On-policy Prediction with Approximation

9.1 Value-function Approximation

- So far we have represented value function by a lookup table
  - Every state s has an entry V(s)
  - Every state-action pair (s,a) has an entry Q(s,a)

- Problem with large MDPs
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually

9.1 Value-function Approximation

Solution for large MDPs:

- Estimate value function with function approximation

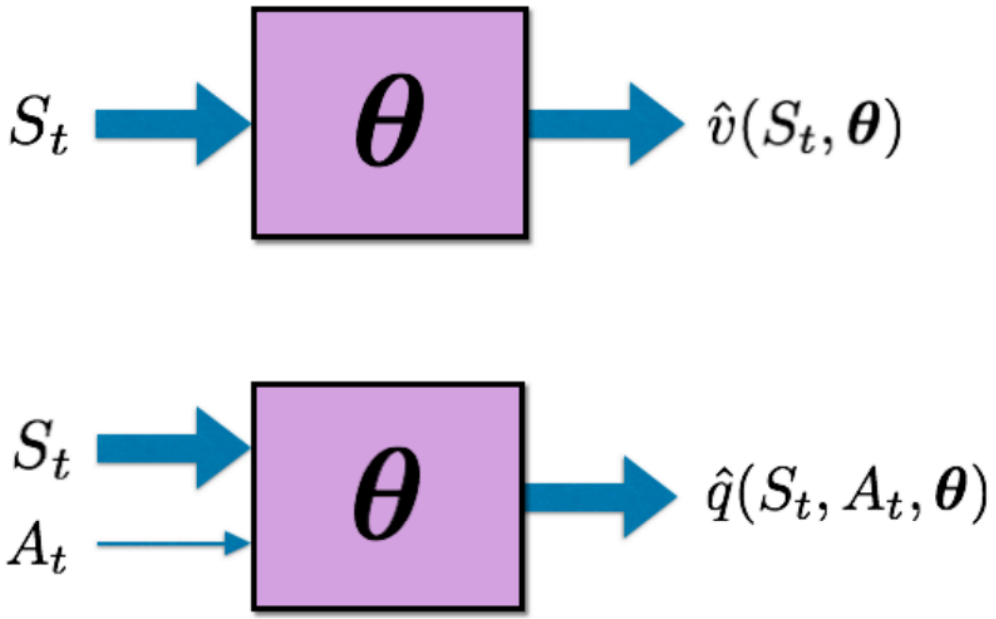
$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$

$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

- Generalize from seen states to unseen states

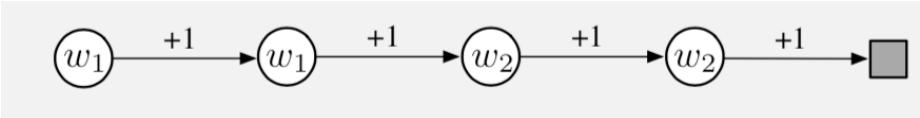
9.1 Value-function Approximation

- Value function approximation (VFA) replaces the table with a general parameterized form:



9.2 Prediction Objective ($\overline{VE}$)

- By assumption we have far more states than weights
  - $d << |\mathcal{S}|$



- Making one state's estimate more accurate invariably means making others less accurate

- So We are obligated then to say which states we care most about (In, 9.11)

- We must specify a state distribution, representing how much we care about the error in each state s
  - $\mu(s) \geq 0, \sum_s \mu(s) = 1$

9.2 Prediction Objective ($\overline{VE}$)

- By the error in a state s mean the square of the difference between the approximate value $\hat{v}(s,w)$ and the true value $v_\pi(s)$

- Mean squared Value Error($\overline{VE}$)
  - $\overline{VE}(w) \doteq \sum_{s\in\mathcal{S}} \mu(s)[v_\pi(s)-\hat{v}(s,w)]^2$

- On-policy training this is called the On-policy distribution

9.2 Prediction Objective ($\overline{VE}$)

The on-policy distribution in episodic tasks

In an episodic task, the on-policy distribution is a little different in that it depends on how the initial states of episodes are chosen. Let $h(s)$ denote the probability that an episode begins in each state s, and let $\eta(s)$ denote the number of time steps spent, on average, in state s in a single episode. Time is spent in a state s if episodes start in s, or if transitions are made into s from a preceding state $\bar{s}$ in which time is spent:

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_{a} \pi(a|\bar{s}) p(s\,|\,\bar{s}, a), \quad \text{for all } s \in \mathcal{S}. \tag{9.2}$$

This system of equations can be solved for the expected number of visits $\eta(s)$. The on-policy distribution is then the fraction of time spent in each state normalized to sum to one:

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}, \quad \text{for all } s \in \mathcal{S}. \tag{9.3}$$

This is the natural choice without discounting. If there is discounting ($\gamma < 1$) it should be treated as a form of termination, which can be done simply by including a factor of γ in the second term of (9.2).

9.2 Prediction Objective ($\overline{VE}$)

- Ideal goal in terms of $\overline{\boldsymbol{VE}}$ would be to find a global optimum
  - Weight vector w^* for which $\overline{\boldsymbol{VE}}(w^*) \leq \overline{\boldsymbol{VE}}(\mathrm{w})$ for all possible w

- Reaching this goal is sometimes possible in linear(simple), but rarely possible for artificial neural network, decision tree

- Sometimes, we seek to coverage instated to a local optimum
  - Weight vector w^* for which $\overline{\boldsymbol{VE}}(w^*) \leq \overline{\boldsymbol{VE}}(\mathrm{w})$ for all w in some neighborhood of w^*

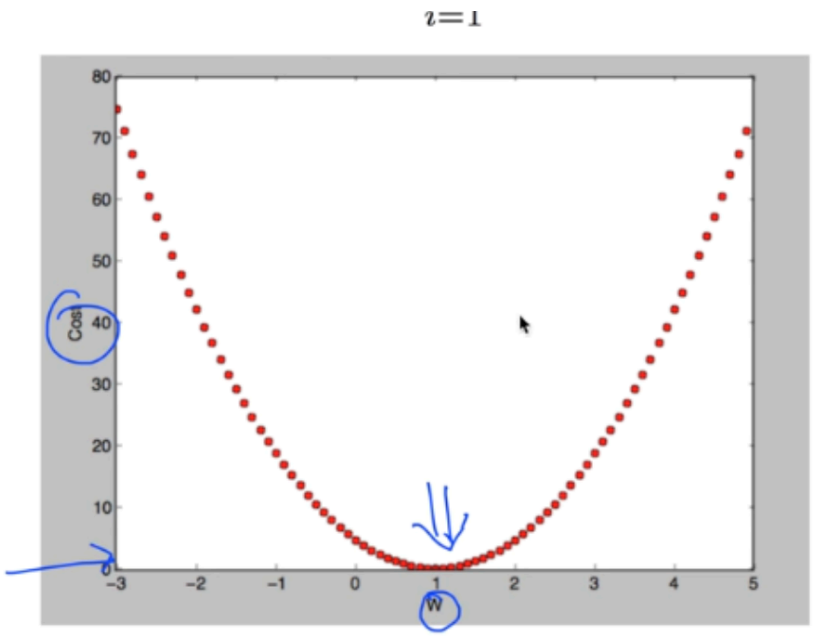
9.3 Stochastic-gradient and Semi-gradient Methods

- SGD(Stochastic-Gradient) methods are among the most widely used of all function approximation methods and are well suited to online reinforcement learning
- Weight vector is a column vector with a fixed number of real valued components
  - $w \doteq (w_1, w_2, \ldots, w_d)^\intercal, 1$
- Approximate value function $\hat{v}(s, w)$ is differentiable function of w
- Updating w at each of a series of discrete time steps, so we will need a notation w_t

9.3 Stochastic-gradient and Semi-gradient Methods

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t),\end{aligned}$$

$$\nabla f(\mathbf{w}) \doteq \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \frac{\partial f(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d}\right)^\top.$$



9.3 Stochastic-gradient and Semi-gradient Methods

- Target output U_t of the Tth training example, $S_t \mapsto U_t$ is not the true value but some, possibly random approximation to it.

