IEOR 8100: Reinforcement learning

Lecture 2: TD-learning, Q-learning: tabular

By Shipra Agrawal

This lecture describes approximate dynamic programming based approaches of TD-learning and Q-learning. These are essentially extensions of policy iteration and Q-value iteration, respectively. We first discuss the tabular versions of these methods, which work for small scale MDPs. Next, we describe large-scale versions of these methods using function approximation. We focus on the infinite horizon discounted case.

1 TD-learning

Recall policy iteration uses *policy evaluation* and *policy improvement*. In every iteration, the current policy π is evaluated by computing the value function $V^\pi(s) = \lim_{T\to\infty} \mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1 = s, \pi]$ for each state s. When the MDP model (i.e, reward function $R(\cdot,\cdot)$ and transition matrix $P(\cdot,\cdot,\cdot)$) is known, the policy evaluation can be done by computing value function using a value iteration like algorithm, where in every step the value vector would be updated as $\mathbf{v}^k = L^\pi \mathbf{v}^{k-1} = \mathbb{E}_{a\sim\pi(s)}[R(s,a) + \gamma P(s,a)^\top \mathbf{v}^{k-1}]$. Then, the policy is improved by remapping each state s to action (greedy update):

$$\arg\max_a R(s,a) + \gamma P(s,a)^\top V^\pi$$

TD-learning is essentially an approximate version of policy evaluation without knowing the model, and using samples instead. Adding policy improvement gives an approximate version of policy iteration.

Since the value of a state $V^\pi(s)$ is defined as the expectation of the random (discounted) return when the process is started from the given state s, a natural way of estimating this value is to compute an average over multiple independent realizations started from the given state. This is an instance of the so-called *Monte-Carlo* method. Unfortunately, the variance of the observed returns can be high, which means that the quality of this estimates can be poor. The Monte-Carlo technique is further difficult to apply if the system is not accessible through a simulator, but rather samples are collected by actually interacting with the system. That is, the rewards obtained while actually taking the actions are used as feedback to learn in a closed loop. In this case, or even in presence of simulator, it might be difficult to reset the state of the system to an arbitrary state. Temporal difference (TD) learning (Sutton [1988]), which is one of the most significant ideas in reinforcement learning, is a method that can be used to address these issues.

1.1 TD(0)-learning

Policy evaluation is about estimating value $V^\pi(\cdot)$ of a given policy π, which by Bellman equations is equivalent to finding a fixed point of the following equations:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}\left[R(s,a) + \gamma \sum_{s'} P(s,a,s') V^\pi(s')\right], \forall s$$

However, now we need to estimate these using only sample observations $\{r_t, s_{t+1}\}$, the reward and next state observed on playing some action a_t in the current state s_t.

Let the current estimate of value function for any state s is $\hat{V}(s)$. Let on taking action $a_t = \pi(s_t)$ in the current state s_t, the observed (sample) next state is s_{t+1}. The current prediction of the value function of the state s_t, s_{t+1} are $\hat{V}(s_t), \hat{V}(s_{t+1})$. Then, we can obtain another prediction of the value of state s_t using one-lookahead (Bellman equations) as $r_t + \gamma\hat{V}(s_{t+1})$. Note that we have

$$\mathbb{E}\left[r_t + \hat{V}(s_{t+1}) | s_t, \hat{V}\right] = \mathbb{E}_{a\sim\pi(s_t)}\left[R(s_t,a) + \sum_{s'} P(s_t,a,s')\hat{V}(s') | s_t, \hat{V}\right]$$

Therefore, to satisfy Bellman equations, we are looking for $\hat{V}$ such that

$$\hat{V}(s_t) \approx r_t + \hat{V}(s_{t+1})$$

The TD method performs the following update to the value function estimate at s_t, moving it towards the new estimate:

$$\hat{V}(s_t) \leftarrow (1-\alpha_t)\hat{V}(s_t) + \alpha_t(r_t + \gamma\hat{V}(s_{t+1}))$$

Let δ_t be the following gap :

$$\delta_t := r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t),$$

referred to as *temporal difference*, i.e., the difference between current estimate, and one-lookahead estimate. Then, the above also be written as:

