transpose

ith components

$$\cancel{V(s)} \approx v_\pi(s) \approx \hat{v}(s,\boldsymbol{\theta}) \doteq \boldsymbol{\theta}^\top \boldsymbol{\phi}(s) \doteq \sum_{i=1}^{n} \theta_i \cdot \phi_i(s) = 1.71$$

inner product

$$\boldsymbol{\theta} \in \mathbb{R}^n, \quad e.g., \quad \boldsymbol{\theta} = \begin{bmatrix} 2.1 \\ 0.01 \\ -1.1 \\ 1.2 \\ -0.1 \\ 0.01 \\ 4.93 \\ 0.5 \end{bmatrix}, \qquad \boldsymbol{\phi}(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^n$$

parameter vector

feature vector

$$\cancel{Q(s,a)} \approx q_\pi(s,a) \approx \hat{q}(s,a,\boldsymbol{\theta}) \doteq \boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a) \doteq \sum_{i=1}^{n} \theta_i \cdot \phi_i(s,a)$$

Chapter 9: 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 Function Approx.

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

for a given policy π, estimate the state-value function v_π

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

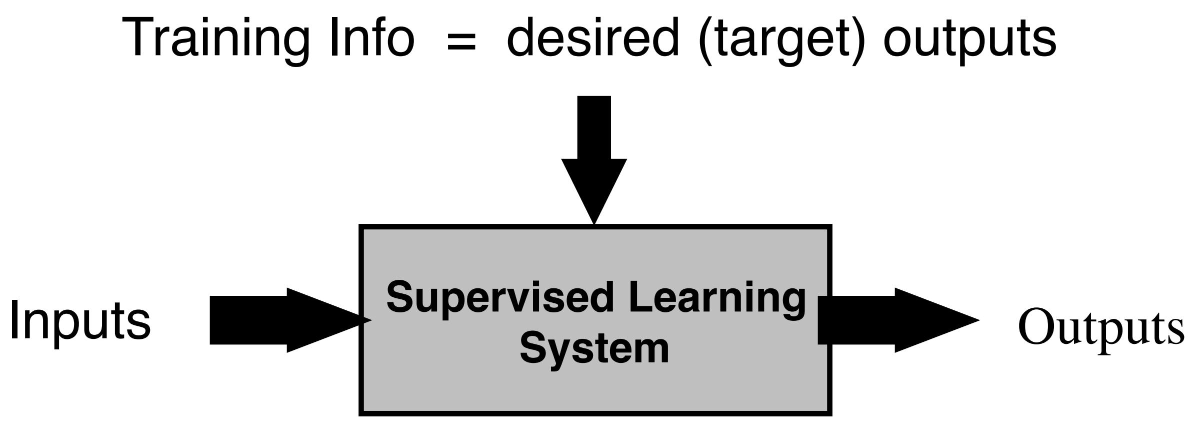
Now, the value function estimate at time t, V_t, depends on a vector of parameters $\boldsymbol{\theta}$:

$$\hat{v}(s,\boldsymbol{\theta}) \approx v_\pi(s)$$

only the parameters are updated

e.g., $\boldsymbol{\theta}$ could be the modifiable connection weights and thesholds of a deep neural network

Adapt Supervised Learning Algorithms


Training Info = desired (target) outputs
Inputs
Supervised Learning System
Outputs

Training example = {input, target output}

Error = (target output – actual output)

Backups as Training Examples

For example, the TD(0) backup:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

As a training example:

$$\underset{\text{input}}{\underset{\uparrow}{\text{features of } S_t}} \longrightarrow \underset{\text{target output}}{\underset{\uparrow}{R_{t+1} + \gamma V(S_{t+1})}}$$

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 (online)
  - ability to handle nonstationarity
  - other?

Gradient Descent Methods

transpose

$$\boldsymbol{\theta} \doteq (\theta_1, \theta_2, \ldots, \theta_n)^\top$$

Assume $\hat{v}(s, \boldsymbol{\theta})$ is a differentiable function of $\boldsymbol{\theta}$, for all $s \in \mathcal{S}$

Assume, for now, training examples of this form:

$$\text{features of } S_t \longrightarrow v_\pi(S_t)$$

Gradient Descent

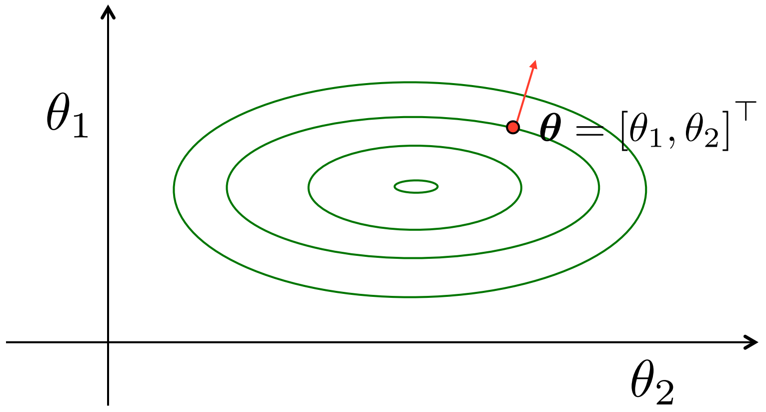
Let $f(\boldsymbol{\theta})$ be a function to be minimized, e.g., an error

Its gradient with respect to $\boldsymbol{\theta}$ is

$$\nabla f(\boldsymbol{\theta}) \doteq \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \doteq \left(\frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1}, \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_2}, \ldots, \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_n} \right)^\top$$

Iteratively move "down" the gradient:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla f(\boldsymbol{\theta})$$



Performance Measures

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

$$\mathrm{MSVE}(\boldsymbol{\theta}) = \sum_{s\in\mathcal{S}} d(s)\Big[v_\pi(s) - \hat{v}(s,\boldsymbol{\theta})\Big]^2$$

