ECE276B: Planning & Learning in Robotics
Lecture 13: Value Function Approximation

Instructor:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Zhichao Li: zhl355@eng.ucsd.edu

Ehsan Zobeidi: ezobeidi@eng.ucsd.edu

Ibrahim Akbar: iakbar@eng.ucsd.edu

UC San Diego

JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

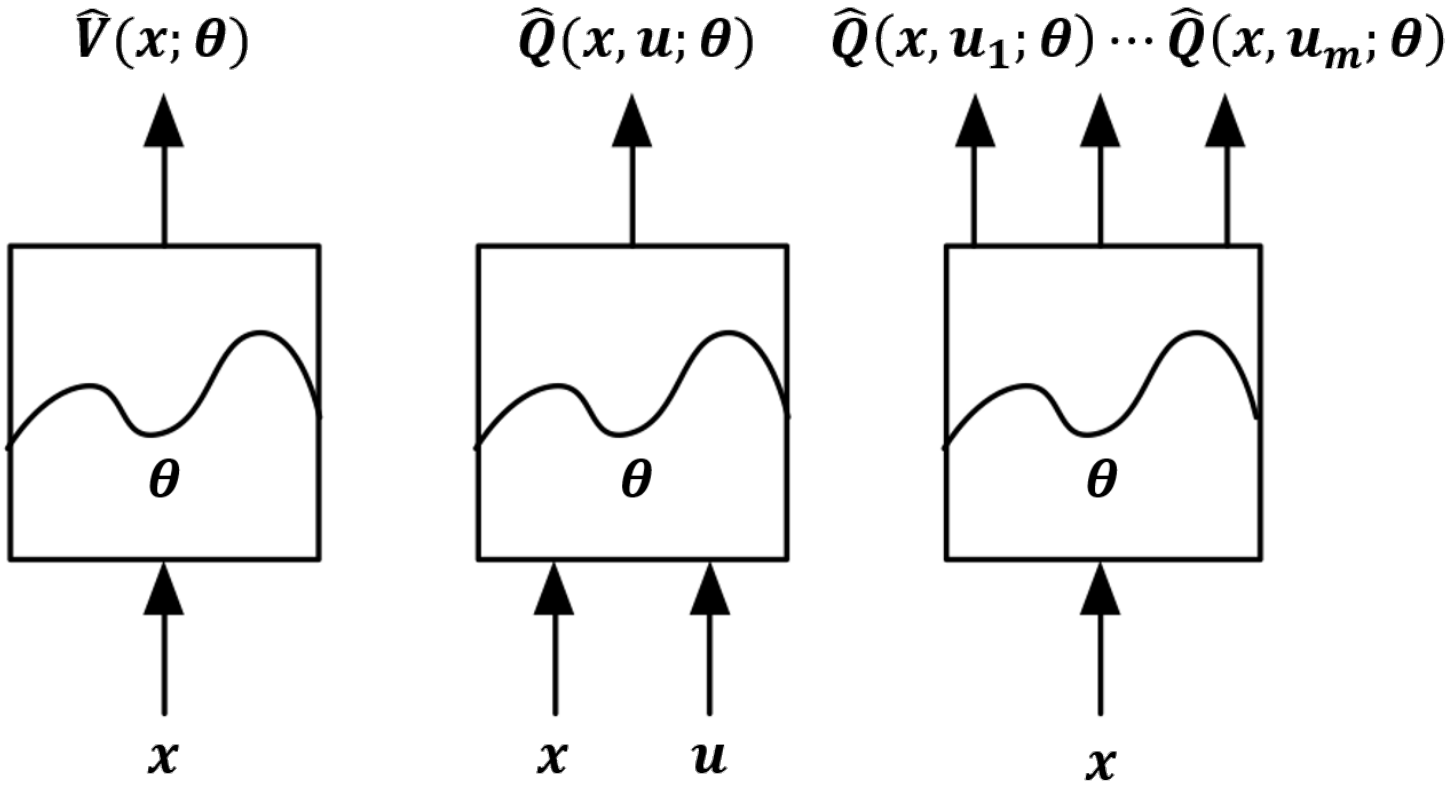
Value Function Approximation

- So far we have been using vectors to represent the value function:
  - Every state x has an entry $V^\pi(x)$
  - Every state-control pair (x, u) has an entry $Q^\pi(x, u)$
- Challenges in very large and continuous state and control spaces:
  - There might be too many (even infinitely many) states and controls to store in memory
  - It might be too slow to learn the value of each state individually
- Approach for large and continuous state and control spaces:
  - Represent the value function using function approximation with parameters $\boldsymbol{\theta}$:

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

  - Update the parameters $\boldsymbol{\theta}$ using MC or TD learning and generalize from seen to unseen states

Value Function Approximation



Value Function Approximation

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

Unconstrained Optimization

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

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

- Two approaches to solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$:
  - **Incremental**: use a (stochastic) descent method:

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

  - **Batch**: obtain $\boldsymbol{\theta}^*$ from the (KKT) optimality conditions:

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

Optimality Conditions

First-order Necessary Condition

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

Second-order Necessary Condition

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

Second-order Sufficient Condition

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

Necessary and Sufficient Condition

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

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 $\delta\boldsymbol{\theta}_k$:

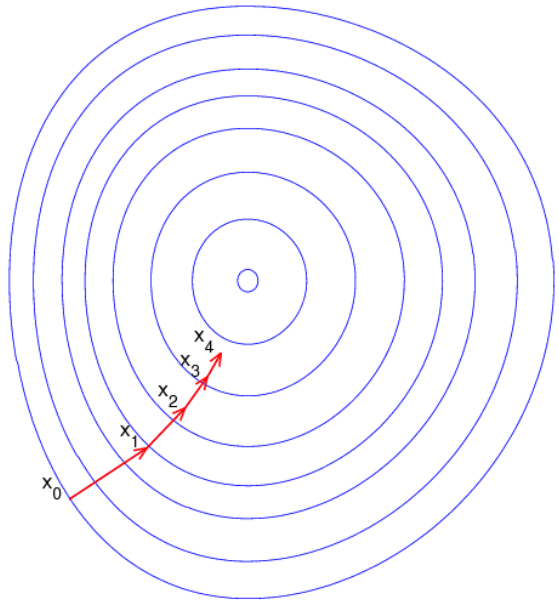
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \delta\boldsymbol{\theta}_k$$

