Value Function Approximation II

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

avereshc@buffalo.edu

September 26, 2019

*Slides are based on David Silver Course. Lecture 6

Overview

1. Value Function Approximation

2. Gradient Descent Recap

3. Model Free VFA Prediction / Policy Evaluation

Recap: Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For i = 1, … , H

For all states s in S:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

$$\pi_{i+1}^*(s) \leftarrow \arg\max_{a \in A} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

This is called a value update or Bellman update/back-up

$V_i^*(s)$ = expected sum of rewards accumulated starting from state s, acting optimally for i steps

$\pi_i^*(s)$ = optimal action when in state s and getting to act for i steps

Recap: Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

Impractical for large state spaces

For i = 1, ... , H

For all states s in S:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

$$\pi_{i+1}^*(s) \leftarrow \arg\max_{a \in A} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

This is called a value update or Bellman update/back-up

$V_i^*(s)$ = expected sum of rewards accumulated starting from state s, acting optimally for i steps

$\pi_i^*(s)$ = optimal action when in state s and getting to act for i steps

Table of Contents

What is the main motivation for function approximation?

Large-Scale Reinforcement Learning

- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

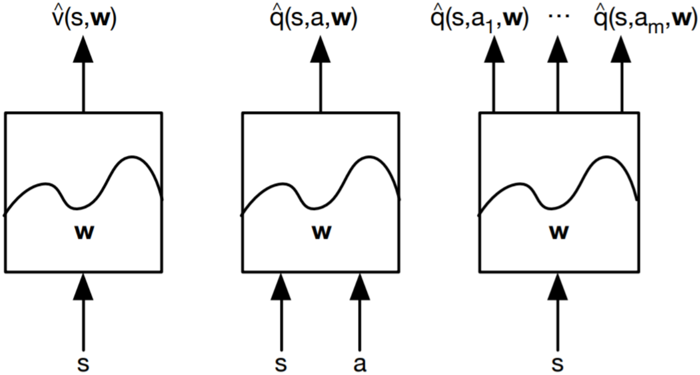
  - *Generalise* from seen states to unseen states
  - Update parameter **w** using MC or TD learning

Motivation for VFA

- Don't want to have to explicitly store or learn for every single state a
  - Dynamics or reward model
  - Value
  - State-action value
  - Policy
- Want more compact representation that generalizes across state or states and actions

Value Function Approximation (VFA)

Represent a (state-action/state) value function with a parameterized function instead of a table



Which function approximator?

Linear combinations of features

Table of Contents

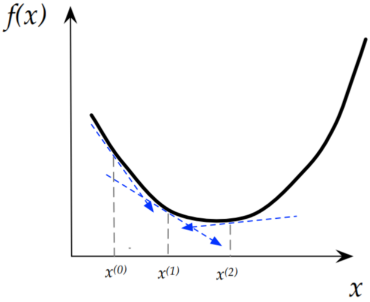
Recap: Gradient Descent

1. Start at an arbitrary point
2. Move along the gradient at that point towards the next point
3. Repeat until (hopefully) converging to a stationary point

Gradient Descent Algo

Algorithm 1 Gradient Descent

```
1: Guess x^(0), set k ← 0
2: while ||∇f(x^(k))|| ≥ ε do
3:     x^(k+1) = x^(k) − t_k ∇f(x^(k))
4:     k ← k + 1
5: end while
6: return x^(k)
```



Recap: Gradient Descent Example

Consider the problem of minimizing

$$f(x,y) = 4x^2 - 4xy + 2y^2$$

Notice that the optimal solution is $(x,y) = (0,0)$. Solution:

1. Compute the gradient:

$$\nabla_f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} =$$

Recap: Gradient Descent Example

Consider the problem of minimizing

$$f(x, y) = 4x^2 - 4xy + 2y^2$$

Notice that the optimal solution is $(x, y) = (0, 0)$. Solution:

1. Compute the gradient:

$$\nabla_f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 8x - 4y \\ \end{bmatrix}$$

Recap: Gradient Descent Example

Consider the problem of minimizing

$$f(x, y) = 4x^2 - 4xy + 2y^2$$

Notice that the optimal solution is $(x, y) = (0, 0)$. Solution:

1. Compute the gradient:

$$\nabla_f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 8x - 4y \\ -4x + 4y \end{bmatrix}$$

2. Start from the point $(x^{(0)}, y^{(0)}) = (2, 3)$, $\alpha = 0.5$

$$(x^{(1)}, y^{(1)}) = (x^{(0)}, y^{(0)}) - \alpha \nabla f(x^{(0)}, y^{(0)})$$

Recap: Gradient Descent Example

Consider the problem of minimizing

$$f(x, y) = 4x^2 - 4xy + 2y^2$$

Notice that the optimal solution is $(x, y) = (0, 0)$. Solution:

1. Compute the gradient:

$$\nabla_f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 8x - 4y \\ -4x + 4y \end{bmatrix}$$

2. Start from the point $(x^{(0)}, y^{(0)}) = (2, 3)$, $\alpha = 0.5$

$$(x^{(1)}, y^{(1)}) = (x^{(0)}, y^{(0)}) - \alpha \nabla f(x^{(0)}, y^{(0)})$$
$$= \begin{bmatrix} 2 \\ 3 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

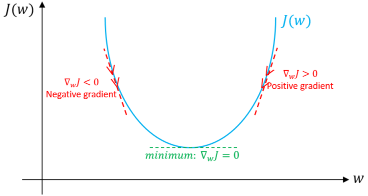
Recap: Gradient Descent

Let $J(w)$ be a differentiable function of parameter vector $\mathbf{w}$

Goal: find parameter $\mathbf{w}$ that minimizes J

1. Define the gradient of $J(w)$ to be

$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{bmatrix}$$



2. To find a local minimum of $J(w)$ adjust $\mathbf{w}$ in direction of -ve gradient

$$\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w})$$

where α is a step-size parameter

Stochastic Gradient Descent

Goal: Find the parameter vector **w** that minimizes the loss between a true value function $V_\pi(s)$ and its approximation $\hat{V}(s; \mathbf{w})$.

$$J(\mathbf{w}) = E_\pi[(V_\pi(S) - \hat{V}(S, \mathbf{w}))^2]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha E_\pi[(V_\pi(S) - \hat{V}(S, \mathbf{w}))\nabla_w \hat{V}(S, \mathbf{w})] \end{aligned}$$

Stochastic Gradient Descent

Goal: Find the parameter vector $\mathbf{w}$ that minimizes the loss between a true value function $V_\pi(s)$ and its approximation $\hat{V}(s;\mathbf{w})$.

$$J(\mathbf{w}) = E_\pi[(V_\pi(S) - \hat{V}(S,\mathbf{w}))^2]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta\mathbf{w} &= -\frac{1}{2}\alpha\nabla_{\mathbf{w}}J(\mathbf{w}) \\ &= \alpha E_\pi[(V_\pi(S) - \hat{V}(S,\mathbf{w}))\nabla_w \hat{V}(S,\mathbf{w})]\end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta\mathbf{w} = \alpha(V_\pi(S) - \hat{V}(S,\mathbf{w}))\nabla_w \hat{V}(S,\mathbf{w})$$

- Expected update is equal to full gradient update

Table of Contents

Model Free VFA Prediction / Policy Evaluation

- Recall model-free policy evaluation
  - Following a fixed policy π
  - Goal is to estimate V_π and/or Q_π
- Maintain a look up table to store estimates V_π and/or Q_π
- Update these estimates after each episode

Model Free VFA Prediction / Policy Evaluation

- Recall model-free policy evaluation
  - Following a fixed policy π
  - Goal is to estimate V_π and/or Q_π
- Maintain a look up table to store estimates V_π and/or Q_π
- Update these estimates after each episode (Monte Carlo methods) or after each step

Model Free VFA Prediction / Policy Evaluation

- Recall model-free policy evaluation
  - Following a fixed policy π
  - Goal is to estimate V_π and/or Q_π
- Maintain a look up table to store estimates V_π and/or Q_π
- Update these estimates after each episode (Monte Carlo methods) or after each step (TD methods)
- **Now: in value function approximation, change the estimate update step to include fitting the function approximator**

Feature Vectors

- Represent state by a *feature vector*

$$x(S) = \begin{bmatrix} x_1(S) \\ \vdots \\ x_n(S) \end{bmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S) \mathbf{w}_j$$

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S) \mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = E_\pi \left[(V_\pi(S) - x(S)^T \mathbf{w})^2 \right]$$

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = E_\pi\left[(V_\pi(S) - x(S)^T \mathbf{w})^2\right]$$

- Stochastic gradient descent converges on global optimum

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S) \mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = E_\pi \left[(V_\pi(S) - x(S)^T \mathbf{w})^2 \right]$$

