Value Function Approximation

岩延 yany@ucas.ac.cn

Value Function Approximation

Introduction

- **Large-Scale Reinforcement Learning**
  - Reinforcement learning can be used to solve large problems, e.g.
    - Computer Go: 10^{170} states
    - Helicopter: continuous state space
- **Problem with large MDPs:**
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- **How can we scale up the RL methods for prediction and control?**

Value Function Approximation

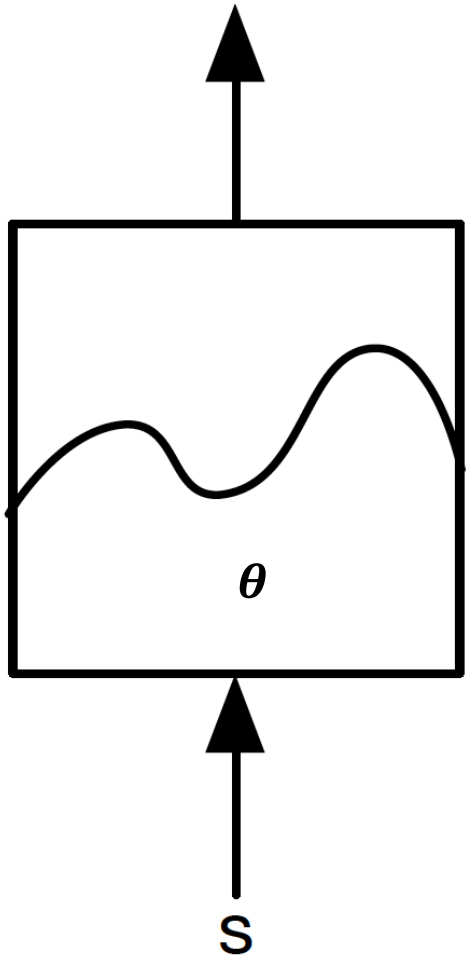
Value Function Approximation

- **Policy Evaluation (the prediction problem)**
  - for a given policy π, compute the state-value function V_π
- **So far value functions were stored in lookup tables**
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s,a)$
- **Value Function Approximation**
  - Solution for large MDPs
  - the value function estimate at time t, $\boldsymbol{V}_t$, depends on a parameter vector $\boldsymbol{\theta}_t$
  - Estimate value function with function approximation
    - $V(s,\boldsymbol{\theta}) \approx V_\pi(s)$
    - $Q(s,a,\boldsymbol{\theta}) \approx Q_\pi(s,a)$
  - Generalize from seen states to unseen states
  - Only the parameter vector is updated

