Lecture 9: Value Function Approximation

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

February 25, 2020

Announcement

1. Homework 2 will be released by the end of tomorrow, due in 2 weeks
2. Tomorrow after the lecture we will have a project proposal feedback session and Homework 1 review session to be joined by TA Zhenghao Peng

Today's Plan

1. Introduction on function approximation
2. Value function approximation for prediction
3. Value function approximation for control
4. Batch RL and least square prediction and control

Introduction: Scaling up RL

1. Previous lectures on small RL problems:
    1. Cliff walk: 4×16 states
    2. Mountain car: 1600 states
    3. Tic-Tac-Toe: 10^3 states
2. Large-scale problems:
    1. Backgammon: 10^{20} states
    2. Chess: 10^{47} states
    3. Game of Go: 10^{170} states
    4. Robot Arm and Helicopter have sontinuous state space
    5. Number of atomics in universe: 10^{80}
3. Challenge: How can we scale up the model-free methods for prediction and control?

Introduction: Scaling up RL

1. In tabular methods we represent value function by a lookup table:
    1. Every state s has an entry $V(s)$
    2. Every state-action pair s, a has an entry $Q(s, a)$
2. Challenges with large MDPs:
    1. too many states or actions to store in memory
    2. too slow to learn the value of each state individually

Scaling up RL with Function Approximation

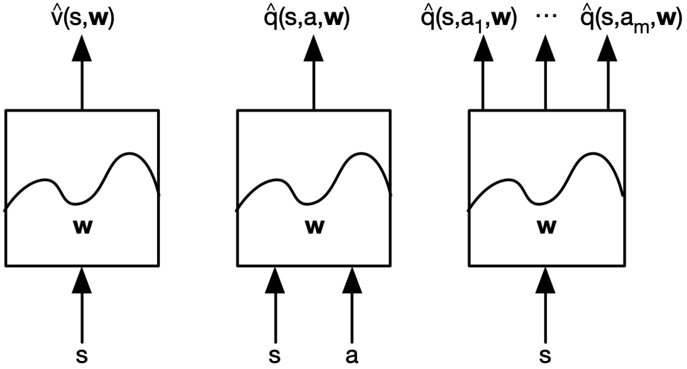
1. How to avoid explicitly learning or storing for every single state:
    1. Dynamics or reward model
    2. Value function, state-action function
    3. Policy
2. Solution: Estimate with function approximation

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

    1. Generalize from seen states to unseen states
    2. Update the parameter **w** using MC or TD learning

Types of value function approximation

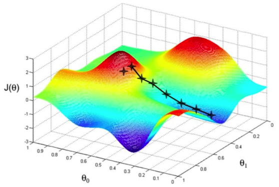
Several function designs:


$\hat{v}(s,\mathbf{w})$
$\hat{q}(s,a,\mathbf{w})$
$\hat{q}(s,a_1,\mathbf{w})$ ··· $\hat{q}(s,a_m,\mathbf{w})$
w
w
w
s
s
a
s

Function Approximators

1. Many possible function approximators:
    1. Linear combinations of features
    2. Neural networks
    3. Decision trees
    4. Nearest neighbors
2. We will focus on function approximators that are differentiable
    1. Linear feature representations
    2. Neural networks

Review on Gradient Descend



1. Consider a function $J(\mathbf{w})$ that is a differentiable function of a parameter vector $\mathbf{w}$
2. Goal is to find parameter $\mathbf{w}^*$ that minimizes J
3. Define the gradient of $J(\mathbf{w})$ to be

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \left(\frac{\partial J(\mathbf{w})}{\partial w_1}, \frac{\partial J(\mathbf{w})}{\partial w_2}, ..., \frac{\partial J(\mathbf{w})}{\partial w_n}\right)^T$$

4. Adjust $\mathbf{w}$ in the direction of the negative gradient, where α is step-size

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

Value Function Approximation with an Oracle

1. We assume that we have the oracle for knowing the true value for $v_\pi(s)$ for any given state s
2. Then the objective is to find the best approximate representation of $v_\pi(s)$
3. Thus use the mean squared error and define the loss function as

$$J(\mathbf{w}) = \mathbb{E}_\pi\Big[\big(v_\pi(S) - \hat{v}(s, \mathbf{w})\big)^2\Big]$$

4. Follow the gradient descend to find a local minimum

$$\begin{aligned}\Delta\mathbf{w} =& -\frac{1}{2}\alpha\nabla_{\mathbf{w}}J(\mathbf{w})\\ \mathbf{w}_{t+1} =& \mathbf{w}_t + \Delta\mathbf{w}\end{aligned}$$

