Chapter 8:
Generalization and Function Approximation

Objectives of this chapter:

- Look at how experience with a limited part of the state set be used to produce good behavior over a much larger part.
- Overview of function approximation (FA) methods and how they can be adapted to RL

Value Prediction with FA

As usual: **Policy Evaluation (the prediction problem)**:

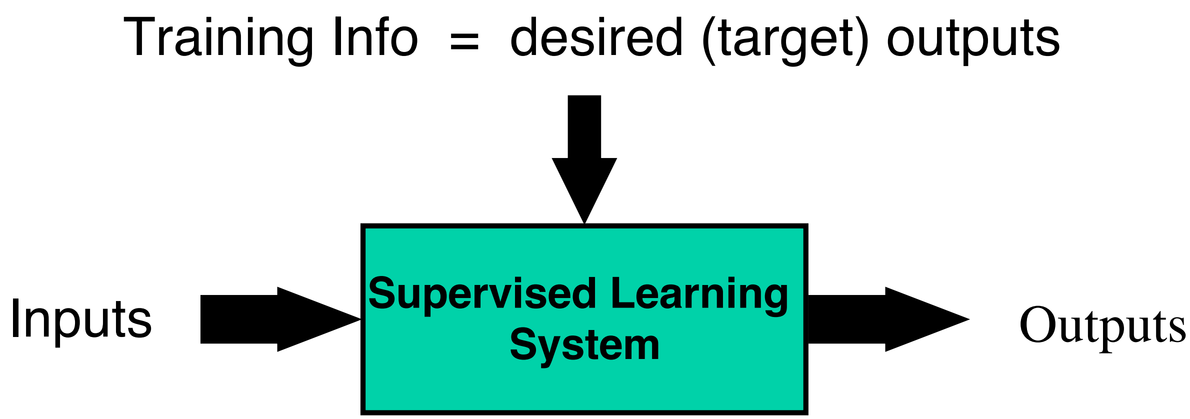
for a given policy π, compute the state-value function V^π

In earlier chapters, value functions were stored in lookup tables.

Here, the value function estimate at time t, V_t, depends on a **parameter vector** $\vec{\theta}_t$, and only the parameter vector is updated.

e.g., $\vec{\theta}_t$ could be the vector of connection weights of a neural network.

Adapt Supervised Learning Algorithms



Training example = {input, target output}

Error = (target output – actual output)

Backups as Training Examples

e.g., the TD(0) backup :

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)\right]$$

As a training example:

$$\left\{\text{description of } s_t,\;\; r_{t+1} + \gamma\, V(s_{t+1})\right\}$$

input — target output

Any FA Method?

- In principle, yes:
  - artificial neural networks
  - decision trees
  - multivariate regression methods
  - etc.
- But RL has some special requirements:
  - usually want to learn while interacting
  - ability to handle nonstationarity
  - other?

Gradient Descent Methods

$$\vec{\theta}_t = \left(\theta_t(1), \theta_t(2), \ldots, \theta_t(n)\right)^T$$

transpose

Assume V_t is a (sufficiently smooth) differentiable function of $\vec{\theta}_t$, for all $s \in S$.

Assume, for now, training examples of this form :

$$\left\{\text{description of } s_t,\ V^\pi(s_t)\right\}$$

Performance Measures

- Many are applicable but…
- a common and simple one is the mean-squared error (MSE) over a distribution *P* :

$$MSE(\theta_t) = \sum_{s \in S} P(s) \left[V^{\pi}(s) - V_t(s) \right]^2$$

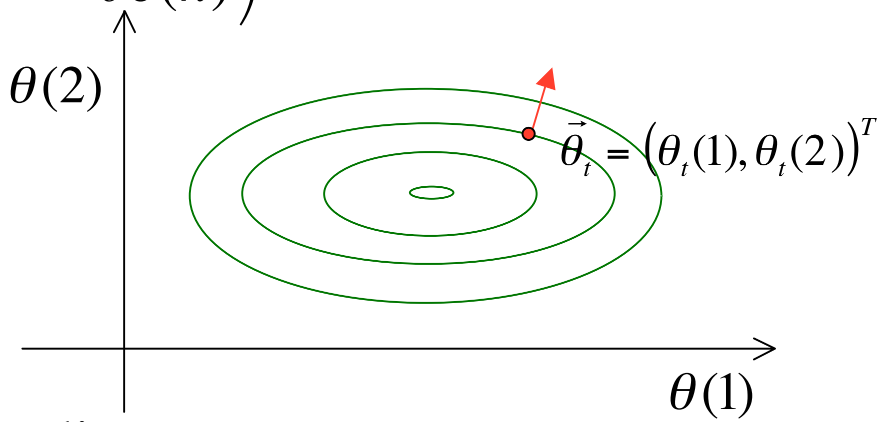
- Why *P* ?
- Why minimize MSE?
- Let us assume that *P* is always the distribution of states with which backups are done.
- The **on-policy distribution**: the distribution created while following the policy being evaluated. Stronger results are available for this distribution.

Gradient Descent

Let *f* be any function of the parameter space.