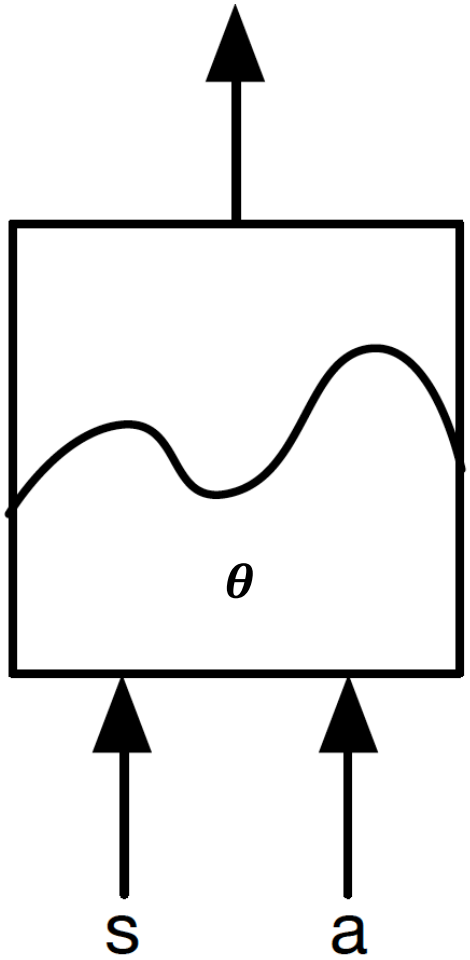
Value Function Approximation

Types of Value Function Approximation

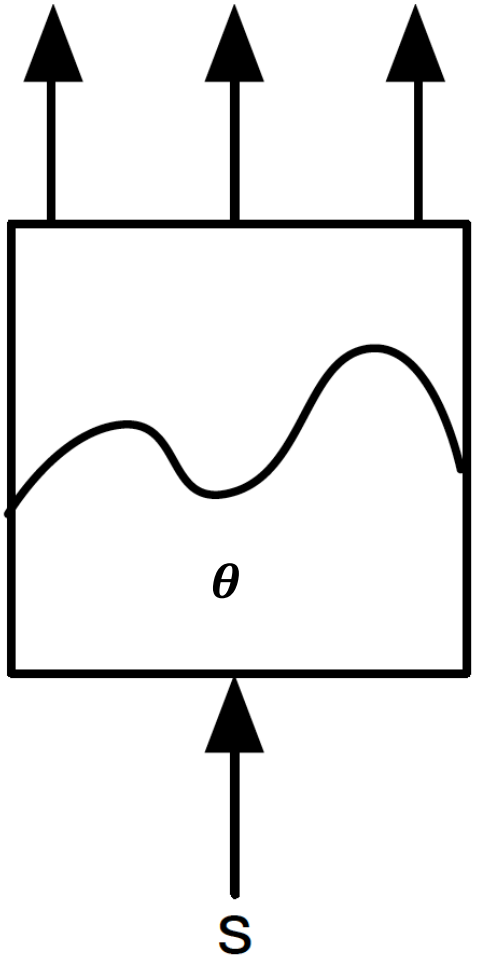
- $V(s, \boldsymbol{\theta})$



$Q(s, a, \boldsymbol{\theta})$



$Q(s, a_1, \boldsymbol{\theta}) Q(s, a_2, \boldsymbol{\theta})$



Value Function Approximation

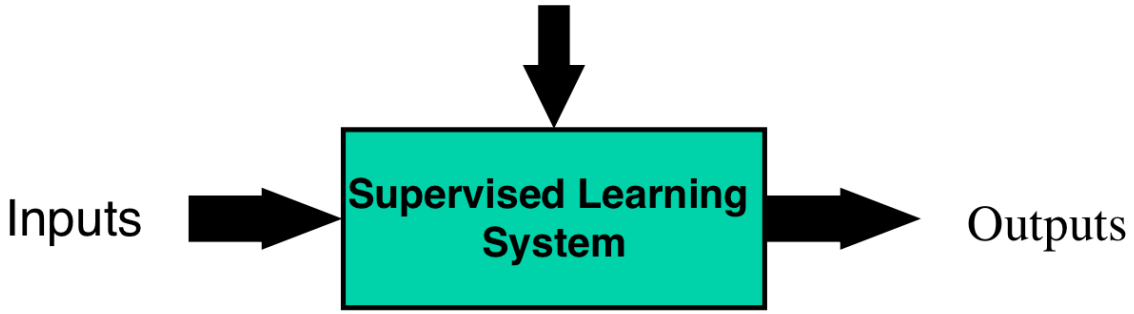
Backups as Training Examples

- **Backup** $s \rightarrow u$
  - estimated value function update its value at particular states toward a backed-up value
  - TD(0):
    - $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
    - $S_t \rightarrow R_{t+1} + \gamma V(S_{t+1})$
  - MC: $S_t \rightarrow G_t$
  - n-step TD: $S_t \rightarrow G_{t:t+n}$
  - DP: $S_t \rightarrow \mathrm{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]$

Value Function Approximation

Adapt Supervised Learning Algorithms

- **Training Info = desired (target) outputs**



- **Training example = {input, target output}**
- **Error = (target output – actual output)**
- **e.g., TD(0) backup:**
  - $V(S_t) \leftarrow V(S_t)+\alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
  - $S_t \rightarrow R_{t+1} + \gamma V(S_{t+1})$
- **As a training example**
  - $\{S_t, R_{t+1} + \gamma V(S_{t+1})\}$

Value Function Approximation

Which Function Approximator?

- **There are many function approximators, e.g.**
  - Linear combinations of features
  - Neural network
  - Decision tree
  - Nearest neighbor
  - Fourier / wavelet bases
  - …
- **But RL has some special requirements:**
  - usually want to learn while interacting
  - ability to handle nonstationarity
  - other?

Value Function Approximation

Which Function Approximator?

- **We consider differentiable function approximators, e.g.**
  - **Linear combinations of features**
  - **Neural network**
  - Decision tree
  - Nearest neighbor
  - Fourier / wavelet bases
  - ...
- **Furthermore, we require a training method that is suitable for non-stationary, non-iid data**