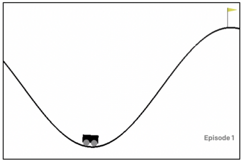
Representing State with Feature Vectors

1. Represent state using a feature vector

$$\mathbf{x}(S) = \big(x_1(S), ..., x_n(S)\big)^T$$

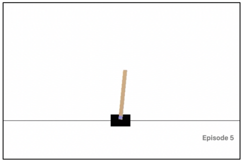
2. For example:

Mountain Car



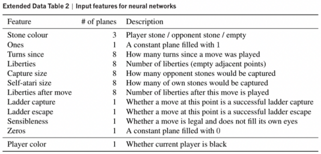
Position of car, velocity of car

Cart Pole



Position of cart, velocity of cart, angle of pole, rotation rate of pole

Game of Go in AlphaGo



Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

48 places of 19x19 feature maps

Linear Value Function Approximation

1. Represent value function by a linear combination of features

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

2. The objective function is quadratic in parameter **w**

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

3. Thus the update rule is as simple as

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

$Update = StepSize \times PredictionError \times FeatureValue$

4. Stochastic gradient descent converges to global optimum. Because in the linear case, there is only one optimum, thus local optimum is automatically converge to or near the global optimum.

Linear Value Function Approximation with Table Lookup Feature

1. Table lookup is a special case of linear value function approximation
2. Table lookup feature is one-hot vector as follows

$$\mathbf{x}^{table}(S) = \left(\mathbf{1}(S = s_1), ..., \mathbf{1}(S = s_n)\right)^T$$

3. Then we can see that each element on the parameter vector **w** indicates the value of each individual state

$$\hat{v}(S, \mathbf{w}) = \left(\mathbf{1}(S = s_1), ..., \mathbf{1}(S = s_n)\right)\left(w_1, ..., w_n\right)^T$$

4. Thus we have $\hat{v}(s_k, \mathbf{w}) = w_k$

Value Function Approximation for Model-free Prediction

1. In practice, no access to oracle of the true value $v_\pi(s)$ for any state s
2. Recall model-free prediction
    1. Goal is to evaluate v_π following a fixed policy π
    2. A lookup table is maintained to store estimates v_π or q_π
    3. Estimates are updated after each episode (MC method) or after each step (TD method)
3. Thus what we can do is to **include the function approximation step in the loop**

Incremental VFA Prediction Algorithms

1. We assumed that true value function $v_\pi(s)$ given by supervisor/oracle

$$\Delta \mathbf{w} = \alpha\Big(v_\pi(S) - \hat{v}(S, \mathbf{w})\Big)\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

2. But in RL there is no supervisor, only rewards
3. In practice, we substitute the target for $v_\pi(s)$
    1. For MC, the target is the actual return G_t

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

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

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

Monte-Carlo Prediction with VFA

1. Return G_t is an unbiased, but noisy sample of true value $v_\pi(S_t)$
2. Why unbiased? $\mathbb{E}[G_t] = v_\pi(S_t)$
3. So we have the training data that can be used for supervised learning in VFA:

$$< S_1, G_1 >, < S_2, G_2 >, ..., < S_T, G_T >$$

4. Using linear Monte-Carlo policy evaluation

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

5. Monte-Carlo prediction converges, in both linear and non-linear value function approximation.

TD Prediction with VFA

1. TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$ is a biased sample of true value $v_\pi(S_t)$
2. Why biased? It is drawn from our previous estimate, rather than the true value: $\mathbb{E}[R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})] \neq v_\pi(S_t)$
3. We have the training data used for supervised learning in VFA:

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

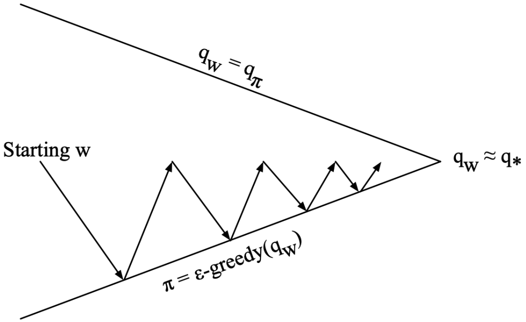
4. Using linear TD(0), the stochastic gradient descend update is

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

   This is also called as semi-gradient, as we ignore the effect of changing the weight vector **w** on the target
5. Linear TD(0) converges(close) to global optimum

Control with Value Function Approximation

Generalized policy iteration



1. Policy evaluation: approximate policy evaluation, $\hat{q}(., ., \mathbf{w}) \approx q_\pi$
2. Policy improvement: ϵ-greedy policy improvement