Descent Optimization Methods

- Different methods differ in the way $\delta\boldsymbol{\theta}_k$ and α_k are chosen but
  - $\delta\boldsymbol{\theta}_k$ should be a descent direction: $\nabla J(\boldsymbol{\theta}_k)^T \delta\boldsymbol{\theta}_k < 0$ for all $\boldsymbol{\theta}_k \neq \boldsymbol{\theta}^*$
  - α_k needs to ensure sufficient decrease in J to guarantee convergence:

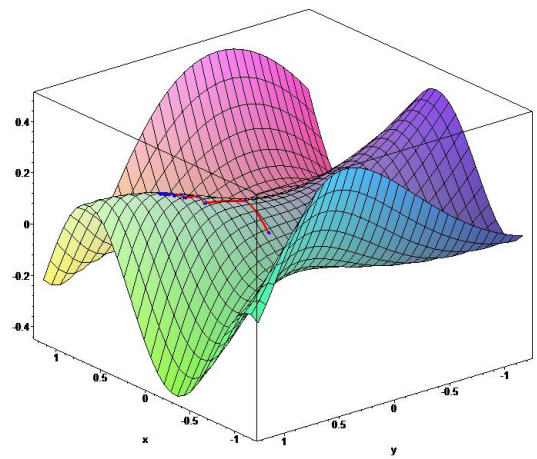
$$\alpha_k^* \in \arg\min_{\alpha>0} J(\boldsymbol{\theta}_k + \alpha\delta\boldsymbol{\theta}_k)$$

Usually obtained via inexact line search



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

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)$$



Value Function Approximation

- **Idea**: find parameters $\boldsymbol{\theta}$ minimizing the mean-squared error between the approximate and true value function of policy π:

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

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

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

Incremental vs Batch optimization

- **Incremental Optimization**:
  - **Gradient descent**:

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

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

  - **Stochastic gradient descent**: uses noisy samples x_t, u_t rather than computing the exact expectation.

$$\delta\boldsymbol{\theta} = \alpha\left(V^{\pi}(x_t) - \hat{V}(x_t;\boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}}\hat{V}(x_t,\boldsymbol{\theta})$$

$$\delta\boldsymbol{\theta} = \alpha\left(Q^{\pi}(x_t,u_t) - \hat{Q}(x_t,u_t;\boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}}\hat{Q}(x_t,u_t;\boldsymbol{\theta})$$

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

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

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

Linear Value Function Approximation

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

$$\hat{V}(x; \boldsymbol{\theta}) = \boldsymbol{\theta}^T \phi(x) = \sum_j \phi_j(x) \boldsymbol{\theta}_j$$

$$\hat{Q}(x, u; \boldsymbol{\theta}) = \boldsymbol{\theta}^T \phi(x, u) = \sum_j \phi_j(x, u) \boldsymbol{\theta}_j$$

Linear Value Function Approximation

- The finite-space representation of $V^{\pi}(x)$ 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 $\boldsymbol{\theta}$ contains the values of the states:

$$\hat{V}(x; \boldsymbol{\theta}) = \begin{bmatrix} \mathbb{1}\{x = 1\} \\ \vdots \\ \mathbb{1}\{x = n\} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\theta}_1 \\ \vdots \\ \boldsymbol{\theta}_n \end{bmatrix}$$

Incremental Methods

Incremental Prediction for Linear Approximation

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

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

- Stochastic gradient descent converges to a *global* optimum
- The update rule is simple:

$$\underbrace{\delta\boldsymbol{\theta}}_{\text{Update}} = \underbrace{\alpha}_{\text{step-size}} \underbrace{\left(V^{\pi}(x_t) - \hat{V}(x_t;\boldsymbol{\theta})\right)}_{\text{prediction error}} \underbrace{\phi(x_t)}_{\text{feature value}}$$

$$\underbrace{\delta\boldsymbol{\theta}}_{\text{Update}} = \underbrace{\alpha}_{\text{step-size}} \underbrace{\left(Q^{\pi}(x_t,u_t) - \hat{Q}(x_t,u_t;\boldsymbol{\theta})\right)}_{\text{prediction error}} \underbrace{\phi(x_t,u_t)}_{\text{feature value}}$$

Incremental Prediction Algorithms

- The (stochastic) gradient descent for optimizing $\boldsymbol{\theta}$ can be performed only if $V^{\pi}(x)$ is available
- In practice, we substitute a *target* for $V^{\pi}(x)$ obtained from noisy samples from the true value $V^{\pi}(x)$:
  - MC: $\mathcal{D} := \{(x_t, L_t)\}$
  - TD: $\mathcal{D} := \left\{\left(x_t, \ell(x_t, u_t) + \gamma \hat{V}(x_{t+1}; \boldsymbol{\theta})\right)\right\}$
  - TD(λ): $\mathcal{D} := \left\{\left(x_t, L_t^{\lambda}\right)\right\}$

Incremental Prediction Algorithms

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

$$\delta\boldsymbol{\theta} = \alpha \left(L_t(\rho) - \hat{V}(x_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{V}(x_t; \boldsymbol{\theta})$$

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

$$\delta\boldsymbol{\theta} = \alpha \left(\ell(x_t, u_t) + \gamma \hat{V}(x_{t+1}; \boldsymbol{\theta}) - \hat{V}(x_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{V}(x_t; \boldsymbol{\theta})$$

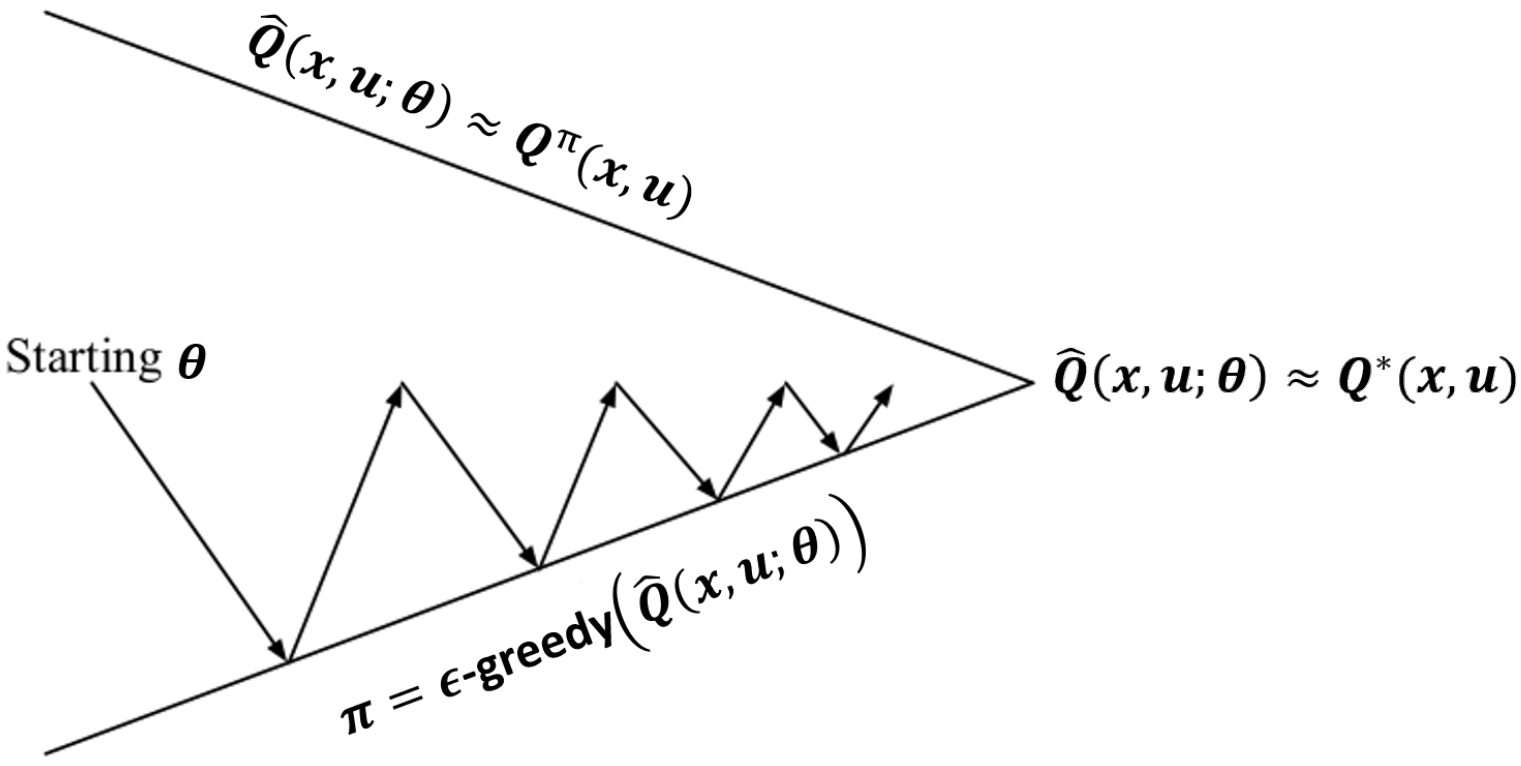
- **Forward-view TD**(λ): the target is the λ-return $L_t^\lambda(\rho)$:

$$\delta\boldsymbol{\theta} = \alpha \left(L_t^\lambda(\rho) - \hat{V}(x_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{V}(x_t; \boldsymbol{\theta})$$

- **Backward-view TD**(λ):

$$\begin{aligned}
\delta_t &= \ell(x_t, u_t) + \gamma \hat{V}(x_{t+1}; \boldsymbol{\theta}) - \hat{V}(x_t; \boldsymbol{\theta}) \\
\mathbf{e}_t &= \gamma\lambda \mathbf{e}_{t-1} + \nabla_{\boldsymbol{\theta}} \hat{V}(x_t; \boldsymbol{\theta}) \\
\delta\boldsymbol{\theta} &= \alpha \delta_t \mathbf{e}_t
\end{aligned}$$

Control with Value Function Approximation



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

Incremental Control Algorithms

- Like for prediction, we must substitute a *target* for $Q^\pi(x, u)$
- **MC**:

$$\Delta\boldsymbol{\theta} = \alpha \left(L_t(\rho) - \hat{Q}(x_t, u_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{Q}(x_t, u_t; \boldsymbol{\theta})$$

- **TD**:

$$\Delta\boldsymbol{\theta} = \alpha \left(\ell(x_t, u_t) + \gamma \hat{Q}(x_{t+1}, u_{t+1}; \boldsymbol{\theta}) - \hat{Q}(x_t, u_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{Q}(x_t, u_t; \boldsymbol{\theta})$$

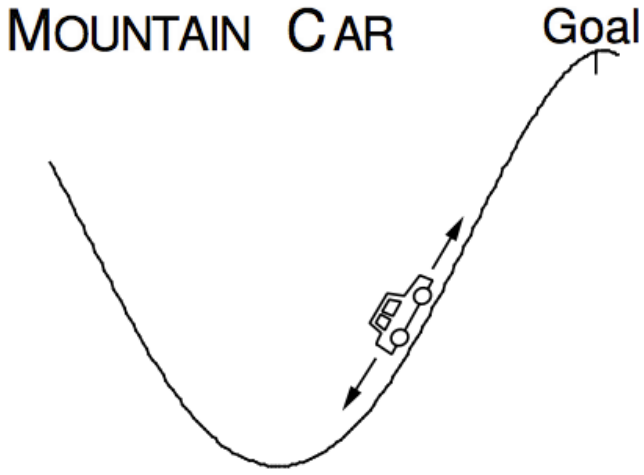
- **Forward-view TD(λ)**:

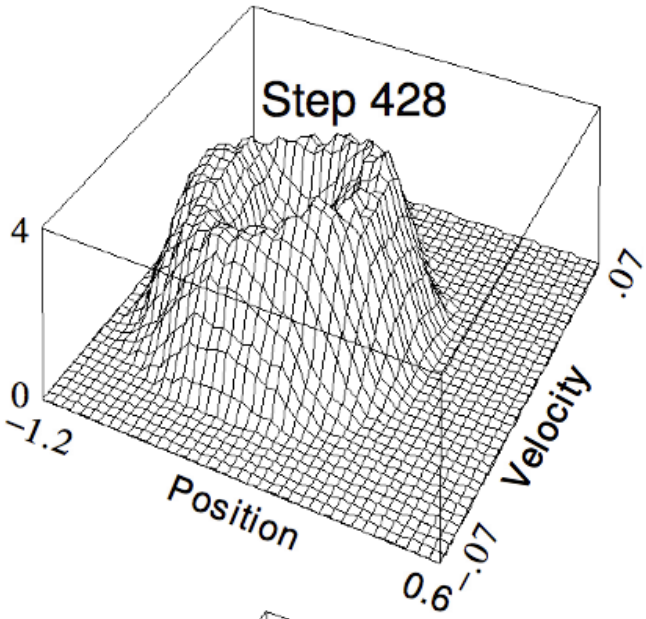
$$\Delta\boldsymbol{\theta} = \alpha \left(L_t^\lambda(\rho) - \hat{Q}(x_t, u_t; \boldsymbol{\theta}) \right) \nabla_{\boldsymbol{\theta}} \hat{Q}(x_t, u_t; \boldsymbol{\theta})$$

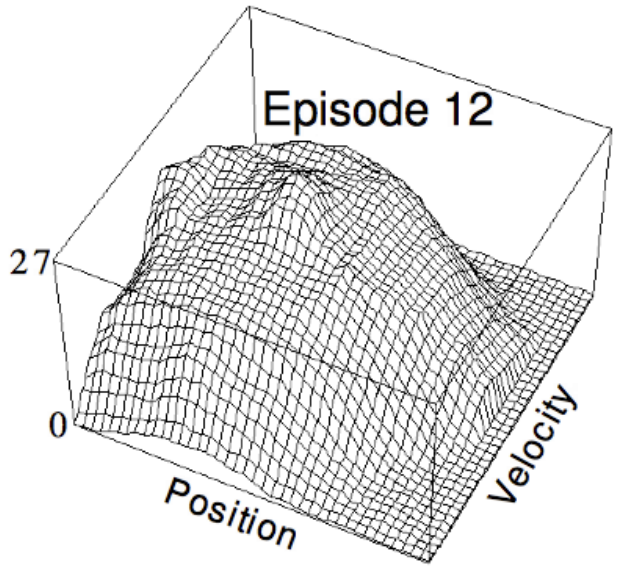
- **Backward-view TD(λ)**:

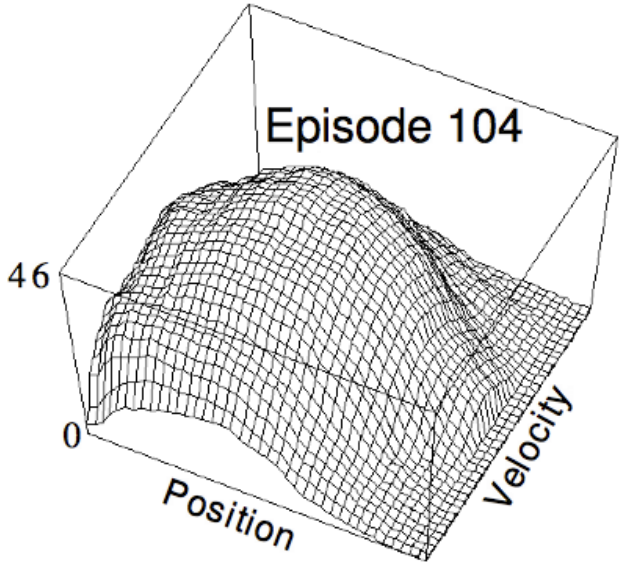
$$\begin{aligned} \delta_t &= \ell(x_t, u_t) + \gamma \hat{Q}(x_{t+1}, u_{t+1}; \boldsymbol{\theta}) - \hat{Q}(x_t, u_t; \boldsymbol{\theta}) \\ \mathbf{e}_t &= \gamma\lambda \mathbf{e}_{t-1} + \nabla_{\boldsymbol{\theta}} \hat{Q}(x_t, u_t; \boldsymbol{\theta}) \\ \Delta\boldsymbol{\theta} &= \alpha \delta_t \mathbf{e}_t \end{aligned}$$

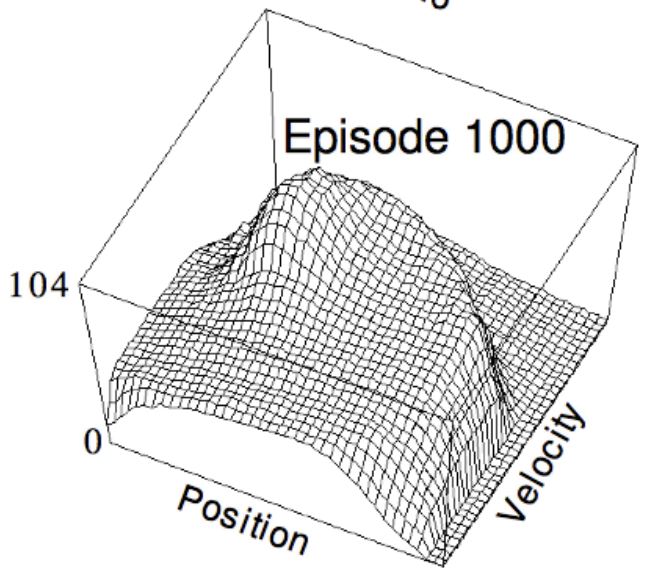
Linear SARSA with Coarse Coding in Mountain Car

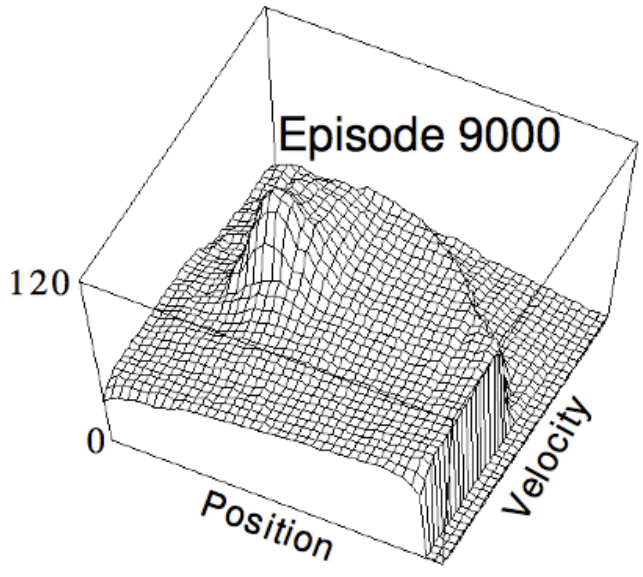












Convergence of Prediction and Control Algorithms

- 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 $\hat{V}(x;\boldsymbol{\theta}) \approx V^{\pi}(x)$
  - Experience $\mathcal{D} := \{(x_t, V^{\pi}(x_t))\}$
- **Goal**: find the best fitting value function approximation:

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

- **Stochastic Gradient Descent with Experience Replay**:
  1. Sample: $(x_t, V^{\pi}(x_t)) \sim \mathcal{D}$
  2. Apply SDG update: $\delta\boldsymbol{\theta} = \alpha\left(V^{\pi}(x_t) - \hat{V}(x_t;\boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}}\hat{V}(x_t,\boldsymbol{\theta})$
  - SDG 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 $J(\boldsymbol{\theta})$:

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

  - Try to 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 $\hat{V}(x;\boldsymbol{\theta}) = \boldsymbol{\theta}^T\phi(x)$, the function $J(\boldsymbol{\theta})$ is quadratic in $\boldsymbol{\theta}$:

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

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

$$0 = \mathbb{E}\left[\delta\boldsymbol{\theta}\right] = \alpha\sum_{x_t\in\mathcal{D}}(V^\pi(x_t) - \boldsymbol{\theta}^T\phi(x_t))\phi(x_t)$$

$$\left(\sum_{x_t\in\mathcal{D}}\phi(x_t)\phi(x_t)^T\right)\boldsymbol{\theta} = \sum_{x_t\in\mathcal{D}}V^\pi(x_t)\phi(x_t)$$

Linear Least Squares Prediction Algorithms

- In practice, we do not know the true values $V^\pi(x_t)$ and must use noisy/biased samples instead:
- **Least-squares Monte Carlo**:

$$V^\pi(x_t) \approx L_t(\rho)$$

- **Least-squares Temporal Difference**:

$$V^\pi(x_t) \approx \ell(x_t, u_t) + \gamma \hat{V}(x_{t+1}; \boldsymbol{\theta})$$

- **Least-squares TD(λ)**:

$$V^\pi(x_t) \approx L_t^\lambda(\rho)$$

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

Linear Least Squares Prediction Algorithms

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

- **LSMC**:

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

$$0 = \sum_{t=0}^{T} \alpha \left(\ell(x_t, u_t) + \gamma \hat{V}(x_{t+1}; \boldsymbol{\theta}) - \hat{V}(x_t; \boldsymbol{\theta}) \right) \phi(x_t)$$

- **LSTD**:

$$\boldsymbol{\theta}^* = \left(\sum_{t=0}^{T} \phi(x_t) \left(\phi(x_t) - \gamma \phi(x_{t+1}) \right)^T \right)^{-1} \sum_{t=0}^{T} \phi(x_t) \ell(x_t, u_t)$$

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

- **LSTD**(λ):

