

ECE276B: Planning & Learning in Robotics

Lecture 14: Value Function Approximation

Instructor:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Thai Duong: tduong@eng.ucsd.edu

UC San Diego

JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

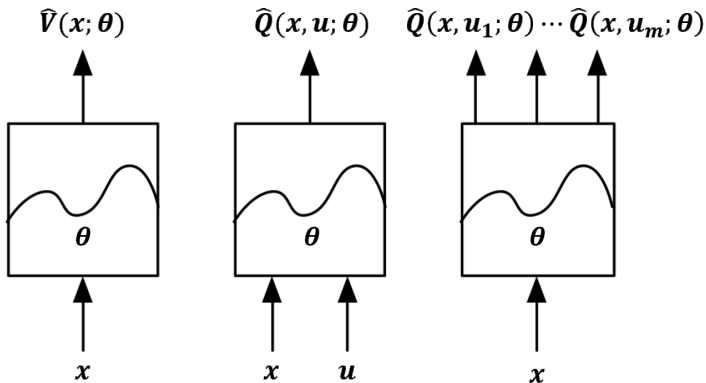
Optimal Control in Large and Infinite Spaces

- ▶ So far we have been using a vector to represent the value function:
 - ▶ Every state \mathbf{x} has an entry $V^\pi(\mathbf{x})$
 - ▶ Every state-control pair (\mathbf{x}, \mathbf{u}) has an entry $Q^\pi(\mathbf{x}, \mathbf{u})$
- ▶ In very large and continuous state and control spaces:
 - ▶ There are too many states and controls to store in memory
 - ▶ It is too slow to approximate the value of each state individually
- ▶ **Key Idea:**
 - ▶ Represent the value function using function approximation with parameters θ :

$$V^\pi(\mathbf{x}) \approx \hat{V}(\mathbf{x}; \theta) \qquad Q^\pi(\mathbf{x}, \mathbf{u}) \approx \hat{Q}(\mathbf{x}, \mathbf{u}; \theta)$$

- ▶ Update the parameters θ using MC or TD learning
- ▶ Allows generalization from seen to unseen states and controls

Value Function Approximation



Value Function Approximation

- ▶ Many function approximators are possible:
 - ▶ Linear combination of features (differentiable)
 - ▶ Fourier / wavelet bases (differentiable)
 - ▶ Neural network (differentiable)
 - ▶ Nearest neighbor
 - ▶ Decision tree
- ▶ A **differentiable** function approximator is necessary to allow parameter updates
- ▶ A training method for non-stationary, non-iid data is required

Value Approximation via Unconstrained Optimization

- ▶ **Main idea:**

- ▶ define a function $J(\theta)$ measuring the error between $V^\pi(\mathbf{x})$ and $\hat{V}(\mathbf{x}; \theta)$
- ▶ determine the parameters through an optimization problem:

$$\theta^* = \arg \min_{\theta} J(\theta)$$

- ▶ Two approaches to solving $\min_{\theta} J(\theta)$:

- ▶ **Incremental:** use a (stochastic) descent method:

$$\theta_{k+1} = \theta_k + \alpha_k \delta \theta_k$$

- ▶ **Batch:** obtain θ^* from the optimality conditions:

$$\nabla_{\theta} J(\theta) = 0$$

Optimality Conditions

First-order Necessary Condition

Suppose $J(\theta)$ is differentiable at $\bar{\theta}$. If $\bar{\theta}$ is a local minimizer, then $\nabla J(\bar{\theta}) = 0$.

Second-order Necessary Condition

Suppose $J(\theta)$ is twice-differentiable at $\bar{\theta}$. If $\bar{\theta}$ is a local minimizer, then $\nabla J(\bar{\theta}) = 0$ and $\nabla^2 J(\bar{\theta}) \succeq 0$.

Second-order Sufficient Condition

Suppose $J(\theta)$ is twice-differentiable at $\bar{\theta}$. If $\nabla J(\bar{\theta}) = 0$ and $\nabla^2 J(\bar{\theta}) \succ 0$, then $\bar{\theta}$ is a local minimizer.

Necessary and Sufficient Condition

Suppose $J(\theta)$ is differentiable at $\bar{\theta}$. If J is **convex**, then $\bar{\theta}$ is a global minimizer **if and only if** $\nabla J(\bar{\theta}) = 0$.

Descent Optimization Methods

Descent Direction Theorem

Suppose $J(\boldsymbol{\theta})$ is differentiable at $\bar{\boldsymbol{\theta}}$. If $\exists \delta\boldsymbol{\theta}$ such that $\nabla J(\bar{\boldsymbol{\theta}})^T \delta\boldsymbol{\theta} < 0$, then $\exists \epsilon > 0$ such that $J(\bar{\boldsymbol{\theta}} + \alpha\delta\boldsymbol{\theta}) < J(\bar{\boldsymbol{\theta}})$ for all $\alpha \in (0, \epsilon)$.

- ▶ The vector $\delta\boldsymbol{\theta}$ is called a **descent direction**
- ▶ The theorem states that if a descent direction exists at $\bar{\boldsymbol{\theta}}$, then it is possible to move to a new point that has a lower J value.
- ▶ **Descent method:** given an initial guess $\boldsymbol{\theta}_k$, take a step of size $\alpha_k > 0$ along a descent direction $\delta\boldsymbol{\theta}_k$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \delta\boldsymbol{\theta}_k$$

Descent Optimization Methods

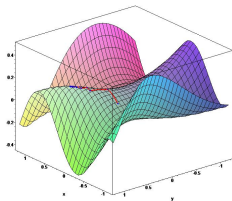
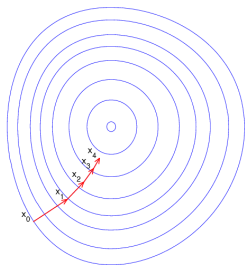
- ▶ Methods differ in the way $\delta\theta_k$ and α_k are chosen:
 - ▶ $\delta\theta_k$ should be a descent direction:
 $\nabla J(\theta_k)^T \delta\theta_k < 0$ for all $\theta_k \neq \theta^*$
 - ▶ α_k needs to ensure sufficient decrease in J to guarantee convergence:

$$\alpha_k^* \in \arg \min_{\alpha > 0} J(\theta_k + \alpha \delta\theta_k)$$

Usually obtained via line search

- ▶ **Steepest descent direction:** $\delta\theta_k := -\frac{\nabla J(\theta_k)}{\|\nabla J(\theta_k)\|}$
- ▶ **Gradient descent:** $\delta\theta_k := -\nabla_{\theta} J(\theta_k)$ points in the direction of steepest local descent and we can iterate:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_{\theta} J(\theta_k)$$



MSE Value Function Approximation

- ▶ **Idea:** find parameters θ minimizing the mean-squared error between the true and approximate value function of policy π :

$$J(\theta) = \frac{1}{2} \mathbb{E} \left[\left(V^\pi(\mathbf{x}) - \hat{V}(\mathbf{x}; \theta) \right)^2 \right] \quad \text{OR} \quad J(\theta) = \frac{1}{2} \mathbb{E} \left[\left(Q^\pi(\mathbf{x}, \mathbf{u}) - \hat{Q}(\mathbf{x}, \mathbf{u}; \theta) \right)^2 \right]$$

where the expectation is over the state-control distribution induced by π

- ▶ Need to choose:
 - ▶ An incremental or batch optimization approach
 - ▶ A representation for $\hat{V}(\mathbf{x}; \theta)$ or $\hat{Q}(\mathbf{x}, \mathbf{u}; \theta)$

Incremental vs Batch optimization

► Incremental Optimization:

► Gradient descent:

$$\delta\theta = -\nabla_{\theta}J(\theta) = \mathbb{E} \left[\left(V^{\pi}(\mathbf{x}) - \hat{V}(\mathbf{x}; \theta) \right) \nabla_{\theta} \hat{V}(\mathbf{x}, \theta) \right]$$

$$\delta\theta = -\nabla_{\theta}J(\theta) = \mathbb{E} \left[\left(Q^{\pi}(\mathbf{x}, \mathbf{u}) - \hat{Q}(\mathbf{x}, \mathbf{u}; \theta) \right) \nabla_{\theta} \hat{Q}(\mathbf{x}, \mathbf{u}; \theta) \right]$$

- **Stochastic gradient descent:** uses samples $\mathbf{x}_t, \mathbf{u}_t$ from π rather than computing the exact expectation.

$$\delta\theta = \left(V^{\pi}(\mathbf{x}_t) - \hat{V}(\mathbf{x}_t; \theta) \right) \nabla_{\theta} \hat{V}(\mathbf{x}_t, \theta)$$

$$\delta\theta = \left(Q^{\pi}(\mathbf{x}_t, \mathbf{u}_t) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta) \right) \nabla_{\theta} \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta)$$

The stochastic update is equal to the full gradient update in expectation

- **Batch Optimization:** the expected update $\mathbb{E}[\delta\theta]$ must be zero at the minimum of $J(\theta)$. Determine θ^* directly by solving:

$$\mathbb{E}[\delta\theta] = 0$$

Linear Value Function Approximation

- ▶ Represent a state \mathbf{x} by a feature vector $\phi(\mathbf{x})$ or a state-control pair (\mathbf{x}, \mathbf{u}) by a feature vector $\phi(\mathbf{x}, \mathbf{u})$, e.g.:
 - ▶ Distance of the robot to landmarks
 - ▶ Piece and pawn configurations in chess
- ▶ Represent the value function by a linear combination of features:

$$\hat{V}(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \phi(\mathbf{x}) = \sum_j \phi_j(\mathbf{x}) \theta_j$$

$$\hat{Q}(\mathbf{x}, \mathbf{u}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \phi(\mathbf{x}, \mathbf{u}) = \sum_j \phi_j(\mathbf{x}, \mathbf{u}) \theta_j$$

Linear Value Function Approximation

- ▶ A finite-space representation of $V^\pi(x)$ over $\mathcal{X} := \{1, \dots, n\}$ is a special case of a linear value function approximation with:

$$\phi(x) := \begin{bmatrix} \mathbb{1}\{x = 1\} \\ \vdots \\ \mathbb{1}\{x = n\} \end{bmatrix}$$

- ▶ The parameter vector θ contains the values of the states:

$$\hat{V}(x; \theta) = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}^\top \begin{bmatrix} \mathbb{1}\{x = 1\} \\ \vdots \\ \mathbb{1}\{x = n\} \end{bmatrix}$$

Incremental Methods

Incremental Prediction for Linear Approximation

- ▶ When the value function is represented by a linear combination of features, the objective function $J(\theta)$ is quadratic in θ :

$$J(\theta) = \frac{1}{2} \mathbb{E} \left[\left(V^\pi(\mathbf{x}) - \theta^\top \phi(\mathbf{x}) \right)^2 \right] \quad J(\theta) = \frac{1}{2} \mathbb{E} \left[\left(Q^\pi(\mathbf{x}, \mathbf{u}) - \theta^\top \phi(\mathbf{x}, \mathbf{u}) \right)^2 \right]$$

- ▶ Stochastic gradient descent converges to a *global* optimum
- ▶ A descent direction $\delta\theta$ is easy to obtain:

$$\delta\theta = \underbrace{\left(V^\pi(\mathbf{x}_t) - \hat{V}(\mathbf{x}_t; \theta) \right)}_{\text{prediction error}} \underbrace{\phi(\mathbf{x}_t)}_{\text{feature value}}$$

$$\delta\theta = \underbrace{\left(Q^\pi(\mathbf{x}_t, \mathbf{u}_t) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta) \right)}_{\text{prediction error}} \underbrace{\phi(\mathbf{x}_t, \mathbf{u}_t)}_{\text{feature value}}$$

Incremental Prediction Algorithms

- ▶ The (stochastic) gradient descent for optimizing θ can be performed only if $V^\pi(\mathbf{x})$ is available to compute the prediction error
- ▶ In practice, we substitute a *target* for $V^\pi(\mathbf{x})$ obtained from noisy samples along an episode $\rho = \mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \dots \sim \pi$:
 - ▶ MC: uses a dataset $\mathcal{D} := \{(\mathbf{x}_t, L_t(\rho_t))\}$
 - ▶ TD: uses a dataset $\mathcal{D} := \{(\mathbf{x}_t, \ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{V}(\mathbf{x}_{t+1}; \theta))\}$
 - ▶ TD(λ): uses a dataset $\mathcal{D} := \{(\mathbf{x}_t, L_t^\lambda(\rho_t))\}$

Incremental Prediction Algorithms

- ▶ **MC**: the target is the return $L_t(\rho_t)$:

$$\delta\theta = \left(L_t(\rho_t) - \hat{V}(\mathbf{x}_t; \theta) \right) \nabla_{\theta} \hat{V}(\mathbf{x}_t; \theta)$$

- ▶ **TD**: the target is the TD target:

$$\delta\theta = \left(\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{V}(\mathbf{x}_{t+1}; \theta) - \hat{V}(\mathbf{x}_t; \theta) \right) \nabla_{\theta} \hat{V}(\mathbf{x}_t; \theta)$$

- ▶ **Forward-view TD**(λ): the target is the λ -return $L_t^{\lambda}(\rho_t)$:

$$\delta\theta = \left(L_t^{\lambda}(\rho_t) - \hat{V}(\mathbf{x}_t; \theta) \right) \nabla_{\theta} \hat{V}(\mathbf{x}_t; \theta)$$

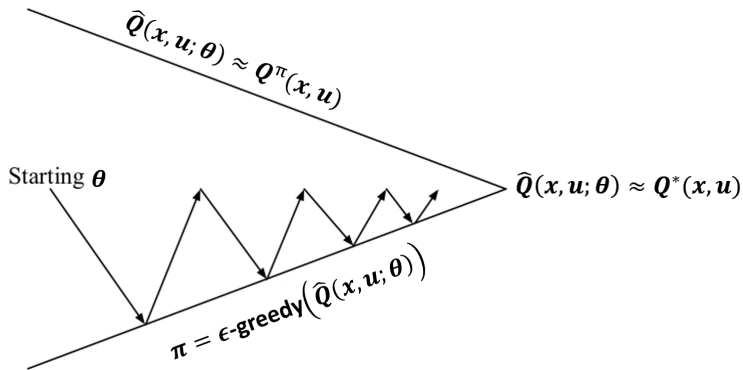
- ▶ **Backward-view TD**(λ):

$$\delta_t = \ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{V}(\mathbf{x}_{t+1}; \theta) - \hat{V}(\mathbf{x}_t; \theta)$$

$$\mathbf{e}_t = \gamma \lambda \mathbf{e}_{t-1} + \nabla_{\theta} \hat{V}(\mathbf{x}_t; \theta)$$

$$\delta\theta = \delta_t \mathbf{e}_t$$

Control with Value Function Approximation



- ▶ **Policy Evaluation:** approximate $Q^\pi(\mathbf{x}, \mathbf{u}) \approx \hat{Q}(\mathbf{x}, \mathbf{u}; \theta)$ via stochastic gradient descent
- ▶ **Policy Improvement:** ϵ -greedy policy improvement based on $\hat{Q}(\mathbf{x}, \mathbf{u}; \theta)$

Incremental Control Algorithms

▶ Q-Prediction: we must substitute a *target* for $Q^\pi(\mathbf{x}, \mathbf{u})$

▶ MC:

$$\delta\theta = \left(L_t(\rho) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta) \right) \nabla_\theta \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta)$$

▶ TD:

$$\delta\theta = \left(\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{Q}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1}; \theta) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta) \right) \nabla_\theta \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta)$$

▶ Forward-view TD(λ):

$$\delta\theta = \left(L_t^\lambda(\rho) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta) \right) \nabla_\theta \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta)$$

▶ Backward-view TD(λ):

$$\delta_t = \ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{Q}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1}; \theta) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta)$$

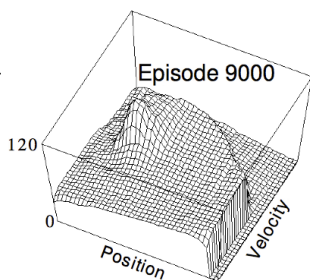
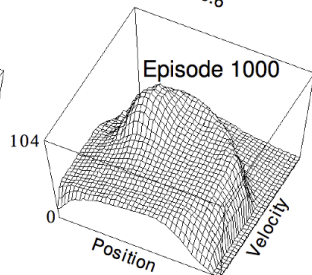
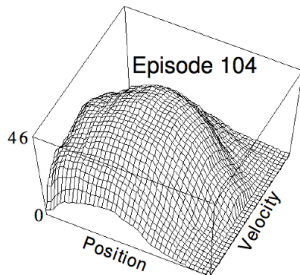
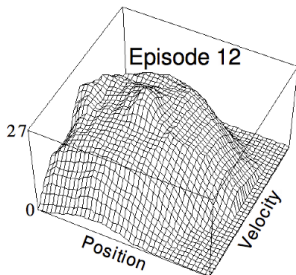
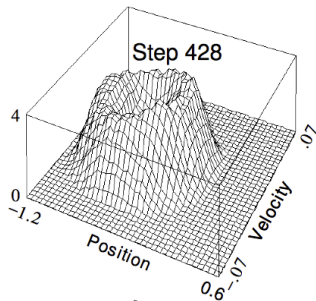
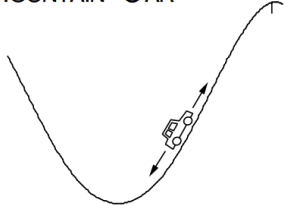
$$\mathbf{e}_t = \gamma \lambda \mathbf{e}_{t-1} + \nabla_\theta \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta)$$

$$\delta\theta = \delta_t \mathbf{e}_t$$

Linear SARSA with Coarse Coding in Mountain Car

MOUNTAIN CAR

Goal



Convergence of Prediction and Control Algorithms

► Model-free Prediction:

Algorithm	Finite Space	Linear	Non-Linear
On-Policy MC	✓	✓	✓
On-Policy TD	✓	✓	✗
Off-Policy MC	✓	✓	✓
Off-Policy TD	✓	✗	✗

- There is a version of TD that follows the gradient of the projected Bellman error and converges in all cases

► Model-free Control:

Algorithm	Finite Space	Linear	Non-Linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-learning	✓	✗	✗

- (✓) = chatters around a near-optimal value function
- There is a gradient Q-learning version that converges in the linear case

Batch Methods

Batch Prediction

▶ **Given:**

- ▶ Value function approximation $\hat{V}(\mathbf{x}; \boldsymbol{\theta}) \approx V^\pi(\mathbf{x})$
- ▶ Experience $\mathcal{D} := \{(\mathbf{x}_t, V^\pi(\mathbf{x}_t))\}$

▶ **Goal:** find the best fitting value function approximation:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) := \frac{1}{2} \mathbb{E} \left[\left(V^\pi(\mathbf{x}) - \hat{V}(\mathbf{x}; \boldsymbol{\theta}) \right)^2 \right] \approx \frac{1}{2} \sum_{\mathbf{x}_t \in \mathcal{D}} \left(V^\pi(\mathbf{x}_t) - \hat{V}(\mathbf{x}_t; \boldsymbol{\theta}) \right)^2$$

▶ **Stochastic Gradient Descent with Experience Replay:**

1. Sample: $(\mathbf{x}_t, V^\pi(\mathbf{x}_t)) \sim \mathcal{D}$
2. Apply SGD update with $\delta\boldsymbol{\theta} = \left(V^\pi(\mathbf{x}_t) - \hat{V}(\mathbf{x}_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{V}(\mathbf{x}_t, \boldsymbol{\theta})$
 - ▶ SGD with experience replay finds the least-squares solution but it may take many iterations

▶ **Batch method:** the expected update must be zero at the min of $J(\boldsymbol{\theta})$:

$$0 = \mathbb{E}[\delta\boldsymbol{\theta}] \approx \sum_{\mathbf{x}_t \in \mathcal{D}} \left(V^\pi(\mathbf{x}_t) - \hat{V}(\mathbf{x}_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{V}(\mathbf{x}_t, \boldsymbol{\theta})$$

- ▶ Obtain $\boldsymbol{\theta}^*$ directly by solving the above equation

Batch Prediction for Linear Approximation

- ▶ When the value function is represented by a linear combination of features $\hat{V}(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x})$, the function $J(\boldsymbol{\theta})$ is quadratic in $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E} \left[\left(V^\pi(\mathbf{x}) - \boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x}) \right)^2 \right] \approx \frac{1}{2} \sum_{\mathbf{x}_t \in \mathcal{D}} \left(V^\pi(\mathbf{x}_t) - \boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x}_t) \right)^2$$

- ▶ We can obtain the least squares solution $\boldsymbol{\theta}^*$ directly:

$$0 = \mathbb{E}[\delta \boldsymbol{\theta}] = \sum_{\mathbf{x}_t \in \mathcal{D}} (V^\pi(\mathbf{x}_t) - \boldsymbol{\theta}^\top \boldsymbol{\phi}(\mathbf{x}_t)) \boldsymbol{\phi}(\mathbf{x}_t)$$
$$\left(\sum_{\mathbf{x}_t \in \mathcal{D}} \boldsymbol{\phi}(\mathbf{x}_t) \boldsymbol{\phi}(\mathbf{x}_t)^\top \right) \boldsymbol{\theta} = \sum_{\mathbf{x}_t \in \mathcal{D}} V^\pi(\mathbf{x}_t) \boldsymbol{\phi}(\mathbf{x}_t)$$

Linear Least Squares Prediction Algorithms

- ▶ In practice, we do not know the true values $V^\pi(\mathbf{x}_t)$ and must use noisy/biased samples instead:

- ▶ **Least-squares Monte Carlo:**

$$V^\pi(\mathbf{x}_t) \approx L_t(\rho)$$

- ▶ **Least-squares Temporal Difference:**

$$V^\pi(\mathbf{x}_t) \approx \ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{V}(\mathbf{x}_{t+1}; \boldsymbol{\theta})$$

- ▶ **Least-squares TD(λ):**

$$V^\pi(\mathbf{x}_t) \approx L_t^\lambda(\rho)$$

- ▶ In each case, we can solve directly for the fixed point $\boldsymbol{\theta}^*$

Linear Least-Squares Prediction Algorithms

$$0 = \sum_{t=0}^T \alpha \left(L_t(\rho) - \hat{V}(\mathbf{x}_t; \boldsymbol{\theta}) \right) \phi(\mathbf{x}_t)$$

► **LSMC:**

$$\boldsymbol{\theta}^* = \left(\sum_{t=0}^T \phi(\mathbf{x}_t) \phi(\mathbf{x}_t)^T \right)^{-1} \sum_{t=0}^T \phi(\mathbf{x}_t) L_t(\rho)$$

$$0 = \sum_{t=0}^T \alpha \left(\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{V}(\mathbf{x}_{t+1}; \boldsymbol{\theta}) - \hat{V}(\mathbf{x}_t; \boldsymbol{\theta}) \right) \phi(\mathbf{x}_t)$$

► **LSTD:**

$$\boldsymbol{\theta}^* = \left(\sum_{t=0}^T \phi(\mathbf{x}_t) (\phi(\mathbf{x}_t) - \gamma \phi(\mathbf{x}_{t+1}))^T \right)^{-1} \sum_{t=0}^T \phi(\mathbf{x}_t) \ell(\mathbf{x}_t, \mathbf{u}_t)$$

$$0 = \sum_{t=0}^T \alpha \left(\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{V}(\mathbf{x}_{t+1}; \boldsymbol{\theta}) - \hat{V}(\mathbf{x}_t; \boldsymbol{\theta}) \right) \mathbf{e}_t$$

► **LSTD(λ):**

$$\boldsymbol{\theta}^* = \left(\sum_{t=0}^T \mathbf{e}_t (\phi(\mathbf{x}_t) - \gamma \phi(\mathbf{x}_{t+1}))^T \right)^{-1} \sum_{t=0}^T \mathbf{e}_t \ell(\mathbf{x}_t, \mathbf{u}_t)$$

Convergence of Linear-Least Squares Prediction Algorithms

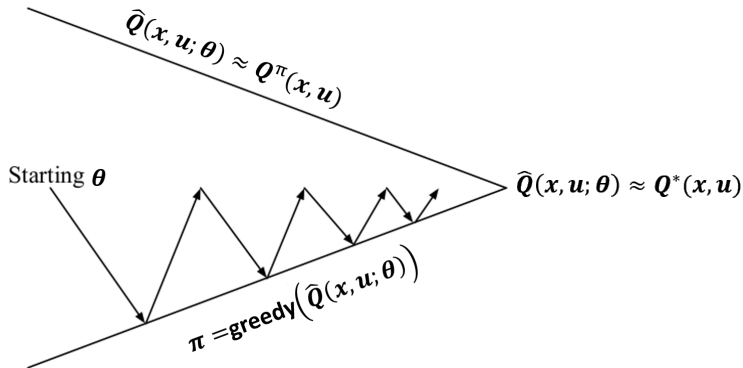
► On-Policy:

Algorithm	Finite Space	Linear	Non-Linear
MC	✓	✓	✓
LSMC	✓	✓	—
TD	✓	✓	✗
LSTD	✓	✓	—

► Off-Policy:

Algorithm	Finite Space	Linear	Non-Linear
MC	✓	✓	✓
LSMC	✓	✓	—
TD	✓	✗	✗
LSTD	✓	✓	—

Least Squares Policy Iteration



- ▶ **Policy Evaluation:** least-squares Q estimation using data from old policies
- ▶ **Policy Improvement:** does not have to be ϵ -greedy since data from old policies is stored

Least Squares Policy Iteration

- ▶ **Policy Evaluation:** efficiently use all experience $\mathcal{D} := \{(\mathbf{x}_t, \mathbf{u}_t, V^\pi(\mathbf{x}_t))\}$ to compute $\hat{Q}(\mathbf{x}, \mathbf{u}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \phi(\mathbf{x}, \mathbf{u})$
- ▶ Since the policy is changing, the experience is generated from many different policies
- ▶ We must approximate Q^π using **off-policy** learning
- ▶ Instead of importance sampling, use an idea from Q-learning:
 - ▶ Use experience: $\mathbf{x}_t, \mathbf{u}_t, \ell(\mathbf{x}_t, \mathbf{u}_t), \mathbf{x}_{t+1} \sim \pi_{old}$
 - ▶ With new action: $\mathbf{u}_{t+1} = \pi_{new}(\mathbf{x}_{t+1})$
 - ▶ Update $\hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta})$ towards the value of the new action:
 $\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{Q}(\mathbf{x}_t, \mathbf{u}_{t+1}; \boldsymbol{\theta})$

Least Squares Policy Iteration

- ▶ Experience: $\mathbf{x}_t, \mathbf{u}_t, \ell(\mathbf{x}_t, \mathbf{u}_t), \mathbf{x}_{t+1} \sim \pi_{old}$
- ▶ Incremental update:

$$\delta\theta = \left(\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{Q}(\mathbf{x}_{t+1}, \pi(\mathbf{x}_{t+1}); \theta) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta) \right) \phi(\mathbf{x}_t, \mathbf{u}_t)$$

- ▶ **LSTDQ**: least-squares TD Q estimation algorithm using the fact that the expected update must be zero at the minimum of $J(\theta)$:

$$0 = \sum_{t=0}^T \alpha \left(\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{Q}(\mathbf{x}_{t+1}, \pi(\mathbf{x}_{t+1}); \theta) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \theta) \right) \phi(\mathbf{x}_t, \mathbf{u}_t)$$

$$\theta^* = \left(\sum_{t=0}^T \phi(\mathbf{x}_t, \mathbf{u}_t) (\phi(\mathbf{x}_t, \mathbf{u}_t) - \gamma \phi(\mathbf{x}_{t+1}, \pi(\mathbf{x}_{t+1})))^T \right)^{-1} \sum_{t=0}^T \phi(\mathbf{x}_t, \mathbf{u}_t) \ell(\mathbf{x}_t, \mathbf{u}_t)$$

Algorithm 1 LSPI-TD

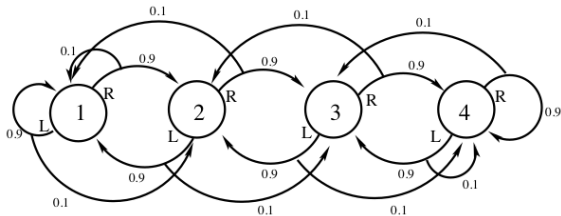
- 1: **Input**: experience \mathcal{D} and base policy π
- 2: **loop**
- 3: $\theta^* \leftarrow \mathbf{LSTDQ}(\pi, \mathcal{D})$
- 4: $\pi(\mathbf{x}) \leftarrow \arg \min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} \hat{Q}(\mathbf{x}, \mathbf{u}; \theta^*)$

Convergence of Control Algorithms

Algorithm	Finite Space	Linear	Non-Linear
MC Control	✓	(✓)	✗
SARSA	✓	(✓)	✗
Q-learning	✓	✗	✗
LSPI-TD	✓	(✓)	—

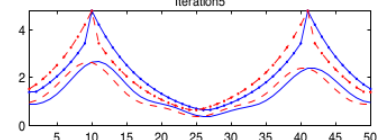
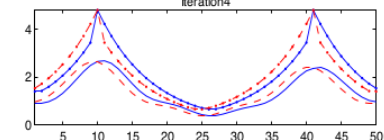
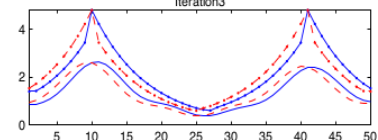
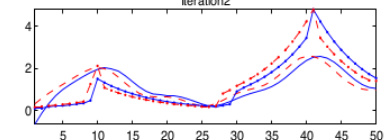
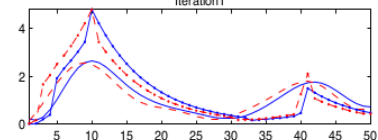
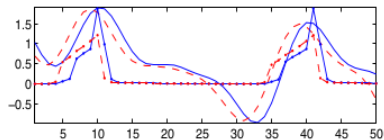
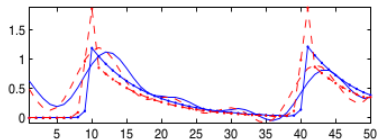
- ▶ (✓) = chatters around a near-optimal value function

Example: Chain Walk



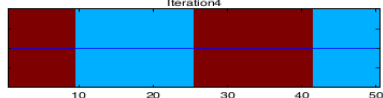
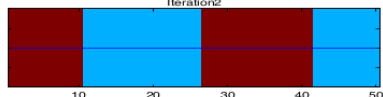
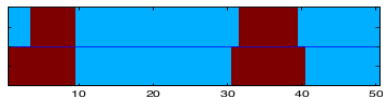
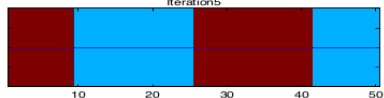
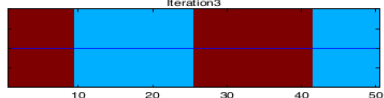
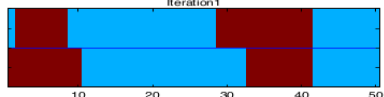
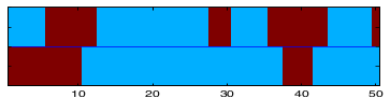
- ▶ Consider a 50 state version of the problem
- ▶ Cost: -1 in states 10 and 41 and 0 elsewhere
- ▶ Optimal policy:
$$\pi(x) = \begin{cases} R & \text{if } x \in \{1, \dots, 9\} \cup \{26, \dots, 41\} \\ L & \text{if } x \in \{10, \dots, 25\} \cup \{42, \dots, 50\} \end{cases}$$
- ▶ Features: 10 evenly spaced Gaussians ($\sigma = 4$) for each control
- ▶ Experience: 10,000 steps from a random walk policy

Chain Walk LSPI: Action-Value Function



- ▶ True (dotted) and approximate (smooth) action-value function
- ▶ Left (blue) and right (red) control

Chain Walk LSPI: Policy



► Left (blue) and right (red) control