

Chapter 9:

On-policy Prediction with Approximation

In this chapter, we approximate $v(\pi)$ from experience generated using a known policy π .

$$\text{Target or } V(s_t) \leftarrow V(s_t) + \alpha [Target_t - V(s_t)]$$

$$\text{Now } V_\pi(s) \approx \hat{V}(s, \theta) \quad \theta \in \mathbb{R}^n \quad n \ll |S|$$

value

$$\text{Function } \theta \leftarrow \theta + \alpha [Target - \hat{V}(s_t, \theta)] \nabla_\theta \hat{V}(s_t, \theta)$$

Approximation
(VFA)

$$\text{e.g., } \phi(s_t)$$

Stochastic gradient descent (SGD)

$$\theta \leftarrow \theta - \alpha \nabla_\theta E_{\text{Error}}^2 \quad \nabla_\theta f(\theta) = \left[\frac{\partial f(\theta)}{\partial \theta_1}, \dots, \frac{\partial f(\theta)}{\partial \theta_n} \right]^T$$

$$\leftarrow \theta - \alpha \nabla_\theta [Target - \hat{V}(s_t, \theta)]^2$$

$$\leftarrow \theta - \alpha [Target - \hat{V}(s_t, \theta)] \nabla_\theta [Target - \hat{V}(s_t, \theta)]$$

$$\leftarrow \theta + 2\alpha [Target_t - \hat{V}(s_t, \theta)] \nabla_\theta \hat{V}(s_t, \theta)$$

QED

Linear VFA

$$\hat{V}(s, \theta) = \theta^T \phi(s) = \sum_{i=1}^n \theta_i \phi_i(s), \quad \phi(s) \in \mathbb{R}^n, \quad \phi: S \rightarrow \mathbb{R}^n$$

$$\nabla_\theta \hat{V}(s, \theta) = \nabla_\theta \theta^T \phi(s) = \nabla_\theta \sum_{i=1}^n \theta_i \phi_i(s) = \phi(s) \quad \text{QED}$$

Semi-gradient Algorithms are when $Target_t$ depends on θ ,
but we ignore $\nabla_\theta Target$ (take it as 0) anyway.

