Approximate Solution Methods.

In this lecture...

Value-function approximation

Gradient methods

Linear methods

Least-Squares TD

Policy gradient methods

REINFORCE

Value-function approximation

- Representing value function as a table is not possible for large state spaces or continuous state spaces
- In this case value function can be a parameterised function with weight vector $\boldsymbol{\theta} \in \mathbb{R}^n$:

$$V_\pi(s) \approx \hat{V}_\pi(s, \boldsymbol{\theta})$$

- The number of components of $\boldsymbol{\theta}$ is much less than the number of states ($n \ll |\mathcal{S}|$), and changing one weight changes the estimated value of many states.
- When a single state is updated, the change generalises from that state to affect the values of many other states.

Back-ups as input-output pairs

Value function estimation can be described as a series of back-ups:

Monte Carlo back-up $s_t \mapsto R_t$

TD back-up $s_t \mapsto r_t + \gamma \hat{V}(s_{t+1}, \boldsymbol{\theta})$

DP back-up $s \mapsto E_\pi[r_t + \gamma \hat{V}(s_{t+1}, \boldsymbol{\theta}) \mid s_t = s]$

Each can be seen as an example of the desired input-output behaviour of the value function.

This means that we can apply function approximation **but** only such that allows data to be obtained sequentially.

Prediction Objective

In the case of approximation it is not possible to get the prediction in all states correct. Therefore, we produce a distribution over states which specifies how much we care about the error in each particular state $d(s)$.
The objective function is then **Mean Squared Value Error**:

$$MSVE(\boldsymbol{\theta}) = \sum_{s} d(s) \left(V_\pi(s) - \hat{V}(s, \boldsymbol{\theta}) \right)^2$$

Typically one chooses $d(s)$ to be the fraction of time spent in s under the target policy π - *occupancy frequency*.

On-policy distribution

- $h(s)$ denotes the probability that an episode begins in state s
- $e(s)$ denotes the average time steps spent in state s in a single episode.

$$e(s) = h(s) \overset{?}{+} \sum_{\hat{s}} e(\hat{s}) \sum_{\hat{a}} \pi(\hat{a} \mid \hat{s}) p(s \mid \hat{s}, \hat{a})$$

This system of equations can be solved for the expected number of visits $e(s)$ yielding the distribution:

$$d(s) = \frac{e(s)}{\sum_{s'} e(s')}$$

Stochastic gradient descent

$$MSVE(\boldsymbol{\theta}) = \sum_s d(s)\left(V_\pi(s) - \hat{V}(s,\boldsymbol{\theta})\right)^2$$

$$e(s) = h(s) + \sum_{\hat{s}} e(\hat{s}) \sum_{\hat{a}} \pi(\hat{a} \mid \hat{s}) p(s \mid \hat{s}, \hat{a})$$

$$d(s) = \frac{e(s)}{\sum_{s'} e(s')}$$

- $\hat{V}(s,\boldsymbol{\theta})$ is differentiable wrt $\boldsymbol{\theta} = (\theta_1, \theta_2, \cdots, \theta_n)^\mathsf{T}$.
- $\boldsymbol{\theta}_t$ is updated at each of a series of discrete time steps, $t = 0, 1, 2, 3, \ldots$.
- A sample $s_t \mapsto V_\pi(s_t)$ consists of a (possibly random) state s_t and its true value under the policy π. We assume that states appear in examples with the same distribution, $d(s)$, over which we are trying to minimize the $MSVE$:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \frac{1}{2}\alpha\nabla_{\boldsymbol{\theta}}\left(V_\pi(s_t) - \hat{V}(s_t, \boldsymbol{\theta}_t)\right)^2 \\ &= \boldsymbol{\theta}_t + \alpha\left(V_\pi(s_t) - \hat{V}(s_t, \boldsymbol{\theta}_t)\right)\nabla\hat{V}(s_t, \boldsymbol{\theta}_t),\end{aligned}$$

  $\alpha > 0$ is a step-size parameter.
- It's called stochastic because the update is done on only a single example, which has been selected stochastically.

Target output

- In practise true value $V_\pi(s_t)$ is not available during learning.
- Instead, we have $s_t \mapsto U_t$ where U_t is a noisy estimate of $V_\pi(s_t)$. The general SGD method for state-value prediction is:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(U_t - \hat{V}(s_t, \boldsymbol{\theta}_t) \right) \nabla \hat{V}(s_t, \boldsymbol{\theta}_t)$$

- If Ut is an unbiased estimate $(E[U_t] = V_\pi(s_t))$ for each t, then $\boldsymbol{\theta}_t$ is guaranteed to converge to a local optimum for decreasing α.

What's U_t ?