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha_t\delta_t \tag{1}$$

SGD interpretation. The above update can be interpreted as a stochastic gradient descent step for minimizing square loss (least squares method). Following supervised learning terminology, let $\hat{V}(s_t)$ denote a prediction for an observed data point s_t. Let $z_t = r_t + \gamma\hat{V}(s_{t+1})$ is the label (also referred to as 'target'). Then, the least squares method would fit a $\hat{V}(\cdot)$ by minimizing squared loss over the observed s_ts:

$$\arg\min_V \sum_e \sum_{t\in\text{episode } e} \frac{1}{2}(V(s_t) - z_t)^2.$$

The loss function aims to minimize distance between the prediction and target. To minimize the above loss function, one could use stochastic gradient descent. The gradient at t^{th} step with respect to $V(s_t)$ is $V(s_t) - z$ whose value at the current point $V = \hat{V}$ is $\hat{V}(s_t) - r_t - \gamma\hat{V}(s_{t+1}) = -\delta_t$. Thus, (1) is a gradient step with step size α_t, which moves the prediction closer to the observations. This is also a popular rule in supervised learning called the LMS update rule (LMS stands for "least mean squares"), and is also known as the Widrow-Hoff learning rule. However, an important difference here compared to the use of squared error minimization in supervised learning is that the 'target' $z_t = r_t + \gamma\hat{V}(s_{t+1})$ in the the above squared loss objective itself depends on the current estimate, thus the algorithm uses a form of 'bootstrapping'.

The gradient descent interpretation is even more insightful in the large-scale case when function approximation is used ($\hat{V}(s)$ will be estimated as a parametric function of state features and the gradient will be with respect to the parameters).

Algorithm 1 Tabular TD(0) method for policy evaluation

Initialization: Given a starting state distributiion D_0, policy π, the method evaluates $V^\pi(s)$ for all states s. Initialize $\hat{V}$ as an empty list/array for storing the value estimates. Initialize estimates $\hat{V}(s) = 0$ for all states s. Initialize episode $e = 0$.
repeat
  $e = e + 1$.
  Set $t = 1, s_1 \sim D_0$. Choose step sizes $\alpha_1, \alpha_2, \ldots$.
  Perform TD(0) updates over an episode:
  repeat
    Take action $a_t \sim \pi(s_t)$. Observe reward r_t, and new state s_{t+1}.
    $\delta_t := r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)$.
    Update $\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha_t\delta_t$.
    $t = t + 1$
  until the episode terminates
until change in $\hat{V}$ over consecutive episodes is small

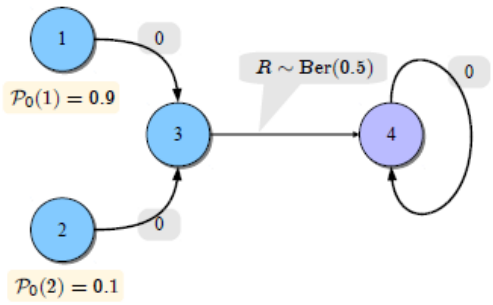

Figure 1: [Szepesvári, 1999] In this example, all transitions are deterministic. The reward is zero, except when transitioning from state 3 to state 4, when it is given by a Bernoulli random variable with parameter 0.5. State 4 is a terminal state. When the process reaches the terminal state, it is reset to start at state 1 or 2. The probability of starting at state 1 is 0.9, while the probability of starting at state 2 is 0.1.

1.2 Monte Carlo method

Why use only 1-step lookahead to construct the target z? Why not look ahead the entire trajectory (in problems where there is a terminal state, also referred to as episodic MDPs)?

The resulting method is referred to as Monte Carlo method. In this method, a sample trajectory starting at s_t is used to construct z:

$$z = \sum_{n=0}^{\infty} \gamma^n r_{t+n} =: \mathcal{R}_t$$

so that

$$\delta_t = z - \hat{V}(s_t) = \mathcal{R}_t - \hat{V}(s_t)$$
$$\hat{V}(s_t) \leftarrow (1 - \alpha_t)\hat{V}(s_t) + \alpha_t \mathcal{R}_t = \hat{V}(s_t) + \alpha_t \delta_t$$

Therefore, unlike TD-learning, the Monte-Carlo method does not use "bootstrapping".

1.3 TD(0) or Monte-Carlo?

Note: This example is taken from page 22-23, Szepesvári [1999].