Its gradient at any point $\vec{\theta}_t$ in this space is:

$$\nabla_{\vec{\theta}} f(\vec{\theta}_t) = \left(\frac{\partial f(\vec{\theta}_t)}{\partial \theta(1)}, \frac{\partial f(\vec{\theta}_t)}{\partial \theta(2)}, \ldots, \frac{\partial f(\vec{\theta}_t)}{\partial \theta(n)} \right)^T$$



Iteratively move down the gradient:

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \alpha \nabla_{\vec{\theta}} f(\vec{\theta}_t)$$

Gradient Descent Cont.

For the MSE given above and using the chain rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \frac{1}{2}\alpha\nabla_{\vec{\theta}} MSE(\vec{\theta}_t)$$

$$= \vec{\theta}_t - \frac{1}{2}\alpha\nabla_{\vec{\theta}} \sum_{s\in S} P(s)\left[V^{\pi}(s) - V_t(s)\right]^2$$

$$= \vec{\theta}_t + \alpha \sum_{s\in S} P(s)\left[V^{\pi}(s) - V_t(s)\right]\nabla_{\vec{\theta}} V_t(s)$$

Gradient Descent Cont.

Use just the **sample gradient** instead:

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \frac{1}{2}\alpha \nabla_{\vec{\theta}} \left[V^\pi(s_t) - V_t(s_t) \right]^2$$

$$= \vec{\theta}_t + \alpha \left[V^\pi(s_t) - V_t(s_t) \right] \nabla_{\vec{\theta}} V_t(s_t),$$

Since each sample gradient is an **unbiased estimate** of the true gradient, this converges to a local minimum of the MSE if α decreases appropriately with t.

$$E\left[V^\pi(s_t) - V_t(s_t) \right] \nabla_{\vec{\theta}} V_t(s_t) = \sum_{s \in S} P(s) \left[V^\pi(s) - V_t(s) \right] \nabla_{\vec{\theta}} V_t(s)$$

But We Don't have these Targets

Suppose we just have targets v_t instead :

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\left[v_t - V_t(s_t)\right]\nabla_{\vec{\theta}} V_t(s_t)$$

If each v_t is an unbiased estimate of $V^\pi(s_t)$,
i.e., $E\{v_t\} = V^\pi(s_t)$, then gradient descent converges
to a local minimum (provided α decreases appropriately).

e.g., the Monte Carlo target $v_t = R_t$:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\left[R_t - V_t(s_t)\right]\nabla_{\vec{\theta}} V_t(s_t)$$

What about TD(λ) Targets?

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\left[R_t^\lambda - V_t(s_t)\right]\nabla_{\vec{\theta}} V_t(s_t)$$

Not unbiased for $\lambda < 1$

But we do it anyway, using the backwards view :

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \delta_t \vec{e}_t,$$

where :

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t), \text{ as usual, and}$$

$$\vec{e}_t = \gamma \lambda \vec{e}_{t-1} + \nabla_{\vec{\theta}} V_t(s_t)$$

On-Line Gradient-Descent TD(λ)

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
  $\vec{e} = 0$
  $s \leftarrow$ initial state of episode
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a, observe reward, r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $\vec{e} \leftarrow \gamma\lambda\vec{e} + \nabla_{\vec{\theta}} V(s)$
    $\vec{\theta} \leftarrow \vec{\theta} + \alpha\,\delta\,\vec{e}$
    $s \leftarrow s'$
  until s is terminal

Linear Methods

Represent states as feature vectors:

for each $s \in S$:

$$\vec{\phi}_s = \left(\phi_s(1), \phi_s(2), \ldots, \phi_s(n)\right)^T$$

$$V_t(s) = \vec{\theta}_t^{\,T} \vec{\phi}_s = \sum_{i=1}^{n} \theta_t(i)\,\phi_s(i)$$