Action-Value Function Approximation

1. Approximate the action-value function

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

2. Minimize the MSE (mean-square error) between approximate action-value and true action-value (assume oracle)

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

3. Stochastic gradient descend to find a local minimum

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

Linear Action-Value Function Approximation

1. Represent state and action using a feature vector

$$\mathbf{x}(S,A) = \big(x_1(S,A), ..., x_n(S,A)\big)^T$$

2. Represent action-value function by a linear combination of features

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

3. Thus the stochastic gradient descend update

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

Incremental Control Algorithm

Same to the prediction, there is no oracle for the true value $q_\pi(S, A)$, so we substitute a target

1. For MC, the target is the return G_t

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

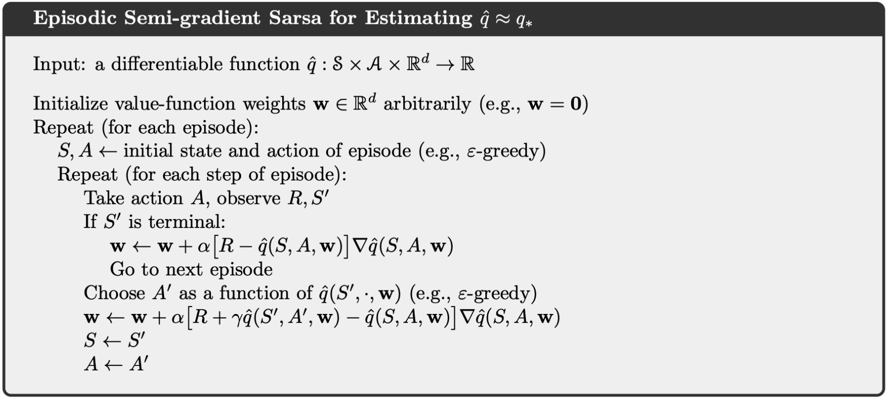
2. For Sarsa, the target is the TD target $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})$

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

3. For Q-learning, the target is the TD target $R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w})$

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

Semi-gradient Sarsa for VFA Control

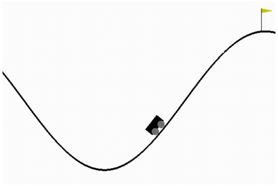


Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

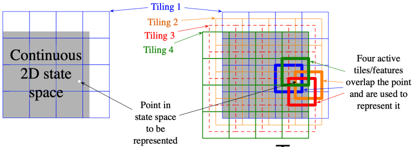
```
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
    $S, A \leftarrow$ initial state and action of episode (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R, S'$
        If $S'$ is terminal:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
            Go to next episode
        Choose $A'$ as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., $\varepsilon$-greedy)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
        $S \leftarrow S'$
        $A \leftarrow A'$
```

Mountain Car Example



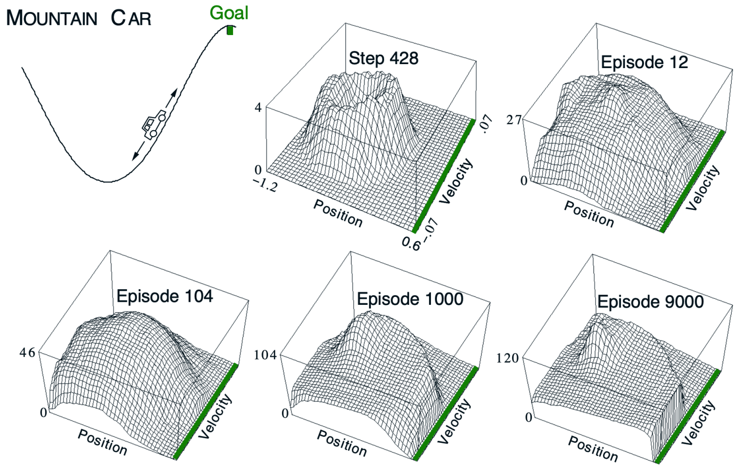
1. Control task: move the car up the hill with forward and backward acceleration
   1. Continuous state: [position of the car, velocity of the car]
   2. Action: [full throttle forward, full throttle backward, zero throttle]
2. Grid-tiling coding of the continuous 2D state space with 4 tilings


Continuous 2D state space
Tiling 1
Tiling 2
Tiling 3
Tiling 4
Point in state space to be represented
Four active tiles/features overlap the point and are used to represent it

3. Q function approximator: $\hat{q}(s, a, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s, a) = \sum_{i=1}^{4} w_i x_i(s, a)$