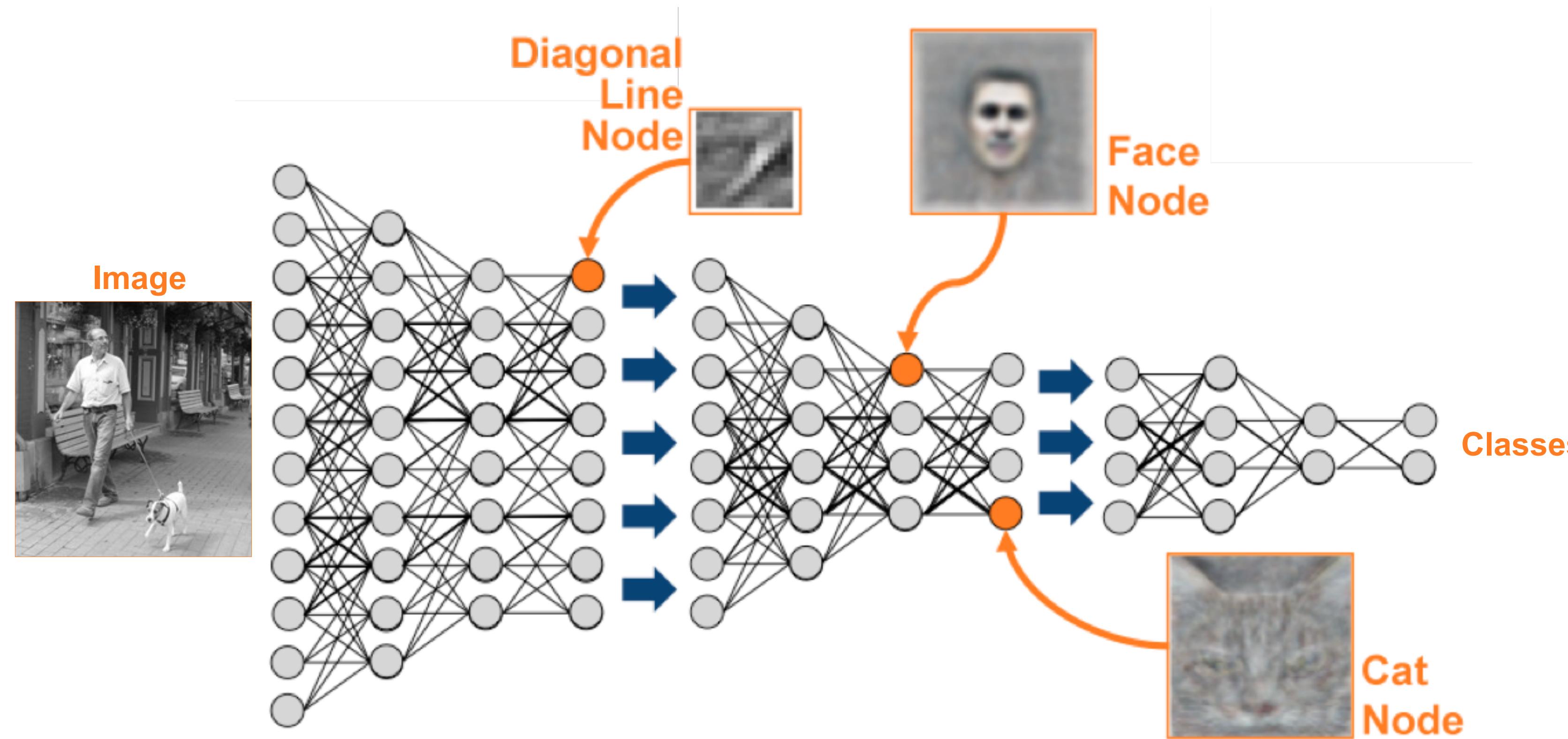
3 waves of neural networks

- First explored in the 1950-60s: Perceptron, Adaline...
 - only one learnable layer
- Revived in the 1980-90s as Connectionism, Neural Networks
 - exciting multi-layer learning using backpropagation (SGD); many successful applications; remained popular in engineering
- Revived again in ~2010 as Deep Learning
 - dramatically improved over state-of-the-art in speech recognition and visual object recognition, *transforming these fields*
 - the best algorithms were essentially the same as in the 1980s, except with faster computers and larger training sets

i.e., NNs won (eventually) because their performance scaled with Moore's law, whereas competing methods did not

