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

Nikolay Atanasov

natanasov@ucsd.edu

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

# Outline

Value Function Approximation

Incremental Methods

Batch Methods

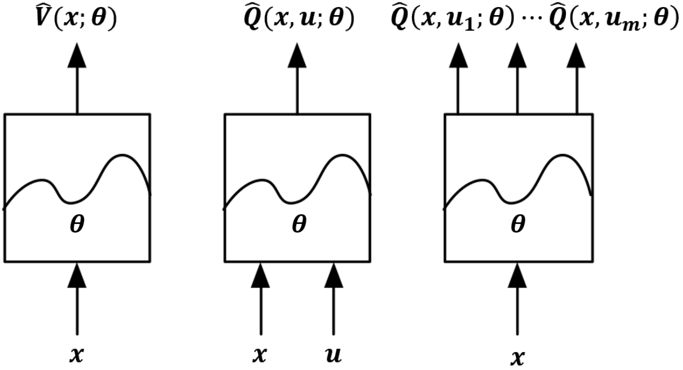
# Optimal Control in Large and Infinite Spaces

- 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 $\boldsymbol{\theta}$:

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

  - update the parameters $\boldsymbol{\theta}$ using MC or TD learning
  - this 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)
  - Neural network (differentiable)
  - Fourier / wavelet base (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 $J(\boldsymbol{\theta})$ measuring the error between $V^{\pi}(\mathbf{x})$ and $\hat{V}(\mathbf{x};\boldsymbol{\theta})$
  - determine the parameters through an optimization problem:

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

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

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

    where $\delta\boldsymbol{\theta}_k$ is a valid descent direction with $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)^\top \delta\boldsymbol{\theta}_k < 0$

  - **Batch**: obtain $\boldsymbol{\theta}^*$ from the 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

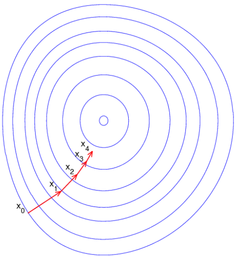
## Descent Direction Theorem

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

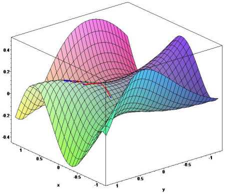



- Methods differ in the way $\delta\boldsymbol{\theta}_k$ and $\alpha_k$ are chosen:
  - $\delta\boldsymbol{\theta}_k$ should be a descent direction: $\nabla J(\boldsymbol{\theta}_k)^T \delta\boldsymbol{\theta}_k < 0$ for all $\boldsymbol{\theta}_k \neq \boldsymbol{\theta}^*$
  - $\alpha_k$ needs to 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 line search

- **Steepest descent direction**: $\delta\boldsymbol{\theta}_k := -\frac{\nabla J(\boldsymbol{\theta}_k)}{\|\nabla J(\boldsymbol{\theta}_k)\|}$



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

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

# Min Square Error Value Function Approximation

- Find parameters $\boldsymbol{\theta}$ minimizing the mean-square error (MSE) between the true and approximate value function of policy $\pi$:

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

  where the expectation is over the state-control distribution induced by $\pi$

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

# Incremental vs Batch optimization

- **Incremental optimization**:
  - **Gradient descent**:

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

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

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

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

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

    The stochastic gradient equals the true gradient in expectation $\mathbb{E}[\delta\boldsymbol{\theta}_t] = \delta\boldsymbol{\theta}$

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

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

# Linear Value Function Approximation

- Associate state $\mathbf{x}$ with feature vector $\phi(\mathbf{x})$ or state-control pair $(\mathbf{x}, \mathbf{u})$ with feature vector $\phi(\mathbf{x}, \mathbf{u})$, e.g.:
  - kernel distance to $n$ landmarks: $\phi(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_n)]^\top$
  - piece and pawn configurations in chess

- Represent the value function by a linear combination of features:

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

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

- Example: finite-space representation of $V^\pi(\mathbf{x})$ over $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ is a special case of linear function approximation with $\phi(\mathbf{x}) = [\mathbb{1}_{\{\mathbf{x}=\mathbf{x}_1\}}, \ldots, \mathbb{1}_{\{\mathbf{x}=\mathbf{x}_n\}}]^\top$ and $\boldsymbol{\theta}$ stores the values of the $n$ points: $\hat{V}(\mathbf{x}; \boldsymbol{\theta}) = \sum_j \theta_j \mathbb{1}_{\{\mathbf{x}=\mathbf{x}_j\}}$

# Outline

Value Function Approximation

Incremental Methods

Batch 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 $\boldsymbol{\theta}$:

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

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

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

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

# Incremental Prediction Algorithms

- The (stochastic) gradient descent for optimizing $\boldsymbol{\theta}$ 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 $\rho = \mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots \sim \pi$:
  - MC: uses a dataset $\mathcal{D} := \{(\mathbf{x}_t, L_t(\rho_t))\}$
  - TD: uses a dataset $\mathcal{D} := \left\{(\mathbf{x}_t, \ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{V}(\mathbf{x}_{t+1}; \boldsymbol{\theta})\right\}$
  - TD($\lambda$): uses a dataset $\mathcal{D} := \left\{(\mathbf{x}_t, L_t^{\lambda}(\rho_t))\right\}$

