ORCS 4529: Reinforcement Learning

Shipra Agrawal

Columbia University
Industrial Engineering and Operations Research

Contents I

Contents I

Recall: Q-value iteration

1. Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
2. In every iteration k, improve the Q-value vector as:

$$\mathbf{Q}^k(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \left(\max_{a'} Q^{k-1}(s',a') \right), \forall s, a$$

3. Stop if $\|Q^k - Q^{k-1}\|_\infty$ is small.
4. Output policy π^k defined as $\pi^k(s) = \arg\max_a Q^k(s,a)$

Recall: Policy iteration using Q-values

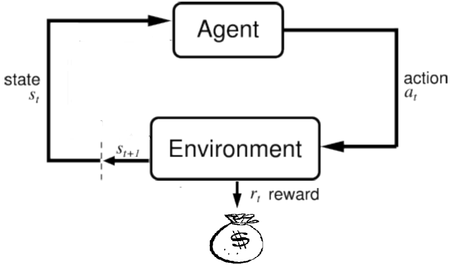
1. Initialize policy π^0.
2. In every iteration $k = 0, 1, 2, \ldots,$
   - (Policy evaluation) Compute value $Q^{\pi^k}(s, a)$, the Q-values of policy π^k for all s, a.
   - (Greedy Policy improvement) Compute new policy

$$\pi^{k+1}(s) := \arg\max_a Q^{\pi_k}(s, a), \forall s$$

3. Stop when $\pi^{k+1} = \pi^k$.

Relaxed stopping criteria: not much change in policy or its value.

Learning from observations



Unknown MDP model (R, P) of the environment, but can interact with the environment (take action a_t) to observe sample reward r_t and transition $s_t \to s_{t+1}$

$$\mathbb{E}[r_t | s_t = s, a_t = a] = R(s, a)$$

$$\Pr(s_{t+1} = s' | s_t = s, a_t = a) = P(s, a, s')$$

Reinforcement Learning: DP based Algorithms

How to implement Q-value iteration and policy iteration using samples?

How to interact with the environment to generate useful samples?

Contents I

Recall: Q-value iteration

1. Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
2. In every iteration k, improve the Q-value vector as:

$$\mathbf{Q}^k(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \left(\max_{a'} Q^{k-1}(s',a') \right), \forall s, a$$

3. Stop if $\|Q^k - Q^{k-1}\|_\infty$ is small.
4. Output policy π^k defined as $\pi^k(s) = \arg\max_a Q^k(s,a)$

First attempt: Implementing Q-VI with samples from environment

1. Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
2. For $t = 1, 2, \ldots,$,
   observe s_t, take action a_t, observe reward r_t, next state s_{t+1}
   improve the Q-value at s_t, a_t only as:

$$\mathbf{Q}^t(s_t, a_t) = r_t + \gamma \left(\max_{a'} Q^{t-1}(s_{t+1}, a') \right)$$

3. Stop if $\|Q^t - Q^{t-1}\|_\infty$ is small.
4. Output policy π^t defined as $\pi^t(s) = \arg\max_a Q^t(s, a)$

Justification

For the state action pair s_t, a_t, we have an unbiased estimator of the DP update

$$\mathbb{E}[r_t + \gamma \left(\max_{a'} Q^{t-1}(s_{t+1}, a') \right) | s_t = s, a_t = a]$$

$$= R(s,a) + \gamma \sum_{s'} P(s,a,s') \left(\max_{a'} Q^{t-1}(s', a') \right)$$

Concerns

- Error in update: Single sample used in every update.
  - We have an unbiased estimate of the Q-VI update but a large variance
  - (Why not take multiple samples before updating?)
  - Do not have unbiased estimate of Q-value
- Coverage: Update of only one component s_t, a_t: the visited state and action

Will it still converge? How to explore and update all states and actions? How to get more samples for important states and actions?

Q-learning

Mitigating the concerns

- Idea 1: Slow down the update

$$\mathbf{Q}^t(s_t, a_t) = (1-\alpha_t)Q^{t-1}(s_t, a_t) + \alpha_t \left(r_t + \gamma \left(\max_{a'} Q^{t-1}(s_{t+1}, a') \right) \right)$$

where $\alpha_t \in [0, 1]$ acts as a "learning rate".

