Lecture 6: Value Function Approximation

David Silver

Outline

Large-Scale Reinforcement Learning

Reinforcement learning can be used to solve *large* problems, e.g.

- Backgammon: 10^{20} states
- Computer Go: 10^{170} states
- Helicopter: continuous state space

How can we scale up the model-free methods for *prediction* and *control* from the last two lectures?

Value Function Approximation

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- 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
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$V_\theta(s) \approx V^\pi(s)$$
$$\text{or } Q_\theta(s, a) \approx Q^\pi(s, a)$$

  - *Generalise* from seen states to unseen states
  - *Update* parameter θ using MC or TD learning

Which Function Approximator?

There are many function approximators, e.g.

- Artificial neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- Coarse coding

In principle, *any* function approximator can be used. However, the choice may be affected by some properties of RL:

- Experience is not i.i.d. - successive time-steps are correlated
- During control, value function $V^{\pi}(s)$ is *non-stationary*
- Agent's actions affect the subsequent data it receives
- Feedback is delayed, not instantaneous

Gradient Descent

- Let $J(\theta)$ be a differentiable function of parameter vector θ
- Define the *gradient* of $J(\theta)$ to be

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- To find a local minimum of $J(\theta)$
- Adjust the parameter θ in the direction of -ve gradient

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

where α is a step-size parameter

Value Function Approx. By Stochastic Gradient Descent

- Goal: find parameter vector θ minimising mean-squared error between approximate value fn $V_\theta(s)$ and true value fn $V^\pi(s)$

$$J(\theta) = \mathbb{E}_\pi\left[(V^\pi(s) - V_\theta(s))^2 \mid s_t = s\right]$$

- Gradient descent finds a local minimum

$$\begin{aligned}\Delta\theta &= -\frac{1}{2}\alpha\nabla_\theta J(\theta) \\ &= \alpha\mathbb{E}_\pi\left[(V^\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s) \mid s_t = s\right]\end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta\theta = \alpha(V^\pi(s) - V_\theta(s))\nabla_\theta V_\theta(s)$$

- Expected update is equal to full gradient update

Feature Vectors

- Represent state by a *feature vector*

$$\phi(s) = \begin{pmatrix} \phi_1(s) \\ \vdots \\ \phi_n(s) \end{pmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$V_\theta(s) = \phi(s)^\top \theta = \sum_{j=1}^{n} \phi_j(s)\theta_j$$

- Objective function is quadratic in parameters θ

$$J(\theta) = \mathbb{E}_\pi \left[(V^\pi(s) - \phi(s)^\top \theta)^2 \mid s_t = s \right]$$

- Stochastic gradient descent converges on *global* optimum
- Update rule is particularly simple

$$\begin{aligned} \nabla_\theta V_\theta(s) &= \phi(s) \\ \Delta\theta &= \alpha(V^\pi(s) - V_\theta(s))\phi(s) \end{aligned}$$

Update $=$ *step-size* $\times$ *prediction error* $\times$ *feature value*

Table Lookup Features

- Table lookup is a special case of linear value function approximation
- Using *table lookup features*

$$\phi^{table}(s) = \begin{pmatrix} \mathbf{1}(s = s_1) \\ \vdots \\ \mathbf{1}(s = s_n) \end{pmatrix}$$

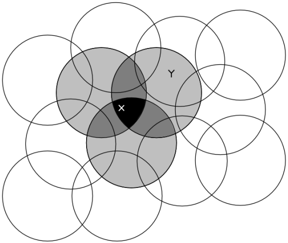
- Parameter vector θ gives value of each individual state

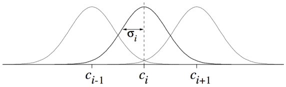
$$V(s) = \begin{pmatrix} \mathbf{1}(s = s_1) \\ \vdots \\ \mathbf{1}(s = s_n) \end{pmatrix} \cdot \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$$

