IEOR 8100: Reinforcement learning

Lecture 2: TD-learning, Q-learning (Tabular versions)

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. Here, we discuss the tabular versions of these methods, which work for small scale MDPs. Large scale versions of these methods using function approximation will be discussed in the next lecture.

For simplicity, 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 $\mathbf{v}^k = L^\pi v^{k-1} = \mathbb{E}_{a\in\pi(s)}[R(s,a) + \gamma P(s,a)^\top \mathbf{v}^{k-1}]$. 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 approximate version of policy evaluation without knowing the model (using samples). 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 return when the process is started from the given state s, an obvious 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 the estimates will be poor. The Monte-Carlo technique is further difficult to apply if the system is not accessible through a simulator but rather estimation happens while actually interacting with the system. In this case, 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 some particular state. In this case, the Monte-Carlo technique cannot be applied without introducing some additional bias. 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 $V^\pi(\cdot)$, which by Bellman equations is equivalent to finding a stationary point of the following equations:

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

However, we need to estimate this using only observations r_t, s_{t+1} on playing some action a_t at 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, s_{t+1} is the observed (sample) next state, The predicted value function of for the next state s_{t+1} is $\hat{V}(s_{t+1})$, giving another prediction of value function at state s_t (by one-lookahead using Bellman equations is) $r_t + \gamma\hat{V}(s_{t+1})$. Note that

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

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

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

Then, 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 terminology, let $\hat{V}(s_t)$ denotes a predictor. Let $z_t = r_t + \gamma\hat{V}(s_{t+1})$ is an observation (also referred to as 'target'). Then, the least squares method fits $\hat{V}(s)$ for any s by minimizing squared loss:

$$\sum_{t:s_t=s} \frac{1}{2}(\hat{V}(s_t) - z_t)^2,$$

The loss function aims to minimize distance between prediction and observation.

To minimize the above loss function, one could use stochastic gradient descent. The gradient at t^{th} step with respect to $\hat{V}(s_t)$ is $(\hat{V}(s_t) - z) = -\delta_t$. 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 from use of squared error minimization 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 will be more significant in the large scale case when function approximation is used ($\hat{V}$ will be estimated as a parametric function of state features).

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 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 until episode terminates
until change in $\hat{V}$ over consecutive episodes is small

1.2 Monte Carlo method

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

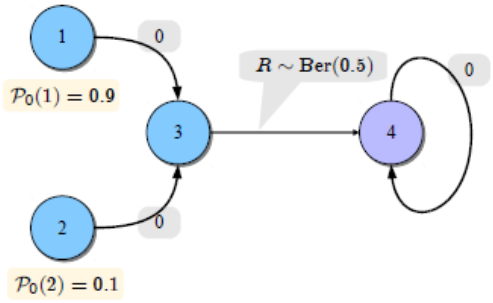

Figure 1: [Szepesvári, 1999] In this example, all transitions are deterministic. The reward is zero, except when transitioning from state 3 to state 4, when it is given by a Bernoulli random variable with parameter 0.5. State 4 is a terminal state. When the process reaches the terminal state, it is reset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

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

$$\delta_t = z - \hat{V}(s_t) = \mathcal{R}_t - \hat{V}(s_t)$$
$$\hat{V}(s_t) = (1 - \alpha_t)\hat{V}(s_t) + \alpha_t \mathcal{R}_t - \hat{V}(s_1) + \alpha_t \delta_t$$

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

1.3 TD(0) or Monte-Carlo?

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

First, let us consider an example when $TD(0)$ converges faster. Consider the undiscounted episodic MRP shown on Figure 4. (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. With high probability the process starts at state 1, while the process starts at state 2 less frequently. 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 has already been visited 10 k times. Assume that $\alpha_t = 1/(t+1)$ (the TD updates with this step size reduce to averaging of target observations). 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, the target of the update of state 2 will be an estimate of the true value of state 2 with accuracy increasing with k.

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, while 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 slows 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.

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 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 simple modification, TD-learning can be used to estimate Q-value of a policy. There, 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 $\pi^{k+1}(s) \leftarrow \arg\max_a \hat{Q}^{\pi^k}(s,a)$

2 Q-learning (tabular)

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 $s_{t+1} \sim P(\cdot, s_t, a_t)$ **for some action** 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

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

How to select actions, the issue of exploration The convergence results (discussed below) for Q-learning will say 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 have higher error levels. And, we may get stuck at a subset of (suboptimal) actions. 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 occurs 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.

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

3 Convergence analysis through Stochastic Approximation method

Convergence theorem. The following result appears in Watkins and Dayan [1992] for 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). 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 (nonlinear) equations of the form

$$h(\theta) = 0$$

for unknown $h(\cdot)$, based on noisy measurements of $h(\theta)$.