Q-learning

Mitigating the concerns

- Idea 1: Slow down the update

$$\mathbf{Q}^t(s_t, a_t) = (1 - \alpha_t)Q^{t-1}(s_t, a_t) + \alpha_t \left(r_t + \gamma \left(\max_{a'} Q^{t-1}(s_{t+1}, a') \right) \right)$$

  where $\alpha_t \in [0, 1]$ acts as a "learning rate".

- Idea 2: Choose action a_t smartly:
  Greedy:

$$a_t = \max_a Q^{t-1}(s_t, a_t)$$

  Or ϵ-**Greedy**: Set a_t as a random action with probability ϵ, and

$$a_t = \max_a Q^{t-1}(s_t, a_t), \text{ w.p. } 1 - \epsilon$$

  Or, **other exploration-exploitation based methods**

Q-learning algorithm (tabular setting)

1. Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
2. For $t = 1, 2, \ldots,$
   Observe s_t, take action a_t using a smart policy , observe reward r_t, next state s_{t+1}
   Improve the Q-value at s_t, a_t only as:

$$\mathbf{Q}^t(s_t, a_t) = (1-\alpha_t)Q^{t-1}(s_t, a_t)+\alpha_t(r_t+\gamma\left(\max_{a'} Q^{t-1}(s_{t+1}, a')\right))$$

3. Stop if $\|Q^t - Q^{t-1}\|_\infty$ is small.
4. Output policy π^t defined as $\pi^t(s) = \arg\max_a Q^t(s, a)$

How to pick the actions? If ϵ-greedy that which ϵ? How to set the learning rate? Varies across implementations.

SGD interpretation of Q-learning update

Recall goal is to have $Q^t \to_{t\to\infty} Q^*$ where for all s, a

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

At the time t, we get an estimates z_t of rhs at $s = s_t, a = a_t$.

$$z_t := r_t + \gamma \left(\max_{a'} Q^{t-1}(s_{t+1}, a') \right)$$

(z_ts are referred to as **target**).
We want to find $\hat{Q}$ such that

$$\hat{Q}(s_t, a_t) \approx z_t, \forall t$$

Supervised learning view: find best fit by minimizing the squared loss over all samples:

$$\min_{\hat{Q}} \sum_t (\hat{Q}(s_t, a_t) - z_t)^2$$

SGD interpretation

$$\min_{\hat{Q}} \sum_t (\hat{Q}(s_t, a_t) - z_t)^2$$

Gradient of t^{th} term in objective with respect to $\hat{Q}$

$$2(\hat{Q}(s_t, a_t) - z_t)\mathbf{1}_{s_t, a_t}$$

SGD or online gradient descent: At iteration t:

$$\begin{aligned} \hat{Q}(s_t, a_t) \quad &\leftarrow \quad \hat{Q}(s_t, a_t) - \alpha_t \underbrace{(\hat{Q}(s_t, a_t) - z_t)}_{\text{temporal difference}} \\ &= \quad (1 - \alpha_t)\hat{Q}(s_t, a_t) + \alpha_t z_t \end{aligned}$$

Useful interpretation for studying convergence and extensions to deep learning.

Temporal difference (TD) in Q-learning

Temporal difference: Difference of current estimate compared to one-lookahead estimate

$$\delta_t = r_t + \gamma \left(\max_{a'} Q^{t-1}(s_{t+1}, a') \right) - Q^{t-1}(s_t, a_t)$$

Q-learning

$$Q^t(s_t, a_t) = Q^{t-1}(s_t, a_t) + \alpha_t \delta_t$$

Why one-lookhead? Could we play out the further future? More on this later in TD-learning.

Lab 1 Problem 1

Implement Tabular Q-learning for a simple OpenAI Gym (gymnasium) environment "Frozen-lake".

Q-learning with function approximation and deep learning

How to handle large state space?

Large-scale Q-learning

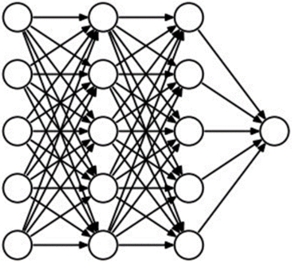
Use a parametric approximation $Q_\theta(s, a) = f_\theta(x_{s,a})$ for $Q(s, a)$.