Value Function Approximation

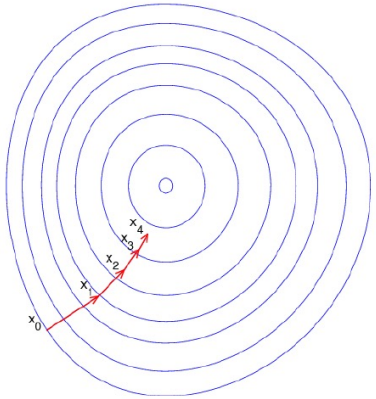
Performance Measures

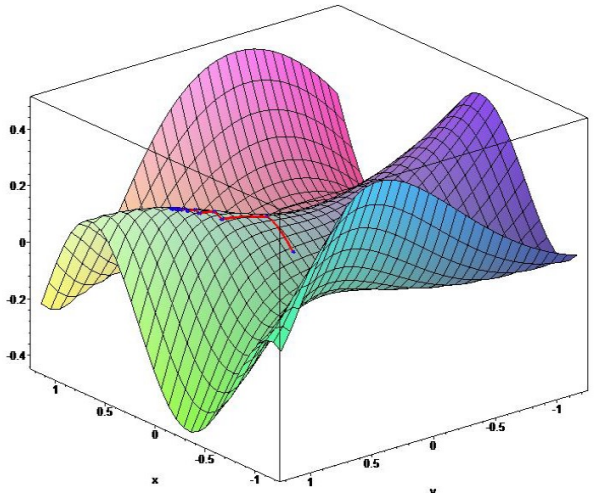
- **The Prediction Objective**
  - a common and simple one is the mean-squared error (MSE) over a state distribution μ under policy π
    - $\text{MSE}(\boldsymbol{\theta}) = \sum_{s \in S} \mu(s)[V_\pi(s) - V(s, \boldsymbol{\theta})]^2$
    - $= \text{E}_\pi[[V_\pi(s) - V(s, \boldsymbol{\theta})]^2]$
- **The on-policy distribution**
  - the distribution created while following the policy being evaluated. Stronger results are available for this distribution.
- **Global optimum and local optimum**
  - $\text{MSE}(\boldsymbol{\theta}^*) \leq \text{MSE}(\boldsymbol{\theta})$

Value Function Approximation

Gradient Descent

- Let $J(\boldsymbol{\theta})$ be a differentiable function of parameter vector $\boldsymbol{\theta}$
  - $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)^T$
- Define the gradient of $J(\boldsymbol{\theta})$
  - $\nabla_\theta J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial j(\theta)}{\partial \theta_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial j(\theta)}{\partial \theta_n} \end{pmatrix}$
- To find a local minimum of $J(\boldsymbol{\theta})$
- Adjust $\boldsymbol{\theta}$ in direction of gradient
  - $\Delta\boldsymbol{\theta} = -\frac{1}{2}\alpha\nabla_\theta J(\boldsymbol{\theta})$
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{1}{2}\alpha\nabla_\theta J(\boldsymbol{\theta})$





Value Function Approximation

Gradient Descent

- **Let $J(\boldsymbol{\theta})$ be a differentiable function of parameter vector $\boldsymbol{\theta}$**
  - $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)^T$
- **Assume $\boldsymbol{V_t}$ is a (sufficiently smooth) differentiable function**
- $S_t \rightarrow V_\pi(S_t)$
  - Training examples form
  - $\{S_t, V_\pi(S_t)\}$
- **For the MSE object**
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{1}{2}\alpha\nabla_\theta J(\boldsymbol{\theta}_t)$
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{1}{2}\alpha\nabla_\theta \mathrm{MSE}(\boldsymbol{\theta}_t)$
  - $= \boldsymbol{\theta}_t - \frac{1}{2}\alpha\nabla_\theta \mathrm{E}_\pi[[V_\pi(s) - V(s, \boldsymbol{\theta}_t)]^2]$
  - $= \boldsymbol{\theta}_t + \alpha\mathrm{E}_\pi[V_\pi(s) - V(s, \boldsymbol{\theta}_t)]\,\nabla_\theta V(s, \boldsymbol{\theta}_t)$
  - $= \boldsymbol{\theta_t} + \alpha\sum_{s\in S}\mu(s)[V_\pi(s) - V(s, \boldsymbol{\theta}_t)]\,\nabla_\theta V(s, \boldsymbol{\theta}_t)$
- **Gradient descent finds a local minimum**

Value Function Approximation

Stochastic Gradient Descent

- **Gradient Descent**
  - $\Delta\boldsymbol{\theta} = -\frac{1}{2}\alpha\nabla_\theta J(\boldsymbol{\theta})$
- **Stochastic gradient descent samples the gradient**
  - $\Delta\boldsymbol{\theta} = \alpha(V_\pi(S_t) - V(S_t, \boldsymbol{\theta}))\nabla_\theta V(S_t, \boldsymbol{\theta})$
- **Expected update is equal to full gradient update**
- **Each sample gradient is an unbiased estimate of the true gradient**
  - $\mathrm{E}_\pi[V_\pi(s) - V(s, \boldsymbol{\theta})]\,\nabla_\theta V(s, \boldsymbol{\theta}) = \sum_{s\in S}\mu(s)[V_\pi(s) - V(s, \boldsymbol{\theta})]\,\nabla_\theta V(s, \boldsymbol{\theta})$
  - SGD converges to a local minimum of the MSE if α decreases appropriately with t.