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

Convergence of Linear Least Squares Prediction Algorithms

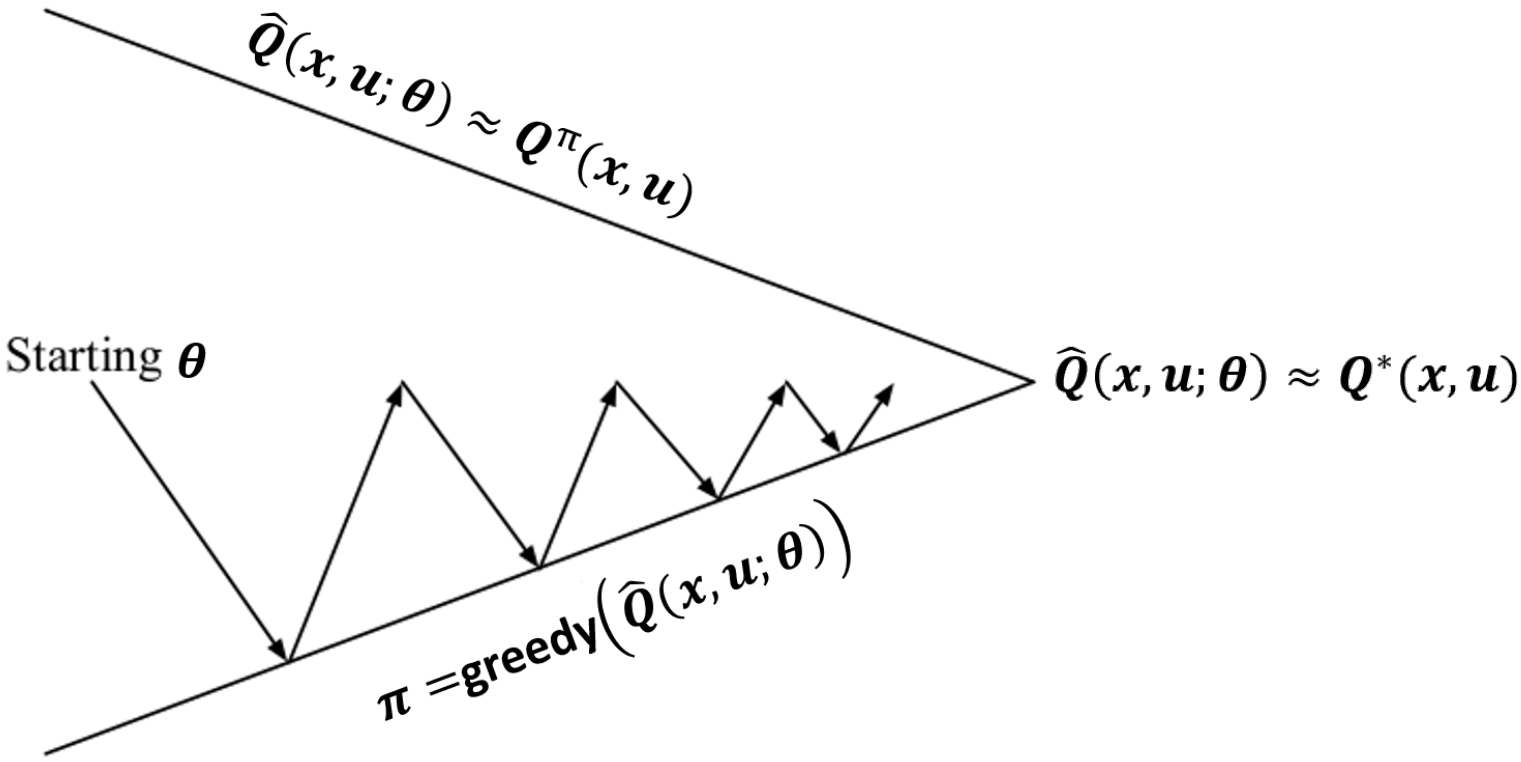
- On-Policy:

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 $\mathcal{D} := \{(x_t, u_t, V^\pi(x_t))\}$ to compute $\hat{Q}(x, u; \boldsymbol{\theta}) = \boldsymbol{\theta}^T \phi(x, u)$
- 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: $x_t, u_t, \ell(x_t, u_t), x_{t+1} \sim \pi_{old}$
  - With new action: $u_{t+1} = \pi_{new}(x_{t+1})$
  - Update $\hat{Q}(x_t, u_t; \boldsymbol{\theta})$ towards the value of the new action: $\ell(x_t, u_t) + \gamma \hat{Q}(x_t, u_{t+1}; \boldsymbol{\theta})$

Least Squares Policy Iteration

- Experience: $x_t, u_t, \ell(x_t, u_t), x_{t+1} \sim \pi_{old}$
- Incremental update:

$$\delta\boldsymbol{\theta} = \alpha \left(\ell(x_t, u_t) + \gamma \hat{Q}(x_{t+1}, \pi(x_{t+1}); \theta) - \hat{Q}(x_t, u_t; \boldsymbol{\theta}) \right) \phi(x_t, u_t)$$

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

$$0 = \sum_{t=0}^{T} \alpha \left(\ell(x_t, u_t) + \gamma \hat{Q}(x_{t+1}, \pi(x_{t+1}); \theta) - \hat{Q}(x_t, u_t; \boldsymbol{\theta}) \right) \phi(x_t, u_t)$$

$$\boldsymbol{\theta}^* = \left(\sum_{t=0}^{T} \phi(x_t, u_t) \left(\phi(x_t, u_t) - \gamma \phi(x_{t+1}, \pi(x_{t+1})) \right)^T \right)^{-1} \sum_{t=0}^{T} \phi(x_t, u_t) \ell(x_t, u_t)$$