First, we discuss an example where TD(0) converges faster. Consider the undiscounted episodic MRP shown in Figure 1. (Markov Reward Process or MRP is the process obtained on fixing a policy in an MDP, i.e., a Markov chain with rewards). The initial states are either 1 or 2. The process starts at state 1 with a higher probability (0.9), less frequently in state 2. Consider now how $TD(0)$ will behave at state 2. By the time state 2 is visited the k^{th} time, on the average state 3 would ha already been visited $10k$ times. Assume that $\alpha_t = 1/(t+1)$ (the TD updates with this step size reduce to $\frac{1}{t+1}(t\hat{V}(s_t) + z_t)$, i.e., averaging of targets). At state 1 and 2, the target is $\hat{V}(3)$ (since immediate reward is 0 and transition probability to state 3 is 1). Therefore, whenever state 2 is visited the TD(0) sets its value as the average of estimates $\hat{V}^t(3)$ over the time steps t when state 1 was visited (similarly for state 2). At state 3 the TD(0) update reduces to averaging the Bernoulli rewards incurred upon leaving state 3. At the k^{th} visit of state 2, $\text{Var}(\hat{V}(3)) \simeq 1/(10k)$ Clearly, $\mathbb{E}[\hat{V}(3)] = 0.5$. Thus, each update to the TD(0) prediction for state 2 is unbiased, and variance of the k^{th} target decreases as $1/(10k)$.

Now, consider the Monte-Carlo method. The Monte-Carlo method ignores the estimate of the value of state 3 and uses the Bernoulli rewards directly. In particular, $Var(\mathcal{R}_t | s_t = 2) = 0.25$, i.e., the variance of the target does not change with time. On this example, this makes the Monte-Carlo method slower to converge, showing that sometimes bootstrapping might indeed help.

To see an example when bootstrapping is not helpful, imagine that the problem is modified so that the reward associated with the transition from state 3 to state 4 is made deterministically equal to one. In this case, the Monte-Carlo method becomes faster since $\mathcal{R}_t = 1$ is the true target value. On the other hand, for the value of state

2 to get close to its true value, TD(0) has to wait until the estimate of the value at state 3 becomes close to its true value. This can slow down the convergence of TD(0). In fact, one can imagine a longer chain of states, where state $i+1$ follows state i, for $i \in 1, ..., N$ and the only time a nonzero reward is incurred is when transitioning from state $N-1$ to state N. In this example, the rate of convergence of the Monte-Carlo method is not impacted by the value of N, while TD(0) would get slower with N increasing (in TD(0), it will take N episodes for state 3's value to be updated to its correct value).

1.4 TD(λ)

TD(0) looks only one step in the "future" to update $\hat{V}(s_t)$, based on r_t and $\hat{V}(s_{t+1})$. Monte Carlo method looks at the entire trajectory. TD(λ) is a "middle-ground" between TD(0) and Monte-Carlo evaluation.

Here, the algorithm considers ℓ-step predictions, where for $\ell \geq 0$ case, the target

$$z_t^\ell = \sum_{n=0}^{\ell} \gamma^n r_{t+n} + \gamma^{\ell+1} \hat{V}(s_{t+\ell+1})$$

with temporal difference:

$$\begin{aligned} \delta_t^\ell &= z - \hat{V}(s_t) \\ &= \sum_{n=0}^{\ell} \gamma^n r_{t+n} + \gamma^{\ell+1} \hat{V}(s_{t+\ell+1}) - \hat{V}(s_t) \\ &= \sum_{n=0}^{\ell} \gamma^n (r_{t+n} + \gamma \hat{V}(s_{t+n+1}) - \hat{V}(s_{t+n})) \\ &= \sum_{n=0}^{\ell} \gamma^n \delta_{t+n} \end{aligned}$$

In TD(λ) method, a mixture of ℓ-step predictions is used, with weight $(1-\lambda)\lambda^\ell$ for $\ell \geq 0$. Therefore, $\lambda = 0$ gives $TD(0)$, and $\lambda \to 1$ gives Monte-Carlo method. $\lambda > 1$ gives a multi-step method. To summarize, the $TD(\lambda)$ update is given as:

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha_t \sum_{\ell=0}^{\infty} (1-\lambda)\lambda^\ell \delta_t^\ell = \hat{V}(s_t) + \alpha_t \sum_{n=0}^{\infty} \lambda^n \gamma^n \delta_{t+n}$$

1.5 Policy improvement with TD-learning

TD-learning allows evaluating a policy. For using TD-learning for finding optimal policy, we need to be able to improve the policy. Recall that policy iteration effectively requires evaluating Q-value of a policy, where

