7 On-policy Value Approximation

Melih Kandemir

Özyeğin University
Computer Science Department
melih.kandemir@ozyegin.edu.tr

7 Nov 2017

Value function approximation

Up till now we have stored the value functions in tabular form, but what if

- the state space is too large to allocate memory for every state?
- or, the state description is very high-dimensional (e.g. a Go table, or the scene image)?

The remedy is to approximate the value function v_π by a function $\hat{v}_\pi$ parametrized by a vector $\mathbf{w}$

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

that applies its parameters $\mathbf{w} \in \mathbb{R}^d$ on a $d-$dimensional vector of state features.

Value function approximation as supervised learning problem

- Typically, $d << |\mathcal{S}|$, where $|\mathcal{S}|$ is the state count.
- Hence, unlike the tabular approach, updating a single state affects the values of many other states!
- This attempt to *generalize* the value function over states is essentially the well-known *regression* setting in classical *supervised* learning.

RL-friendly supervised learning models

- **Online:** In RL, samples are generated on the fly, in a non-i.i.d. fashion. Hence, the chosen learning algorithm should be eligible for online learning (train incrementally by introducing one sample at a time).
- **Non-stationary**: The behavior of most RL target functions is *non-stationary* (i.e. the data distribution changes over time). The supervised learning method should adaptively account for drifts in data distributions.

Stochastic Gradient Descent (SGD)

Given a data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$ of N input observations $\mathbf{x}_n$, the corresponding output observations y_n, and an estimator $f(\mathbf{x}_n, \mathbf{w})$, in a typical regression task, we aim to minimize the Mean Squared Error (MSE)

$$MSE = \frac{1}{N} \sum_{n=1}^{N} \Big[y_n - f(\mathbf{x}_n, \mathbf{w}_t) \Big]^2$$

subject to $\mathbf{w}$ by gradient descent

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \frac{1}{N} \sum_{n=1}^{N} \Big[y_n - f(\mathbf{x}_n, \mathbf{w}_t) \Big] \nabla f(\mathbf{x}_n, \mathbf{w}_t).$$

Stochastic Gradient Descent (SGD)

One update requires a full pass on the entire data set

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \frac{1}{N} \sum_{n=1}^{N} \Big[y_n - f(\mathbf{x}_n, \mathbf{w}_t) \Big] \nabla f(\mathbf{x}_n, \mathbf{w}_t).$$

This

- is expensive if the data set is large.
- delays model fit.

Try out the divide-and-conquer strategy instead!

Stochastic Gradient Descent (SGD)

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \underbrace{\frac{1}{N} \sum_{n=1}^{N} \Big[y_n - f(\mathbf{x}_n, \mathbf{w}_t) \Big] \nabla f(\mathbf{x}_n, \mathbf{w}_t)}_{true\ gradient}.$$

Approximate the true gradient by sampling a random data point $\{\mathbf{x}_j, y_j\}$ and assuming that $\mathcal{D}$ consists of N copies of it. Then the gradient-descent update reads

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \underbrace{\Big[y_j - f(\mathbf{x}_j, \mathbf{w}_t) \Big] \nabla f(\mathbf{x}_j, \mathbf{w}_t)}_{stochastic\ gradient}.$$

Stochastic Gradient Descent (SGD)

SGD can be generalized to updating by small randomly-chosen data subsets, called **minibatches**, trivially. Given a random subset $\{(\mathbf{x}_{i(1)}, y_{i(1)}), \cdots, (\mathbf{x}_{i(\tilde{N})}, y_{i(\tilde{N})})\}$ of size $\tilde{N} << N$, where $i(n)$ is a random permutation, perform the update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \frac{1}{\tilde{N}} \sum_{n=1}^{\tilde{N}} \left[y_{i(n)} - f(\mathbf{x}_{i(n)}, \mathbf{w}_t) \right] \nabla f(\mathbf{x}_{i(n)}, \mathbf{w}_t).$$

Robbins-Monro Theorem (1951)

The stochastic gradient is an *unbiased* estimate of the true gradient if updates are performed following a learning rate series satisfying the two properties below

$$\sum_{t=1}^{\infty} \epsilon_t = \infty, \tag{1}$$

$$\sum_{t=1}^{\infty} \epsilon_t^2 < \infty. \tag{2}$$

(1) reach at points arbitrarily far away
(2) stop learning at some point

Data Sets in RL

Monte Carlo

$$\mathcal{D}_{MC} = \begin{bmatrix} (\boldsymbol{\phi}(s_1), G_1), \\ (\boldsymbol{\phi}(s_2), G_2), \\ \vdots \\ (\boldsymbol{\phi}(s_T), G_T) \end{bmatrix}$$