- Example 1 Linear regression: Given feature embedding $x_{s,a}$ for state-action pair s, a, define $Q_\theta(s, a) = \theta_0 + \theta_1 x_1(s, a) + \ldots + \theta_n x_n(s, a)$
- Example 2 DNN: $Q_\theta(s, a) = f_\theta(x_{s,a})$ with θ being the parameters of the DNN

How do we find a good parameter θ?

Recall supervised learning basics

- Prediction Model (e.g. linear model or DNN)

$x \rightarrow$  $\rightarrow f_\theta(x)$

- Training: Given labeled dataset (x_i, y_i) a training algorithm (e.g., training in tensorflow, pytorch etc.) fits model parameters θ such that a loss function (e.g. square loss or cross-entropy loss) is minimized

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x_i), y_i)$$

Q-learning via supervised learning

- Task: Find a θ such that for every s, a, the Bellman equation,

  $$Q(s, a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s, a, s') + \gamma \max_{a'} Q(s', a')]$$

  can be approximated well for all s, a, i.e.,

  $$f_\theta(x_{s,a}) \approx \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s, a, s') + \gamma \max_{a'} f_\theta(x_{s',a'})]$$

Q-learning via supervised learning

- Task: Find a θ such that for every s, a, the Bellman equation,

$$Q(s, a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s, a, s') + \gamma \max_{a'} Q(s', a')]$$

  can be approximated well for all s, a, i.e.,

$$f_\theta(x_{s,a}) \approx \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s, a, s') + \gamma \max_{a'} f_\theta(x_{s',a'})]$$

- Given an observation (s_t, a_t, r_t, s_{t+1}) and current θ_t, construct sample

$$(x_t, z_t) = \{x_{s_t,a_t}, \quad r_t + \gamma \max_{a'} f_{\theta_t}(x_{s_{t+1},a'})\}$$

Q-learning via supervised learning

- Task: Find a θ such that for every s, a, the Bellman equation,

$$Q(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

  can be approximated well for all s, a, i.e.,

$$f_\theta(x_{s,a}) \approx \mathbb{E}_{s' \sim P(\cdot|s,a)}[R(s,a,s') + \gamma \max_{a'} f_\theta(x_{s',a'})]$$

- Given an observation (s_t, a_t, r_t, s_{t+1}) and current θ_t, construct sample

$$(x_t, z_t) = \{x_{s_t,a_t}, \quad r_t + \gamma \max_{a'} f_{\theta_t}(x_{s_{t+1},a'})\}$$

- Find θ to minimize a loss function, e.g., square loss (or later cross-entropy loss)

$$\min_\theta \sum_t (f_\theta(x_t) - z_t)^2$$

  Can do batch update or online update of θ (e.g. by stochastic gradient descent).

Deep Q-learning algorithm (DQN)

Let $L(x, y)$ be a loss function like squared loss function $L(x, y) = (x - y)^2$. Let $Q_\theta(s, a) = f_\theta(x_{s,a})$ be the function approximation, e.g., a DNN.

Repeat for $t = 1, 2, ..., T$,

- Observe state s_t, take an action a_t
  - ϵ-greedy policy $a_t = \arg\max_a Q_{\theta_t}(s_t, a) = \arg\max_a f_{\theta_t}(x_{s_t,a})$ with probability $1 - \epsilon$ and random action with probability ϵ
- Observe reward r_t, transition to state s_{t+1}.
- Construct feature embedding $x_t = x_{s_t,a_t}$. Use current θ_t to construct target

$$z_t = r_t + \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a') = r_t + \gamma \max_{a'} f_{\theta_t}(x_{s_{t+1},a'})$$

- Set learning rate α_t, and perform one stochastic gradient step

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \nabla_{\theta|_{\theta_t}} L(f_\theta(x_t), z_t)$$

Output parameter θ_T (learned policy $\pi(s) = \arg\max_a f_{\theta_T}(x_{s,a})$).