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P^\pi(s,a,s') V^\pi(s'),$$

with $V^\pi(s') = Q^\pi(s', \pi(s'))$. Policy improvement step then sets the new policy to take action $\arg\max_a Q^\pi(s,a)$ in state s. When the MDP model is known, Q-value of a policy can be evaluated by Q-value iteration, where $(k+1)^{th}$ update is given by:

$$Q_{k+1}^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P^\pi(s,a,s') Q_k^\pi(s', \pi(s')).$$

With simple modification, TD-learning can be used to estimate Q-value of a policy from observations. Given current state and action s_t, a_t, and reward, next state r_t, s_{t+1}, the temporal difference is now computed as:

$$\delta_t = r_t + \gamma \hat{Q}(s_{t+1}, \pi(s_{t+1})) - \hat{Q}(s_t, a_t)$$

and the updates in Step 8 and 9 would be replaced by:

8: $\delta_t := r_t + \gamma \hat{Q}(s_{t+1}, \pi(s_{t+1})) - \hat{Q}(s_t, a_t)$.

9: Update $\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha_t \delta_t$.

Then, the general scheme for policy improvement is similar to policy iteration: Start with some policy π^1. Repeat the following until convergence to some policy:

- Use TD-learning to evaluate the policy π^k. The method outputs $\hat{Q}^{\pi^k}(s,a), \forall s, a$.
- Compute new 'improved policy' π^{k+1} as $\pi^{k+1}(s) \leftarrow \arg\max_a \hat{Q}^{\pi^k}(s,a)$

The issue of exploration. A challenge here is that during iteration k, updates would be made only for those actions a that are chosen by policy π^k. Therefore, unless the policy π^k explores the action space well, the estimates of Q-value for other actions wouldn't be good enough for a meaningful policy improvement step. The policy update (aka policy improvement)step therefore often includes adding some exploration to the greedy update. One option is to replace policy improvement step by an ϵ-greedy choice. That is, the policy improvement step will now compute the new 'improved policy π^{k+1} as the following randomized policy:

$$\pi^{k+1}(s) = \begin{cases} a_k^* \in \arg\max_a \hat{Q}^{\pi^k}(s,a), & \text{with probability} 1-\epsilon, \\ \text{uniformly random } a \in A, & \text{with probability } \epsilon \end{cases}$$

Then, in policy evaluation, this 'randomized policy' must be used.

8: $\delta_t := r_t + \gamma \mathbb{E}_{a \sim \pi^{k+1}(s_{t+1})}[\hat{Q}(s_{t+1}, a)] - \hat{Q}(s_t, a_t)$.

9: Update $\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha_t \delta_t$.

More on this later, during the discussion on policy gradient methods.

2 Q-learning

Q-learning is a sample based version of Q-value iteration. This method attempts to directly find optimal Q-values, instead of computing Q-values of a given policy.

Recall Q-value iteration: for all s, a update,

$$Q_{k+1}(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} P(s,a,s') \left(\max_{a'} Q_k(s', a') \right)$$

Q-learning approximates these updates using sample observations, similar to TD-learning.

In steps $t = 1, 2, ...$ of an episode, the algorithm observes reward r_t and next state $s_{t+1} \sim P(\cdot, s_t, a_t)$ **<u>for some action</u>** a_t. It updates the Q-estimates for pair (s_t, a_t) as follows:

$$Q_{k+1}(s_t, a_t) = (1-\alpha) Q_k(s_t, a_t) + \alpha \underbrace{\left(r_t + \gamma \max_{a'} Q_k(s_{t+1}, a') \right)}_{\text{target}}$$

Algorithm 2 Tabular Q-learning method

Initialization: Given a starting state distributiion D_0.
Initialize $\hat{Q}$ as an empty list/array for storing the Q-value estimates.
Initialize episode $e = 0$.
repeat
  $e = e + 1$.
  Set $t = 1, s_1 \sim D_0$. Choose step sizes $\alpha_1, \alpha_2, \ldots$.
  Perform Q-learning updates over an episode:
  repeat
    Take action a_t. Observe reward r_t, and new state s_{t+1}.
    $\delta_t := \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a')\right) - \hat{Q}(s_t, a_t)$.
    Update $\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha_t \delta_t$.
    $t = t + 1$
  until episode terminates
until change in $\hat{Q}$ over consecutive episodes is small