More specifically, consider a (continuous) function $\mathbb{R}^d \to \mathbb{R}^d$, with $d \geq 1$, which depends on a set of parameters $\theta \in \mathbb{R}^d$. Suppose that $h(\theta)$ is unknown. However, for any θ we can measure $Z = h(\theta) + \omega$, where ω is some 0-mean noise. The classical SA algorithm (Robbins and Monro [1951]) is of the form

$$\begin{aligned} \theta_{n+1} &= \theta_n + \alpha_n Z_n \\ &= \theta_n + \alpha_n (h(\theta_n) + \omega_n), \quad n \geq 0 \end{aligned} \tag{2}$$

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

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:

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

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

- **Estimating Mean:** Let Y_n be iid samples of a r.v. with mean μ. Consider the following method for estimating mean μ:

  $$\theta_{n+1} = \theta_n + \frac{1}{n+1}(Y_n - \theta_n).$$

  This is SA with $h(\theta_n) = \mu - \theta_n = \theta^* - \theta_n$, $Z_n = Y_n - \theta_n$, $\alpha_n = 1/(n+1)$. Converges to $\lim_{n\to\infty} \theta_n = \lim_{n\to\infty} \frac{\sum_{i=0}^{n} Y_{n-1} + \theta_0}{n} = \mu$.

- **TD(0)-learning for estimating V^π:** Let θ^* denotes V^π, the value function of policy π. And, let

$$h(\theta) = R_\pi + \gamma P_\pi^\top \theta - \theta = L^\pi \theta - \theta$$

Then $h(\theta^*) = 0$ means θ^* (i.e., V^π) satisfies Bellman equations. We can now show that TD(0) is equivalent to the asynchronous SA algorithm. In every step n, we observe the s^{th} coordinate:

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

with expected value (given θ_n) as:

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

(Clearly, $\mathbb{E}[\omega_n(s)|\mathcal{F}_{n-1}] = 0$, and further the variance is bounded by a quadratic function of $\|V\|_\infty$.)
And, the TD(0)-update is same as the SA method:

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

- **Q learning for estimating Q^*:** Let $\theta^* \in \mathbb{R}^{S \times A}$ denote $[Q^*(s,a)]_{s,a}$, the optimal Q-value function. Define

$$h(\theta)[s,a] = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a'} \theta[s',a'] - \theta[s,a].$$

Therefore, finding θ^* satisfying $h(\theta^*) = 0$ means finding $Q^*(s,a)$ that satisfy Bellman optimality equations. 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

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

where

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

with expected value (given θ_n) as:

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

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:

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty \tag{3}$$

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

$$\mathbb{E}[\omega_n|\mathcal{F}_{n-1}] = 0, \quad \mathbb{E}[||\omega_n||^2|\mathcal{F}_{n-1}] \leq A + B||\theta||^2 \tag{4}$$

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* (3) *and* (4)*, 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*.*

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* (3) *and* (4)*, and further assume that*

$$h(\theta_n)^\top (\theta_n - \theta^*) \leq -c\|\theta_n - \theta^*\|^2 \tag{5}$$

$$\|h(\theta_n)\|^2 \leq K_1 + K_2 \|\theta_n - \theta^*\|^2 \tag{6}$$

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

$$h(\theta_n)^\top (\theta_n - \theta) \leq -c\|\theta_n - \theta\|^2 \tag{7}$$

$$\|h(\theta_n)\|^2 \leq K_1 + K_2 \|\theta_n - \theta\|^2 \tag{8}$$

Then,

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

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. Summing up, and moving the terms around

$$\sum_{n \geq n_0}^{N} \alpha_n c b_n \leq b_{n_0} - b_{N+1} + \mathbb{E}[\sum_{n \geq n_0}^{N} \alpha_n^2 (\omega_n^2 + K_1)] \tag{9}$$

$$\text{(using } b_{N+1} \geq 0) \quad \leq b_{n_0} + \mathbb{E}[\sum_{n \geq n_0}^{N} \alpha_n^2 (\omega_n^2 + K_1)] \tag{10}$$

$$< \infty \text{ (using assumption of bounded variance of noise and } \sum_n \alpha_n^2 < \infty) \tag{11}$$

Now, substituting this back in in (9), we have

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

This implies $b_{n+1}-b_n \to 0$, i.e., b_n converges. Now, if $\lim_{n\to\infty} b_n = \delta > 0$, then $\sum_{n\geq n_0}\alpha_n c b_n \to \infty$ (since $\sum_n \alpha_n = \infty$), contradicting the above observation. Therefore, it must hold that $\lim_{n\to\infty} b_n = 0$. □

Condition verification. Lets first verify if the conditions in Theorem 3 hold for interesting examples: 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 following $\mathcal{L}$ operator,

$$h(\theta_n)[s,a] = r(s,a) + \gamma\sum_{s'}P(s,a,s')\max_{a'}Q_n(s',a') - Q_n(s,a) = (\mathcal{L}Q_n - Q_n)[s,a],$$

where $\mathcal{L}$ is also a contraction operator with respect to $\|\cdot\|_\infty$.

Therefore, we are interested in functions $h(\theta)$ of the form $h(\theta) = L\theta - \theta$, where L satisfies contraction property with respect to sup norm. These do not always satisfy conditions in Theorem 3, but a closely related case does: when L satisfies the contraction property for 2 norm. If only sup-norm contraction is available, the proofs are quite technical due to non-differentiability of this norm, see Bertsekas and Tsitsiklis [1996], Proposition 4.5. For developing intuition, below we show that for closely related case of $h(\theta) = L\theta - \theta$, where L satisfies contraction property with respect to 2-norm, the required conditions in Theorem 3 are satisfied.

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

$$h(\theta_n)^\top(\theta_n - \theta^*) \leq -c\|\theta_n - \theta^*\|^2 \tag{12}$$

That is, the first condition 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 are satisfied.

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.