Value Function Approximation

General SGD method

- **Use target $s \rightarrow U_t$ instead of $V_\pi(s)$**
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(U_t - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
- **If each U_t is an unbiased estimate of $V_\pi(S_t)$**
  - $\mathrm{E}[U_t] = V_\pi(S_t)$
  - Gradient descent converges to a local minimum (provided α decreases appropriately)
- **Example**
  - The MC target $S_t \rightarrow G_t$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
  - The TD(0) target $S_t \rightarrow R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t)$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
  - The TD(λ) target $S_t \rightarrow G_t^\lambda$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t^\lambda - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$

Value Function Approximation

Stochastic gradient descent

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
    Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

Value Function Approximation

Semi-gradient Methods

- **No guarantees if bootstrapping is used as the target U_t**
- **Bootstrapping U_t is not unbiased**
  - all depend on the current value of the weight vector θ
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{1}{2}\alpha\nabla_\theta[U_t - V(S_t, \boldsymbol{\theta}_t)]^2$
    - $= \boldsymbol{\theta}_t + \alpha(U_t - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
  - will not produce a true gradient-descent method
- **Include only a part of the gradient**
  - take into account the effect of changing the weight vector θ on the estimate, but ignore its effect on the target.

Value Function Approximation

Semi-gradient Methods

- **semi-gradient TD(0)**
  - $S_t \rightarrow R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t)$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop 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 Function Approximation

Linear Function Approximation

- **Feature Vectors**
  - Represent state by a feature vector
    - $\boldsymbol{x}(s) = \begin{pmatrix} x_1(s) \\ . \\ . \\ . \\ x_n(s) \end{pmatrix}$
- **Linear methods**
  - Represent value function by a linear combination of features
    - $v(s, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{x}(s) = \sum_{i=1}^{n} \theta_i x_i(s)$

Value Function Approximation

Linear Function Approximation

- **Objective function is quadratic in parameters $\boldsymbol{\theta}$**
  - $\text{MSE}(\boldsymbol{\theta}) = \sum_{s \in S} \mu(s)[V_\pi(s) - V(s, \boldsymbol{\theta})]^2$
  - $= \text{E}_\pi[[V_\pi(s) - V(s, \boldsymbol{\theta})]^2]$
  - $= \text{E}_\pi[[V_\pi(s) - \boldsymbol{\theta}^T \boldsymbol{x}]^2]$
- **Stochastic gradient descent converges on global optimum**
- **Update rule**
  - $\nabla_\theta V(s, \boldsymbol{\theta}) = \boldsymbol{x}(s)$
  - $\Delta\boldsymbol{\theta} = \alpha(V_\pi(s) - V(s, \boldsymbol{\theta}))\nabla_\theta V(s, \boldsymbol{\theta})$
  - $= \alpha(V_\pi(s) - V(s, \boldsymbol{\theta}))\boldsymbol{x}(s)$
  - Update = step-size × prediction error × feature value

Value Function Approximation

Linear Function Approximation

- **semi-gradient TD(0) also converges**
  - Need different theorem
  - Not global optimum, near the local optimum
  - The TD(0) target $S_t \rightarrow R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t)$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma \boldsymbol{\theta}_t^T \boldsymbol{x}_{t+1} - \boldsymbol{\theta}_t^T \boldsymbol{x}_t)\boldsymbol{x}_t$
    - $= \boldsymbol{\theta}_t + \alpha(R_{t+1}\boldsymbol{x}_t - \boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^T \boldsymbol{\theta}_t)$
    - For steady state
    - $\mathrm{E}(\boldsymbol{\theta}_{t+1} | \boldsymbol{\theta}_t) = \boldsymbol{\theta}_t + a(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{\theta}_t)$
    - Where
      - $\boldsymbol{b} = \mathrm{E}[R_{t+1}\boldsymbol{x}_t]$ and $\boldsymbol{A} = \mathrm{E}[\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^T]$
  - LSTD: Converge to the TD fixed point $\boldsymbol{\theta}_{TD}$
    - $\boldsymbol{b} - \boldsymbol{A}\boldsymbol{\theta}_{TD} = \boldsymbol{0}$
    - $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{\theta}_{TD}$
    - $\boldsymbol{\theta}_{TD} = \boldsymbol{A}^{-1}\boldsymbol{b}$