Remark on large action spaces

Can embed them using neural network just like states,

$$Q_\theta(s, a) = f_\theta(x_{s,a})$$

but in order to compute the policy and the target, we need to be able to compute

$$\max_a Q_\theta(s, a) \equiv \max_a f_\theta(x_{s,a})$$

Many practical scenarios: actions space is large but only few of them available in a given state, so above can be done by enumerating all feasible actions.

DQN: Tricks and tips

- No manual computation of gradient required: Autograd :
- Exploration:
- Batch learning and experience replay:
- Lazy update of target network:

DQN: Tricks and tips

- No manual computation of gradient required: Autograd : Deep learning packages provide autograd (efficient automatic differentiation) using backpropagation
- Exploration: Through the adaptive setting of ϵ in ϵ-greedy or other advanced exploration-exploitation schemes like Posterior Sampling.
- Batch learning and experience replay: Reuse older samples with current θ_t, and batch them for efficient updates.
- Lazy update of target network: If the parameters of the network (Q_{θ_t}) used to compute the target (z_t) are changed frequently with each update, an unstable target function will make training difficult.

Lab 3

Deep Q-learning and the OpenAI gym environment Cartpole

Contents I

Recall: Policy iteration using Q-values

1. Initialize policy π^0.
2. In every iteration $k = 0, 1, 2, \ldots,$
   - (Policy evaluation) Compute value $Q^{\pi^k}(s, a)$, the Q-values of policy π^k for all s, a.
   - (Greedy Policy improvement) Compute new policy

$$\pi^{k+1}(s) := \arg\max_a Q^{\pi_k}(s, a), \forall s$$

3. Stop when $\pi^{k+1} = \pi^k$.

Relaxed stopping criteria: not much change in policy or its value.

Computing Q-values of a fixed policy π

- Monte Carlo method:

$$Q^\pi(s,a) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | s_1 = s, a_1 = a; a_t = \pi(s)]$$

  Simulate the policy π starting from every state s and action a, and use the discounted sum of sample rewards to estimate the expected value.
  Easy extension to learning from observations

- Q-value-iteration : Based on Bellman equation (DP)

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

  TD-learning uses ideas similar to Q-learning

Q-value iteration for learning value of a policy π

A subroutine of policy iteration

1. Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
2. In every iteration k, improve the Q-value vector as:

$$\mathbf{Q}^k(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \mathbb{E}_{a' \sim \pi(s)} \left[Q^{k-1}(s',a') \right], \forall s, a$$

3. Stop if $\|Q^k - Q^{k-1}\|_\infty$ is small.
4. Output policy π^k defined as $\pi^k(s) = \arg\max_a Q^k(s,a)$

TD (Temporal difference) learning algorithm for learning Q-values of policy π

A subroutine of policy iteration

1. Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
2. For $t = 1, 2, \ldots,$
   Observe s_t, take action a_t using a smart policy, observe reward r_t, next state s_{t+1}
   Update the Q-value at s_t, a_t only as:
   $$\mathbf{Q}^t(s_t, a_t) = (1 - \alpha_t) Q^{t-1}(s_t, a_t) + \alpha_t (r_t + \gamma \left(\mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{s_{t+1}})} [Q^{t-1}(s_{t+1}, a')] \right))$$
3. Stop if $\|Q^t - Q^{t-1}\|_\infty$ is small.

Question: What's the difference compared to Q-learning? Why is this learning Q^π and not Q^*?

Concerns and Choices

Concerns

- Sampling estimation error in update
- Coverage of all state-action pairs (Why?)

Choices

- How do we set learning rate α_t?
- How do we pick the actions a_t?
  Should we use the policy π? Why? Why not?

Concerns and Choices

Concerns

- Sampling estimation error in update
- Coverage of all state-action pairs (Why?)

Choices

- How do we set learning rate α_t?

  A hyper-parameter that is tuned, but in theory decreasing with time, e.g. $1/t$ or $1/\sqrt{t}$.

- How do we pick the actions a_t?
  Should we use the policy π? Why? Why not?

  A Common choice is a randomized policy (perturbed version of π): with probability $1 - \epsilon$, play the action according to π, with probability $(1 - \epsilon)$ play a random action.

Monte Carlo Method for estimating Q-values of a fixed policy

Simulate policy π for T steps for a large enough T such that γ^T is very small.