Deep learning

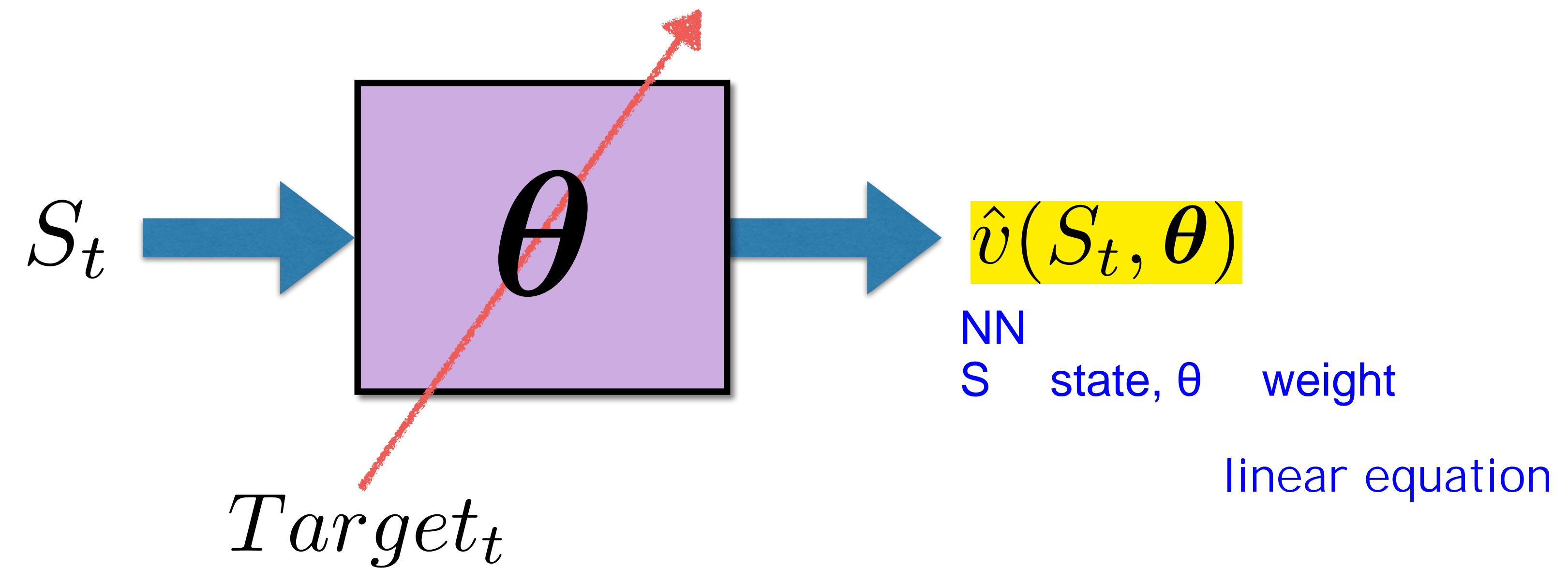
≡ multi-layer neural networks with many layers



- Each line has a learned connection weight
- Each node combines its weighted inputs, then applies a nonlinear transformation
- For each image, the network produces class labels as output, and true class labels are provided by people (supervised learning)
- Then each weight is incremented so as to reduce the squared error (stochastic gradient descent, backpropagation)

Value function approximation (VFA) replaces the table with a general parameterized form

for the purpose of Generalization
- more powerful
- more difficult



Stochastic Gradient Descent (SGD) is the idea behind most approximate learning

Stochastic
- Gradient

- is fast and non-deterministic, but has some error

Semi-gradient decent
- Target

- θ 가
- θ = semi-gradient

Linear case

General SGD:

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} Error_t^2$$

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

For VFA:

$$\leftarrow \theta - \alpha \nabla_{\theta} [Target_t - \hat{v}(S_t, \theta)]^2$$

Chain rule:

$$\leftarrow \theta - 2\alpha [Target_t - \hat{v}(S_t, \theta)] \nabla_{\theta} [Target_t - \hat{v}(S_t, \theta)]$$

Semi-gradient:

$$\leftarrow \theta + \alpha [Target_t - \hat{v}(S_t, \theta)] \nabla_{\theta} \hat{v}(S_t, \theta)$$

Linear case:

$$\leftarrow \theta + \alpha [Target_t - \hat{v}(S_t, \theta)] \phi(S_t)$$

Action-value form:

$$\theta \leftarrow \theta + \alpha [Target_t - \hat{q}(S_t, A_t, \theta)] \phi(S_t, A_t)$$

A natural objective in VFA
is to minimize the Mean Square Value Error

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

where $d(s)$ is the fraction of time steps spent in state s

True SGD will converge to a local minimum of the error objective
In *linear* VFA, there is only one minimum: local=global

Gradient Monte Carlo Algorithm for Approximating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \rightarrow \mathbb{R}$

Initialize value-function weights θ as appropriate (e.g., $\theta = 0$)

Repeat forever:

 Generate an episode $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using π

 For $t = 0, 1, \dots, T - 1$:

$$\theta \leftarrow \theta + \alpha [G_t - \hat{v}(S_t, \theta)] \nabla \hat{v}(S_t, \theta)$$

State aggregation is the simplest kind of VFA

- States are partitioned into disjoint subsets (groups)
- One component of θ is allocated to each group