How to select actions, the issue of exploration The convergence results (discussed later) for tabular Q-learning will prove that if all actions and states are infinitely sampled, learning rate is small, but does not decrease too quickly, then Q-learning converges. (Does not mater how you select actions, as long as they are infinitely sampled). One option is to select actions greedily according to the current estimate $\max_a Q_k(s, a)$. But, this will reinforce past errors, and may fail to sample and estimate Q-values for actions which currently have higher error levels. As a result, the algorithm may get stuck in a subset of (suboptimal) actions and a subspace of states. Therefore, exploration is required. The ϵ-greedy approach (i.e., with ϵ probability pick an action uniformly at random instead of greedy choice) can ensure infinite sampling of every action, but can be very inefficient.

The same issue occured in TD-learning based policy improvement methods, there the choice of action is specified as the greedy policy according to the previous episode estimates. Without exploration this may not ensure that all actions and states are infinitely sampled.

These issues will be discussed further throughout the course.

3 Convergence analysis for tabular Q-learning

Convergence theorem. The following result appears in Watkins and Dayan [1992] for tabular Q-learning (the result for TD(0) is similar).

Theorem 1 (Watkins and Dayan [1992])**.** *Given bounded rewards $|r_t| \leq R$, learning rates $0 \leq \alpha_t < 1$, and*

$$\sum_{i=1}^{\infty} \alpha_{n^i(s,a)} = \infty, \sum_{i=1}^{\infty} (\alpha_{n^i(s,a)})^2 < \infty,$$

then $\hat{Q}^t(s, a) \to Q(s, a)$ as $t \to \infty$ for all s, a with probability 1. Here, $n^i(s, a)$ is the index of the i^{th} time the action a is tried in state s, and $\hat{Q}^t(s, a)$ is the estimate $\hat{Q}$ in round t.

This means if all actions and states are infinitely sampled, learning rate is small, but does not decrease too quickly, then Q-learning converges. (Does not mater how you select actions, as long as they are infinitely sampled). For example, consider $\alpha_t = 1/t$; for finite state and action, this satisfies the given conditions as long as every state and action is visited periodically. The proof of this and many similar results in RL algorithms follow the analysis of a more general online learning/optimization method – the stochastic approximation method.

3.1 Stochastic Approximation method

The stochastic approximation (SA) algorithm essentially solves a system of fixed point (and potentially nonlinear) equations of the form

$$\theta = H(\theta)$$

for unknown $h(\cdot)$, based on noisy measurements $H(\theta) + \omega$ of the function $H(\theta)$.

The classical SA algorithm (Robbins and Monro [1951]) is of the form

$$\theta_{n+1} \quad = \quad (1 - \alpha_n)\theta_n + \alpha_n(H(\theta_n) + \omega_n), \quad n \geq 0 \tag{2}$$

Or equivalently,

$$\theta_{n+1} \quad = \quad \theta_n + \alpha_n \underbrace{(H(\theta_n) + \omega_n - \theta_n)}_{\delta_n}, \quad n \geq 0 \tag{3}$$

Since ω_n is 0-mean noise, the stationary points of the above algorithm coincide with the solutions of $H(\theta) = \theta$. We will show that (under certain conditions) this method converges to θ such that $H(\theta) = \theta$.

Asynchronous version. More relevant to the RL methods discussed here is the asynchronous version of the SA method. In the asynchronous version of SA method, we may observe only one coordinate (say i^{th}) of $Z_n = H(\theta_n) + \omega_n$ at a time step, and we use that to update i^{th} component of our parameter estimate:

$$\theta_{n+1}[i] = \theta_n[i] + \alpha_n \delta_n[i]$$

The convergence for this method will be proven similarly to the synchronous version, under the assumption that every coordinate is sampled infinitely often.