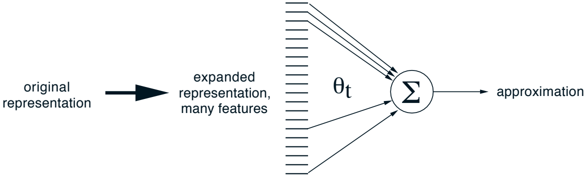
Coarse Coding

Example of linear value function approximation:

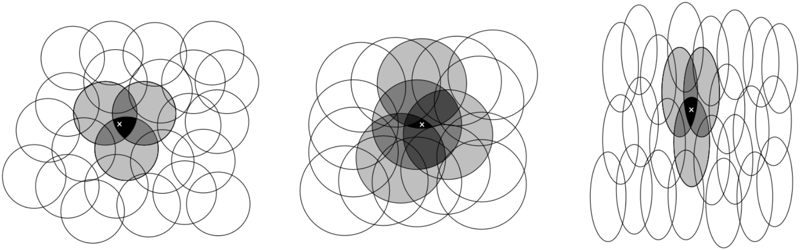
- *Coarse coding* provides large feature vector $\phi(s)$
- Parameter vector θ gives a value to each feature





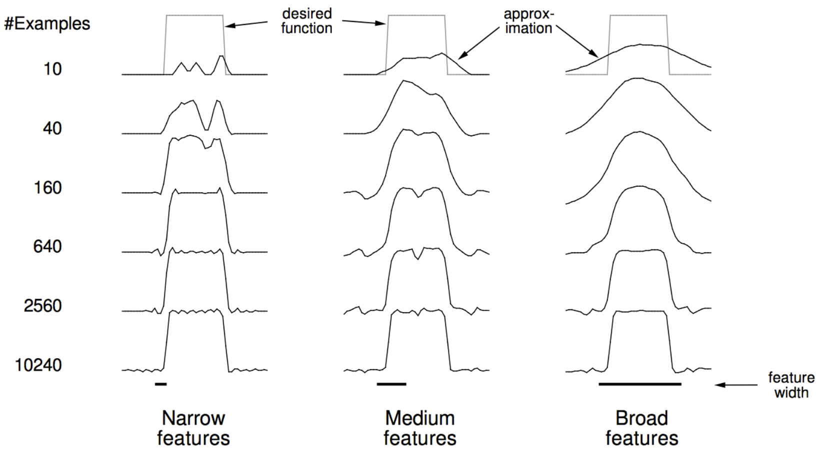


Generalization in Coarse Coding



a) Narrow generalization b) Broad generalization c) Asymmetric generalization

Stochastic Gradient Descent with Coarse Coding



Incremental Prediction Algorithms

- Have assumed true value function $V^{\pi}(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $V^{\pi}(s)$
  - For MC, the target is the return v_t

$$\Delta\theta = \alpha(v_t - V_\theta(s))\nabla_\theta V_\theta(s)$$

  - For TD(0), the target is the TD target $r + \gamma V_\theta(s')$

$$\Delta\theta = \alpha(r + \gamma V_\theta(s') - V_\theta(s))\nabla_\theta V_\theta(s)$$

  - For TD(λ), the target is the λ-return v_t^λ

$$\Delta\theta = \alpha(v_t^\lambda - V_\theta(s))\nabla_\theta V_\theta(s)$$

Monte-Carlo with Value Function Approximation

- The return v_t is an unbiased, noisy sample of true value $V^\pi(s)$
- Can therefore apply supervised learning to "training data":

$$\langle s_1, v_1 \rangle, \langle s_2, v_2 \rangle, ..., \langle s_T, v_T \rangle$$

- For example, using *linear Monte-Carlo policy evaluation*

$$\begin{aligned} \Delta\theta &= \alpha({\color{red}v_t} - V_\theta(s))\nabla_\theta V_\theta(s) \\ &= \alpha(v_t - V_\theta(s))\phi(s) \end{aligned}$$

- Monte-Carlo evaluation converges to a local optimum
- Even when using non-linear value function approximation