Algorithm 1 LSPI-TD

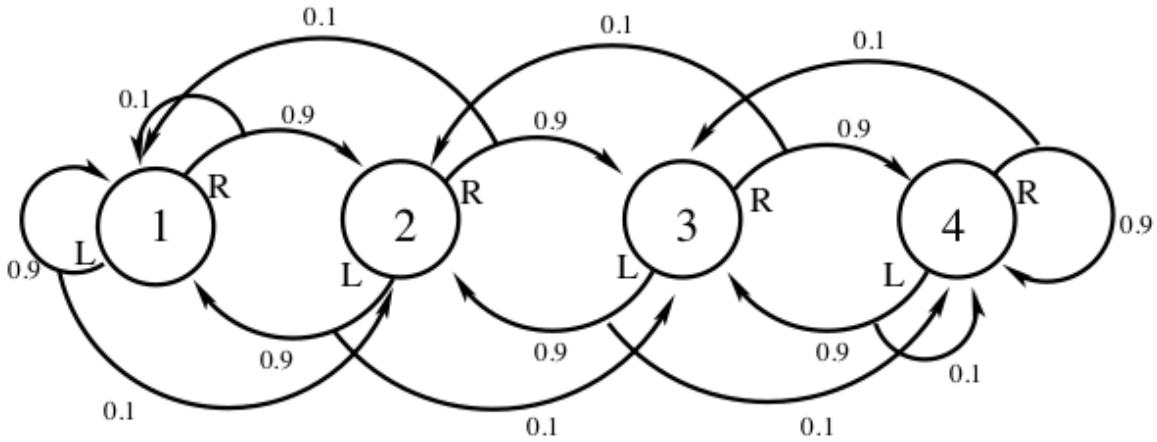
1: **Input**: experience $\mathcal{D}$ and base policy π
2: **loop**
3: $\quad \boldsymbol{\theta}^* \leftarrow$ **LSTDQ**$(\pi, \mathcal{D})$
4: $\quad \pi(x) \leftarrow \arg\min_{u \in \mathcal{U}(x)} \hat{Q}(x, u; \theta^*)$

Convergence of Control Algorithms

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

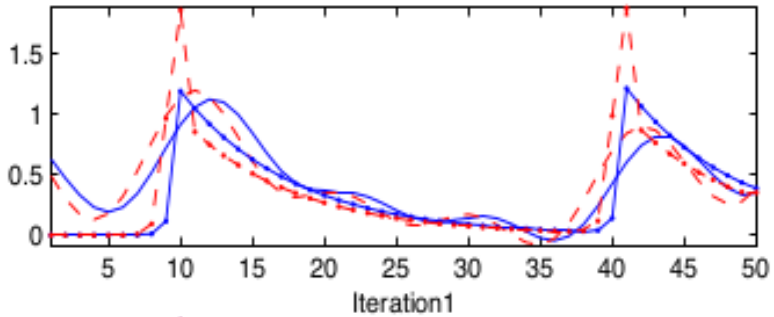
- (✓) = chatters around a near-optimal value function

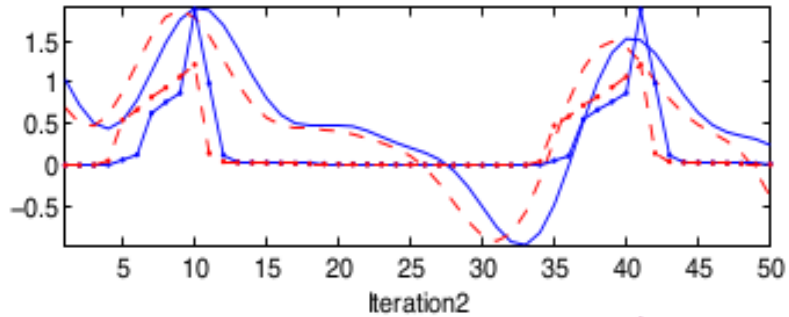
Example: Chain Walk

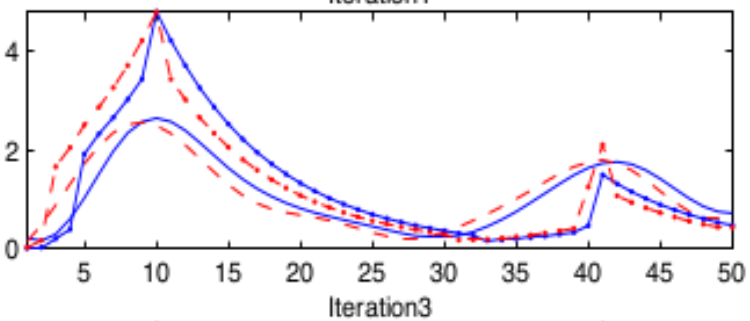


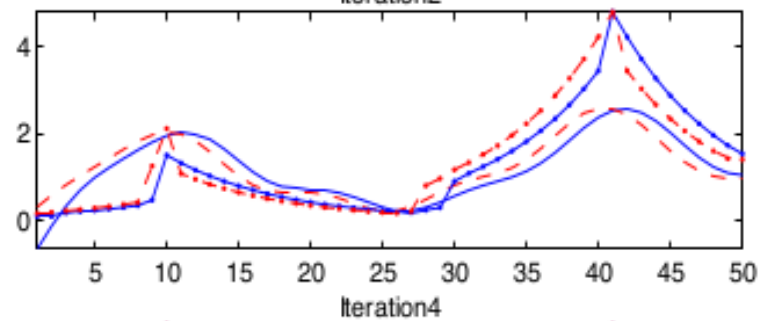
- Consider a 50 state version of the problem
- Cost: -1 in states 10 and 41 and 0 elsewhere
- Optimal policy: $\pi(x) = \begin{cases} R & \text{if } x \in \{1, \ldots, 9\} \cup \{26, \ldots, 41\} \\ L & \text{if } x \in \{10, \ldots, 25\} \cup \{42, \ldots, 50\} \end{cases}$
- Features: 10 evenly spaced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

