Reinforcement Learning

# Lecture 7: Value Function Approximation

*Lecturer: Haim Permuter* *Scribe: Nave Algarici*

## I. Introduction

## II. Value Function Approximation

In previous lectures, we have represented the value function using a lookup table, where every state $s$ has an entry $V(s)$, or alternatively, every state-action pair $s, a$ has an entry $Q(s, a)$. The problem comes when we start dealing with large MDPs. In such cases, there are to many states and/or actions to store in memory, and the process of learning each state's value individually takes too long. The solution in this case is to Estimate the value function with function approximation:

$$\hat{v}(s, w) \approx v_\pi(s). \tag{1}$$

where $w$ a trainable parameter vector. Using this approximation we can generalize from seen states to unseen states, and update parameter vector $w$ to improve the accuracy of the function approximation.

The function approximator can be one of many options, including: Linear combinations of features, Neural Network, Decision Tree, Nearest Neighbour, etc. We will consider differentiable function approximators: Linear combinations of features and Neural Networks.

### *A. Value Function Approximation By Stochastic Gradient Descent*

**Reminder 1 (Gradient Descent)** Let $J(w)$ be a differentiable function of parameter vector $w$. Define the gradient of $J(w)$ to be

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}. \tag{2}$$

To find a local minimum of $J(w)$, adjust $w$ in the opposite direction of the gradient

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w). \tag{3}$$

Our goal is to find a parameter vector $w$ that minimizes the mean-squared error between the approximate value function $\hat{v}(s, w)$ and the true value function $v_\pi(s)$. The objective function:

$$J(w) = \mathbb{E}_\pi \left[ (v_\pi(S) - \hat{v}(S, w))^2 \right]. \tag{4}$$

Gradient descent finds a local minimum

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w) \tag{5}$$

$$= \alpha \mathbb{E}_\pi \left[ (v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w) \right], \tag{6}$$

and Stochastic gradient descent (SGD) samples the gradient

$$\Delta w = \alpha (v_\pi(S) - \hat{v}(S, w)) \nabla_w \hat{v}(S, w). \tag{7}$$

The expected update of SGD is equal to the full gradient update.
Each state will be represented by a feature vector

$$x(S) = \begin{pmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{pmatrix}. \tag{8}$$

This feature vector can many things, such as: distance of a robot from landmarks, Chess pieces configuration on a board, etc.

**Example 1 (Linear Value function Approximation)** In this case, the value function is represented by a linear combination of features

$$\hat{v}(S, w) = x(S)^T w = \sum_{j=1}^{n} x_j(S) w_j, \tag{9}$$

and the objective function is quadratic in parameters w

$$J(w) = \mathbb{E}_\pi \left[ (v_\pi(S) - x(S)^T w)^2 \right]. \tag{10}$$

By deriving the objective function by $w$

$$\nabla_w \hat{v}(S, w) = x(S), \tag{11}$$

we get the update rule

$$\Delta w = \alpha(v_\pi(S) - x(S)^T w)x(S). \tag{12}$$

**Note 1** Table lookup is a special case of linear value function approximation, where the table lookup features are

$$x^{table}(S) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix}, \tag{13}$$

and the parameter vector w is the value of each individual state

$$\hat{v}(S, w) = \begin{pmatrix} \mathbf{1}(S = s_1), & \ldots, & \mathbf{1}(S = s_n) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}. \tag{14}$$

### *B. Incremental Prediction Algorithms*

Up until this point, we have assumed the true value function $v_\pi(s)$ is given, but in real world problems, we observe only rewards. In practise, we substitute a *target* for $v_\pi(s)$ using different methods. We will discuss four of them.