- Why *d* ?
- Why minimize MSVE?
- Let us assume that *d* is always the distribution of states at 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 Derivation

$$
\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \alpha \nabla \mathrm{MSVE}(\boldsymbol{\theta}_t) \\
&= \boldsymbol{\theta}_t - \alpha \nabla \sum_{s \in \mathcal{S}} d(s) \left[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}_t) \right]^2 \\
&= \boldsymbol{\theta}_t - \alpha \sum_{s \in \mathcal{S}} d(s) \nabla \left[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}_t) \right]^2 \\
&= \boldsymbol{\theta}_t - 2\alpha \sum_{s \in \mathcal{S}} d(s) \left[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}_t) \right] \nabla \left[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}_t) \right] \\
&= \boldsymbol{\theta}_t + \alpha \sum_{s \in \mathcal{S}} d(s) \left[v_\pi(s) - \hat{v}(s, \boldsymbol{\theta}_t) \right] \nabla \hat{v}(s, \boldsymbol{\theta}_t)
\end{aligned}
$$

(sampling)

$$
= \boldsymbol{\theta}_t + \alpha \Big[v_\pi(S_t) - \hat{v}(S_t, \boldsymbol{\theta}_t) \Big] \nabla \hat{v}(S_t, \boldsymbol{\theta}_t)
$$

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

But We Don't have these Targets

Suppose we just have targets V_t instead :

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big[V_t - \hat{v}(S_t,\boldsymbol{\theta}_t)\Big]\nabla\hat{v}(S_t,\boldsymbol{\theta}_t)$$

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 $V_t = G_t$ (unbiased):

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big[G_t - \hat{v}(S_t,\boldsymbol{\theta}_t)\Big]\nabla\hat{v}(S_t,\boldsymbol{\theta}_t)$$

What about TD(λ) Targets?

What about the λ-return, G_t^λ ?

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\Big[G_t^\lambda - \hat{v}(S_t,\boldsymbol{\theta}_t)\Big]\nabla\hat{v}(S_t,\boldsymbol{\theta}_t)$$

Unfortunately, G_t^λ is biased for $\lambda < 1$

$\Rightarrow$ standard gradient descent results don't apply

But we do it anyway!

first, some meta comments on

Understanding Algorithms

1. Do I understand the symbols and their meaning?
    - Could I write a program to do it?
    - Does it make intuitive sense?
2. Can I derive the algorithm from some objective?
3. Can I prove that the algorithm converges to some objective?
4. Can I prove something about the rate of convergence?

and some meta comments on

Efficient Scaling

Bengio & LeCun, 2007

3 Kinds of Efficiency in Machine Learning & AI

1. Data efficiency (rate of learning)
2. Computational efficiency (memory, computation, communication)
3. User efficiency (autonomy, ease of setup, lack of parameters, priors, labels, expertise)

Computational Resources

1. Memory
2. Computation
3. Communication (wires)

Natural Scaling

- Every learning system has two parts
  1. the thing that is learned (e.g., the neural network and its weights)
  2. the algorithm that learns it (e.g., the algorithm that learns the weights)
- *Natural scaling* is when the computational complexities of the two parts scale similarly

Gradient-based TD(λ), backwards view

$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1},\boldsymbol{\theta}_t) - \hat{v}(S_t,\boldsymbol{\theta}_t)$$

$$\mathbf{e}_t \doteq \gamma\lambda\mathbf{e}_{t-1} + \nabla\hat{v}(S_t,\boldsymbol{\theta}_t)$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\,\delta_t\,\mathbf{e}_t$$

On-Line Gradient-Descent TD(λ)

```
Initialize $\boldsymbol{\theta}$ as appropriate for the problem, e.g., $\boldsymbol{\theta} = \mathbf{0}$
Repeat (for each episode):
    $\mathbf{e} = 0$
    $S \leftarrow$ initial state of episode
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe reward, $R$, and next state, $S'$
        $\delta \leftarrow R + \gamma \hat{v}(S',\boldsymbol{\theta}) - \hat{v}(S,\boldsymbol{\theta})$
        $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \nabla\hat{v}(S,\boldsymbol{\theta})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\mathbf{e}$
        $S \leftarrow S'$
    until $S'$ is terminal
```

Linear Methods

Represent states as feature vectors:

for each $s \in \mathcal{S}$:

$$\hat{v}(s,\boldsymbol{\theta}) \doteq \boldsymbol{\theta}^\top \boldsymbol{\phi}(s) = \sum_{i=1}^{n} \theta_i x_i(s)$$

$$\nabla \hat{v}(s,\boldsymbol{\theta}) = \ ?$$

Linear Methods

Represent states as feature vectors:

for each $s \in \mathcal{S}$:

$$\hat{v}(s,\boldsymbol{\theta}) \doteq \boldsymbol{\theta}^\top \boldsymbol{\phi}(s) = \sum_{i=1}^{n} \theta_i x_i(s)$$