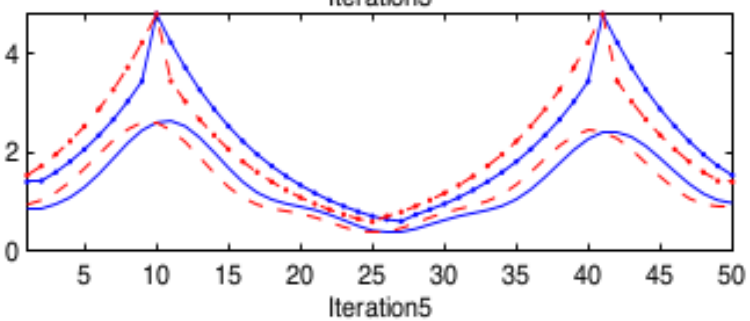
Chain Walk LSPI: Action-Value Function

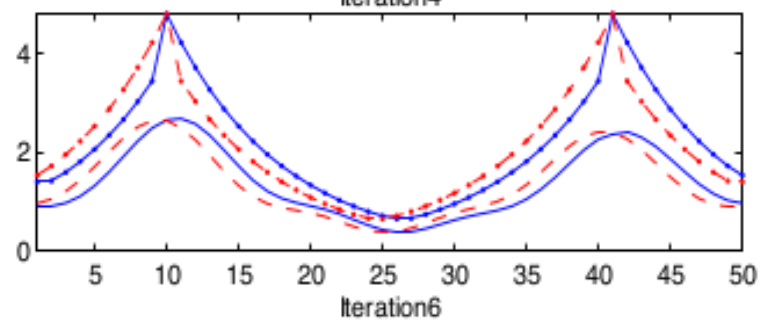


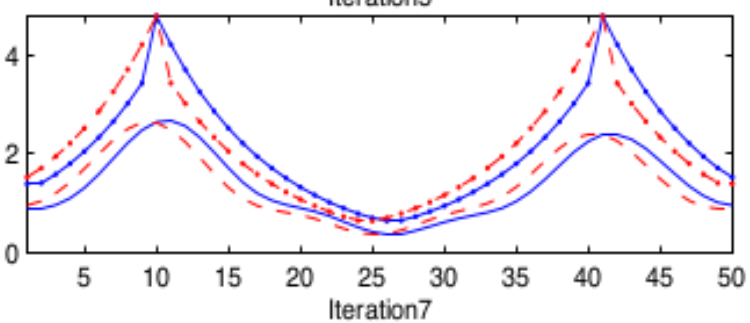






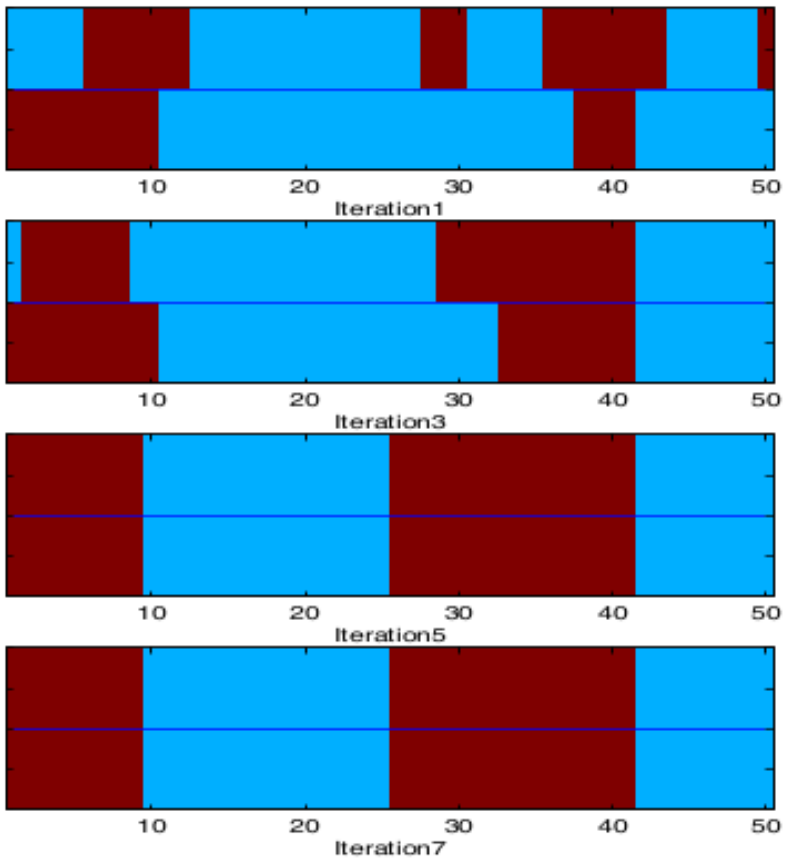


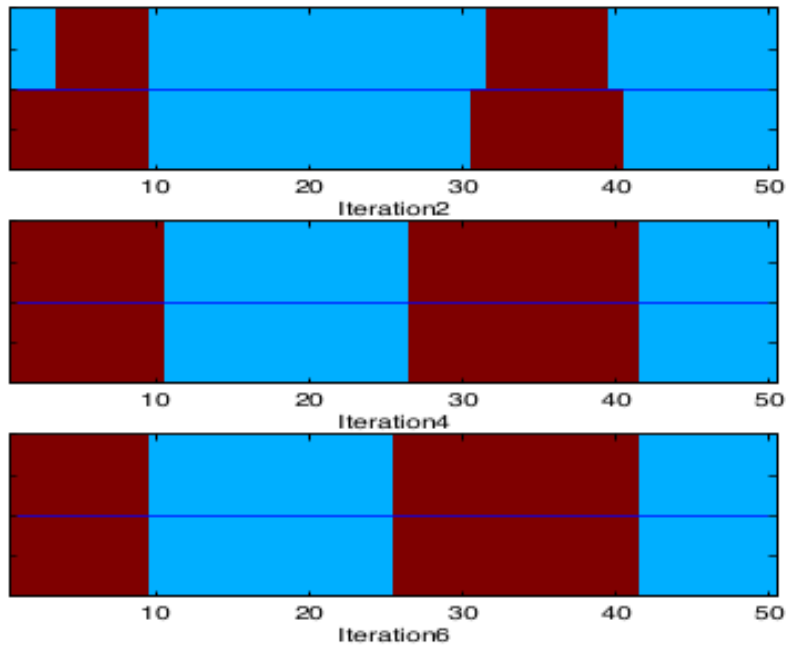




- True (dotted) and approximate (smooth) action-value function
- Left (blue) and right (red) control

Chain Walk LSPI: Policy





- Left (blue) and right (red) control