$$\nabla_{\vec{\theta}} V_t(s) = \quad ?$$

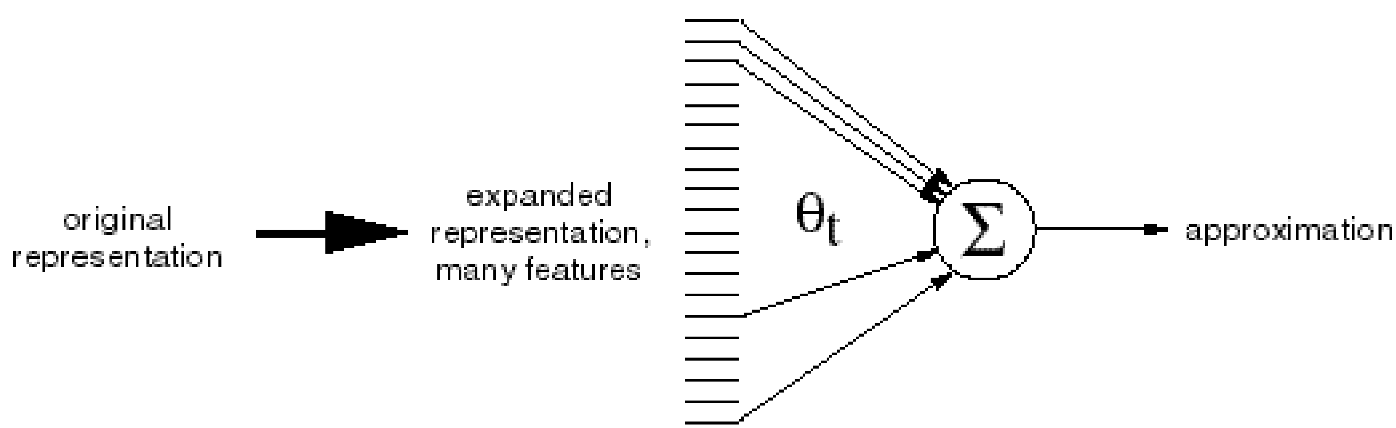
Nice Properties of Linear FA Methods

- The gradient is very simple: $\nabla_{\vec{\theta}} V_t(s) = \vec{\phi}_s$
- For MSE, the error surface is simple: quadratic surface with a single minumum.
- Linear gradient descent TD(λ) converges:
  - Step size decreases appropriately
  - On-line sampling (states sampled from the on-policy distribution)
  - Converges to parameter vector $\vec{\theta}_\infty$ with property:

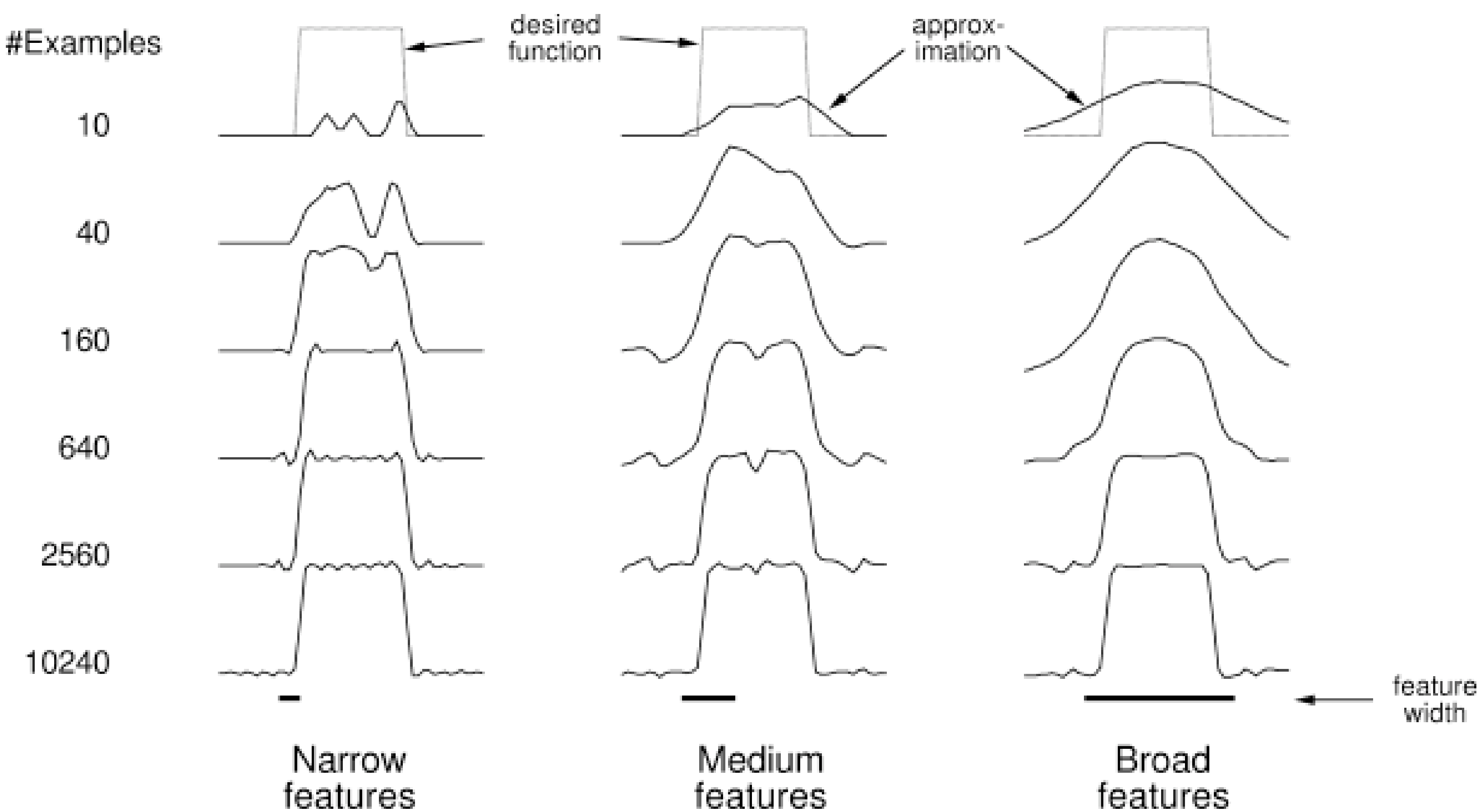
$$MSE(\vec{\theta}_\infty) \leq \frac{1 - \gamma\,\lambda}{1 - \gamma} MSE(\vec{\theta}^*)$$

best parameter vector

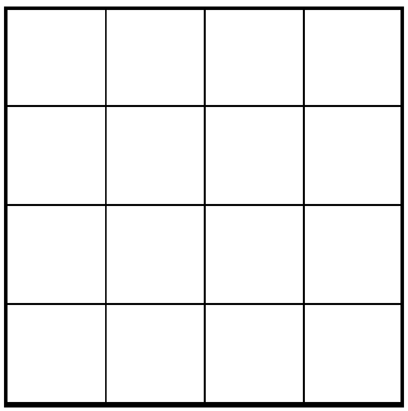
(Tsitsiklis & Van Roy, 1997)


original
representation
expanded
representation,
many features
θ_t
Σ
approximation