TD Learning with Value Function Approximation

- The TD-target $r_{t+1} + \gamma V_\theta(s_{t+1})$ is a *biased* sample of true value $V^\pi(s_t)$
- Can still apply supervised learning to "training data":

$$\langle s_1, r_2 + \gamma V_\theta(s_2) \rangle, \langle s_2, r_3 + \gamma V_\theta(s_3) \rangle, ..., \langle s_{T-1}, r_T \rangle$$

- For example, using *linear TD(0)*

$$\begin{aligned} \Delta\theta &= \alpha(r + \gamma V_\theta(s') - V_\theta(s))\nabla_\theta V_\theta(s) \\ &= \alpha\delta\phi(s) \end{aligned}$$

- Linear TD(0) converges (close) to global optimum

TD(λ) with Value Function Approximation

- The λ-return v_t^λ is also a biased sample of true value $V^\pi(s)$
- Can again apply supervised learning to "training data":

$$\left\langle s_1, v_1^\lambda \right\rangle, \left\langle s_2, v_2^\lambda \right\rangle, ..., \left\langle s_{T-1}, v_{T-1}^\lambda \right\rangle$$

- Forward view linear TD(λ)

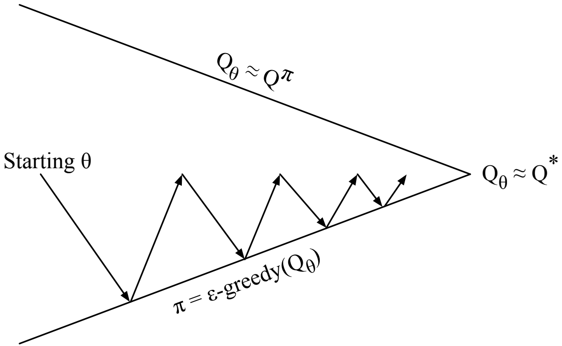
$$\begin{aligned} \Delta\theta &= \alpha(v_t^\lambda - V_\theta(s_t))\nabla_\theta V_\theta(s_t) \\ &= \alpha(v_t^\lambda - V_\theta(s_t))\phi(s_t) \end{aligned}$$

- Backward view linear TD(λ)

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \\ e_t &= \gamma\lambda e_{t-1} + \phi(s_t) \\ \Delta\theta &= \alpha\delta_t e_t \end{aligned}$$

Forward view and backward view linear TD(λ) are equivalent

Control with Value Function Approximation



Policy evaluation Approximate policy evaluation, $Q_\theta \approx Q^\pi$

Policy improvement ϵ-greedy policy improvement

Action-Value Function Approximation

- Approximate the action-value function

$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- Minimise mean-squared error between approximate action-value fn $Q_\theta(s,a)$ and true action-value fn $Q^\pi(s,a)$

$$J(\theta) = \mathbb{E}_\pi\left[\left(Q^\pi(s,a) - Q_\theta(s,a)\right)^2 \mid s_t = s\right]$$

- Use stochastic gradient descent to find a local minimum

$$\begin{aligned} -\frac{1}{2}\nabla_\theta J(\theta) &= (Q^\pi(s,a) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a) \\ \Delta\theta &= \alpha(Q^\pi(s,a) - Q_\theta(s,a))\nabla_\theta Q_\theta(s,a) \end{aligned}$$

Linear Action-Value Function Approximation

- Represent state *and* action by a *feature vector*

$$\phi(s, a) = \begin{pmatrix} \phi_1(s, a) \\ \vdots \\ \phi_n(s, a) \end{pmatrix}$$

- Represent action-value fn by linear combination of features

$$Q_\theta(s, a) = \phi(s, a)^\top \theta = \sum_{j=1}^{n} \phi_j(s, a)\theta_j$$

- Stochastic gradient descent update

$$\nabla_\theta Q_\theta(s, a) = \phi(s, a)$$
$$\Delta\theta = \alpha(Q^\pi(s, a) - Q_\theta(s, a))\phi(s)$$