Mountain Car Example

1. Visualization of cost-to-go function ($-\max_a \hat{q}(s, a, \mathbf{w})$) learned over episodes



Mountain Car Example

1. Code example: `https://github.com/cuhkrlcourse/RLexample/blob/master/modelfree/q_learning_mountaincar.py`

Convergence of Control Methods with VFA

1. For Sarsa,

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

2. For Q-learning,

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

3. TD with VFA doesn't follow the gradient of any objective function
4. The updates involve doing an approximate Bellman backup followed by fitting the underlying value function
5. That is why TD can diverge when off-policy or using non-linear function approximation
6. Challenge for off-policy control: behavior policy and target policy are not identical, thus value function approximation can diverge

The Deadly Triad for the Danger of Instability and Divergence

Potential problematic issues:

1. **Function approximation**: A scalable way of generalizing from a state space much larger than the memory and computational resources
2. **Bootstrapping**: Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods)
3. **Off-policy training**: training on a distribution of transitions other than that produced by the target policy
4. See details at Textbook Chapter 11.3

Convergence of Control Methods

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	$\surd$	$(\surd)$	X
Sarsa	$\surd$	$(\surd)$	X
Q-Learning	$\surd$	X	X

$(\surd)$ moves around the near-optimal value function

Batch Reinforcement Learning

1. Incremental gradient descend update is simple
2. But it is not sample efficient
3. Batch-based methods seek to find the best fitting value function for a batch of the agent's experience

Least Square Prediction

1. Given the value function approximation $\hat{v}(s, \mathbf{w}) \approx v^{\pi}(s)$
2. The experience $\mathcal{D}$ consisting of $< state, value >$ pairs (may from one episode or many previous episodes)

$$\mathcal{D} = \{ < s_1, v_1^{\pi} >, ..., < s_T, v_T^{\pi} > \}$$

3. Objective: To optimize the parameter $\mathbf{w}$ that best fit all the experience $\mathcal{D}$
4. Least squares algorithms are used to minimize the sum-squared error between $\hat{v}(s_t, \mathbf{w})$ and the target values v_t^{π}

$$\begin{aligned}\mathbf{w}^* &= \arg\min_{\mathbf{w}} \mathbb{E}_{\mathcal{D}}[(v^{\pi} - \hat{v}(s, \mathbf{w})^2] \\ &= \arg\min_{\mathbf{w}} \sum_{t=1}^{T} (v_t^{\pi} - \hat{v}(s_t, \mathbf{w}))^2\end{aligned}$$

Stochastic Gradient Descent with Experience Replay

1. Given the experience consisting of $< state, value >$ pairs

$$\mathcal{D} = \{ < s_1, v_1^\pi >, ..., < s_T, v_T^\pi > \}$$

2. Iterative solution could be to repeat the following two steps
    1. Randomly sample one pair $< state, value >$ from the experience

    $$< s, v^\pi > \sim \mathcal{D}$$

    2. Apply the stochastic gradient descent update

    $$\Delta \mathbf{w} = \alpha (v^\pi - \hat{v}(s, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

3. The solution from the gradient descend method converges to the least squares solution

$$\mathbf{w}^{LS} = \arg\min_{\mathbf{w}} \sum_{t=1}^{T} (v_t^\pi - \hat{v}(s_t, \mathbf{w}))^2$$

Solving Linear Least Squares Prediction

1. Experience replay finds least squares solution, but it may take many iterations
2. Using linear value function approximation $\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w}$, we can solve the least squares solution directly

Solving Linear Least Squares Prediction 2

1. At minimum of LS($\mathbf{w}$), the expected update must be zero

$$\mathbb{E}_{\mathcal{D}}[\Delta \mathbf{w}] = 0$$

2. thus

$$\Delta \mathbf{w} = \alpha \sum_{t=1}^{T} \mathbf{x}(s_t)(v_t^{\pi} - \mathbf{x}(x_t)^T \mathbf{w}) = 0$$

$$\sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi} = \sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^T \mathbf{w}$$

$$\mathbf{w} = \Big(\sum_{t=1}^{T} \mathbf{x}(s_t)\mathbf{x}(s_t)^T \Big)^{-1} \sum_{t=1}^{T} \mathbf{x}(s_t) v_t^{\pi}$$

3. For N features, the matrix inversion complexity is $O(N^3)$

Linear Least Squares Prediction Algorithms

1. Again, we do not know the true values v_t^π
2. In practice, the training data must use noisy or biased samples of v_t^π
    1. LSMC: Least squares Monte-Carlo uses return $v_t^\pi \approx G_t$
    2. LSTD: Least squares TD uses TD target $v_t^\pi \approx R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