Value Function Approximation

Prediction Algorithms

- **Have assumed true value function $V_\pi(s)$ given by supervisor**
- **In RL there is no supervisor, only rewards**
- **Use target $s \rightarrow U_t$ instead of $V_\pi(s)$**
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(U_t - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
  - The MC target $S_t \rightarrow G_t$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
  - The TD(0) target $S_t \rightarrow R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t)$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
  - The TD(λ) target $S_t \rightarrow G_t^\lambda$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t^\lambda - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$

Value Function Approximation

Prediction Algorithms

- **Monte-Carlo with Value Function Approximation**
  - Return G_t is an unbiased, noisy sample of true value $V_\pi(s)$
  - Training data:
    - $< S_1, G_1 >, < S_2, G_2 >, \ldots, < S_T, G_T >$
  - e.g., linear MC policy evaluation
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
    - $= \boldsymbol{\theta}_t + \alpha(G_t - V(S_t, \boldsymbol{\theta}_t))\boldsymbol{x}_t$
- **MC evaluation converges to a local optimum**
- **Even when using non-linear value function approximation**

Value Function Approximation

Prediction Algorithms

- **TD Learning with Value Function Approximation**
  - The TD-target $R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t)$ is a biased sample of true value $V_\pi(s)$
  - Can still apply supervised learning to training data:
    - $< S_1, R_2 + \gamma V(S_2, \boldsymbol{\theta}_1) >, < S_2, R_3 + \gamma V(S_3, \boldsymbol{\theta}_2) >, \ldots, < S_{T-1}, R_T >$
  - e.g., linear TD(0)
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
    - $= \boldsymbol{\theta}_t + \alpha(R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t))\boldsymbol{x}_t$
    - $= \boldsymbol{\theta}_t + \alpha\delta_t \boldsymbol{x}_t$
- **Linear TD(0) converges (close) to global optimum**