Incremental Control Algorithms

- Like prediction, we must substitute a *target* for $Q^\pi(s, a)$
  - For MC, the target is the return v_t

$$\Delta\theta = \alpha(v_t - Q_\theta(s_t, a_t))\phi(s_t, a_t)$$

  - For TD(0), the target is the TD target $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$

$$\Delta\theta = \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t))\phi(s_t, a_t)$$

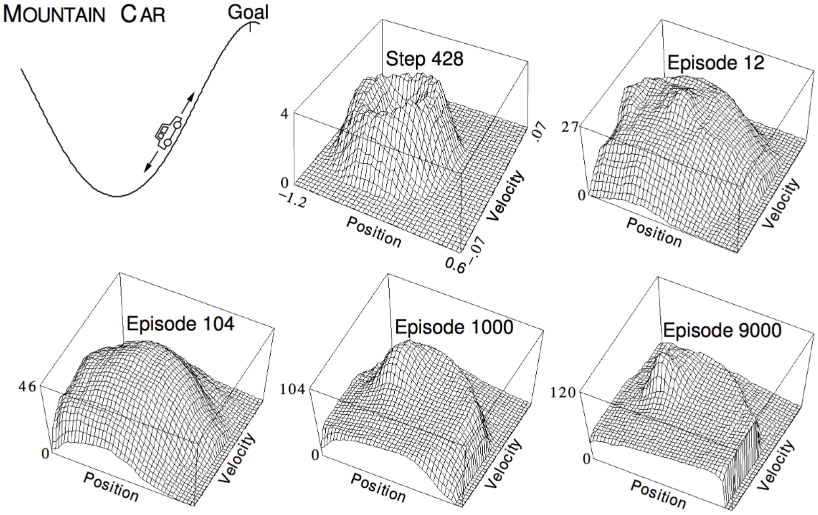
  - For forward-view TD(λ), target is the action-value λ-return

$$\Delta\theta = \alpha(q_t^\lambda - Q_\theta(s_t, a_t))\phi(s_t, a_t)$$

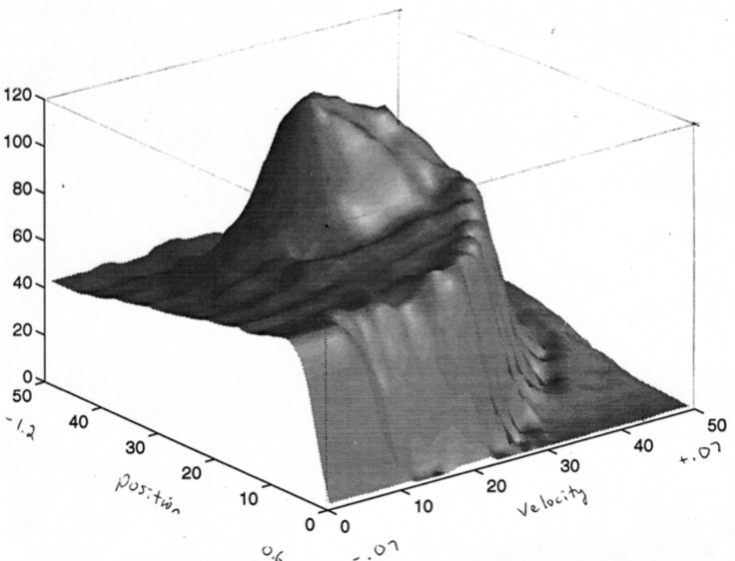
  - For backward-view TD(λ), equivalent update is

$$\begin{aligned}
\delta_t &= r_{t+1} + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t) \\
e_t &= \gamma\lambda e_{t-1} + \phi(s_t, a_t) \\
\Delta\theta &= \alpha\delta_t e_t
\end{aligned}$$