$$\nabla \hat{v}(s,\boldsymbol{\theta}) = \boldsymbol{\phi}(s)$$

Nice Properties of Linear FA Methods

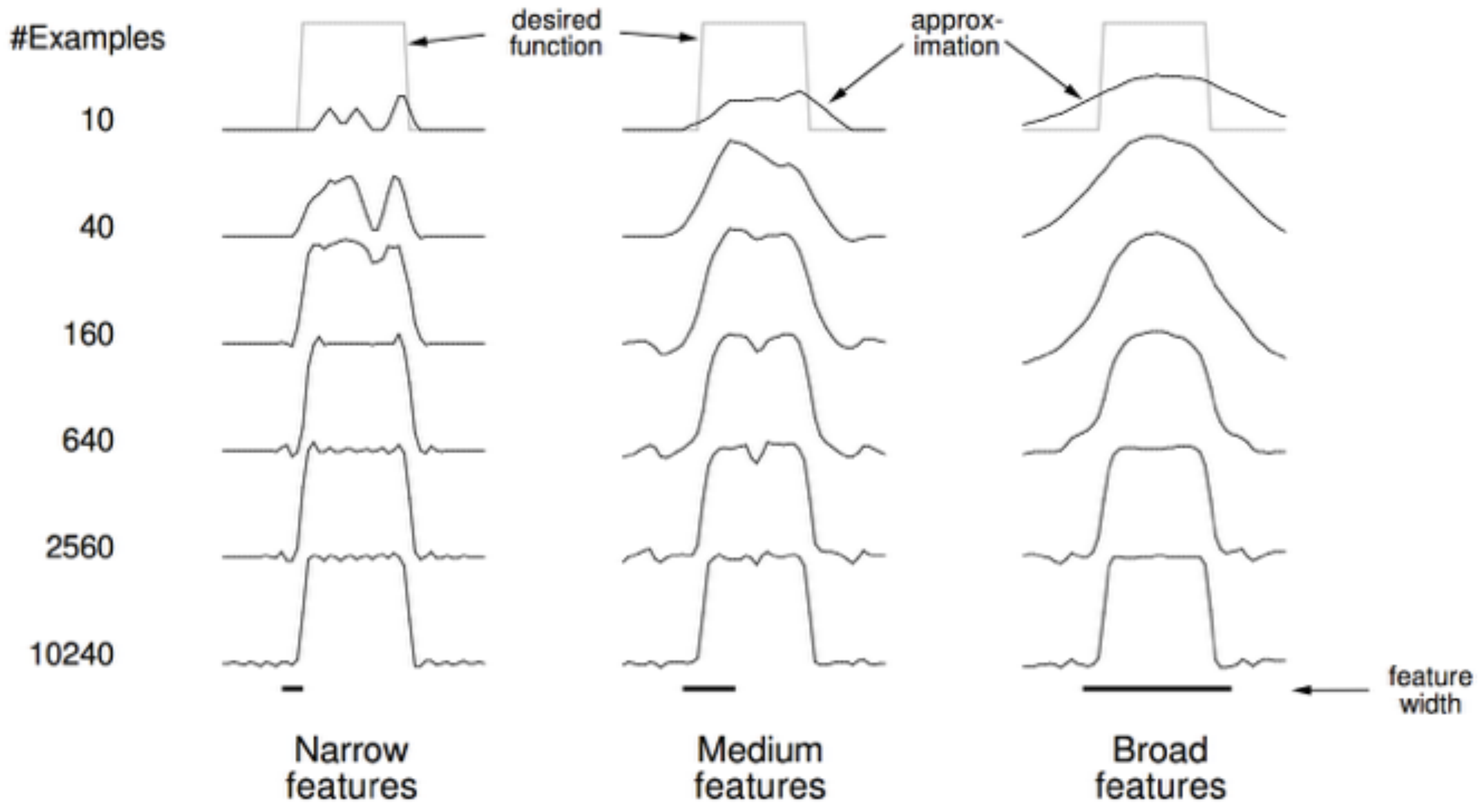
- The gradient is very simple: $\nabla \hat{v}(s,\boldsymbol{\theta}) = \boldsymbol{\phi}(s)$
- For MSE, the error surface is simple: quadratic surface with a single minimum.
- Linear gradient descent TD(λ) converges:
  - Step size decreases appropriately
  - On-line sampling (states sampled from the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

$$\mathrm{MSVE}(\boldsymbol{\theta}_\infty) \quad \leq \quad \frac{1-\gamma\lambda}{1-\gamma}\,\mathrm{MSVE}(\underset{\uparrow}{\boldsymbol{\theta}^*})$$

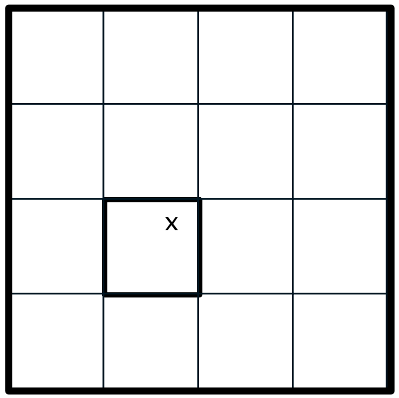
best weight vector

(Tsitsiklis & Van Roy, 1997)