TD(0)

$$\mathcal{D}_{TD} = \begin{bmatrix} (\boldsymbol{\phi}(s_1), R_1 + \gamma \hat{v}(s_2, \mathbf{w})), \\ (\boldsymbol{\phi}(s_2), R_2 + \gamma \hat{v}(s_3, \mathbf{w})), \\ \vdots \\ (\boldsymbol{\phi}(s_T), R_T) \end{bmatrix}$$

Legend: (**input**,**output**)

SGD with and without replacement

- **without replacement:**
  - Choose a sample only once until all samples are chosen (i.e. until an **epoch** is complete).
  - More common in standard ML due mainly to practical reasons (to decide whether the epoch is over).
- **with replacement:**
  - Keep random sampling without caring about coverage.
  - Allow multiple selection of a sample within an epoch.
  - More common in RL.
  - Called **Experience Replay**.

The Mean Squared Value Error (MSVE)

$$MSVE(\mathbf{w}) \triangleq \sum_{s \in \mathcal{S}} \mu(s) \Big[v_\pi(s) - \hat{v}(s, \mathbf{w}) \Big]^2$$

where $\mu(s) \geq 0$ is a distribution assigning each state a degree of importance.
Here,

- $v_\pi(s)$ is the true value of state s, which is our *target*,
- and $\hat{v}(s, \mathbf{w})$ is a prediction estimating this target.

If we choose $\mu(s)$ to be the fraction of time spent in s, then we perform *on-policy RL*.

SGD for value approximation

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

where U_t is the approximation target for $v_\pi(S_t)$. For Monte Carlo RL, $U_t = G_t$.

Gradient MC for value approximation

Input: π, $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Initialize $\mathbf{w}$
repeat forever
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \cdots, R_T, S_t$ from π
  For $t = 0, 1, \cdots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \Big[G_t - \hat{v}(S_t, \mathbf{w}) \Big] \nabla \hat{v}(S_t, \mathbf{w})$

Convergence of Gradient Monte Carlo Value Approximation

- If $\mathbb{E}[U_t] = v_\pi(S_t)$, then Robbins-Monro is satisfied following a proper learning rate series ϵ_t. Hence, $\hat{v}(S_t, \mathbf{w})$ converges to a local optimum.
- $G_t = \mathbb{E}[U_t] = v_\pi(S_t)$ by definition.

Semi-gradient Methods

The situation about convergence is more challenging for bootstrapping methods, as the target itself also depends on the parameters being learned!

For the TD(0) target, the gradient-descent update reads

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \Big[\underbrace{R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})}_{U_t \ for \ TD(0)} - \hat{v}(S_t, \mathbf{w}_t) \Big] \nabla \hat{v}(S_t, \mathbf{w}_t).$$

- Robbins-Monro no longer applies!
- Accounts for the effect of the change on the estimate by updating $\mathbf{w}$, but ignores its effect on the target.
- Called **semi-gradient methods.**

Semi-gradient TD(0) for prediction

Input: π, $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ s.t. $\hat{v}(terminal, \cdot) = 0$.
Initialize $\mathbf{w}$
repeat (for each episode)
    Initialize S
    repeat (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

Value approximation with a linear estimator

For a linear value estimator $\hat{v}(S_t, \mathbf{w}) = \boldsymbol{\phi}_t^T \mathbf{w}$, where $\boldsymbol{\phi}_t \in \mathbb{R}^d$ is a $d-$dimensional feature vector for state S_t, the TD(0) semi-gradient update rule is

$$
\begin{aligned}
\mathbf{w}_{t+1} \leftarrow & \mathbf{w}_t + \alpha\Big[R_{t+1} + \gamma \boldsymbol{\phi}_{t+1}^T \mathbf{w}_t - \boldsymbol{\phi}_t^T \mathbf{w}_t\Big]\boldsymbol{\phi}_t \\
& \mathbf{w}_t + \alpha\Big[R_{t+1}\boldsymbol{\phi}_t - \boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1})^T \mathbf{w}_t\Big].
\end{aligned}
$$

At the steady-state, the expected update reads

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

where

$$\mathbf{b} = \mathbb{E}[R_{t+1}\boldsymbol{\phi}_t] \in \mathbb{R}^d, \qquad \mathbf{A} = \mathbb{E}\Big[\boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^T\Big] \in \mathbb{R}^{d\times d}.$$

The linear system converges to $\mathbf{w}_{TD}$ (called the **TD fixpoint**) if the update results in no change

$$\mathbf{b} - \mathbf{A}\mathbf{w}_{TD} = \mathbf{0} \Rightarrow \mathbf{w}_{TD} = \mathbf{A}^{-1}\mathbf{b}.$$

TD error bound for the linear estimator

The linear value estimator of TD satisfies the error bound below

$$\min_{\mathbf{w}} MSVE(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\mathbf{w}} MSVE(\mathbf{w}),$$

which is not such a tight estimator for $\gamma \approx 1$.

Least-squares TD (LSTD)

Actually, why should we do gradient-descent for the linear model? We know from linear regression that it has analytical solution

$$\mathbf{w}_{t+1} = \hat{\mathbf{A}}_t^{-1}\hat{\mathbf{b}}_t,$$

where

$$\hat{\mathbf{A}}_t = \sum_{k=0}^{t} \boldsymbol{\phi}_k(\boldsymbol{\phi}_k - \gamma\boldsymbol{\phi}_{k+1})^T + \epsilon\mathbf{I},$$

$$\hat{\mathbf{b}}_t = \sum_{k=0}^{t} R_{t+1}\boldsymbol{\phi}_k$$

for some $0 < \epsilon << 1$ that guarantees invertibility.

Online learning with LSTD

The Sherman-Morrison formula allows incremental calculation of the matrix inverse

$$(\mathbf{M} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{M}^{-1} - \frac{\mathbf{M}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{M}^{-1}}{1 + \mathbf{v}^T\mathbf{M}^{-1}\mathbf{u}}.$$

Applied to LSTD, we get

$$\begin{aligned}\hat{\mathbf{A}}_t^{-1} &= \Big(\underbrace{\hat{\mathbf{A}}_{t-1}}_{\mathbf{M}} + \underbrace{\phi_t}_{\mathbf{u}}\underbrace{(\phi_t - \gamma\phi_{t+1})}_{\mathbf{v}}{}^T\Big)^{-1} \\ &= \hat{\mathbf{A}}_{t-1}^{-1} - \frac{\hat{\mathbf{A}}_{t-1}^{-1}\phi_t(\phi_t - \gamma\phi_{t+1})^T\hat{\mathbf{A}}_{t-1}^{-1}}{1 + (\phi_t - \gamma\phi_{t+1})^T\hat{\mathbf{A}}_{t-1}^{-1}\phi_t}.\end{aligned}$$

The LSTD algorithm

Input: Feature representation $\phi(s) \in \mathbb{R}^d, \;\; \phi(terminal) = \mathbf{0}$
$\hat{\mathbf{A}}^{-1} \leftarrow \epsilon^{-1}\mathbf{I}$
$\mathbf{b} \leftarrow \mathbf{0}$
repeat (for each episode)
  Initialize S, obtain ϕ
  repeat (for each step of the episode)
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S', obtain ϕ'
    $\mathbf{v} \leftarrow \hat{\mathbf{A}}^{-T}(\boldsymbol{\phi} - \gamma\boldsymbol{\phi}')$
    $\hat{\mathbf{A}}^{-1} \leftarrow \hat{\mathbf{A}}^{-1} - (\hat{\mathbf{A}}^{-1}\boldsymbol{\phi})\mathbf{v}^T/(1 + \mathbf{v}^T\boldsymbol{\phi})$
    $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{b}} + R\boldsymbol{\phi}$
    $\mathbf{w} \leftarrow \hat{\mathbf{A}}^{-1}\hat{\mathbf{b}}$
    $S \leftarrow S', \;\; \boldsymbol{\phi} \leftarrow \boldsymbol{\phi}'$
  until S' is terminal

On-policy control with approximation

As was before, we need to move from $v(s)$ to $q(s, a)$

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

The TD(0) gradient-descent update is

$$\begin{aligned}\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \Big[& R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) \\ & - \hat{q}(S_t, A_t, \mathbf{w}_t) \Big] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t).\end{aligned}$$

Episodic semi-gradient Sarsa for control

Input: a differentiable $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$
Initialize $\mathbf{w} \in \mathbb{R}^d$
repeat (for each episode)
    $S, A \leftarrow$ initial state and action for episode (e.g. $\epsilon-$greedy)
    repeat (for each step of episode)
        Take action A, observe R, S'
        If S' is terminal
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g. $\epsilon-$greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S';\ A \leftarrow A'$

Example: Mountain Car

- Drive a car out of a U-shaped valley.
- Gravity is stronger than the car's engine.
- **Cost-to-go function:** $-\max_a \ \hat{q}(s, a, \mathbf{w})$.
- **Reward:** -1 per time step, +100 for reaching the goal.
- **Actions:**
  - +1 full throttle forward,
  - -1 full throttle backwards,
  - 0 zero throttle.
- The system dynamics are as below

$$
\begin{aligned}
x_{t+1} &\leftarrow x + \dot{x}_{t+1} \\
\dot{x}_{t+1} &\leftarrow \dot{x}_t + 0.001A_t - 0.025\cos(3x_t),
\end{aligned}
$$

where x_t denotes the position and $\dot{x}_t$ the velocity of the car.

Example: Mountain Car

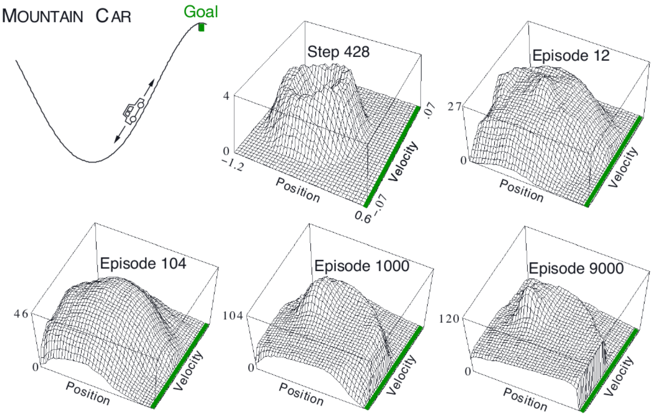

Figure: R. Sutton, A. Barto, MIT Press, 2017

Example: Mountain Car

- Mountain car is a standard application for delayed reward: *Driving towards the exit point is not the right way.*
- Step 428 has a symmetric shape, because all initially visited states are valued worse than the default value unexplored states.
- Consequently, the agent decides to explore for long episodes even though $\epsilon = 0$.

Average reward

- Applicable to continuing problems.
- Discounting causes problems with function approximation.
- The goodness measure for π is average reward / time step

$$\begin{aligned}
\eta(\pi) &\triangleq \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[R_t | A_{0:t-1} \sim \pi] \\
&= \lim_{t\to\infty} \mathbb{E}[R_t | A_{0:t-1} \sim \pi] \\
&= \sum_{s} d_\pi(s) \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a) r,
\end{aligned}$$

where

$$d_\pi(s) \triangleq \lim_{t\to\infty} P[S_t = s | A_{0:t-1} \sim \pi]$$

is the steady-state distribution.

Ergodicity

- The steady-state distribution is independent of the starting state S_0.
- Markov chains with a steady-state distribution are called *ergodic*.
- Taking an action wrt π keeps $d_\pi(s)$ unchanged:

$$\sum_s d_\pi(s) \sum_a \pi(a|s, \mathbf{w}) p(s'|s, a) = d_\pi(s')$$

Differential return

$$G_t \triangleq R_{t+1} - \eta(\pi) + R_{t+2} - \eta(\pi) + \cdots$$

Mind that there is *no discount*! The related value function is also named as the **differential value function**

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a) \Big[r - \eta(\pi) + v_\pi(s') \Big]$$

$$q_\pi(s, a) = \sum_{r,s'} p(s', r|s, a) \Big[r - \eta(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a') \Big]$$

$$v_*(s) = \max_a \sum_{r,s'} p(s', r|s, a) \Big[r - \eta(\pi) + q_*(s', a) \Big]$$

$$q_*(s, a) = \sum_{r,s'} p(s', r|s, a) \Big[r - \eta(\pi) + \max_{a'} \; q_*(s', a') \Big]$$

Differential TD error

For state-value function

$$\delta_t \triangleq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})$$

For action-value function

$$\delta_t \triangleq R_{t+1} - \bar{R}_t + \hat{q}(S_{t+1}, A_t, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})$$

Differential semi-gradient Sarsa for control

Input: a differentiable $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, step sizes $\alpha, \beta > 0$
Initialize $\mathbf{w} \in \mathbb{R}^d$, $\bar{R}$, S, A
repeat (for each step)
    Take action A, observe R, S'
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g. $\epsilon-$greedy)
    $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
    $\bar{R} \leftarrow \bar{R} + \beta\delta$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S';\ A \leftarrow A'$

Futility of the discount factor

$$
\begin{aligned}
J(\pi) &= \sum_s d_\pi(s) v_\pi^\gamma(s) \quad (v_\pi^\gamma(s) \text{ is the discounted value function}) \\
&= \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi^\gamma(s')] \\
&= \eta(\pi) + \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \gamma v_\pi^\gamma(s') \\
&= \eta(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') \sum_s d_\pi(s) \sum_a \pi(a|s) p(s'|s,a) \\
&= \eta(\pi) + \gamma \sum_{s'} v_\pi^\gamma(s') d_\pi(s') \\
&= \eta(\pi) + \gamma J(\pi) = \eta(\pi) + \gamma \eta(\pi) + \gamma^2 J(\pi) \\
&= \eta(\pi) + \gamma \eta(\pi) + \gamma^2 \eta(\pi) + \gamma^3 J(\pi) + \cdots = \frac{1}{1-\gamma} \eta(\pi)
\end{aligned}
$$

Futility of the discount factor

- As $J(\pi) = \dfrac{1}{1-\gamma}\eta(\pi)$, the discount factor scales all policies witht the same factor $1/(1-\gamma)$.
- Hence, does not affect their *ordering*!
- Discounting still matters in the episodic case!