- For $t = 1, 2, \ldots, T$
  Observe s_t, take action $a_t \sim \pi(s_t)$, observe reward r_t, next state s_{t+1}
- Output sample trajectory $\tau = (s_1, a_1, r_1, s_2, a_2, s_3, \ldots, s_T)$
- For each (s_t, a_t) in the trajectory τ,

$$\hat{Q}^{\pi}(s_t, a_t) \leftarrow r_t + \gamma r_{t+1} + \cdots + \gamma^{T-1} r_T$$

Comparison to TD-learning

- Sampling estimation error
- Coverage of state-action pairs:

Comparison to TD-learning

- Sampling estimation error in certain cases can be better than TD-learning due to larger lookahead $\equiv$ less bias
- Coverage of state-action pairs: Perturbing the policy needed for exploration but will introduce bias in the estimation of Q^{π}.

Unlike TD-learning, in MC Sample collection policy has to be the same as the policy being evaluated.

Examples: TD-learning vs. Monte Carlo

Comparison of estimation error

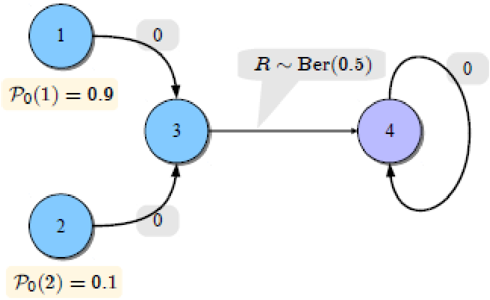


Figure: Source: Algorithms for Reinforcement Learning by Csaba Szepasvari

Lab 1 Problem 2

Implement Tabular Policy iteration with TD-learning on a simple OpenAI gym environment "Frozen-lake".

Policy iteration with function approximation

- Initialize θ^0, initialize policy $\pi^1 = \arg\max_a f_{\theta_0}(x_{s,a})$.
- In every iteration $k = 1, 2, \ldots$
  - (Policy evaluation): Use Deep TD-learning, train DNN $Q^{\pi_k}(s,a) := f_{\theta_k}(x_{s,a})$ to learn the value of policy π_k.
  - (Policy improvement): New policy π_{k+1} is defined by

$$\pi^{k+1}(s) := \arg\max Q^{\pi_k}(s,a) = \arg\max_a f_{\theta_k}(x_{s,a})$$

Note: Only the parameter θ_k needs to be stored.

Deep TD-learning subroutine

Learn the Q-value of given policy π using function approximation.

Repeat for $t = 1, 2, ..., T$,

- Observe state s_t, take an action a_t
  - ϵ-perturbed policy $a_t = \pi(s_t)$ w.p. $1-\epsilon$, random action w.p. ϵ.
- Observe reward r_t, transition to state s_{t+1}.
- Use current θ_t to construct (1-lookahead) target

$$z_t = r_t + \gamma f_{\theta_t}(x_{s_{t+1},a_{t+1}})$$

  or use larger look-ahead up to $\tau + 1$ steps or Monte-Carlo

$$z_t = \sum_{i=t}^{t+\tau} \gamma^{i-1} r_i + \gamma^{\tau+1} f_{\theta_t}(x_{s_{t+\tau+1},a_{t+\tau+1}})$$

  where $a_{t+\tau+1} = \pi(s_{t+\tau+1})$.
- Set learning rate α_t, and perform one stochastic gradient step

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \nabla_{\theta|_{\theta_t}} L(f_\theta(x_{s_t,a_t}), z_t)$$

Output θ_T, where $Q^\pi(s,a) \approx f_{\theta_T}(x_{s,a})$.

Contents I

Convergence of Tabular Q-learning, TD-learning

- We use the connections to the Stochastic Approximation method [Robbins and Monroe 1951]
- Asymptotic convergence proved in "Asynchronous Stochastic Approximation and Q-Learning" by John N. Tsitsiklis, 1994
- More recent works provide finite sample bounds:
  - Sheng Zhang, Zhe Zhang, Siva Theja Maguluri, Finite Sample Analysis of Average-Reward TD Learning and Q-Learning, NeurIPS 2021.
    (... and other more recent works by the last author)
  - Guannan Qu and Adam Wierman, Finite-time analysis of asynchronous stochastic approximation and Q-learning. COLT 2020.
  - Rayadurgam Srikant and Lei Ying. Finite-time error bounds for linear stochastic approximation and TD-learning. COLT 2019.