- $v_\pi(S_t)$ is unknown, but we can approximate it by substituting U_t in

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - \hat{v}(S_t,\mathbf{w}_t)\Big]\nabla\hat{v}(S_t,\mathbf{w}_t).$$

- If U_t is an unbiased estimate, w_t is guaranteed to converge to a local optimum.

9.3 Stochastic-gradient and Semi-gradient Methods

- Because the true value of a state is the expected value of the return following it, Monte carlo target $U_t = G_t$ is by definition an unbiased estimate of $v_\pi(S_t)$

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

9.3 Stochastic-gradient and Semi-gradient Methods

- Bootstrapping target such as n-step, or DP target all depend on the current value of the weight vector w_t
  - They will be biased and that they will not produce a true gradient-descent method
- Bootstrapping is not true gradient descent.
- They only account the effect of changing the weight vector on the estimate, and ignore its effect on the target
  - So we call them semi-gradient methods

9.3 Stochastic-gradient and Semi-gradient Methods

- Semi gradient methods do not converge as robustly as gradient methods, they do converge reliably in important cases such as linear

- And they have important advantages
  - Faster
  - Enable learning to be continual and online, without waiting for the end of an episode

- Prototypical semi-gradient method is semi gradient TD(0)

9.3 Stochastic-gradient and Semi-gradient Methods

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S is terminal

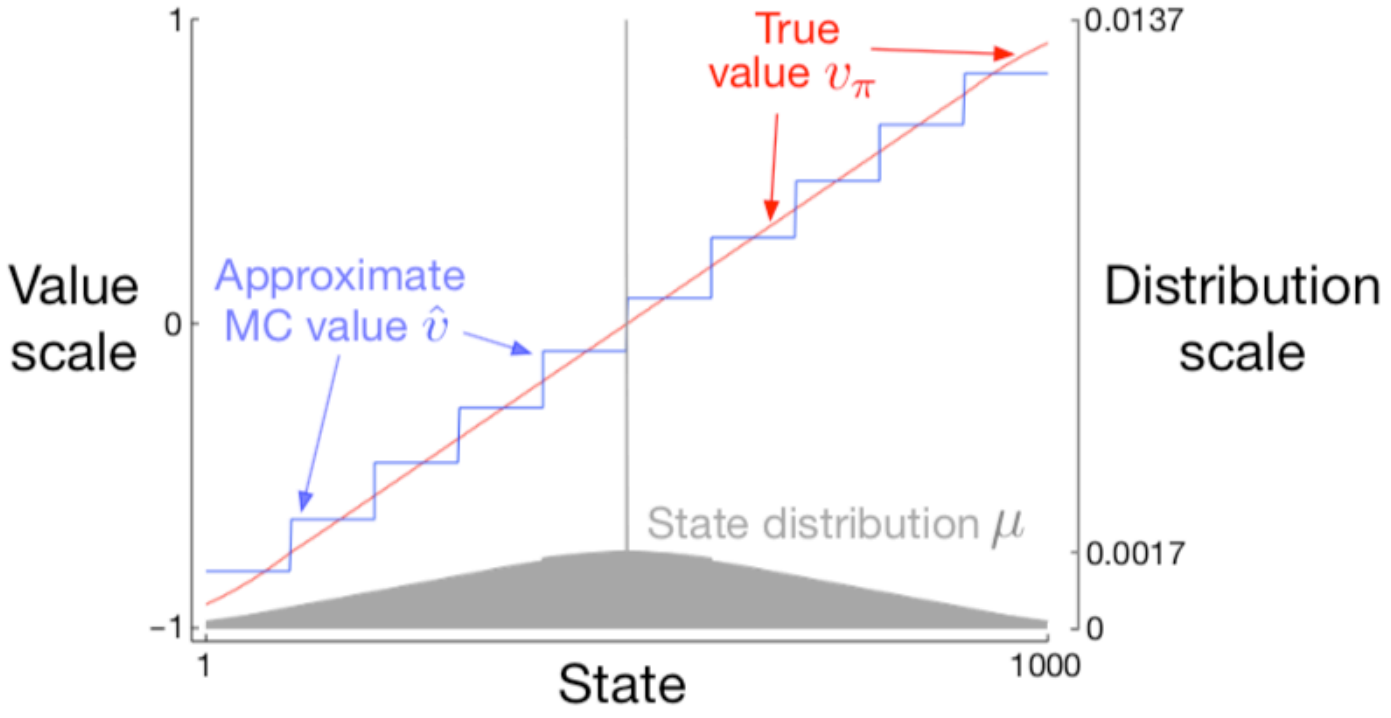
9.3 Stochastic-gradient and Semi-gradient Methods

- State aggregation is simple form of generalizing function in which states are grouped together

- The value of a state is estimated as its group's component, and when the state is updated, that component alone is updated

- State aggregation is a special case of SGD in which the gradient $< \nabla \hat{v}(S_t, W_t)$ is 1 for S_t's group's component and 0 for the other components

9.3 Stochastic-gradient and Semi-gradient Methods



This is due to the states in these areas having the greatest asymmetry in their weightings by μ. For example, in the leftmost group, state 100 is weighted more than 3 times more strongly than state 1. Thus the estimate for the group is biased toward the true value of state 100, which is higher than the true value of state 1.

9.4 Linear Methods

- Vector x(s) is called a feature vector representing state s

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s). \qquad \nabla \hat{v}(s,\mathbf{w}) = \mathbf{x}(s).$$

- General SGD update(9.7)

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[U_t - \hat{v}(S_t,\mathbf{w}_t) \Big] \mathbf{x}(S_t).$$

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha \Big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \Big) \mathbf{x}_t \\ &= \mathbf{w}_t + \alpha \Big(R_{t+1} \mathbf{x}_t - \mathbf{x}_t \big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1} \big)^\top \mathbf{w}_t \Big), \end{aligned}$$

9.4 Linear Methods

- Vector x(s) is called a feature vector representing state s

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s). \qquad \nabla \hat{v}(s,\mathbf{w}) = \mathbf{x}(s).$$

- General SGD update(9.7)

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[U_t - \hat{v}(S_t,\mathbf{w}_t) \Big] \mathbf{x}(S_t).$$

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha \Big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \Big) \mathbf{x}_t \\ &= \mathbf{w}_t + \alpha \Big(R_{t+1} \mathbf{x}_t - \mathbf{x}_t \big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1} \big)^\top \mathbf{w}_t \Big), \end{aligned}$$

9.4 Linear Methods

- Expected next weight vector

$$\mathbb{E}[\mathbf{w}_{t+1}|\mathbf{w}_t] = \mathbf{w}_t + \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t),$$

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}\left[\mathbf{x}_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)^\top\right] \in \mathbb{R}^d \times \mathbb{R}^d$$

$$\begin{aligned} \mathbf{b} - \mathbf{A}\mathbf{w}_{\text{TD}} &= \mathbf{0} \\ \mathbf{b} &= \mathbf{A}\mathbf{w}_{\text{TD}} \\ \mathbf{w}_{\text{TD}} &\doteq \mathbf{A}^{-1}\mathbf{b}. \end{aligned}$$

TD fixed point, linear semi-gradient TD(0) converges to this point.

9.4 Linear Methods

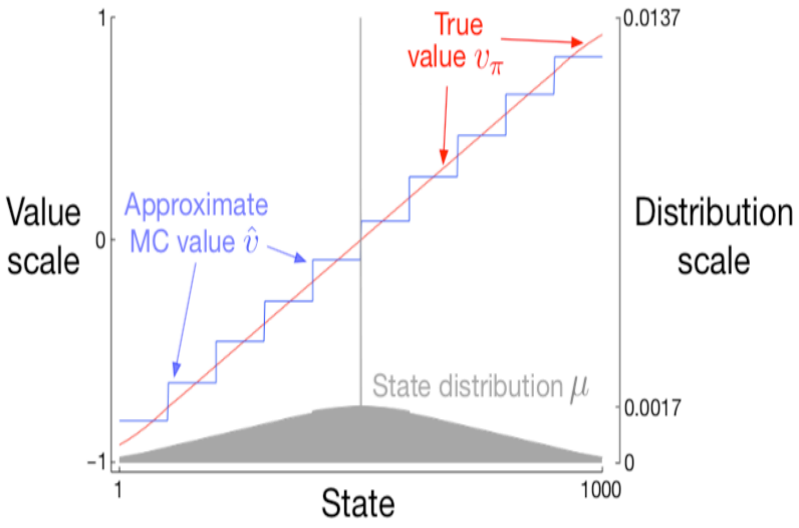
- At the TD fixed point, it has also been proven that the VE is within a bounded expansion of the lowest possible error