Prediction with function approximation

Algorithm 1 Gradient Monte Carlo Algorithm for Approximating $\hat{V} \approx V_\pi$

```
Input: the policy $\pi$ to be evaluated
Input: a differentiable function $\hat{V}(s, \boldsymbol{\theta}) : \mathcal{S} \times \mathbb{R}^n \to \mathbb{R}$
Initialise $\boldsymbol{\theta}_0$
repeat
    Generate an episode $s_0, a_0, r_1, \cdots r_T, s_T$ using $\pi$
    for $t = 0, 1, \cdots T$ do
        $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left( \underset{V_\pi(S_t)}{R_t} - \hat{V}(s_t, \boldsymbol{\theta}_t) \right) \nabla \hat{V}(s_t, \boldsymbol{\theta}_t)$
    end for
until convergence
```

Semi-gradient methods

- If instead of MC, we are using TD or DP updates for prediction using SGD, ie we perform bootstrapping, they all depend on the current value of the weight vector $\boldsymbol{\theta}_t$
- This implies that they will be biased and that they will not produce a true gradient-descent method.
- They include only a part of the gradient and are called *semi-gradient methods*.

Linear methods

One of the most important special cases of function approximation is that in which the approximate function, $\hat{V}(s, \boldsymbol{\theta})$, is a linear function of the weight vector, $\boldsymbol{\theta}$: $\hat{V}(s\theta) = \theta^{T} \cdot \phi(s)$

$$\hat{V}(s, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathsf{T}} \cdot \boldsymbol{\phi}(s)$$
$$= \sum_{i} \theta_i \phi_i(s)$$

$\boldsymbol{\phi} = (\phi_1, \phi_2, \cdots, \phi_n)^{\mathsf{T}}$, $\phi_i(s) : \mathcal{S} \to \mathbb{R}$ are feature functions.

Linear approximation

It is natural to use stochastic gradient descent updates with linear function approximation. The gradient of the approximate value function with respect to $\boldsymbol{\theta}$ in this case is:

$$\nabla \hat{V}(s, \boldsymbol{\theta}) = \phi(s)$$

In linear case there is only one optimum.

Semi-gradient TD update

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha\left(r_{t+1} + \gamma\boldsymbol{\theta}_t^\mathsf{T}\boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\mathsf{T}\boldsymbol{\phi}_t\right)\boldsymbol{\phi}_t \\ &= \boldsymbol{\theta}_t + \alpha\left(r_{t+1}\boldsymbol{\phi}_t - \boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\mathsf{T}\boldsymbol{\theta}_t\right),\end{aligned}$$

where $\boldsymbol{\phi}_t = \phi(s_t)$. Once the system has reached steady state, for any given $\boldsymbol{\theta}_t$, the expected next weight vector is:

$$E[\boldsymbol{\theta}_{t+1} \mid \boldsymbol{\theta}_t] = \boldsymbol{\theta}_t + \alpha(\boldsymbol{b} - A\boldsymbol{\theta}_t),$$

where

$$\begin{aligned}\boldsymbol{b} &= E[r_{t+1}\boldsymbol{\phi}_t] \\ A &= E[\boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma^\mathsf{T}\boldsymbol{\phi}_{t+1})]\end{aligned}$$

If the system converges to $\boldsymbol{\theta}$, then $\boldsymbol{b} - A\boldsymbol{\theta} = 0$

Least-Squares TD

TD with linear function approximation converges asymptotically, for appropriately decreasing step sizes, to the TD fixpoint:

$$\boldsymbol{\theta} = A^{-1}\boldsymbol{b}$$

$$A = E[\phi_{\boldsymbol{t}}(\phi_t - \gamma\phi_{t+1})^{\mathsf{T}}]$$

$$\boldsymbol{b} = E[R_{t+1}\phi_t]$$

If this is so, then we don't need to compute the solution iteratively. Instead, we can calculate A and $\boldsymbol{b}$ separately and then find the fixpoint.

Least-Squares TD prediction

Algorithm 2 Least-Squares TD

```
Input: policy $\pi$, features $\phi(s) \in \mathbb{R}^n, \phi(\mathit{terminal}) = 0$
Initialise $\widehat{A^{-1}} = \epsilon^{-1}\boldsymbol{I}$, $\hat{b} = \mathbf{0}$
repeat
    for each episode do
        Initialise $s$ and obtain $\boldsymbol{\phi}$
        for each step do
            Choose $a \sim \pi(\cdot \mid s)$, take $a$, observe $r, s'$, obtain $\boldsymbol{\phi}'$
            $\boldsymbol{v} = \widehat{A^{-1}}(\boldsymbol{\phi} - \gamma\boldsymbol{\phi}')$
            $\widehat{A^{-1}} = \widehat{A^{-1}} - (\widehat{A^{-1}}\phi)\boldsymbol{v}^\mathsf{T}/(1 + \boldsymbol{v}^\mathsf{T}\phi)$
            $\hat{b} \leftarrow \hat{b} + r\boldsymbol{\phi}$
            $\boldsymbol{\theta} = A^{-1}\boldsymbol{b}$
            $s \leftarrow s', \boldsymbol{\phi} \leftarrow \phi'$
        end for
    end for
until convergence
```

How to update $\phi(s)$ $\forall s$?