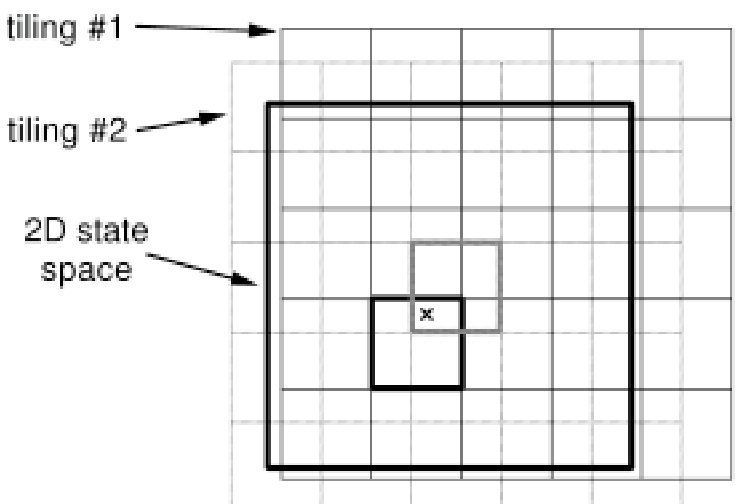
Learning and Coarse Coding



Tile Coding



- Binary feature for each tile
- Number of features present at any one time is constant
- Binary features means weighted sum easy to compute
- Easy to compute indices of the freatures present

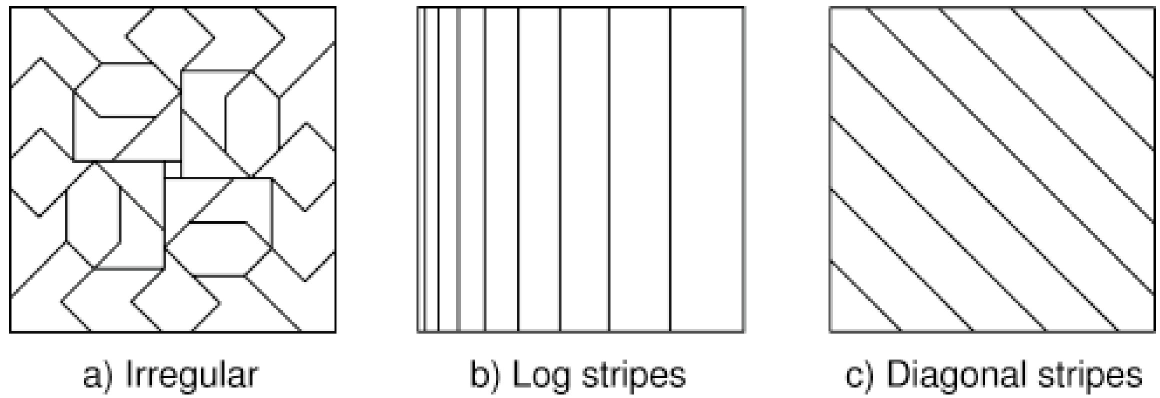


Shape of tiles ⇒ Generalization

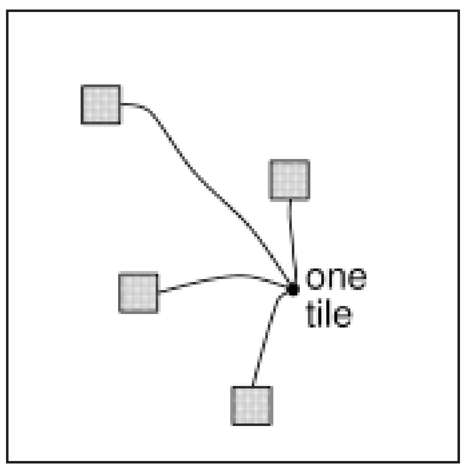
#Tilings ⇒ Resolution of final approximation

Tile Coding Cont.