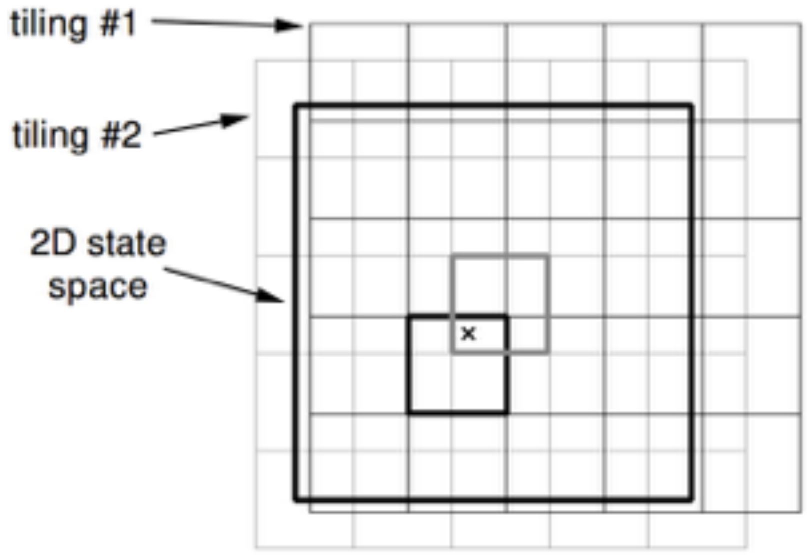
Learning and Coarse Coding



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 features present

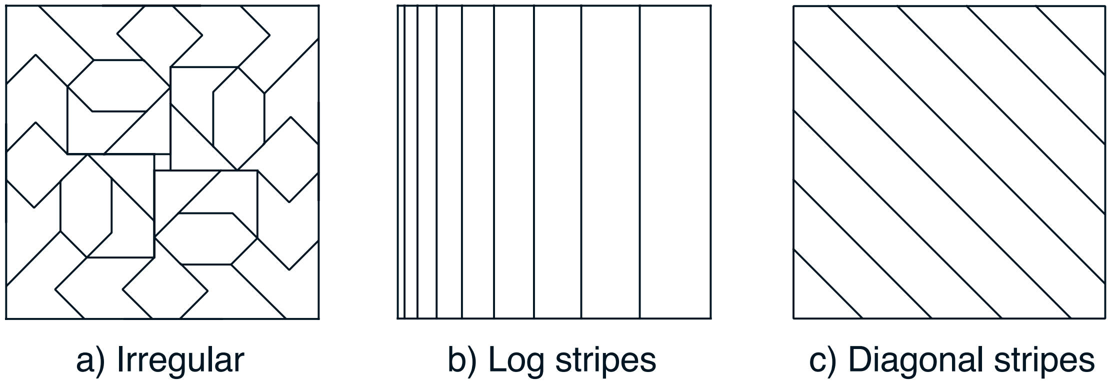


Shape of tiles $\Rightarrow$ Generalization

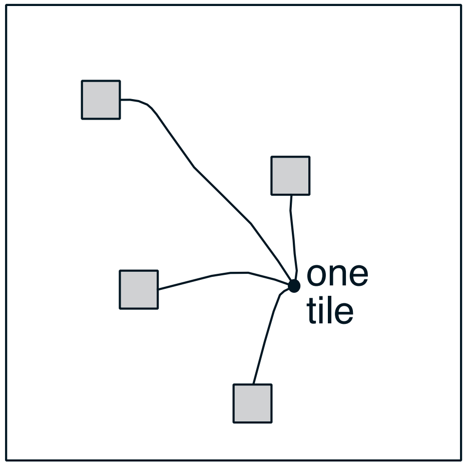
#Tilings $\Rightarrow$ Resolution of final approximation

Tile Coding Cont.

Irregular tilings

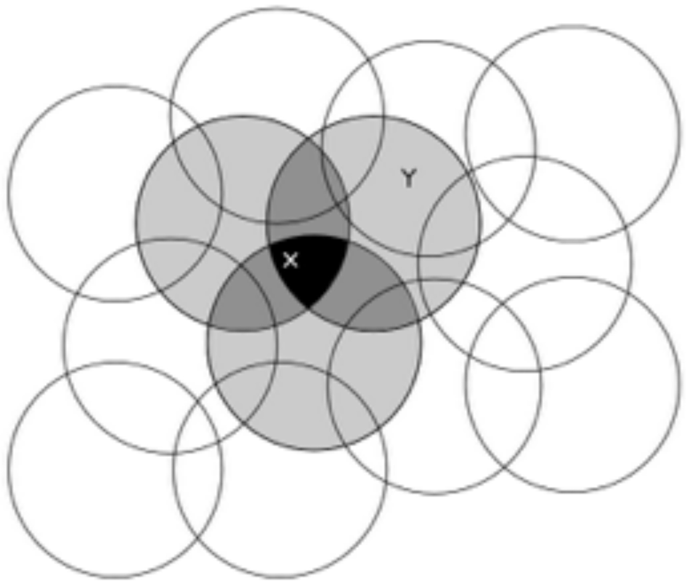


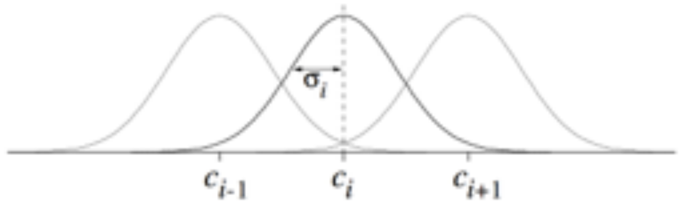
Hashing

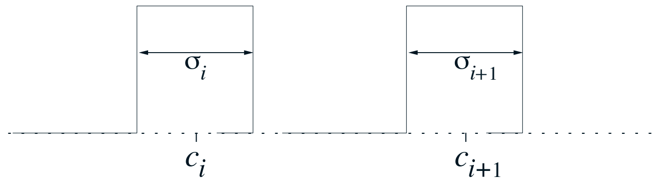


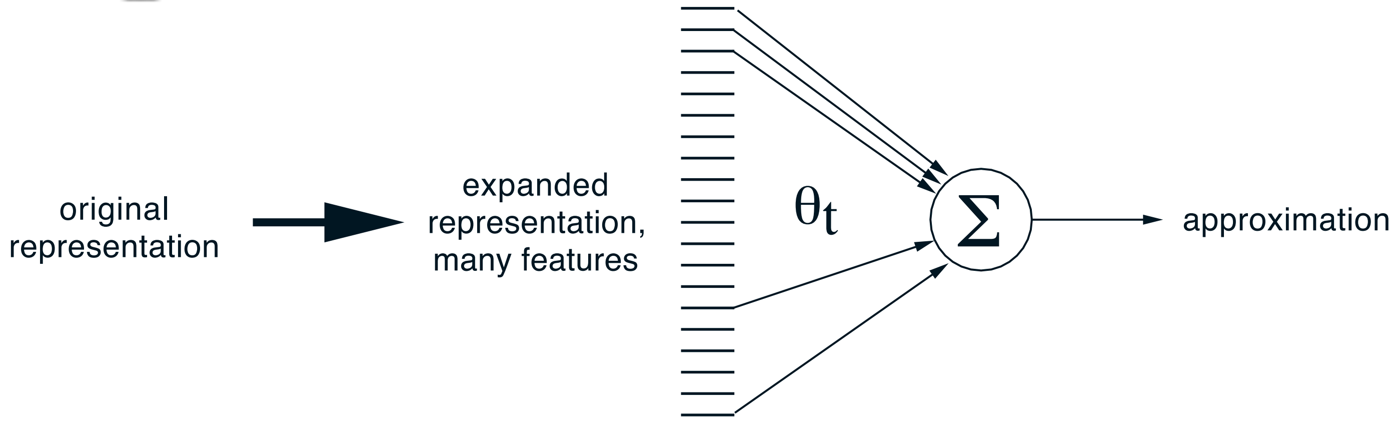
CMAC
"Cerebellar model arithmetic computer"
Albus 1971