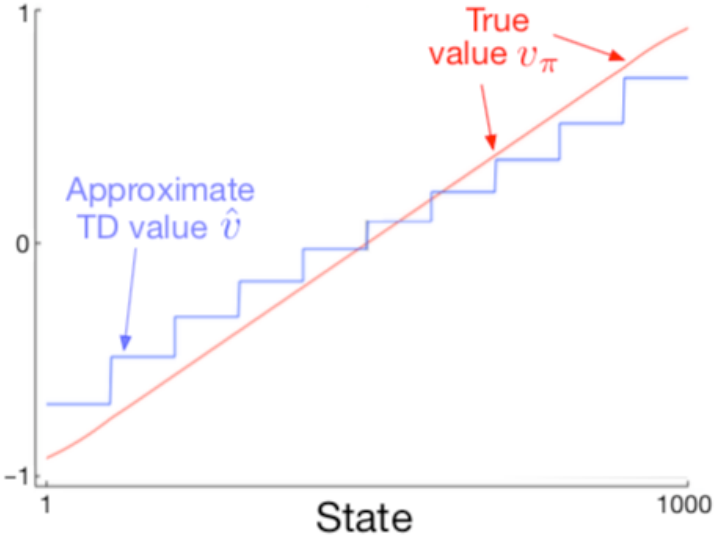
$$\overline{\mathrm{VE}}(\mathbf{w}_{\mathrm{TD}}) \;\leq\; \frac{1}{1-\gamma} \min_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}).$$

- TD method asymptotic error is often near one, this expansion factor can be quite large, so there is substantial potential loss in asymptotic performance with the TD method.

- A bound analogous applies to other on-policy bootstrapping methods as well.

9.4 Linear Methods





TD는 Monte carlo에 비해 True Value로 부터 많이 떨어져있음에도
불구하고 학습률에서 큰 장점을 유지하면서 일반화한다.

9.4 Linear Methods

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, a positive integer n
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n \hat{v}(S_{\tau+n},\mathbf{w})$ $(G_{\tau:\tau+n})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha\,[G - \hat{v}(S_\tau,\mathbf{w})]\,\nabla\hat{v}(S_\tau,\mathbf{w})$
  Until $\tau = T - 1$

9.6 Selecting Step-Size Parameters Manually

- Selecting Alpha(learning rate)

- Slowly decreasing step-size sequence that are sufficient to guarantee convergence, but these tend to result in learning that is too slow.

- intuitive feel for how to set the step-size parameter manually, it is best to go back momentarily to the tabular case.

9.6 Selecting Step-Size Parameters Manually

- In the tabular case, a step size of alpha = 1/10 would take about 10 experiences to converge approximately to their mean target.

- If we wanted to learn in 100 experiences we would use alpha = 1/100 .

9.8 Least-Squares TD

- Why, one might ask, must we compute this solution iteratively?

$$\begin{aligned}\mathbf{b} - \mathbf{A}\mathbf{w}_{\mathrm{TD}} &= \mathbf{0} \\ \mathbf{b} &= \mathbf{A}\mathbf{w}_{\mathrm{TD}} \\ \mathbf{w}_{\mathrm{TD}} &\doteq \mathbf{A}^{-1}\mathbf{b}.\end{aligned}$$

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t] \in \mathbb{R}^d \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}\left[\mathbf{x}_t\left(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\right)^\top\right] \in \mathbb{R}^d \times \mathbb{R}^d$$

Could one not do better by computing estimates of A and b, and then directly computing the TD fixed point?