Irregular tilings



Hashing

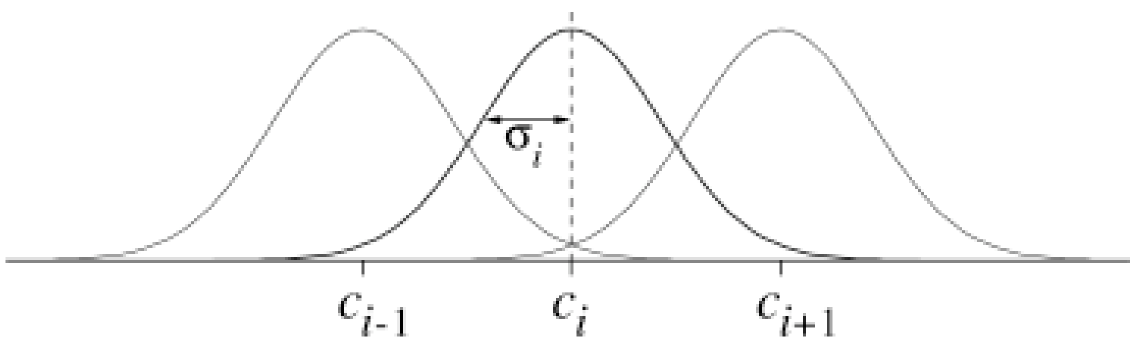


CMAC
"Cerebellar Model Arithmetic Computer"
Albus 1971

Radial Basis Functions (RBFs)

e.g., Gaussians

$$\phi_s(i) = \exp\left(-\frac{\left\| s - c_i \right\|^2}{2\sigma_i^2} \right)$$



Can you beat the "curse of dimensionality"?

- Can you keep the number of features from going up exponentially with the dimension?
- Function complexity, not dimensionality, is the problem.
- Kanerva coding:
  - Select a bunch of binary **prototypes**
  - Use hamming distance as distance measure
  - Dimensionality is no longer a problem, only complexity
- "Lazy learning" schemes:
  - Remember all the data
  - To get new value, find nearest neighbors and interpolate
  - e.g., locally-weighted regression

Control with FA

- Learning state-action values

  Training examples of the form:

$$\left\{\text{description of } (s_t, a_t),\ \ v_t\right\}$$

- The general gradient-descent rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\left[v_t - Q_t(s_t,a_t)\right]\nabla_{\vec{\theta}}Q(s_t,a_t)$$

- Gradient-descent Sarsa(λ) (backward view):

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\,\delta_t\,\vec{e}_t$$

where

$$\delta_t = r_{t+1} + \gamma\,Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$$

$$\vec{e}_t = \gamma\,\lambda\,\vec{e}_{t-1} + \nabla_{\vec{\theta}}\vec{Q}_t(s_t,a_t)$$

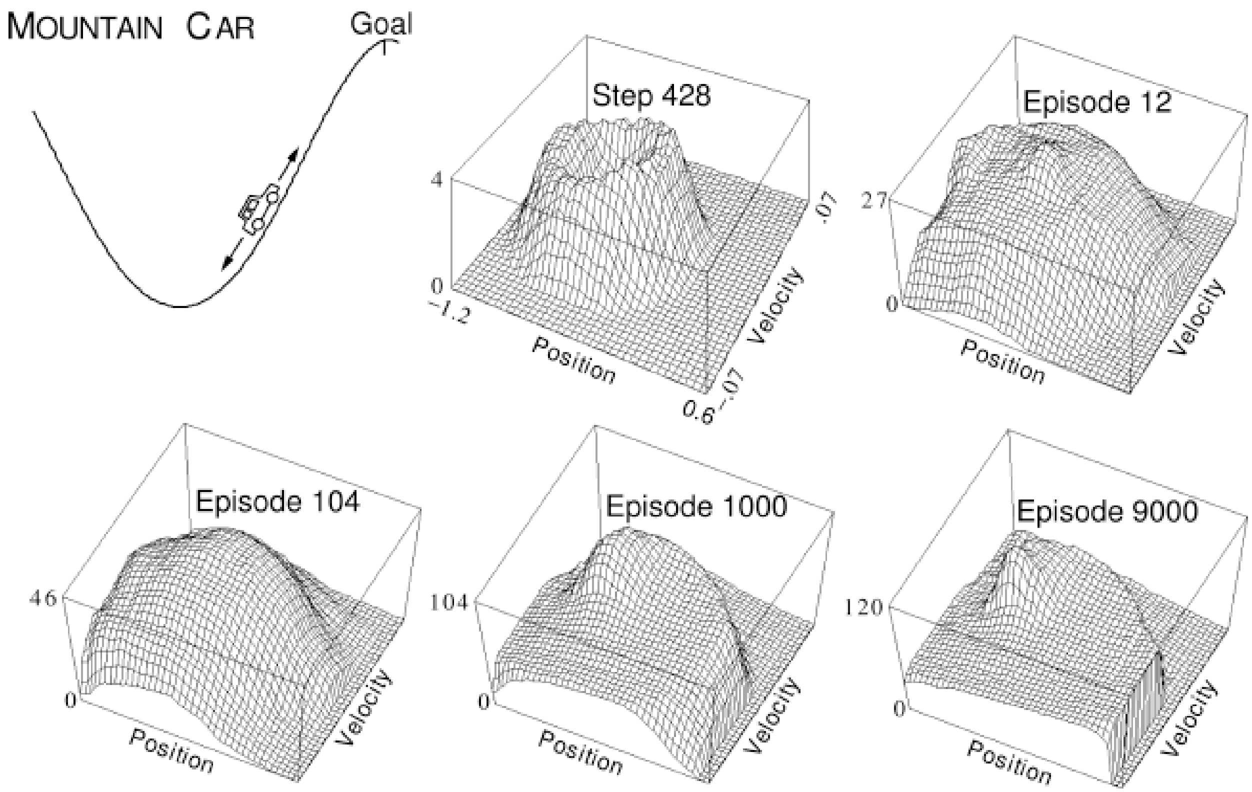
Linear Gradient Descent Sarsa(λ)

Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
    $\vec{e} = \vec{0}$
    $s, a \leftarrow$ initial state and action of episode
    $\mathcal{F}_a \leftarrow$ set of features present in s, a
    Repeat (for each step of episode):
        For all $i \in \mathcal{F}_a$:
            $e(i) \leftarrow e(i) + 1$ (accumulating traces)
            or $e(i) \leftarrow 1$ (replacing traces)
        Take action a, observe reward, r, and next state, s
        $\delta \leftarrow r - \sum_{i \in \mathcal{F}_a} \theta(i)$
        With probability $1 - \varepsilon$:
            For all $a \in \mathcal{A}(s)$:
                $\mathcal{F}_a \leftarrow$ set of features present in s, a
                $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
            $a \leftarrow \arg\max_a Q_a$
        else
            $a \leftarrow$ a random action $\in \mathcal{A}(s)$
            $\mathcal{F}_a \leftarrow$ set of features present in s, a
            $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
        $\delta \leftarrow \delta + \gamma Q_a$
        $\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$
        $\vec{e} \leftarrow \gamma \lambda \vec{e}$
    until s is terminal

GPI Linear Gradient Descent Watkins' Q(λ)

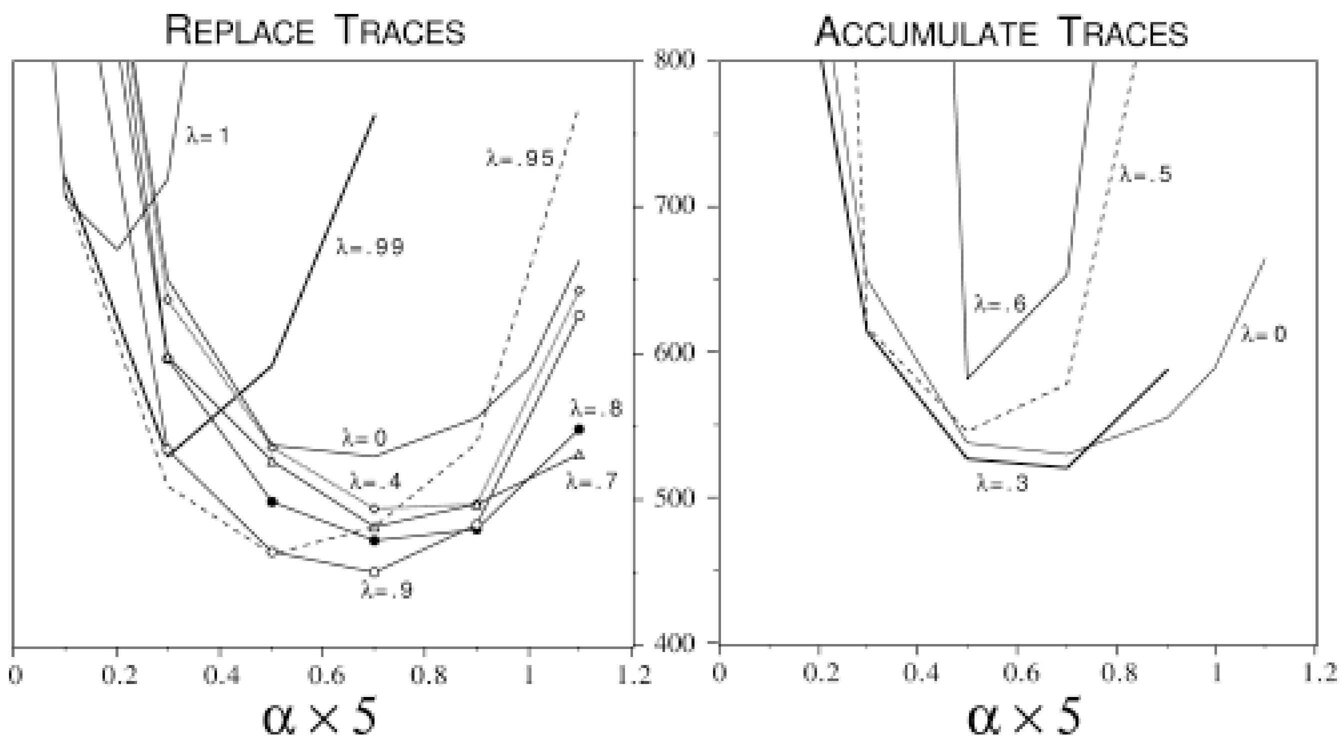
Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
- $\vec{e} = \vec{0}$
- $s, a \leftarrow$ initial state and action of episode
- $\mathcal{F}_a \leftarrow$ set of features present in s, a
- Repeat (for each step of episode):
  - For all $i \in \mathcal{F}_a$: $e(i) \leftarrow e(i) + 1$
  - Take action a, observe reward, r, and next state, s
  - $\delta \leftarrow r - \sum_{i \in \mathcal{F}_a} \theta(i)$
  - For all $a \in \mathcal{A}(s)$:
    - $\mathcal{F}_a \leftarrow$ set of features present in s, a
    - $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
  - $\delta \leftarrow \delta + \gamma \max_a Q_a$
  - $\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$
  - With probability $1 - \varepsilon$:
    - For all $a \in \mathcal{A}(s)$:
      - $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
    - $a \leftarrow \arg\max_a Q_a$
    - $\vec{e} \leftarrow \gamma \lambda \vec{e}$
  - else
    - $a \leftarrow$ a random action $\in \mathcal{A}(s)$
    - $\vec{e} \leftarrow 0$
- until s is terminal