Value Function Approximation

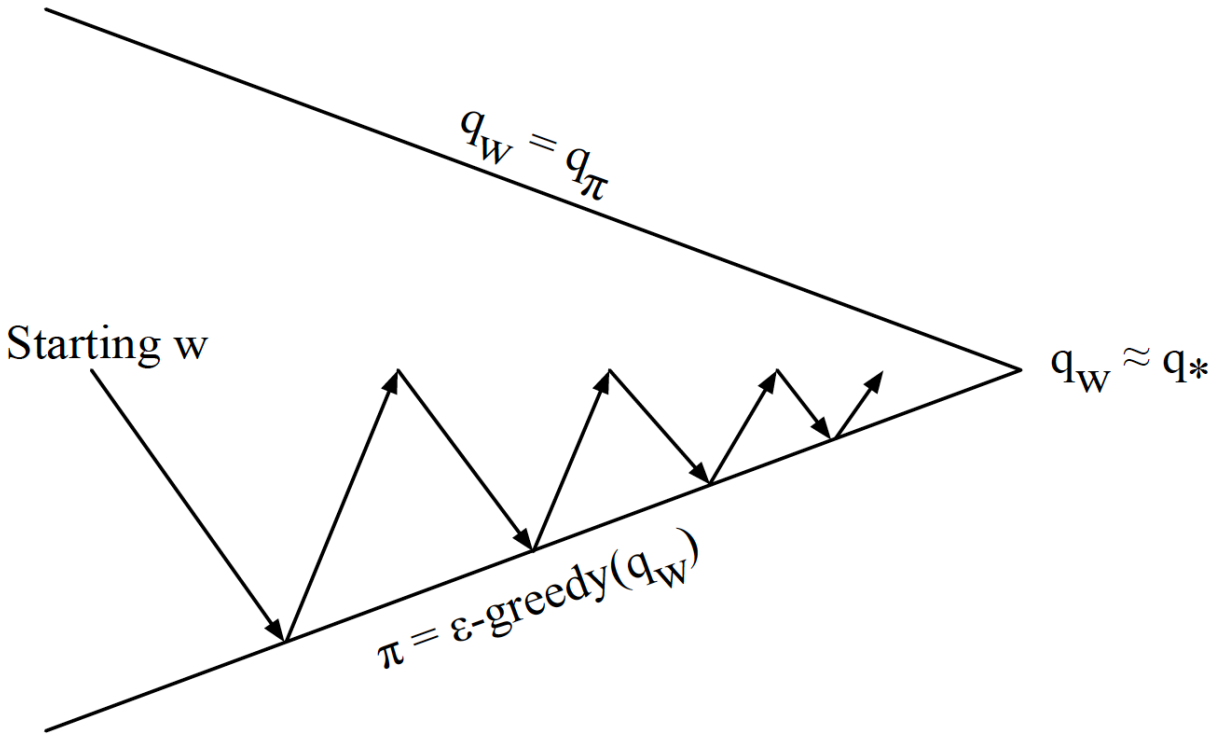
Prediction Algorithms

- **TD(λ) Learning with Value Function Approximation**
  - The $\boldsymbol{\lambda}$-return G_t^λ is a biased sample of true value $V_\pi(s)$
  - Can still apply supervised learning to training data:
    - $< S_1, G_1^\lambda >, < S_2, G_2^\lambda >, \ldots, < S_{T-1}, G_{T-1}^\lambda >$
  - Forward view linear TD(λ)
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t^\lambda - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
    - $\quad = \boldsymbol{\theta}_t + \alpha(G_t^\lambda - V(S_t, \boldsymbol{\theta}_t))\boldsymbol{x}_t$
  - Backward view linear TD(λ)
    - In proportion to TD-error δ_t and eligibility trace $E_t(s)$
    - $\delta_t = R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t)$
    - $\boldsymbol{E}_t(s) = \gamma\lambda \boldsymbol{E}_{t-1}(s) + \boldsymbol{x}_t$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + +\alpha\delta_t \boldsymbol{E}_t$

Value Function Approximation

Control Algorithms

- **Policy evaluation: Approximate policy evaluation $\boldsymbol{Q(.,.,\theta) \approx Q_\pi}$**
- **Policy improvement: $\boldsymbol{\varepsilon}$-greedy policy improvement**



Value Function Approximation

Control Algorithms

- **Action-Value Function Approximation**
  - Approximate the action-value function
    - $Q(s, a, \boldsymbol{\theta}) \approx Q_\pi(s, a)$
  - The MSE Objective
    - $\text{MSE}(\boldsymbol{\theta}) = \sum_{s \in S, a \in A} \mu(s)[Q_\pi(s, a) - Q(s, a, \boldsymbol{\theta})]^2$
    - $= \text{E}_\pi[[Q_\pi(s, a) - Q(s, a, \boldsymbol{\theta})]^2]$
  - Use SGD to find a local minimum
    - $\Delta\boldsymbol{\theta} = \alpha(Q_\pi(s, a) - Q(s, a, \boldsymbol{\theta}))\nabla_\theta Q(s, a, \boldsymbol{\theta})$

Value Function Approximation

Linear Action-Value Function Approximation

- **Feature Vectors**
  - Represent state by a feature vector
    - $\boldsymbol{x}(s,a) = \begin{pmatrix} x_1(s,a) \\ . \\ . \\ . \\ x_n(s,a) \end{pmatrix}$
- **Linear methods**
  - Represent value function by a linear combination of features
    - $Q(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{x}(s,a) = \sum_{i=1}^{n} \theta_i x_i(s,a)$
- **SGD update**
  - $\nabla_\theta Q(s,a,\boldsymbol{\theta}) = \boldsymbol{x}(s,a)$
  - $\Delta\boldsymbol{\theta} = \alpha(Q_\pi(s,a) - Q(s,a,\boldsymbol{\theta}))\boldsymbol{x}$

Value Function Approximation

Control Algorithms

- **Like prediction, use target $s \rightarrow U_t$ instead of $Q_\pi(s,a)$**
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(U_t - Q(S_t, A_t, \boldsymbol{\theta}_t))\nabla_\theta Q(S_t, A_t, \boldsymbol{\theta}_t)$
  - The MC target $S_t \rightarrow G_t$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - Q(S_t, A_t, \boldsymbol{\theta}_t))\nabla_\theta Q(S_t, A_t, \boldsymbol{\theta}_t)$
  - The Sarsa(0) target $S_t \rightarrow R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}, \boldsymbol{\theta}_t)$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1} + Q(S_{t+1}, A_{t+1}, \boldsymbol{\theta}_t) - Q(S_t, A_t, \boldsymbol{\theta}_t))\nabla_\theta Q(S_t, A_t, \boldsymbol{\theta}_t)$
  - The Sarsa(λ) target $S_t \rightarrow Q_t^\lambda$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Q_t^\lambda - Q(S_t, A_t, \boldsymbol{\theta}_t))\nabla_\theta Q(S_t, A_t, \boldsymbol{\theta}_t)$
  - The backview Sarsa(λ)
    - $\delta_t = R_{t+1} + Q(S_{t+1}, A_{t+1}, \boldsymbol{\theta}_t) - Q(S_t, A_t, \boldsymbol{\theta}_t)$
    - $\boldsymbol{E}_t(s) = \gamma\lambda \boldsymbol{E}_{t-1}(s) + \nabla_\theta Q(S_t, A_t, \boldsymbol{\theta}_t)$
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\delta_t \boldsymbol{E}_t$