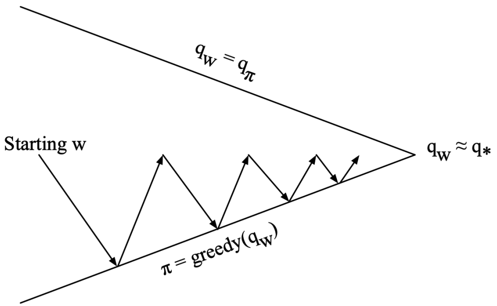
Linear Least Squares Prediction Algorithms

LSMC
$$0 = \sum_{t=1}^{T} \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$
$$\mathbf{w} = \Big(\sum_{t=1}^{T} \mathbf{x}(S_t)\mathbf{x}(S_t)^T \Big)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) G_t$$

LSTD
$$0 = \sum_{t=1}^{T} \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\mathbf{x}(S_t)$$
$$\mathbf{w} = \Big(\sum_{t=1}^{T} \mathbf{x}(S_t)(\mathbf{x}(S_t) - \gamma \mathbf{x}(S_{t+1}))^T \Big)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t) R_{t+1}$$

Least Square Control with Value Function Approximation

Generalized policy iteration



1. Policy evaluation: Policy evaluation by least squares Q-learning
2. Policy improvement: greedy policy improvement

Least Squares Action-Value Function Approximation

1. Approximate action-value function $q_{\pi}(s, a)$
2. using linear combination of features $\mathbf{x}(s, a)$

$$\hat{q}(s, a, \mathbf{w}) = \mathbf{x}(s, a)^T \mathbf{w} \approx q_{\pi}(s, a)$$

3. minimize least squares error between $\hat{q}(s, a, \mathbf{w})$ and $q_{\pi}(s, a)$
4. The experience is generate from using policy π, consisting of $< state, action, action - value >$

Least Squares Control

1. For policy evaluation we want to efficiently use all the experience
2. For control we want to improve the policy
3. The experience is generated from many policies (previous old policies)
4. So to evaluate $q_\pi(S, A)$ we must learn off-policy
5. We follow the same idea of Q-Learning:
    1. Use the experience generated by old policy:

    $$S_t, A_t, R_{t+1}, S_{t+1} \sim \pi_{old}$$

    2. Consider the alternative successor action $A' = \pi_{new}(S_{t+1})$ using greedy
    3. Update $\hat{q}(S_t, A_t, \mathbf{w})$ towards value of alternative action $R_{t+1} + \gamma \hat{q}(S_{t+1}, A', \mathbf{w})$

Least Squares Q-Learning

1. Consider the following linear Q-learning update

$$\begin{aligned}\delta =& R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \\ \Delta \mathbf{w} =& \alpha \delta \mathbf{x}(S_t, A_t)\end{aligned}$$

2. LSTDQ algorithm: solve the total sum of the gradient = zero:

$$\begin{aligned}0 =& \sum_{t=1}^{T} \alpha (R_{t+1} + \gamma \hat{q}(S_{t+1}, \pi(S_{t+1}), \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \mathbf{x}(S_t, A_t) \\ \mathbf{w} =& \Big(\sum_{t=1}^{T} \mathbf{x}(S_t, A_t)(\mathbf{x}(S_t, A_t) - \gamma \mathbf{x}(S_{t+1}, \pi(S_{t+1})))^T \Big)^{-1} \sum_{t=1}^{T} \mathbf{x}(S_t, A_t) R_{t+1}\end{aligned}$$

Least Squares Policy Iteration Algorithm

1. Use LSTDQ for policy evaluation
2. Repeatedly re-evaluate experience $\mathcal{D}$ with different policies

```
function LSPI-TD($\mathcal{D}, \pi_0$)
    $\pi' \leftarrow \pi_0$
    repeat
        $\pi \leftarrow \pi'$
        $Q \leftarrow$ LSTDQ($\pi, \mathcal{D}$)
        for all $s \in \mathcal{S}$ do
            $\pi'(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$
        end for
    until ($\pi \approx \pi'$)
    return $\pi$
end function
```

Convergence of Control Methods

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	$\surd$	$(\surd)$	X
Sarsa	$\surd$	$(\surd)$	X
Q-Learning	$\surd$	X	X
LSPI	$\surd$	$(\surd)$	-

$(\surd)$ moves around the near-optimal value function

Summary

1. Today's content:
    1. Value function approximation for prediction
    2. Value function approximation for control
    3. Least square prediction and control
2. Next lecture: Deep Q-Learning
    1. DeepMind's **Nature** paper: Mnih, Volodymyr; et al. (2015). **Human-level control through deep reinforcement learning**
    2. Entering the period of modern RL: deep learning + reinforcement learning