Coarse Coding

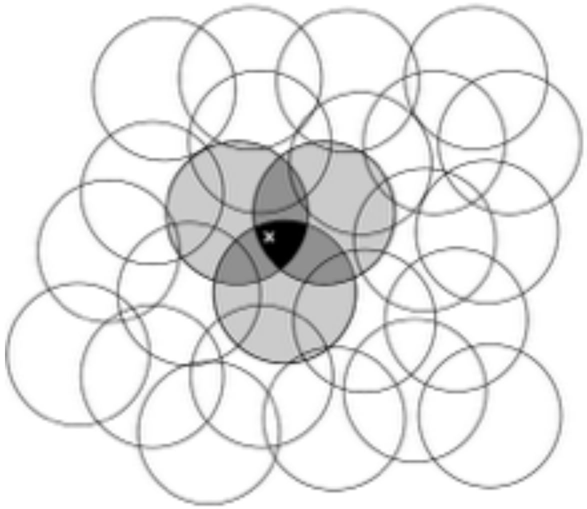


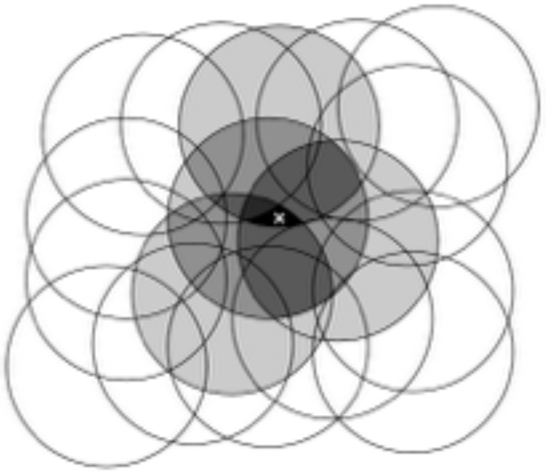


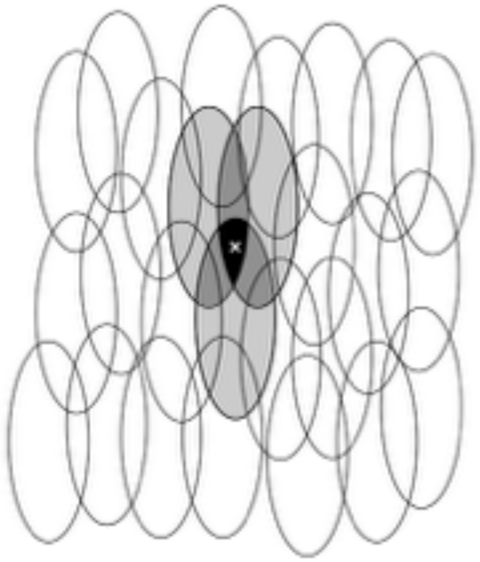




Shaping Generalization in Coarse Coding


a) Narrow generalization


b) Broad generalization


c) Asymmetric generalization

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 neighbours and interpolate
  - e.g., locally-weighted regression

Control with FA

- ❐ Learning state-action values

  Training examples of the form:

$$\left\{ \text{description of } (S_t, A_t),\ Q_t \right\}$$

- ❐ The general gradient-descent rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \Big[Q_t - \hat{q}(S_t, A_t, \mathbf{w}_t) \Big] \nabla_{\mathbf{w}_t} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{e}_t$$

where:

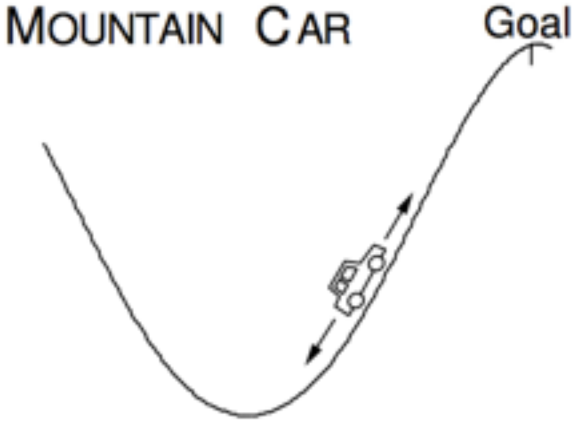
$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$$

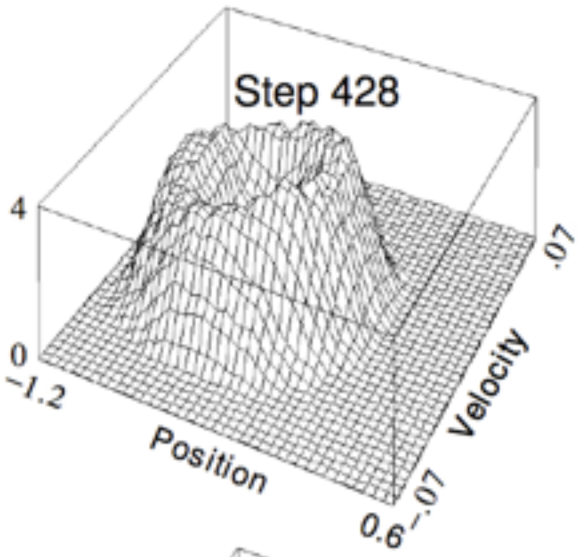
$$\mathbf{e}_t = \gamma \lambda \mathbf{e}_{t-1} + \nabla_{\mathbf{w}_t} \hat{q}(S_t, A_t, \mathbf{w}_t)$$

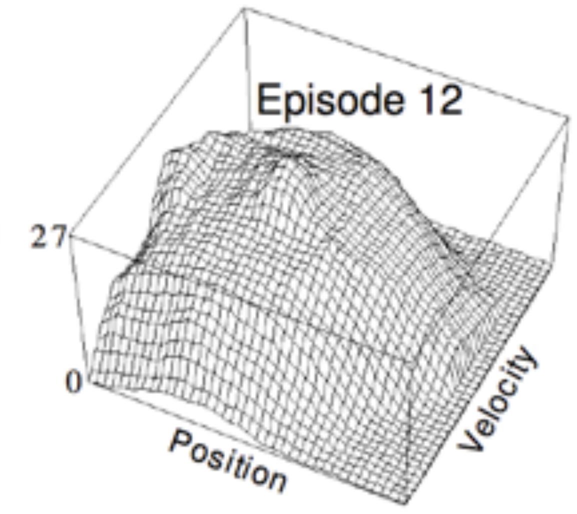
Linear Gradient-based Sarsa(λ)