Properties of LSTD

- Complexity is $O(n^2)$ vs $O(n)$ for semi-gradient TD
- No step size parameter is required
- ϵ-greedy policy is used in the policy improvement step
- This requires setting ϵ: if ϵ is too small the sequence of inverses can vary wildly, and if ϵ is too large then learning is slowed
- It never forgets which is problematic if the target policy changes as it does in reinforcement learning and generalised policy iteration.

Summary

- Reinforcement learning systems must be capable of generalization if they are to be applicable to artificial intelligence or to large engineering applications.
- In parameterised function approximation the value function is parameterised by a weight vector $\boldsymbol{\theta}$
- To find a good weight vector we use a variation of stochastic gradient descent
- Good results can be obtained for semi-gradient methods in the special case of linear function approximation, in which the value estimates are weighted sum of features.
- LSTD is the most data-efficient linear TD prediction method, but has computational complexity $O(n^2)$ for n features

Policy gradient methods

- Policy gradient methods learn a parametrised policy that can select actions without needing to compute a value function
- Policy π is parametrised with $\boldsymbol{\omega} \in \mathbb{R}^n$

$$\pi(a \mid s, \boldsymbol{\omega}) = p(a_t = a \mid s_t = s, \boldsymbol{\omega}_t = \boldsymbol{\omega})$$

- Given a performance measure $J(\boldsymbol{\omega})$ the gradient is

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t + \alpha \nabla J(\boldsymbol{\omega}_{\boldsymbol{t}})$$

- $J(\boldsymbol{\omega})$ is typically the value of the initial state $V_{\pi(\boldsymbol{\omega})}(s_0)$,

Policy approximation

$\psi(s,a)$

- Stochastic policy
- Approximation method such that gradient $\nabla_{\boldsymbol{\omega}} \pi(a|s, \boldsymbol{\omega})$ exists and is finite
- We often use a *Gibbs policy*:

$$\pi(a|s, \boldsymbol{\omega}) = \frac{\exp(\boldsymbol{\omega}^{\mathsf{T}} \boldsymbol{\psi}(s, a))}{\sum_{a'} \exp(\boldsymbol{\omega}^{\mathsf{T}} \boldsymbol{\psi}(s, a'))}$$

where $\boldsymbol{\psi}$ denotes parametrised the feature functions.

Policy gradient theorem

$$\nabla J(\boldsymbol{\omega}) = \sum_{s} d_{\pi}(s) \sum_{a} Q_{\pi}(s, a) \nabla_{\boldsymbol{\omega}} \pi(a \mid s, \boldsymbol{\omega})$$

PROOF

REINFORCE

- The policy gradient theorem gives us an exact expression for the gradient; all we need is some way of sampling whose expectation equals or approximates this expression.
- Notice that the right-hand side is a sum over states weighted by how often the states occurs under the target policy π weighted again by γ times how many steps it takes to get to those states.
- If we just follow π we will encounter states in these proportions, which we can then weight by γ to preserve the expected value.

REINFORCE

$$\begin{aligned}\nabla J(\boldsymbol{\omega}) &= E_\pi\left[\gamma^t R_t \frac{\nabla_{\boldsymbol{\omega}}\pi(a|s,\boldsymbol{\omega})}{\pi(a|s_t,\boldsymbol{\omega})}\right]\\ &= E_\pi\left[\gamma^t R_t \nabla_{\boldsymbol{\omega}} \log \pi(a|s_t,\boldsymbol{\omega})\right]\\ \boldsymbol{\omega}_{t+1} &= \boldsymbol{\omega}_t + \alpha\gamma^t R_t \nabla \log \pi(a|s_t,\boldsymbol{\omega})\end{aligned}$$

In case π is a Gibbs policy:

$$\nabla \log \pi(a|s,\boldsymbol{\omega}) = \boldsymbol{\psi}(s,a) - \sum_b \pi(b|s,\boldsymbol{\omega})\boldsymbol{\psi}(s,b)$$

REINFORCE, A Monte-Carlo Policy-Gradient Method

Algorithm 3 REINFORCE

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\omega})$, $\alpha > 0$
Initialise $\boldsymbol{\omega}$
repeat
    Generate an episode $s_0, a_0, r_1, \cdots, s_T, a_T$ following $\pi(\cdot|\cdot,\boldsymbol{\omega})$
    for each step $t = 0, \cdots, T$ do
        $R_t \leftarrow$ return from step $t$
        $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\gamma^t R_t \nabla \log \pi(a|s_t,\boldsymbol{\omega})$
    end for
until convergence
```

Summary

- Instead of parametrising value functions we can directly parametrise policy
- Policy gradient theorem states the value of the gradient
- An episodic Monte Carlo algorithm which estimates policy parameters using policy gradient theorem is REINFORCE algorithm