3.2 Examples

- **TD(0)-learning for estimating V^π:** Let θ^* denotes V^π, the value function of policy π. Then, our aim is to find V^π such that

  $$L^\pi V^\pi = V^\pi$$

  where the operator $L^\pi : \mathbb{R}^S \to \mathbb{R}^S$ was defined as

  $$L^\pi \theta := R_\pi + \gamma P_\pi^\top \theta$$

  Therefore, we want to solve $H(\theta) = \theta$ where $H = L^\pi$. We can now show that TD(0) is equivalent to the asynchronous SA algorithm. In every step n, we observe the noisy version of the s^{th} coordinate of $H(\theta) + \omega_n$, i.e., we observe

  $$Z_n[s] = r_n + \gamma \hat{V}_n(s') = r_n + \gamma \theta_n[s'] = H(\theta)[s] + \omega_n[s]$$

  where

  $$\mathbb{E}[\omega_n[s] \mid \theta_n, s] = \mathbb{E}[Z_n[s] - H(\theta)[s] \mid \theta_n, s] = R_\pi(s) + \gamma P_\pi(s)^\top \theta_n(s) - H(\theta_n)[s] = 0$$

  and further the variance of ω_n is bounded by a quadratic function of $\|V\|_\infty$.

  And, the TD(0)-update is same as the SA method:

  $$\hat{V}_{n+1}(s) = \hat{V}_n(s) + \alpha_n(r_n + \gamma \hat{V}_n(s') - \hat{V}_n(s)) = \theta_n[s] + \alpha_n \delta_n[s]$$

- **Q learning for estimating Q^*:** Recall that we defined operator $L : \mathbb{R}^{S \times A} \to \mathbb{R}^{S \times A}$ as

  $$LQ(s, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') \max_{a'} Q(s', a')$$

  Then, finding Q^* that satisfies Bellman optimality equation is equivalent to solving for θ satisfying

  $$L\theta - \theta = 0$$

  Therefore, the stochastic approximation algorithm can be applied with $H = L$. We can check if the noise has mean 0, similar to Q-learning. Now, we can show that Q-learning is equivalent to asynchronous SA algorithm.

At time n, we are in state $s_n = s$, take action $a_n = a$ and observe the reward r_n and next state $s_{n+1} = s'$. Q-learning update is given by

$$Q_{n+1}(s,a) = Q_n(s,a) + \alpha_n(r_n + \gamma \max_{a'} Q_n(s',a') - Q_n(s,a)) = \theta_n[s,a] + \alpha_n \delta_n[s,a]$$

where

$$\delta_n[s,a] = r_n + \gamma \max_{a'} Q_n(s',a') - Q_n(s,a) = r_n + \gamma \max_{a'} \theta_n[s',a'] - \theta_n[s,a] = (H(\theta_n[s] + \omega_n[s]) - \theta_n[s]$$

with expected value (given θ_n) as:

$$\mathbb{E}[\delta_n[s,a] \mid \theta_n, s, a] = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a'} \theta_n(s',a') - \theta_n(s,a) = H(\theta_n)[s,a] - \theta_n[s,a].$$

Clearly, $\mathbb{E}[\omega_n(s)|\mathcal{F}_{n-1}] = 0$, and further the variance is bounded by a quadratic function of $\|Q\|_\infty$.

3.3 Convergence results for SA

Under appropriate conditions (on α_n, h and ω_n) the algorithm indeed can be shown to converge to a solution of $h(\theta) = 0$.

Assumptions required for convergence:
On step sizes:

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty \tag{4}$$

On noise: martingale zero-mean and sub-Guassian:

$$\mathbb{E}[\omega_n|\mathcal{F}_{n-1}] = 0, \quad \mathbb{E}[||\omega_n||^2|\mathcal{F}_{n-1}] \leq A + B||\theta_n||^2 \tag{5}$$

for some constant A. Here $\mathcal{F}_{n-1}$ is the σ-algebra defined by the history $\{\theta_1, \alpha_1, \omega_1, \ldots, \theta_{n-1}, \alpha_{n-1}, \omega_{n-1}, \theta_n, \alpha_n\}$. By definition, ω_n is measurable with respect to $\mathcal{F}_n$.

Note that Guassian noise $\omega_n \sim \mathcal{N}(\mathbf{0}, AI)$ satisfies the above assumption.

Theorem 2 (Corollary of Proposition 4.5 in Bertsekas and Tsitsiklis [1996])**.** *Given conditions* (4) *and* (5)*, and further assume that $h(\theta) = L\theta - \theta$, where L is a contraction mapping, i.e., for all iterates θ_n*

$$\|L\theta_n - L\theta^*\|_\infty \leq \gamma \|\theta_n - \theta^*\|_\infty$$

for some scalar $\gamma \in [0,1)$, then $\theta_n \to \theta^*$ as $n \to \infty$ with probability 1*.*

For intuition, we will prove an easier version of this theorem, which requires stronger conditions on $h(\theta)$. Later we show that these conditions are satisfied if $h(\theta) = L\theta - \theta$, and L is a contraction mapping with respect to *Euclidean norm*.