## Incremental Prediction Algorithms

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

$$\delta\boldsymbol{\theta}_t = \Big(L_t(\rho_t) - \hat{V}(\mathbf{x}_t;\boldsymbol{\theta})\Big)\,\nabla_{\boldsymbol{\theta}}\hat{V}(\mathbf{x}_t;\boldsymbol{\theta})$$

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

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

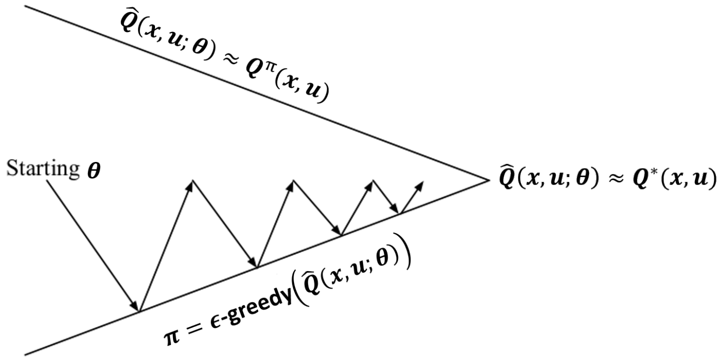
- **Forward-view TD**($\lambda$): the target is the $\lambda$-return $L_t^\lambda(\rho_t)$:

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

- **Backward-view TD**($\lambda$):

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

# Control with Value Function Approximation




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

## Incremental Control Algorithms

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

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

- **TD**:

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

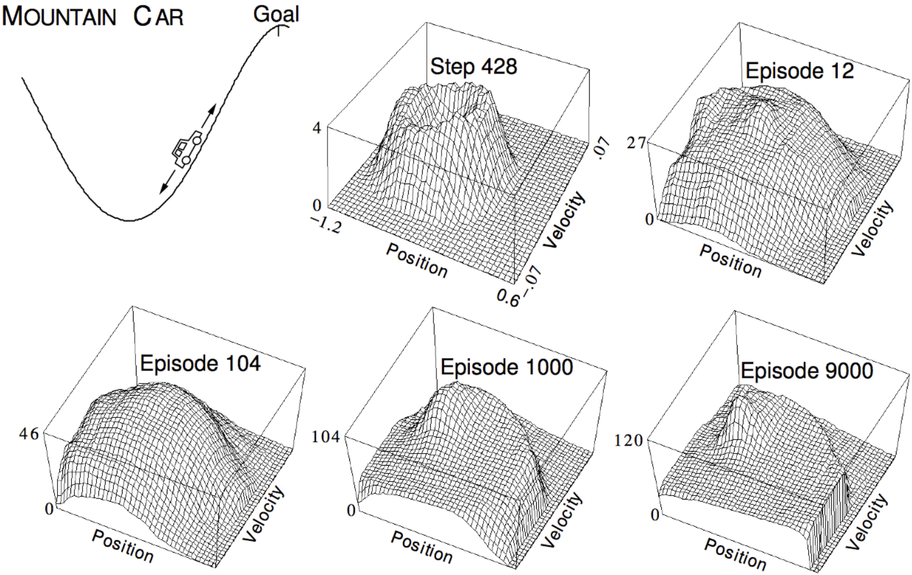
- **Forward-view TD**($\lambda$):

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

- **Backward-view TD**($\lambda$):

$$\begin{aligned} \delta_t &= \ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{Q}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1}; \boldsymbol{\theta}) - \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta}) \\ \mathbf{e}_t &= \gamma\lambda \mathbf{e}_{t-1} + \nabla_{\boldsymbol{\theta}} \hat{Q}(\mathbf{x}_t, \mathbf{u}_t; \boldsymbol{\theta}) \\ \delta\boldsymbol{\theta}_t &= \delta_t \mathbf{e}_t \end{aligned}$$

# 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

# Outline

# 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 (SGD) with experience replay**:
  1. Sample: $(\mathbf{x}_t, V^{\pi}(\mathbf{x}_t)) \sim \mathcal{D}$
  2. Apply SGD update with $\delta\boldsymbol{\theta}_t = \left(V^{\pi}(\mathbf{x}_t) - \hat{V}(\mathbf{x}_t;\boldsymbol{\theta})\right) \nabla_{\boldsymbol{\theta}} \hat{V}(\mathbf{x}_t, \boldsymbol{\theta})$
  - 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 $J(\boldsymbol{\theta})$:

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

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

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

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

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

$$\left(\sum_{\mathbf{x}_t \in \mathcal{D}} \phi(\mathbf{x}_t)\phi(\mathbf{x}_t)^\top\right)\boldsymbol{\theta} = \sum_{\mathbf{x}_t \in \mathcal{D}} V^\pi(\mathbf{x}_t)\phi(\mathbf{x}_t)$$

## Linear Least Squares Prediction Algorithms

- We do not know the true values $V^\pi(\mathbf{x}_t)$ and must use noisy samples instead
- **Least-Squares Monte Carlo (LSMC)**:

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

- **Least-Squares Temporal Difference (LSTD)**:

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

- **Least-Squares TD($\lambda$) (LSTD($\lambda$))**:

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

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

## Linear Least-Squares Prediction Algorithms

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

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

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

# Convergence of Linear-Least Squares Prediction Algorithms

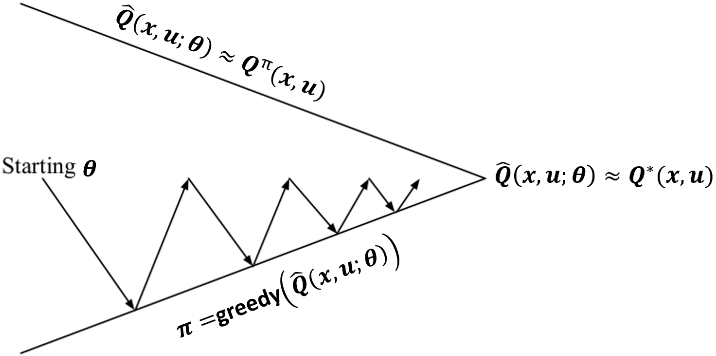
- On-Policy:

| 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 $\epsilon$-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 in PI is changing, the experience is generated from many different policies and we must approximate $Q^{\pi}$ 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 new action value: $\ell(\mathbf{x}_t, \mathbf{u}_t) + \gamma \hat{Q}(\mathbf{x}_t, \mathbf{u}_{t+1}; \boldsymbol{\theta})$

## 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:

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

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

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

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

**Algorithm** LSPI-TD

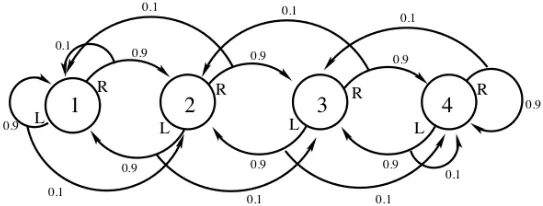
```
Input: experience D and base policy π
loop
    θ* ← LSTDQ(π, D)
    π(x) ← arg min_{u∈U(x)} Q̂(x, u; θ*)
```

## Convergence of Control Algorithms

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

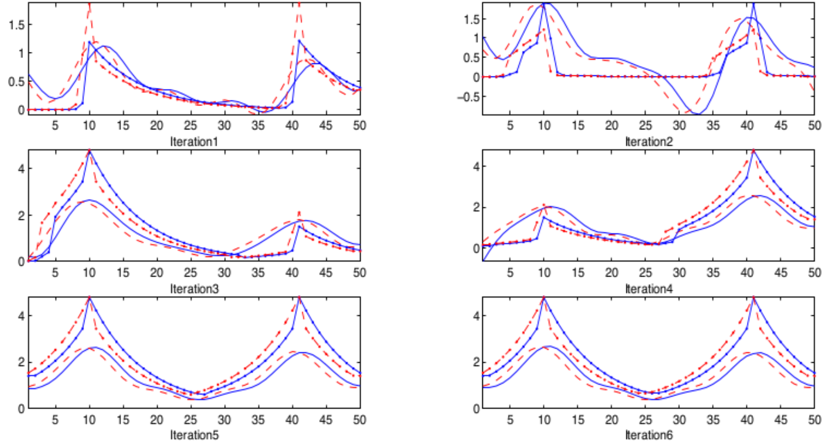
- (✓) = chatters around a near-optimal value function

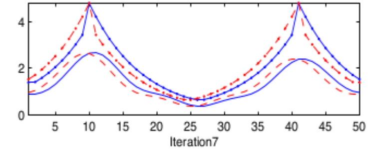
# Example: Chain Walk




- 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

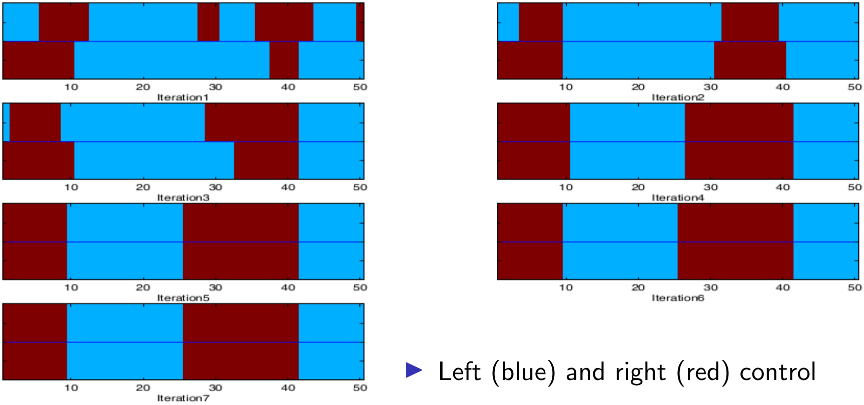
# Chain Walk LSPI: Action-Value Function




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

# Chain Walk LSPI: Policy




- Left (blue) and right (red) control