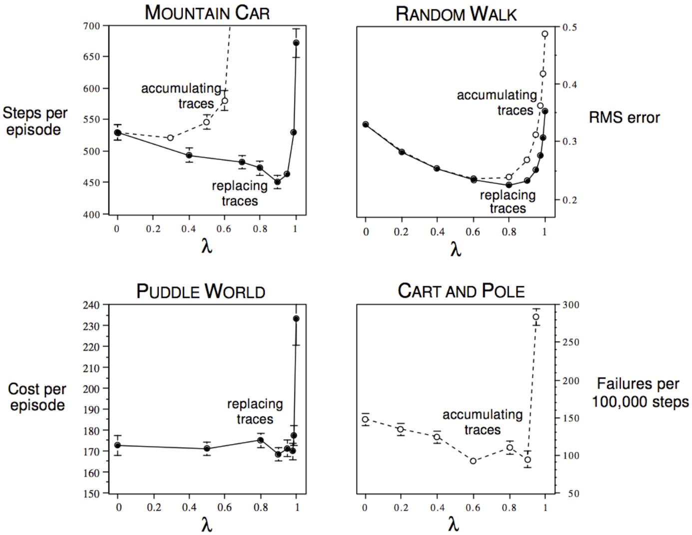
Linear Sarsa with Coarse Coding in Mountain Car



Linear Sarsa with Radial Basis Functions in Mountain Car



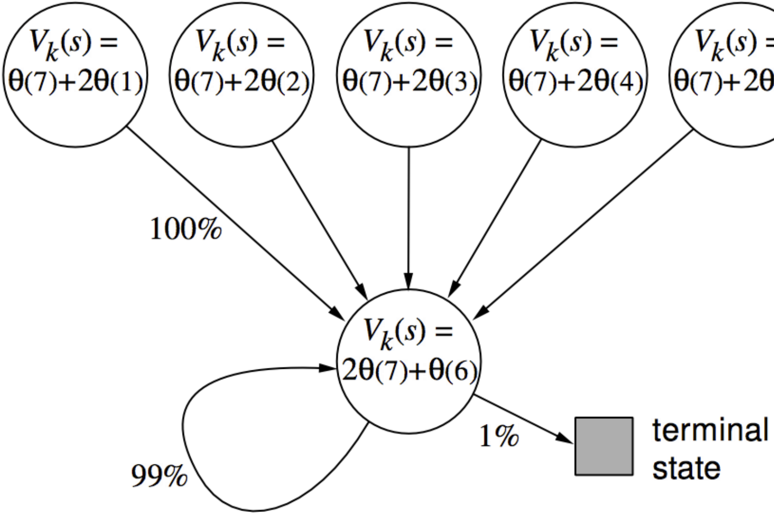
Study of λ: Should We Bootstrap?



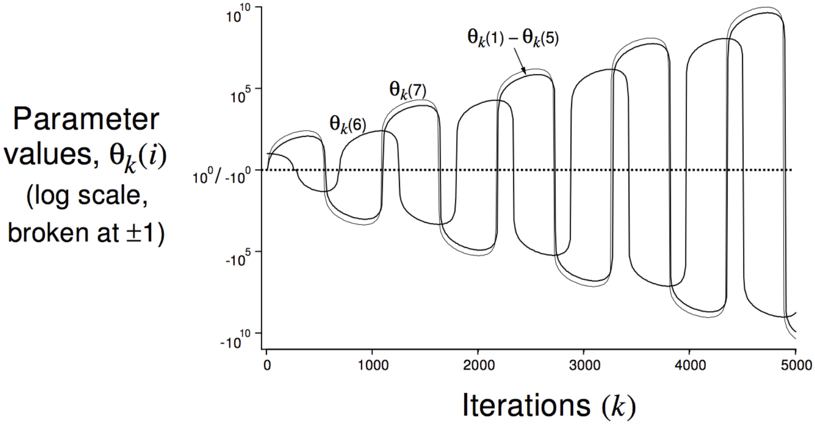
Convergence Questions

- The previous results show it is desirable to bootstrap
- But now we consider convergence issues
- When do incremental prediction algorithms converge?
  - When using bootstrapping (i.e. TD with $\lambda < 1$)?
  - When using linear value function approximation?
  - When using off-policy learning?
- Ideally, we would like algorithms that converge in all cases