Value Function Approximation

Control Algorithms

- **Example: Episodic 1-step semi-gradient Sarsa**
  - Select action and improve policy using an ε-greedy action
  - Converges the same ways as TD(0) with same error bound
    - $\text{MSE}(\boldsymbol{\theta}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$

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

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop 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., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Value Function Approximation

Off-policy Methods with Approximation

- **Off-policy learning**
  - learn a value function for a target policy π
  - given data due to a different behavior policy μ
- **In the prediction case, both policies are static and given**
- **In the control case, both policies typically change during learning**
  - π is the greedy policy
  - μ is ε-greedy policy

Value Function Approximation

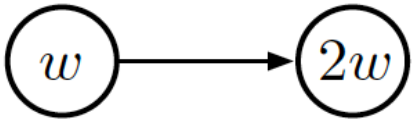
Off-policy Methods with Approximation

- **the first challenge of off-policy learning with function approximation**
  - changing the update targets
- **Semi-gradient Methods**
  - Use per-step importance sampling ratio
    - $\rho_t = \rho_{t:t} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$
- **semi-gradient off-policy TD(0)**
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\rho_t(R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t))\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
  - $= \boldsymbol{\theta}_t + \alpha\rho_t\delta_t\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$

Value Function Approximation

Off-policy Divergence

- **the Second challenge of off-policy learning with function approximation**
  - changing the update distribution: The distribution of updates does not match the on-policy distribution
- **Example**



  - TD error
    - $\delta_t = R_{t+1} + \gamma V(S_{t+1}, \boldsymbol{\theta}_t) - V(S_t, \boldsymbol{\theta}_t)$
    - $= 0 + \gamma 2\boldsymbol{\theta}_t - \boldsymbol{\theta}_t$
    - $= (2\gamma - 1)\boldsymbol{\theta}_t$
  - semi-gradient off-policy TD(0)
    - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\rho_t\delta_t\nabla_\theta V(S_t, \boldsymbol{\theta}_t)$
    - $= \boldsymbol{\theta}_t + \alpha 1(2\gamma - 1)\boldsymbol{\theta}_t 1$
    - $= (1 + \alpha(2\gamma - 1))\boldsymbol{\theta}_t$

Value Function Approximation

Instability and Divergence

- **Danger of instability and divergence arises whenever we combine all of the following three elements**
  - Function approximation
  - Bootstrapping
  - Off-policy training

Value Function Approximation

Convergence

- **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)	✓	✗	✗

Value Function Approximation

Convergence

- **Gradient TD Learning**
  - TD does not follow the gradient of any objective function
  - This is why TD can diverge when off-policy or using non-linear function approximation
  - Gradient TD follows true gradient of projected Bellman error

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD	✓	✓	✗
	Gradient TD	✓	✓	✓
Off-Policy	MC	✓	✓	✓
	TD	✓	✗	✗
	Gradient TD	✓	✓	✓

Value Function Approximation

Convergence

- **Convergence of Control Algorithms**

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

Value Function Approximation

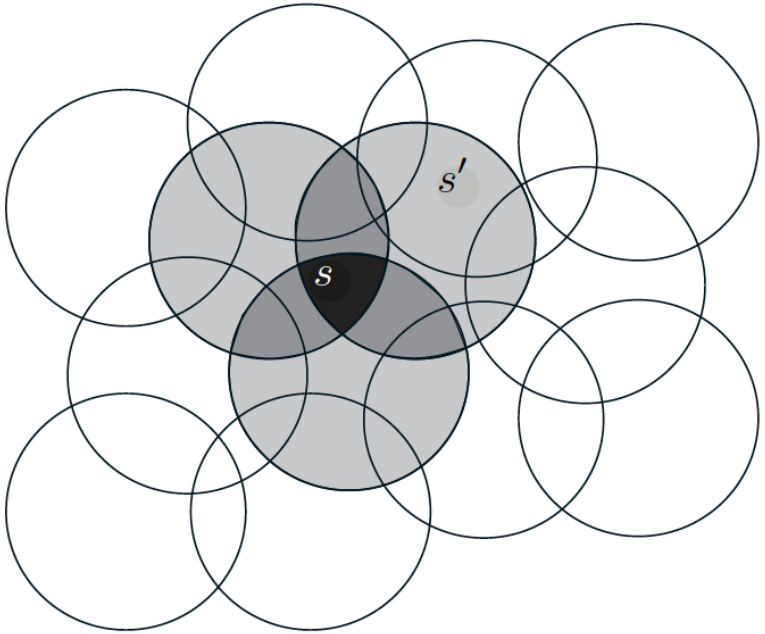
Linear Function Approximation

- **Nice properties of linear method**
  - The gradient is very simple:
    - $\nabla_\theta V(s, \boldsymbol{\theta}) = \boldsymbol{x}(s)$
  - For MSE, the error surface is simple
    - quadratic surface with a single minimum
  - Linear gradient descent TD(λ) converges:
    - Step size decreases appropriately
    - On-line sampling (states sampled from the on-policy distribution)
    - Converges to parameter vector with property:
      - $\text{MSE}(\boldsymbol{\theta}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$
  - Linear gradient descent TD(0) converges:
      - $\text{MSE}(\boldsymbol{\theta}_{TD}) \leq \frac{1-\gamma\lambda}{1-\gamma} \min_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$