9.8 Least-Squares TD

- LSTD

$$\widehat{\mathbf{A}}_t \doteq \sum_{k=0}^{t-1} \mathbf{x}_k (\mathbf{x}_k - \gamma \mathbf{x}_{k+1})^\top + \varepsilon \mathbf{I} \quad \text{and} \quad \widehat{\mathbf{b}}_t \doteq \sum_{k=0}^{t-1} R_{k+1} \mathbf{x}_k, \qquad \mathbf{w}_t \doteq \widehat{\mathbf{A}}_t^{-1} \widehat{\mathbf{b}}_t.$$

I is the identity matrix, ensures that A_t is always invertible
However, the extra t factors cancel out when LSTD uses these estimates to estimate the TD fixed point.

LSTD is more data efficiently than TD(0)
But TD(0) -> O(d), LSTD -> o(d^3)

9.8 Least-Squares TD

- LSTD

$$\begin{aligned}\widehat{\mathbf{A}}_t^{-1} &= \left(\widehat{\mathbf{A}}_{t-1} + \mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top\right)^{-1} \\ &= \widehat{\mathbf{A}}_{t-1}^{-1} - \frac{\widehat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top\widehat{\mathbf{A}}_{t-1}^{-1}}{1 + (\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top\widehat{\mathbf{A}}_{t-1}^{-1}\mathbf{x}_t},\end{aligned}$$

Sherman-Morrison formula -> O(d^2)

O(d^2) is still significantly more expensive than the O(d) of semi-gradient TD.

LSTD requires no step-size parameter is sometimes also touted, but the advantage of this is probably overstated.

9.9 Memory-based Function Approximation

- we have discussed the parametric approach to approximating value functions.

- Each update, s->g, is a training example used by the learning algorithm to change the parameters with the aim of reducing the approximation error.

- They simply save training examples in memory as they arrive (or at least save a subset of the examples) without updating any parameters.

- Then, whenever a query state's value estimate is needed, a set of examples is retrieved from memory and used to compute a value estimate for the query state

9.9 Memory-based Function Approximation

- The simplest example of the memory-based approach is the nearest neighbor method, which simply finds the example in memory whose state is closest to the query state and returns that example's value as the approximate value of the query state.

- Finding nearest neighbors in a large database can take too long to be practical in many applications.

9.10 Kernel-based Function Approximation

- Memory-based methods such as the weighted average and locally weighted regression methods described above depend on assigning weights to examples s->g in the database depending on the distance between s0 and a query states s.

- The function that assigns these weights is called a kernel function, or simply a kernel.

- Kernel regression is the memory-based method that computes a kernel weighted average of the targets of all examples stored in memory, assigning the result to the query state. If D is the set of stored examples, and g(s0) denotes the target for state s0 in a stored example, then kernel regression approximates the target function, in this case a value function depending on D

$$\hat{v}(s,\mathcal{D}) = \sum_{s' \in \mathcal{D}} k(s,s')g(s').$$

9.10 Kernel-based Function Approximation

- The form of the approximation is a linear combination of the pre-determined RBFs.

- First, it is memory-based: the RBFs are centered on the states of the stored examples.
- Second, it is nonparametric: there are no parameters to learn

9.10 Kernel-based Function Approximation

- The form of the approximation is a linear combination of the pre-determined RBFs.

- First, it is memory-based: the RBFs are centered on the states of the stored examples.
- Second, it is nonparametric: there are no parameters to learn

9.11 Looking Deeper at On-Policy Learning

- The algorithms we have considered so far in this chapter have treated all the states encountered equally, as if they were all equally important.

- In some cases, however, we are more interested in some states than others.

- One reason we have treated all states encountered equally is that then we are updating according to the on-policy distribution, for which stronger theoretical results are available for semi-gradient methods.

9.11 Looking Deeper at On-Policy Learning

- First we introduce a non-negative scalar measure, a random variable I_t called interest, indicating the degree to which we are interested in accurately valuing the state (or state–action pair) at time t

- Second, we introduce another non-negative scalar random variable, the emphasis M_t. This scalar multiplies the learning update and thus emphasizes or de-emphasizes the learning done at time t.

9.11 Looking Deeper at On-Policy Learning

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha M_t \left[G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{v}(S_t, \mathbf{w}_{t+n-1}), \quad M_t = I_t + \gamma^n M_{t-n}, \qquad 0 \le t < T,$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G_t - \hat{v}(S_t, \mathbf{w}) \right] \nabla \hat{v}(S_t, \mathbf{w})$$