Baird's Counterexample



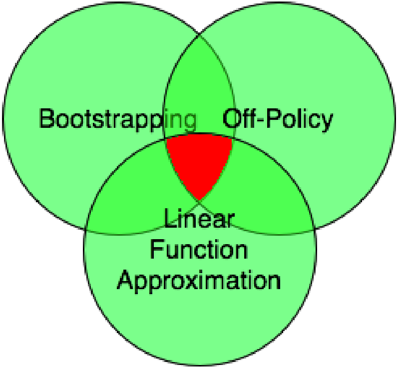
Parameter Divergence in Baird's Counterexample



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)	✓	✗	✗

Gruesome Threesome



We have not *quite* achieved our ideal goal for prediction algorithms.

Gradient Temporal-Difference 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	✓	✓	✓

Convergence of Control Algorithms

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

(✓) = chatters around near-optimal value function

Batch Reinforcement Learning

- Gradient descent is simple and appealing
- But it is *not* sample efficient
- Batch methods seek to find the best fitting value function
- Given the agent's experience ("training data")

Least Squares Prediction

- Given value function approximation $V_\theta(s) \approx V^\pi(s)$
- And *experience* $\mathcal{D}$ consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

- Which parameters θ give the *best fitting* value fn $V_\theta(s)$?
- Least squares algorithms find parameter vector θ minimising sum-squared error between $V_\theta(s_t)$ and target values v_t^π,

$$\begin{aligned} LS(\theta) &= \sum_{t=1}^{T} (v_t^\pi - V_\theta(s_t))^2 \\ &= \mathbb{E}_{\mathcal{D}} \left[(v^\pi - V_\theta(s))^2 \right] \end{aligned}$$

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle \textit{state}, \textit{value} \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, ..., \langle s_T, v_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

$$\Delta\theta = \alpha(v^\pi - V_\theta(s))\nabla_\theta V_\theta(s)$$

Converges to least squares solution

$$\theta^\pi = \underset{\theta}{\operatorname{argmin}}\ LS(\theta)$$

Linear Least Squares Prediction

- Experience replay finds least squares solution
- But it may take many iterations
- Using *linear* value function approximation $V_\theta(s) = \phi(s)^\top \theta$
- We can solve the least squares solution directly

Linear Least Squares Prediction (2)

- At minimum of $LS(\theta)$, the expected update must be zero

$$
\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[\Delta\theta\right] &= 0 \\
\alpha \sum_{t=1}^{T} \phi(s_t)(v_t^\pi - \phi(s_t)^\top \theta) &= 0 \\
\sum_{t=1}^{T} \phi(s_t) v_t^\pi &= \sum_{t=1}^{T} \phi(s_t)\phi(s_t)^\top \theta \\
\theta &= \left(\sum_{t=1}^{T} \phi(s_t)\phi(s_t)^\top \right)^{-1} \sum_{t=1}^{T} \phi(s_t) v_t^\pi
\end{aligned}
$$

- For N features, direct solution time is $O(N^3)$
- Incremental solution time is $O(N^2)$ using Shermann-Morrison

Linear Least Squares Prediction Algorithms

- We do not know true values v_t^π
- In practice, our "training data" must use noisy or biased samples of v_t^π

LSMC Least Squares Monte-Carlo uses return
$v_t^\pi \approx v_t$

LSTD Least Squares Temporal-Difference uses TD target
$v_t^\pi \approx r_{t+1} + \gamma V_\theta(s_{t+1})$

LSTD(λ) Least Squares TD(λ) uses λ-return
$v_t^\pi \approx v_t^\lambda$

- In each case solve directly for fixed point of MC / TD / TD(λ)

Linear Least Squares Prediction Algorithms (2)