Value Function Approximation

Feature Construction for Linear Methods

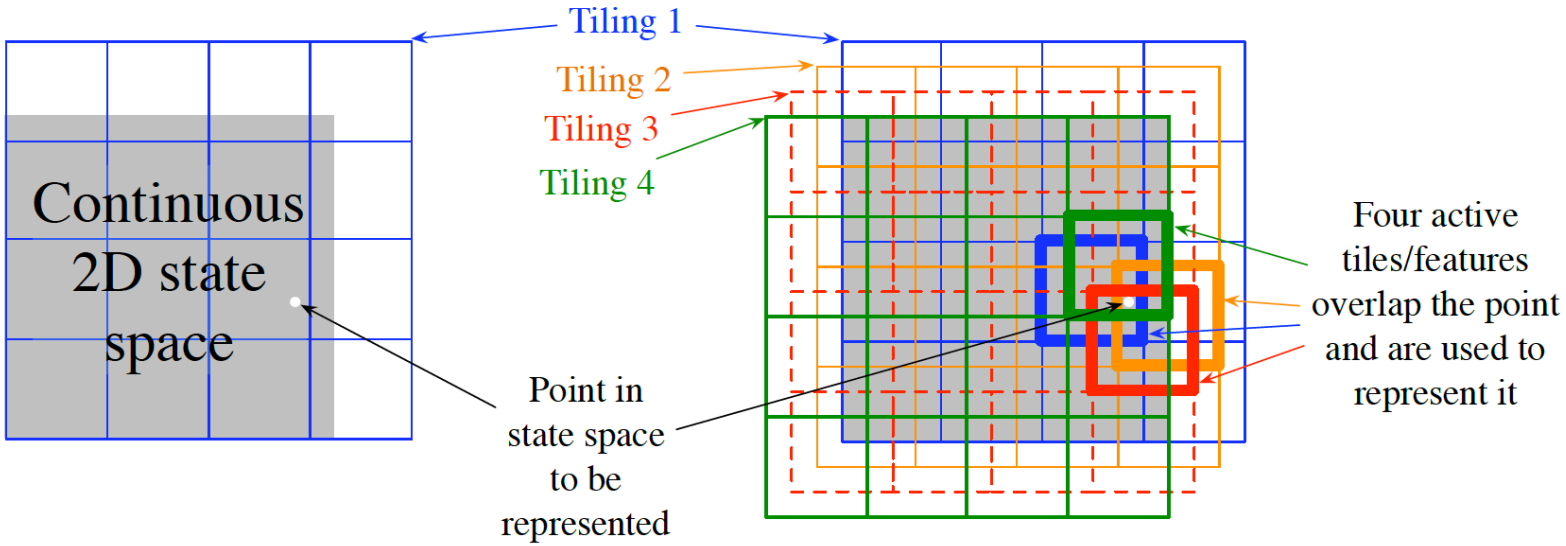
- **Polynomials**
  - e.g., $\boldsymbol{x}(s) = (1, s_1, s_2, s_1^2, s_2^2, s_1\, s_s)^T$
- **Fourier Basis**
  - e.g., $x_i(s) = \cos(i\pi s)$
- **Coarse Coding**



Value Function Approximation

Feature Construction for Linear Methods

- **Tile Coding**



- **Radial Basis Functions**
  - $x_i(s) = \exp(-\frac{||s-c_i||^2}{2\sigma_i^2})$

Value Function Approximation

Selecting Step-Size Parameters

- **The classical choice**
  - $\alpha_t = \frac{1}{t}$
  - not appropriate for TD methods or function approximation
- **Tabular case**
  - $\alpha_t = \frac{1}{\tau}$
  - tabular estimate for a state will approach the mean of its targets after τ experiences with the state
- **general function approximation**
  - $\alpha = (\tau \mathrm{E}[\boldsymbol{x}^T \boldsymbol{x}])^{-1}$

Value Function Approximation

Nonlinear Function Approximation

- **Artificial Neural Networks**
  - widely used for nonlinear function approximation