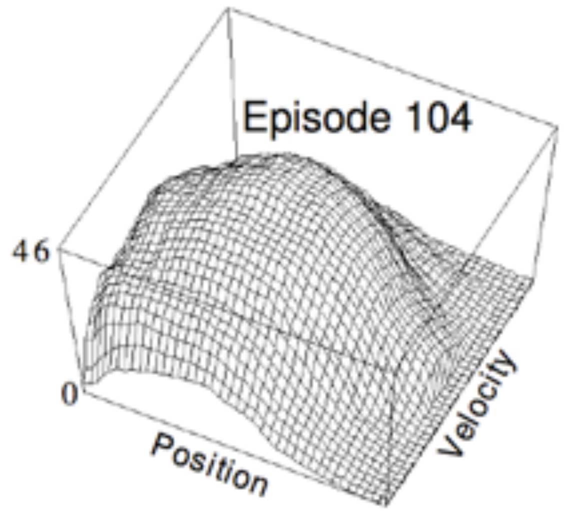
```
Let $\boldsymbol{\theta}$ and $\mathbf{e}$ be vectors with one component for each possible feature
Let $\mathcal{F}_a$, for every possible action $a$, be a set of feature indices, initially empty
Initialize $\boldsymbol{\theta}$ as appropriate for the problem, e.g., $\boldsymbol{\theta} = \mathbf{0}$
Repeat (for each episode):
    $\mathbf{e} = \mathbf{0}$
    $S, A \leftarrow$ initial state and action of episode      (e.g., $\varepsilon$-greedy)
    $\mathcal{F}_A \leftarrow$ 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, $S'$
        $\delta \leftarrow R - \sum_{i \in \mathcal{F}_A} \theta_i$
        If $S'$ is terminal, then $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \mathbf{e}$; go to next episode
        For all $a \in \mathcal{A}(S')$:
            $\mathcal{F}_a \leftarrow$ set of features present in $S', a$
            $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta_i$
        $A' \leftarrow$ new action in $S'$ (e.g., $\varepsilon$-greedy)
        $\delta \leftarrow \delta + \gamma Q_{A'}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \mathbf{e}$
        $\mathbf{e} \leftarrow \gamma \lambda \mathbf{e}$
        $S \leftarrow S'$
        $A \leftarrow A'$
```

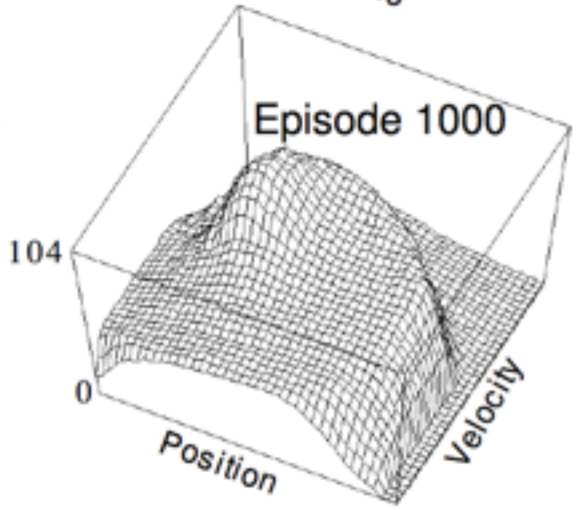
Approx Value Functions on Mountain-Car Task

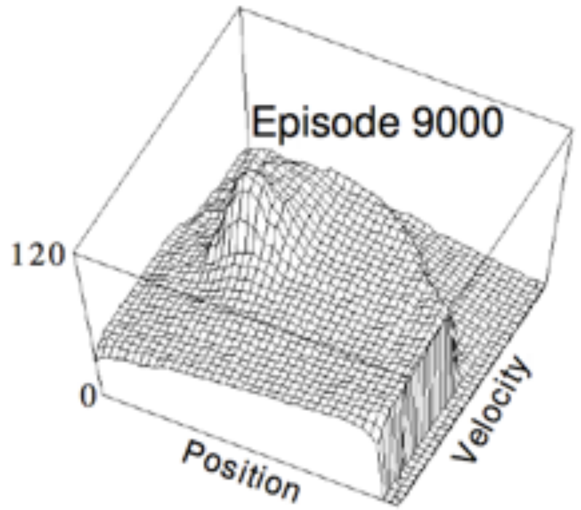








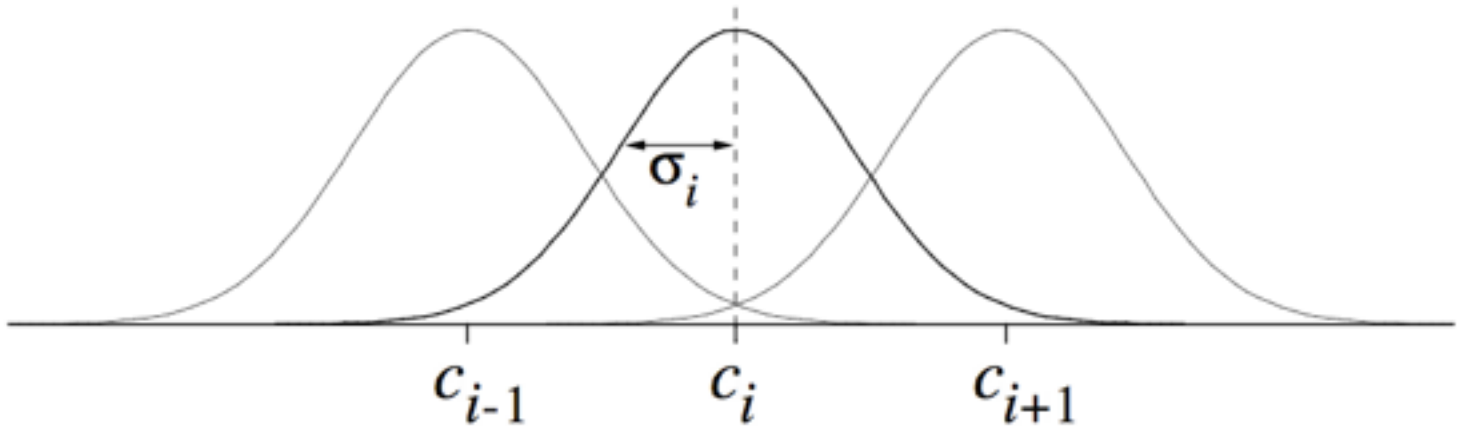




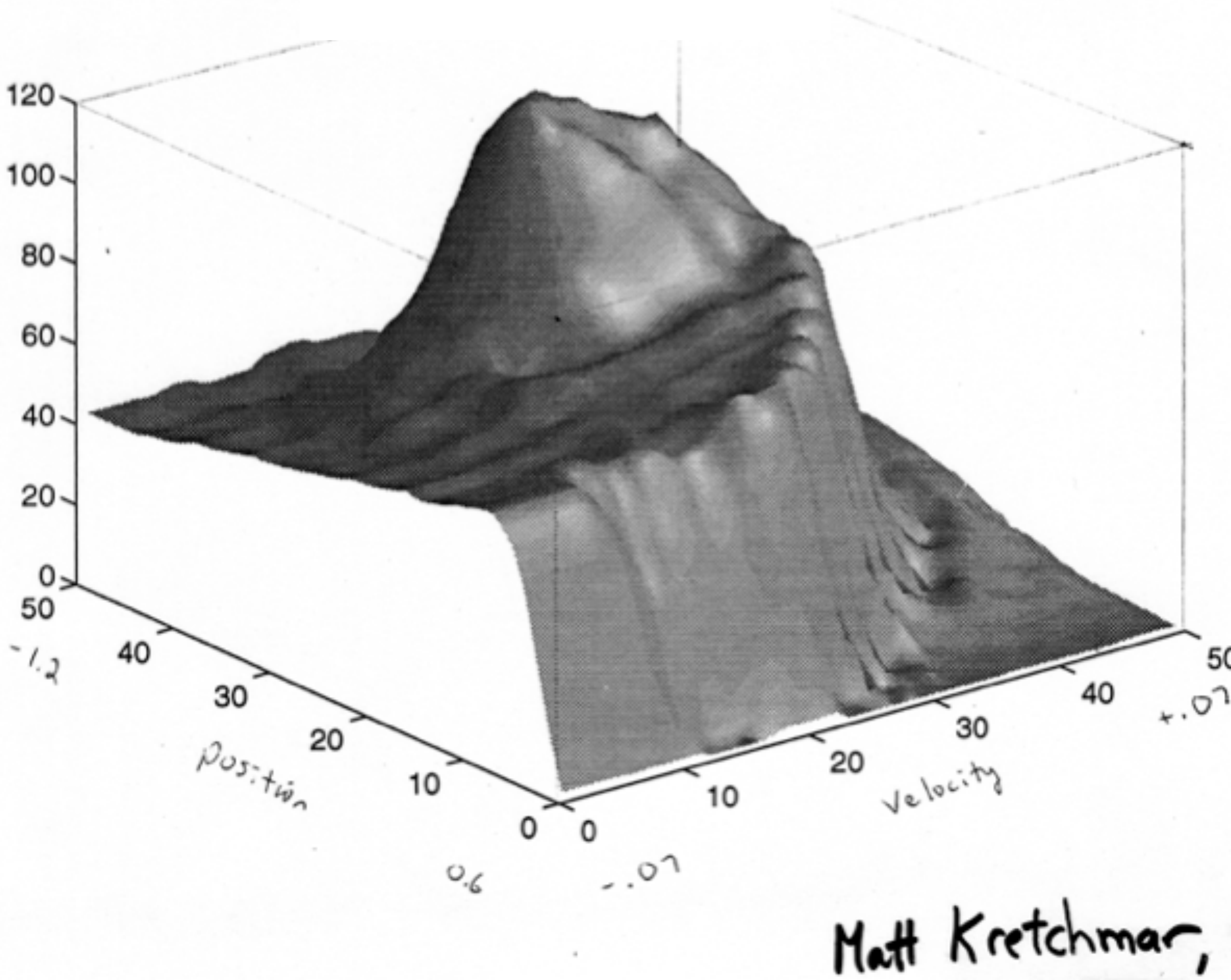
Radial Basis Functions (RBFs)

e.g., Gaussians

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

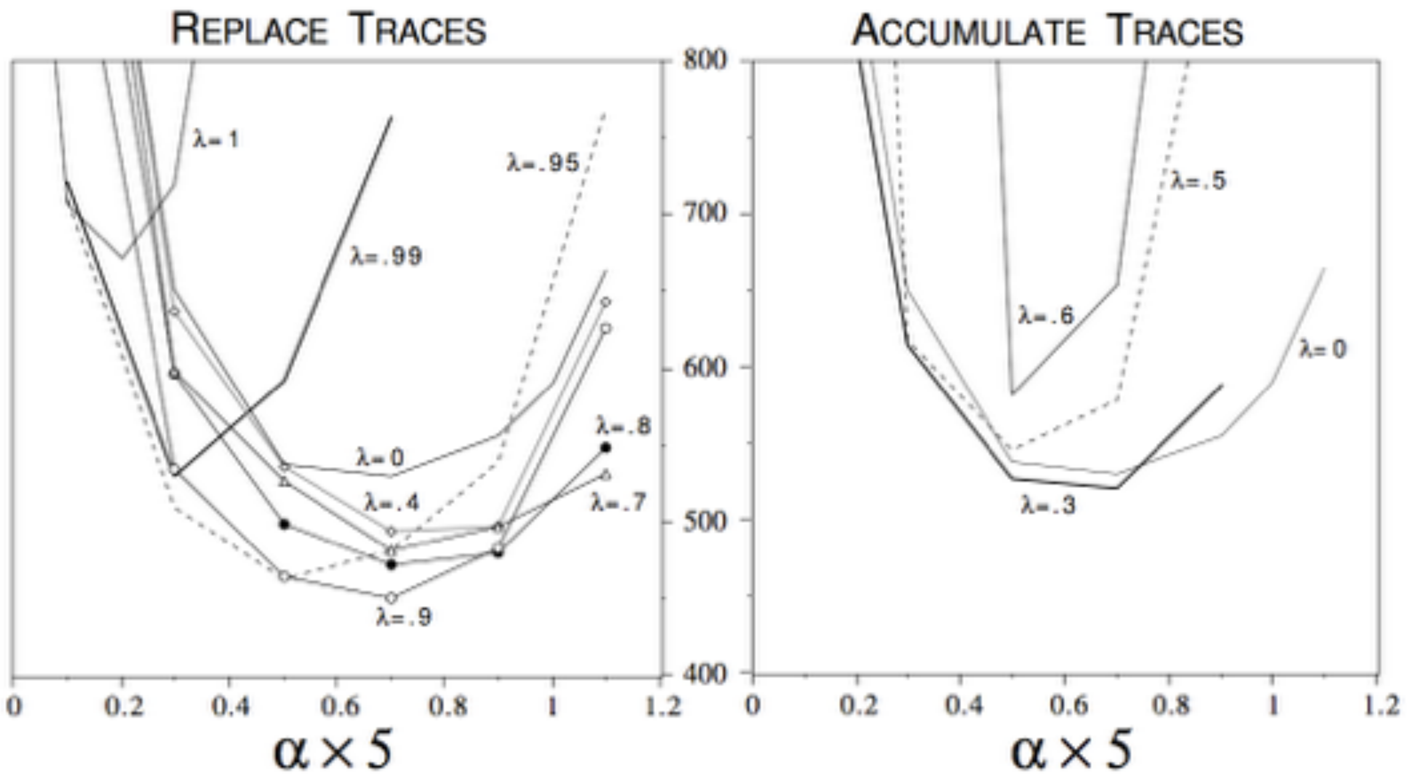


Mountain Car with Radial Basis Functions

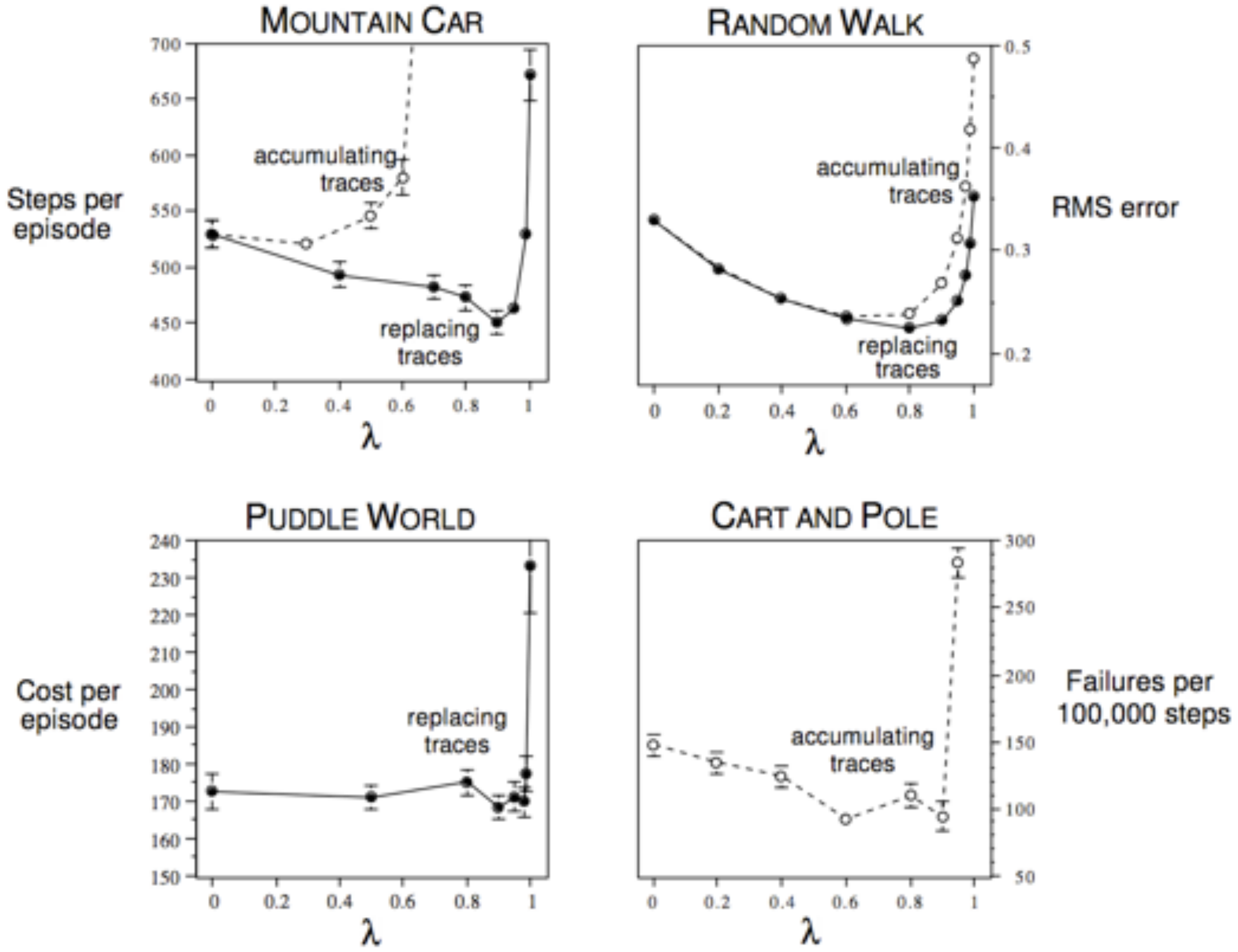


Mountain-Car Results

Steps per episode
averaged over
first 20 trials
and 30 runs



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