Mountain-Car Task

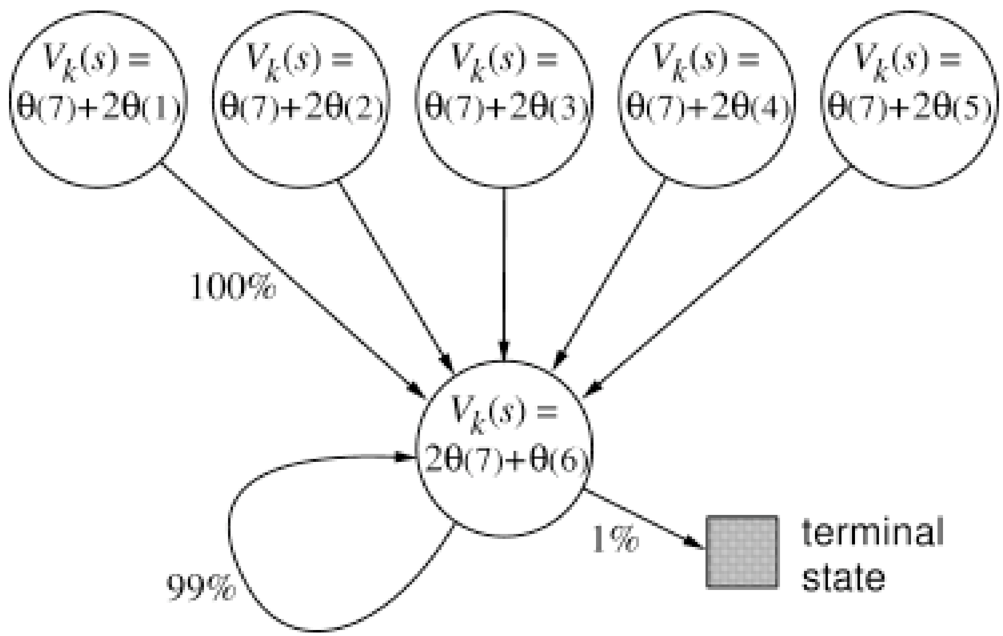


Mountain-Car Results

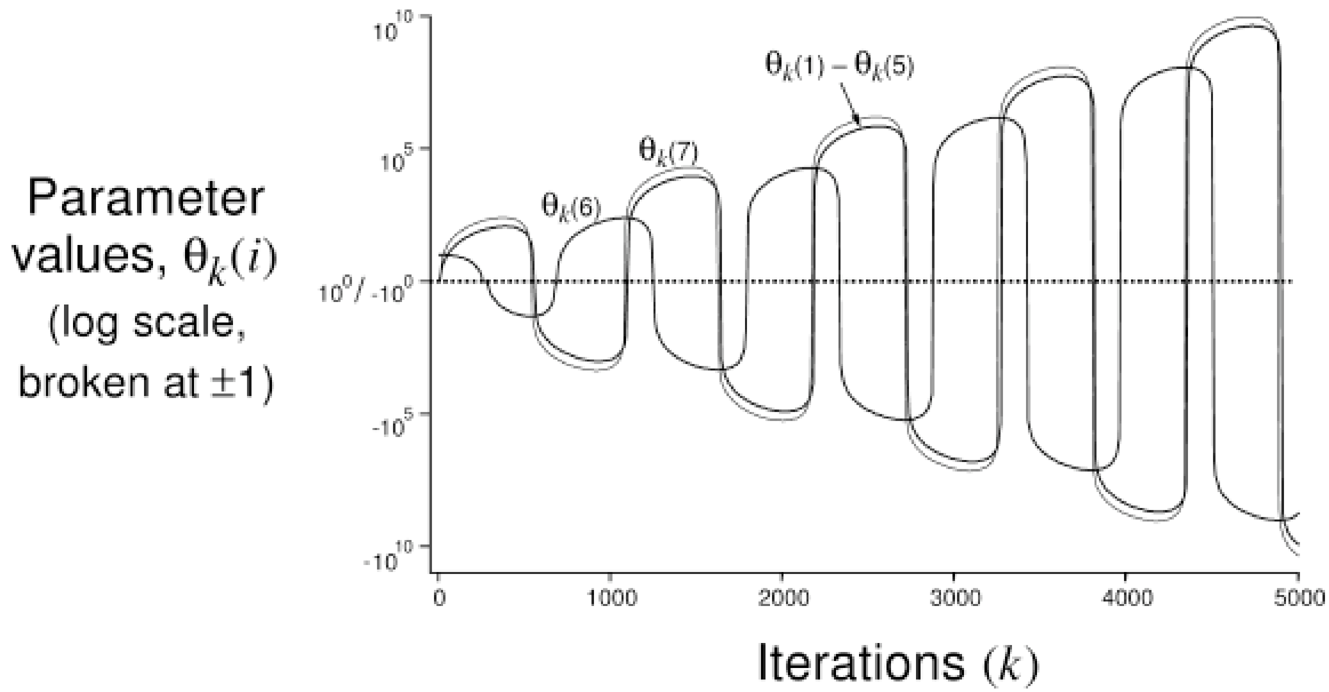
Steps per episode
averaged over
first 20 trials
and 30 runs