Theorem 3. *Given conditions* (4) *and* (5)*, and further assume that*

$$h(\theta_n)^\top (\theta_n - \theta^*) \leq -c\|\theta_n - \theta^*\|^2 \tag{6}$$

$$\|h(\theta_n)\|^2 \leq K_1 + K_2\|\theta_n - \theta^*\|^2 \tag{7}$$

where θ^* is the parameter value such that $h(\theta^*) = 0$, and $\|\cdot\|$ denotes 2*-norm. Then, we have that SA method converges under above assumptions. i.e.,*

$$\lim_{n\to\infty} \|\theta_n - \theta^*\| = 0$$

Proof. Here is a proof sketch. Let θ be the target, i.e., $h(\theta) = 0$

$$b_n = \mathbb{E}[(\theta_n - \theta)^2]$$

$$\begin{aligned}
b_{n+1} &= \mathbb{E}[\|\theta_{n+1} - \theta\|^2] \\
&= \mathbb{E}[\|\theta_{n+1} - \theta_n + \theta_n - \theta\|^2] \\
&= \mathbb{E}[\|\theta_{n+1} - \theta_n\|^2 + 2(\theta_{n+1} - \theta_n)^\top(\theta_n - \theta) + \|\theta_n - \theta\|^2] \\
b_{n+1} - b_n &= \mathbb{E}[\|\theta_{n+1} - \theta_n\|^2] + 2\mathbb{E}[(\theta_{n+1} - \theta_n)^\top(\theta_n - \theta)] \\
&= \mathbb{E}[\alpha_n^2\|Z_n\|^2] + 2\mathbb{E}[\mathbb{E}[\alpha_n(h(\theta_n) + \omega_n)^\top(\theta_n - \theta)|\mathcal{F}_{n-1}]] \\
&= \mathbb{E}[\alpha_n^2\|Z_n\|^2] + 2\alpha_n\mathbb{E}[h(\theta_n)^\top(\theta_n - \theta)] \\
&= \alpha_n^2\mathbb{E}[\mathbb{E}[\|h(\theta_n)\|^2 + \|\omega_n\|^2 + 2h(\theta_n)^\top\omega_n|\mathcal{F}_{n-1}]] + 2\alpha_n\mathbb{E}[h(\theta_n)^\top(\theta_n - \theta)] \\
&= \alpha_n^2\mathbb{E}[\|h(\theta_n)\|^2 + \|\omega_n\|^2] + 2\alpha_n\mathbb{E}[h(\theta_n)^\top(\theta_n - \theta)]
\end{aligned}$$

using $\mathbb{E}[\omega_n|\mathcal{F}_{n-1}] = 0$. Now, we use

$$h(\theta_n)^\top(\theta_n - \theta) \leq -c\|\theta_n - \theta\|^2 \tag{8}$$

$$\|h(\theta_n)\|^2 \leq K_1 + K_2\|\theta_n - \theta\|^2 \tag{9}$$

Then,

$$\begin{aligned}
b_{n+1} - b_n &\leq \alpha_n^2 K_1 + \alpha_n(K_2\alpha_n - 2c)\mathbb{E}[\|\theta_n - \theta\|^2] + \mathbb{E}[\alpha_n^2\omega_n^2] \\
&= \alpha_n^2 K_1 + \alpha_n(K_2\alpha_n - 2c)b_n + \mathbb{E}[\alpha_n^2\omega_n^2]
\end{aligned} \tag{10}$$

Since $\sum_n \alpha_n^2 < \infty$, there exists some finite n after which $K_2\alpha_n < c$. Therefore, for that n and higher, the second term is negative. Therefore for some n_0,

$$\begin{aligned}
\sum_{n\geq n_0}(b_{n+1} - b_n) &\leq \sum_{n\geq n_0}(\alpha_n^2 K_1 + \mathbb{E}[\alpha_n^2\omega_n^2]) < \infty \\
&\quad \text{(using assumption of bounded variance of noise and } \sum_n \alpha_n^2 < \infty)
\end{aligned}$$

This implies $b_{n+1} - b_n \to 0$, i.e., b_n converges.