LSMC

$$0 = \sum_{t=1}^{T} \alpha (v_t - V_\theta(s_t)) \phi(s_t)$$

$$\theta = \left(\sum_{t=1}^{T} \phi(s_t) \phi(s_t)^\top \right)^{-1} \sum_{t=1}^{T} \phi(s_t) v_t$$

LSTD

$$0 = \sum_{t=1}^{T} \alpha (r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)) \phi(s_t)$$

$$\theta = \left(\sum_{t=1}^{T} \phi(s_t) (\phi(s_t) - \gamma \phi(s_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} \phi(s_t) r_{t+1}$$

LSTD(λ)

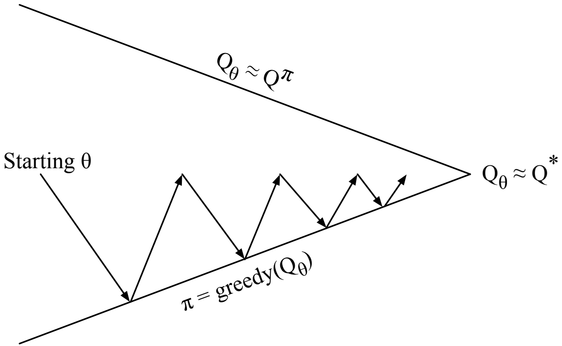
$$0 = \sum_{t=1}^{T} \alpha \delta_t e_t$$

$$\theta = \left(\sum_{t=1}^{T} e_t (\phi(s_t) - \gamma \phi(s_{t+1}))^\top \right)^{-1} \sum_{t=1}^{T} e_t r_{t+1}$$

Convergence of Linear Least Squares Prediction Algorithms

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

Least Squares Policy Iteration



Policy evaluation Policy evaluation by least squares Q-learning

Policy improvement Greedy policy improvement

Least Squares Action-Value Function Approximation

- Approximate action-value function $Q^\pi(s, a)$
- using linear combination of features $\phi(s, a)$

$$Q_\theta(s, a) = \phi(s, a)^\top \theta \approx Q^\pi(s, a)$$

- Minimise least squares error between $Q_\theta(s, a)$ and $Q^\pi(s, a)$
- from experience generated using policy π
- consisting of $\langle (state, action), value \rangle$ pairs

$$\mathcal{D} = \{\langle (s_1, a_1), v_1^\pi \rangle, \langle (s_2, a_2), v_2^\pi \rangle, ..., \langle (s_T, a_T), v_T^\pi \rangle\}$$

Least Squares Control

- For policy evaluation, we want to efficiently use all experience
- For control, we also want to improve the policy
- This experience is generated from many policies
- So to evaluate $Q^{\pi}(s, a)$ we must learn off-policy
- We use the same idea as Q-learning:
  - Use experience generated by old policy $s_t, a_t, r_{t+1}, s_{t+1} \sim \pi_{old}$
  - Consider alternative successor action $a' = \pi_{new}(s_{t+1})$
  - Update $Q_{\theta}(s_t, a_t)$ towards value of alternative action $r_{t+1} + \gamma Q_{\theta}(s_{t+1}, a')$

Least Squares Q-Learning

- Consider the following linear Q-learning update

$$\begin{aligned}\delta &= r_{t+1} + \gamma Q_\theta(s_{t+1}, \pi(s_{t+1})) - Q_\theta(s_t, a_t) \\ \Delta\theta &= \alpha\delta\phi(s_t, a_t)\end{aligned}$$

- LSTDQ algorithm: solve for total update = zero

$$0 = \sum_{t=1}^{T} \alpha(r_{t+1} + \gamma Q_\theta(s_{t+1}, \pi(s_{t+1})) - Q_\theta(s_t, a_t))\phi(s_t, a_t)$$

$$\theta = \left(\sum_{t=1}^{T} \phi(s_t, a_t)(\phi(s_t, a_t) - \gamma\phi(s_{t+1}, \pi(s_{t+1})))^\top\right)^{-1} \sum_{t=1}^{T} \phi(s_t, a_t) r_{t+1}$$