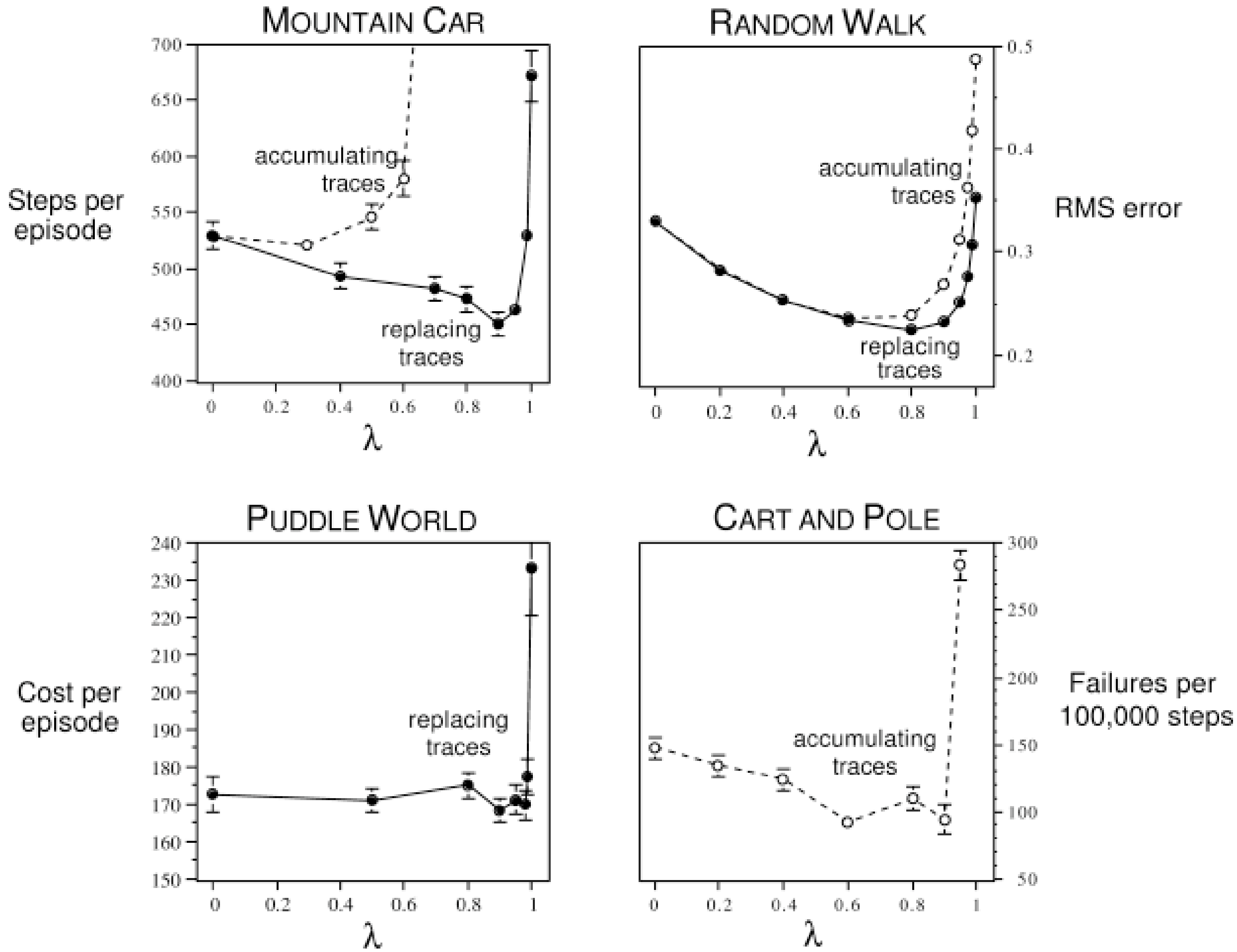
Baird's Counterexample



Baird's Counterexample Cont.



Should We Bootstrap?



Summary

- Generalization
- Adapting supervised-learning function approximation methods
- Gradient-descent methods
- Linear gradient-descent methods
  - Radial basis functions
  - Tile coding
  - Kanerva coding
- Nonlinear gradient-descent methods? Backpropation?
- Subleties involving function approximation, bootstrapping and the on-policy/off-policy distinction