- Stochastic gradient descent converges on global optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{V}(S, \mathbf{w}) = x(S)$$

$$\Delta \mathbf{w} = \alpha (V_\pi(S) - \hat{V}(S, \mathbf{w})) x(S)$$

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S) \mathbf{w}_j$$

- Objective function is quadratic in parameters **w**

$$J(\mathbf{w}) = E_\pi \left[(V_\pi(S) - x(S)^T \mathbf{w})^2 \right]$$

- Stochastic gradient descent converges on global optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{V}(S, \mathbf{w}) = x(S)$$

$$\Delta \mathbf{w} = \alpha (V_\pi(S) - \hat{V}(S, \mathbf{w})) x(S)$$

Update = step-size x prediction error x feature value

Table Lookup Features

- Table lookup is a special case of linear value function approximation
- Using *table lookup features*

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

- Parameter vector $\mathbf{w}$ gives value of each individual state

$$\hat{v}(S, \mathbf{w}) = \begin{pmatrix} \mathbf{1}(S = s_1) \\ \vdots \\ \mathbf{1}(S = s_n) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_n \end{pmatrix}$$

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

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

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

Monte-Carlo with Value Function Approximation

- Return G_t is an unbiased, noisy sample of true value $v_\pi(S_t)$
- Can therefore apply supervised learning to "training data":

$$\langle S_1, G_1 \rangle, \langle S_2, G_2 \rangle, ..., \langle S_T, G_T \rangle$$

- For example, using *linear Monte-Carlo policy evaluation*

$$\begin{aligned}\Delta \mathbf{w} &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w}) \\ &= \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)\end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

TD with Value Function Approximation

- The TD-target $R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})$ is a *biased* sample of true value $v_\pi(S_t)$
- Can still apply supervised learning to "training data":

$$\langle S_1, R_2 + \gamma\hat{v}(S_2, \mathbf{w})\rangle, \langle S_2, R_3 + \gamma\hat{v}(S_3, \mathbf{w})\rangle, ..., \langle S_{T-1}, R_T\rangle$$

- For example, using *linear TD(0)*

$$\begin{aligned}\Delta\mathbf{w} &= \alpha({\color{red}R + \gamma\hat{v}(S', \mathbf{w})} - \hat{v}(S, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S, \mathbf{w}) \\ &= \alpha\delta\mathbf{x}(S)\end{aligned}$$

- Linear TD(0) converges (close) to global optimum

TD with Value Function Approximation

- The λ-return G_t^λ is also a biased sample of true value $v_\pi(s)$
- Can again apply supervised learning to "training data":

$$\left\langle S_1, G_1^\lambda \right\rangle, \left\langle S_2, G_2^\lambda \right\rangle, ..., \left\langle S_{T-1}, G_{T-1}^\lambda \right\rangle$$

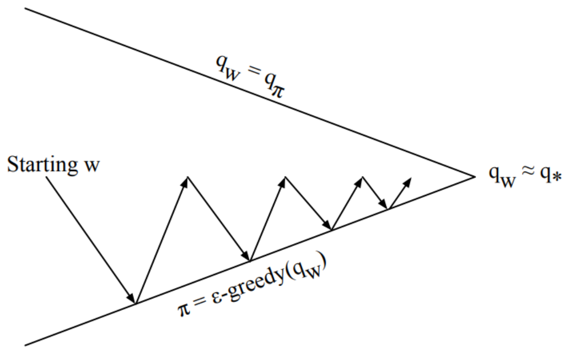
- Forward view linear TD(λ)

$$\begin{aligned} \Delta \mathbf{w} &= \alpha({\color{red}G_t^\lambda} - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w}) \\ &= \alpha({\color{red}G_t^\lambda} - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t) \end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \\ E_t &= \gamma\lambda E_{t-1} + \mathbf{x}(S_t) \\ \Delta \mathbf{w} &= \alpha\delta_t E_t \end{aligned}$$

Control with Value Function Approximation



Policy evaluation **Approximate** policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimise mean-squared error between approximate action-value fn $\hat{q}(S, A, \mathbf{w})$ and true action-value fn $q_\pi(S, A)$

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

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2}\nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

$$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state *and* action by a *feature vector*

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value fn by linear combination of features

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$$

- Stochastic gradient descent update

$$\begin{aligned} \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) &= \mathbf{x}(S, A) \\ \Delta \mathbf{w} &= \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A) \end{aligned}$$

Incremental Control Algorithms

- Like prediction, we must substitute a *target* for $q_\pi(S, A)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

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

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

  - For forward-view TD(λ), target is the action-value λ-return

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w})$$

  - For backward-view TD(λ), equivalent update is

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \\
E_t &= \gamma\lambda E_{t-1} + \nabla_{\mathbf{w}}\hat{q}(S_t, A_t, \mathbf{w}) \\
\Delta \mathbf{w} &= \alpha\delta_t E_t
\end{aligned}$$

Convergence of Prediction Algorithms

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