- For Monte Carlo (MC), the *target* is the return $G_t$

$$\Delta w = \alpha(G_t - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w). \tag{15}$$

- For TD(0), the *target* is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, w)$

$$\Delta w = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w). \tag{16}$$

- For forward-view TD($\lambda$), the *target* is the $lambda$-return $G_t^\lambda$

$$\Delta w = \alpha(G_t^\lambda - \hat{v}(S_t, w))\nabla_w \hat{v}(S_t, w). \tag{17}$$

- For backward-view TD($\lambda$). the equivalent update is

$$\delta_t = R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w), \tag{18}$$

$$E_t = \gamma\lambda E_{t-1} + \nabla_w \hat{v}(S_t, w), \tag{19}$$

$$\Delta w = \alpha\delta_t E_t. \tag{20}$$

## III. Action-Value Function Approximation

In lecture 5, we learned controlling the policy by repeating two steps: policy evaluation and policy improvement. Action-value function approximation is used to perform an approximate policy evaluation, for the same reasons that were mentioned in the introduction. After that, an $\epsilon$-greedy policy improvement can be applied using this approximation. The approximation process of the Action-value function is similar to the value function. The Action-value function approximation:

$$\hat{q}(s, a, w) \approx q_\pi(s, a). \tag{21}$$

The objective function:

$$J(w) = \mathbb{E}_\pi \left[ (q_\pi(S, A) - \hat{q}(S, A, w))^2 \right]. \tag{22}$$

Using SGD we can find a local minimum

$$\Delta w = -\frac{1}{2} \alpha \nabla_w J(w) \tag{23}$$

$$= \alpha (q_\pi(S, A) - \hat{q}(S, A, w)) \nabla_w \hat{q}(S, A, w). \tag{24}$$

Each state-action pair will be represented by a feature vector

$$x(S) = \begin{pmatrix} x_1(S, A) \\ \vdots \\ x_n(S, A) \end{pmatrix}. \tag{25}$$

**Example 2 (Linear Action-Value function Approximation)** In this case, the action-value function is represented by a linear combination of features

$$\hat{q}(S, A, w) = x(S, A)^T w = \sum_{j=1}^{n} x_j(S, A) w_j, \tag{26}$$

The SGD update rule:

$$\nabla_w \hat{q}(S, A, w) = x(S, A), \tag{27}$$

$$\Delta w = \alpha (q_\pi(S, A) - x(S, A)^T w) x(S, A). \tag{28}$$

### A. Incremental Control Algorithms

As in prediction, we will substitute a *target* for $q_\pi(s, a)$.

- For Monte Carlo (MC), the *target* is the return $G_t$

$$\Delta w = \alpha(G_t - \hat{q}(S_t, A_t, w))\nabla_w \hat{q}(S_t, A_t, w). \tag{29}$$

- For TD(0), the *target* is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, A_{t+1}, w)$

$$\Delta w = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w))\nabla_w \hat{q}(S_t, A_t, w). \tag{30}$$

- For forward-view TD($\lambda$), the *target* is the $lambda$-return $G_t^\lambda$

$$\Delta w = \alpha(G_t^\lambda - \hat{q}(S_t, A_t, w))\nabla_w \hat{q}(S_t, A_t, w). \tag{31}$$

- For backward-view TD($\lambda$). the equivalent update is

$$\delta_t = R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w), \tag{32}$$

$$E_t = \gamma\lambda E_{t-1} + \nabla_w \hat{q}(S_t, A_t, w), \tag{33}$$

$$\Delta w = \alpha\delta_t E_t. \tag{34}$$

## IV. Batch Reinforcement Learning

This method addresses the objective of best fitting the value function to the whole training data. Instead of using only the last data point, or *event*, to improve the accuracy of the value function, it can use all of the agent's experience as training data, and improve using it. This method gives us more stability and is more data efficient.

### A. Stochastic Gradient Descent with Experience Replay

In this algorithm we sample randomly a state-value pair from the complete experience, and use it for training the value function approximation.

**Algorithm 1** *SGD Experience Replay*

Given $\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$

**repeat**

Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

Apply SGD update

$$\Delta w = \alpha(v_\pi - \hat{v}(s, w))\nabla_w \hat{v}(s, w)$$

## REFERENCES

[1] UCL course on RL by David Silver, Lecture 6, Value Function Approximation.
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf.