



## 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

$$\vec{\theta}_t = (\theta_t(1), \theta_t(2), \dots, \theta_t(n))^T \quad \leftarrow \text{transpose}$$

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 :

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

## Performance Measures

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

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

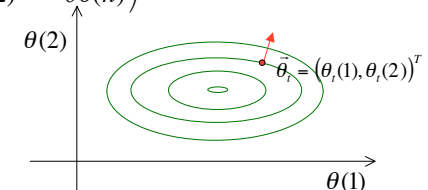
- Why  $P$  ?
- Why minimize MSE?
- Let us assume that  $P$  is always the distribution of states with which backups are done.
- The **on-policy distribution**: the distribution created while following the policy being evaluated. Stronger results are available for this distribution.

## Gradient Descent

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

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

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



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:

$$\begin{aligned}\bar{\theta}_{t+1} &= \bar{\theta}_t - \frac{1}{2} \alpha \nabla_{\bar{\theta}} \text{MSE}(\bar{\theta}_t) \\ &= \bar{\theta}_t - \frac{1}{2} \alpha \nabla_{\bar{\theta}} \sum_{s \in S} P(s) [V^\pi(s) - V_t(s)]^2 \\ &= \bar{\theta}_t + \alpha \sum_{s \in S} P(s) [V^\pi(s) - V_t(s)] \nabla_{\bar{\theta}} V_t(s)\end{aligned}$$

## Gradient Descent Cont.

Use just the **sample gradient** instead:

$$\begin{aligned}\bar{\theta}_{t+1} &= \bar{\theta}_t - \frac{1}{2} \alpha \nabla_{\bar{\theta}} [V^\pi(s_t) - V_t(s_t)]^2 \\ &= \bar{\theta}_t + \alpha [V^\pi(s_t) - V_t(s_t)] \nabla_{\bar{\theta}} V_t(s_t),\end{aligned}$$

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

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

## But We Don't have these Targets

Suppose we just have targets  $v_t$  instead :

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

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  $\alpha$  decreases appropriately).

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

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

## What about TD( $\lambda$ ) Targets?

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

Not unbiased for  $\lambda < 1$

But we do it anyway, using the backwards view :

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

where :

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

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

## On-Line Gradient-Descent TD( $\lambda$ )

```

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  $\pi$  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  $s \in S$  :

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

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

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

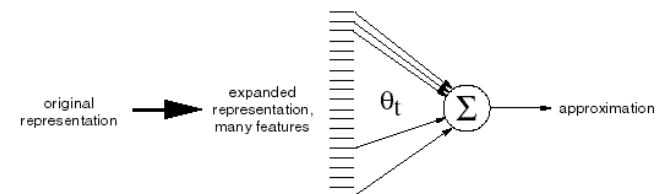
## Nice Properties of Linear FA Methods

- 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 minimum.
- Linear gradient descent TD( $\lambda$ ) converges:
  - Step size decreases appropriately
  - On-line sampling (states sampled from the on-policy distribution)
  - Converges to parameter vector  $\vec{\theta}_\infty$  with property:

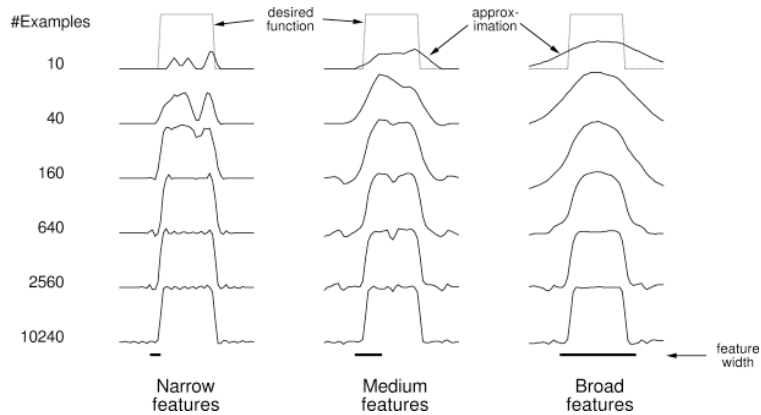
$$MSE(\vec{\theta}_\infty) \leq \frac{1 - \gamma \lambda}{1 - \gamma} MSE(\vec{\theta}^*)$$

(Tsitsiklis & Van Roy, 1997)

best parameter vector



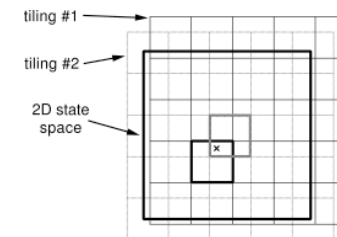
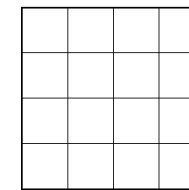
## Learning and Coarse Coding



R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

17

## 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

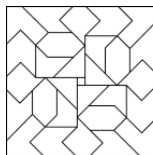
#Tilings  $\Rightarrow$  Resolution of final approximation

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

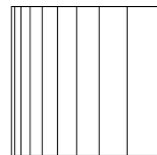
18

## Tile Coding Cont.

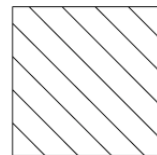
Irregular tilings



a) Irregular

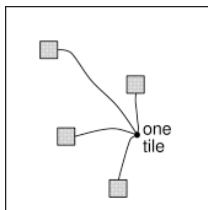


b) Log stripes



c) Diagonal stripes

Hashing



CMAC  
"Cerebellar Model Arithmetic Computer"  
Albus 1971

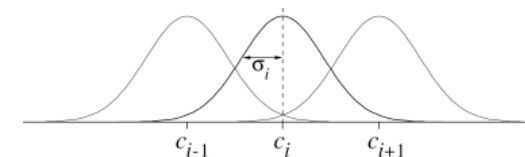
R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

19

## Radial Basis Functions (RBFs)

e.g., Gaussians

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



R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

20

## 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:

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

- ❑ The general gradient-descent rule:

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

- ❑ Gradient-descent Sarsa( $\lambda$ ) (backward view):

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

where

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

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

## Linear Gradient Descent Sarsa( $\lambda$ )

```

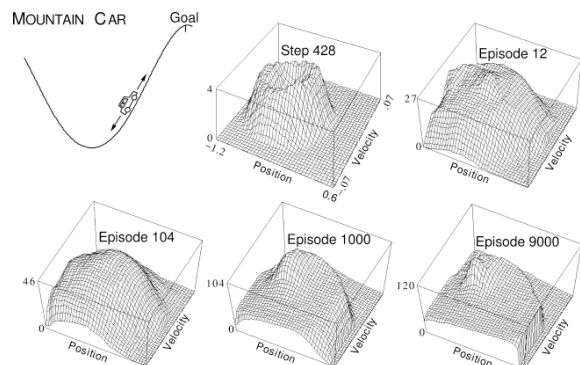
Initialize  $\bar{\theta}$  arbitrarily
Repeat (for each episode):
     $\bar{e} = \vec{0}$ 
     $s, a \leftarrow$  initial state and action of episode
     $\mathcal{F}_a \leftarrow$  set of features present in  $s, a$ 
    Repeat (for each step of episode):
        For all  $i \in \mathcal{F}_a$ :
             $e(i) \leftarrow e(i) + 1$  (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  $s, a$ 
                 $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$ 
             $\delta \leftarrow \delta + \gamma Q_a$ 
             $\bar{\theta} \leftarrow \bar{\theta} + \alpha \delta \bar{e}$ 
             $\bar{e} \leftarrow \gamma \lambda \bar{e}$ 
    until  $s$  is terminal
    
```

## GPI Linear Gradient Descent Watkins' Q( $\lambda$ )

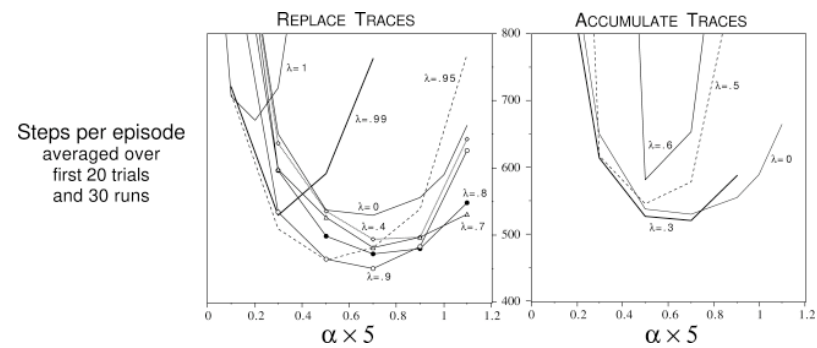
```

Initialize  $\bar{\theta}$  arbitrarily
Repeat (for each episode):
     $\bar{e} = \vec{0}$ 
     $s, a \leftarrow$  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  $s, a$ 
             $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$ 
             $\delta \leftarrow \delta + \gamma \max_a Q_a$ 
             $\bar{\theta} \leftarrow \bar{\theta} + \alpha \delta \bar{e}$ 
        With probability  $1 - \varepsilon$ :
            For all  $a \in \mathcal{A}(s)$ :
                 $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$ 
                 $a \leftarrow \arg \max_a Q_a$ 
             $\bar{e} \leftarrow \gamma \lambda \bar{e}$ 
        else
             $a \leftarrow$  a random action  $\in \mathcal{A}(s)$ 
             $\bar{e} \leftarrow \vec{0}$ 
    until  $s$  is terminal
    
```

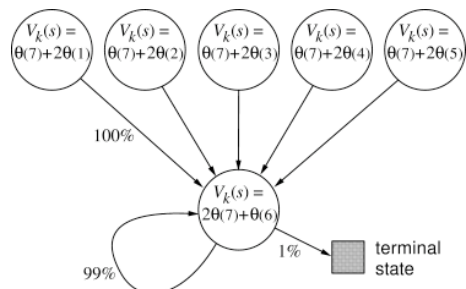
## Mountain-Car Task



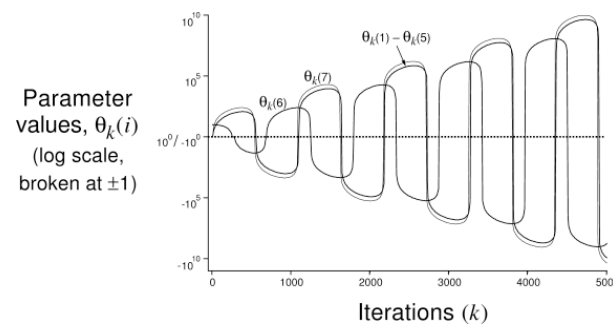
## Mountain-Car Results



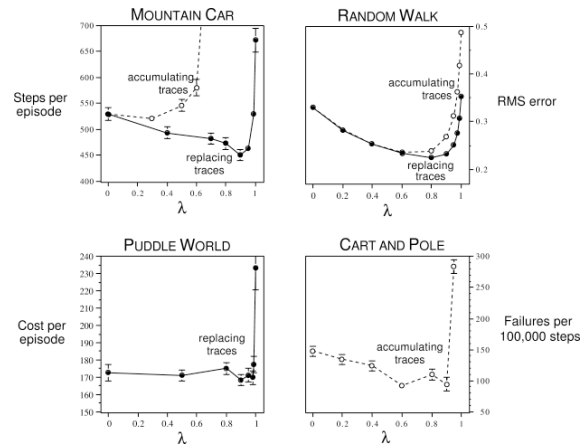
## Baird's Counterexample



## 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?
- Subtleties involving function approximation, bootstrapping and the on-policy/off-policy distinction