Stochastic Approximation (SA):

A method for finding a solution $x^* \in \mathbb{R}^n$ to

$$h(x) = 0$$

using noisy observations $\delta_t = h(x_t) + \omega_t$.

The SA method

- At time t on seeing sample δ_t, update

$$x_{t+1} \leftarrow x_t + \alpha_t \delta_t$$

Classic convergence theorems show $h(x_t) \to 0$ under certain conditions on $h(\cdot)$, the distribution of noise ω_t, and α_ts.

Examples of SA method

- Find solution to $\nabla f(x) = 0$ for $f(x) = \mathbb{E}_y[g(x, y)]$ using samples. SA is essentially the stochastic gradient descent method. (Derive!)

Examples of SA method

- Find solution to $\nabla f(x) = 0$ for $f(x) = \mathbb{E}_y[g(x, y)]$ using samples. SA is essentially the stochastic gradient descent method. (Derive!)
- Q-learning: Find a solution to Bellman equation for Q^*: $LQ = Q$. SA is the asynchronous version of the Q-learning method. (Derive!)

Note: Classic convergence proofs for SA are for synchronous versions, Tsitsiklis 1994, and more recent finite sample results extend them to asynchronous versions, assuming enough exploration.

Examples of SA method

- Find solution to $\nabla f(x) = 0$ for $f(x) = \mathbb{E}_y[g(x, y)]$ using samples. SA is essentially the stochastic gradient descent method. (Derive!)
- Q-learning: Find a solution to Bellman equation for Q^*: $LQ = Q$. SA is the asynchronous version of the Q-learning method. (Derive!)
- TD-learning: Find solution to $L^\pi Q = Q$, or $L^\pi V = V$. SA is the asynchronous version of the TD-learning method.

Note: Classic convergence proofs for SA are for synchronous versions, Tsitsiklis 1994, and more recent finite sample results extend them to asynchronous versions, assuming enough exploration.

SA method convergence

Essential Assumptions:

On step sizes:

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty \tag{1}$$

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

$$\mathbb{E}[\omega_t | \mathcal{F}_{t-1}] = 0, \quad \mathbb{E}[\|\omega_t\|^2 | \mathcal{F}_{t-1}] \leq A + B\|x_n\|^2 \tag{2}$$

for some constant A, B.

Here $\mathcal{F}_{n-1}$ is the σ-algebra defined by the history $\{x_1, \alpha_1, \omega_1, \ldots, x_{n-1}, \alpha_{n-1}, \omega_{n-1}, x_n, \alpha_n\}$. Noise 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.

Classic SA convergence theorem

Theorem

Given conditions (1) *and* (2), *and further assume that*

$$(h(x_n) - h(x^*))^\top (x_n - x^*) \leq -c\|x_n - x^*\|^2 \tag{3}$$

$$\|h(x_n) - h(x^*)\|^2 \leq K_1 + K_2\|x_n - x^*\|^2 \tag{4}$$

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.,*

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

A corollary

Corollary (Corollary of Theorem 1)

Given conditions (1) *and* (2), *and further assume that* $h(V) = LV - V$, *where the L operator satisfied Euclidean norm contraction, ie.,*

$$\|LV - LV'\|_2 \leq \gamma \|V - V'\|_2, \forall V, V'$$

and V^* *is such that* $LV^* = V^*$. *Then, we have that SA method converges, i.e.,*

$$\lim_{n \to \infty} \|V_n - V^*\| = 0$$

Proof.

Both conditions of the previous theorem are satisfied (Check) □

Convergence proofs relevant to Q-learning, TD-learning

Synchronous version

Theorem (Tsitsiklis 1994, Corollary of Proposition 4.5 in Bertsekas, 1996)

Given conditions (1) *and* (2), *and further assume that* $h(V) = LV - V$, *where* L *is a contraction mapping under infinity-norm, i.e., for all iterates* V_n

$$\|LV_n - LV^*\|_\infty \leq \gamma \|V_n - V^*\|_\infty$$

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

Convergence counterexample under function approximation

Contents I