Now, we want to show that b_n converges to 0. Suppose for contradiction that $\lim_{n\to\infty} b_n = \delta > 0$, then we would have $\sum_{n\geq n_0}\alpha_n c b_n \to \infty$ (since $\sum_n \alpha_n = \infty$). However, if we sum up (10) and move the terms around

$$\sum_{n\geq n_0}^{N}\alpha_n c b_n \leq b_{n_0} - b_{N+1} + \mathbb{E}[\sum_{n\geq n_0}^{N}\alpha_n^2(\omega_n^2 + K_1)] \tag{11}$$

$$\text{(using } b_{N+1} \geq 0) \leq b_{n_0} + \mathbb{E}[\sum_{n\geq n_0}^{N}\alpha_n^2(\omega_n^2 + K_1)] \tag{12}$$

$$< \infty \text{ (using assumption of bounded variance of noise and } \sum_n \alpha_n^2 < \infty) \tag{13}$$

This is a contradiction. Therefore, it must hold that $\lim_{n\to\infty} b_n = 0$. □

Corollary 4 (Corollary of Theorem 3)**.** *Given conditions* (4) *and* (5)*, and further assume that $h(\theta) = L\theta - \theta$, where the L operator satisfied Euclidean norm contraction, ie., $\|LV - LV'\|_2 \leq \gamma\|V - V'\|_2$. Then, we have that SA method converges, i.e.,*

$$\lim_{n\to\infty}\|\theta_n - \theta^*\| = 0$$

Proof. Suppose $h(\theta) = L\theta - \theta$, and L operator satisfied Euclidean norm contraction, ie., $\|LV - LV'\| \leq \gamma\|V - V'\|$. Then, using $L\theta^* = \theta^*$, and Cauchy-Schwartz,

$$(L\theta_n - \theta^*)^\top(\theta_n - \theta^*) \leq \|L\theta_n - L\theta^*\| \cdot \|\theta_n - \theta^*\| \leq \gamma\|\theta_n - \theta^*\|^2$$

Subtracting $(\theta_n - \theta^*)^\top(\theta_n - \theta^*)$ from both sides:

$$(L\theta_n - \theta_n)^\top(\theta_n - \theta^*) \leq -(1-\gamma)\|\theta_n - \theta^*\|^2$$

which is same as

$$h(\theta_n)^\top(\theta_n - \theta^*) \leq -c\|\theta_n - \theta^*\|^2 \tag{14}$$

That is, the first condition in Theorem 3 is satisfied. And, further

$$\begin{aligned}
\|h(\theta_n)\| &= \|L\theta_n - \theta_n\| \\
&= \|(L\theta_n - L\theta^*) + (L\theta^* - \theta_n)\| \\
&\leq \|L\theta_n - L\theta^*\| + \|\theta^* - \theta_n\| \\
&\leq (1+\gamma)\|\theta_n - \theta^*\| \\
&= (1+\gamma)\|\theta_n - \theta^*\|
\end{aligned}$$

Therefore, both the conditions of Theorem 3 are satisfied, and the convergence result follows from the theorem statement. □

3.4 Convergence of Q-learning/TD-learning through stochastic approximation

Here, we verify that the conditions in either Theorem 2 or 3 hold for all the three examples (mean computation, TD learning, Q-learning) discussed above. Therefore, convergence follows.

For the mean computation example, $\mathbb{E}[Z_n|\theta_n] = \mu - \theta_n = h(\theta_n)$. Also, $h(\theta_n) = \mu - \theta_n = -(\theta_n - \theta)$, which clearly statisfies the condition in Theorem 3.

For TD-learning,

$$h(\theta_n)[s] = r(s, \pi(s)) + \gamma P(s, \pi(s))^\top\theta_n - \theta_n(s) = (L^\pi\theta_n - \theta_n)[s, a],$$

where L^π is a contraction operator $\|L^\pi\theta - L^\pi\theta'\|_\infty \leq \gamma\|\theta - \theta'\|_\infty$. In Q-learning, L^π is replaced by the L operator which is also a contraction operator with respect to $\|\cdot\|_\infty$. Therefore, the convergence for these methods follows from Theorem 2.

References

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

Csaba Szepesvári. *Algorithms for Reinforcement Learning.* Morgan & Claypool Publishers, 1999. URL `http://old.sztaki.hu/ szcsaba/papers/RLAlgsInMDPs-lecture.pdf`.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.