Least Squares Policy Iteration Algorithm

- The following pseudocode uses LSTDQ for policy evaluation
- It repeatedly re-evaluates experience $\mathcal{D}$ with different policies

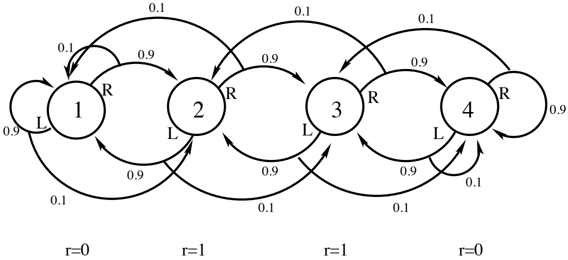
function LSPI-TD$(\mathcal{D}, \pi_0)$
    $\pi' \leftarrow \pi_0$
    repeat
        $\pi \leftarrow \pi'$
        $Q \leftarrow$ **LSTDQ**$(\pi, \mathcal{D})$
        for all $s \in \mathcal{S}$ **do**
            $\pi'(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ Q(s, a)$
  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

Convergence of Control Algorithms

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

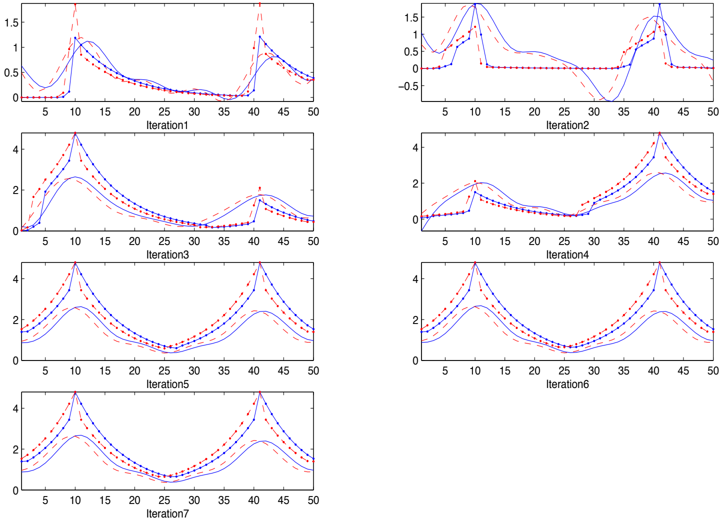
(✓) = chatters around near-optimal value function

Chain Walk Example

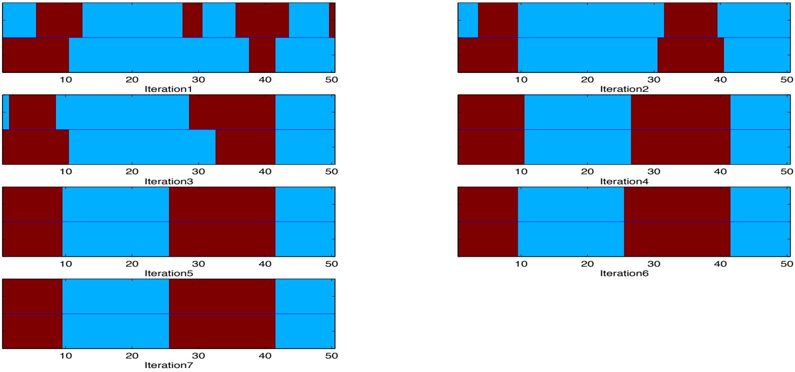


- Consider the 50 state version of this problem
- Reward $+1$ in states 10 and 41, 0 elsewhere
- Optimal policy: R (1-9), L (10-25), R (26-41), L (42, 50)
- Features: 10 evenly spaced Gaussians ($\sigma = 4$) for each action
- Experience: 10,000 steps from random walk policy

LSPI in Chain Walk: Action-Value Function



LSPI in Chain Walk: Policy



Questions?