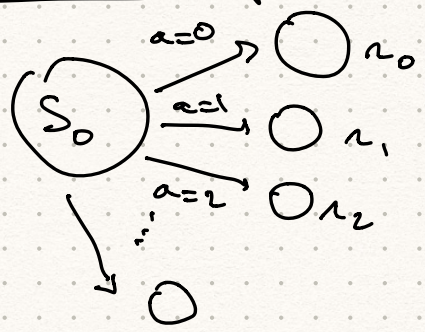
$$\hat{\pi}(s,a) = \frac{1}{Z}\exp\left[-\frac{(a-\mu)^2}{2\sigma^2}\right]$$

σ 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 MDP



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

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

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

$$\nabla_\theta J(\theta) = \sum_a \nabla_\theta \hat{\pi}(s_0, a)\, r_a$$

$$= \sum_a \hat{\pi}(s_0, a)\, \nabla_\theta \log \hat{\pi}(s_0, a)\, r_a$$

$$= \mathbb{E}_{\hat{\pi}_\theta}\left[\underbrace{\nabla_\theta \log \hat{\pi}}_{\text{score function}}\ \underbrace{r}_{\text{reward}}\right]$$

In general

$$\nabla_\theta J(\theta) = \mathbb{E}_{\hat{\pi}_\theta}\left[\nabla_\theta \log \hat{\pi}\ \underbrace{\hat{q}_{\hat{\pi}_\theta}(s,a)}_{\text{can be replaced by an approximation of } \mathbb{E}(G_t \mid s,a).}\right]$$

REINFORCE (MC policy gradient)

- Compute a complete episode with policy $\hat{\pi}$
- Compute for each (S_t, A_t): G_t
- To each (S_t, A_t), update the weights θ 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)

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

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)

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

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

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

Two approximations

$$\tilde{V}(s;\tilde{\theta}) \simeq V_\pi(s)$$

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

Temporal difference algorithm for $\tilde{V}$

$$\tilde{\theta} \leftarrow \tilde{\theta} + \tilde{\alpha}\left(r + \gamma \tilde{V}(s') - \tilde{V}(s)\right)\nabla_{\tilde{\theta}} \tilde{V}$$

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

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

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

→ A3C : asynchronous advantage actor-critic

→ ACER : Actor-critic w/ experience replay