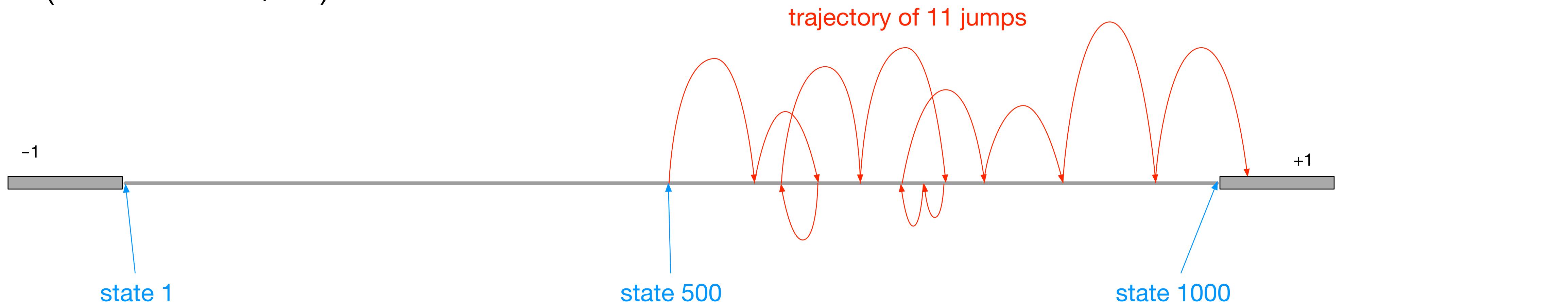
$$\hat{v}(s, \theta) \doteq \theta_{group(s)}$$

$$\nabla_{\theta} \hat{v}(s, \theta) \doteq [0, 0, \dots, 0, 1, 0, 0, \dots, 0]$$

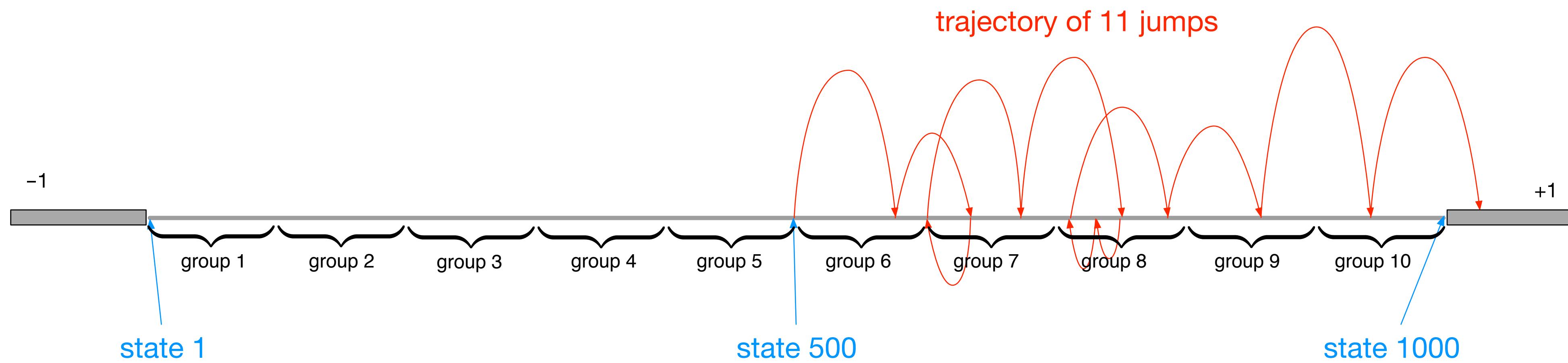
Recall: $\theta \leftarrow \theta + \alpha [Target_t - \hat{v}(S_t, \theta)] \nabla_{\theta} \hat{v}(S_t, \theta)$

The 1000-state random walk example

- States are numbered 1 to 1000
- Walks start in the near middle, at state 500 $S_0 = 500$
- At each step, *jump* to one of the 100 states to the right, or to one of the 100 states to the left $S_1 \in \{400..499\} \cup \{501..600\}$
- If the jump goes beyond 1 or 1000, terminates with a reward of -1 or $+1$ (otherwise $R_t=0$)



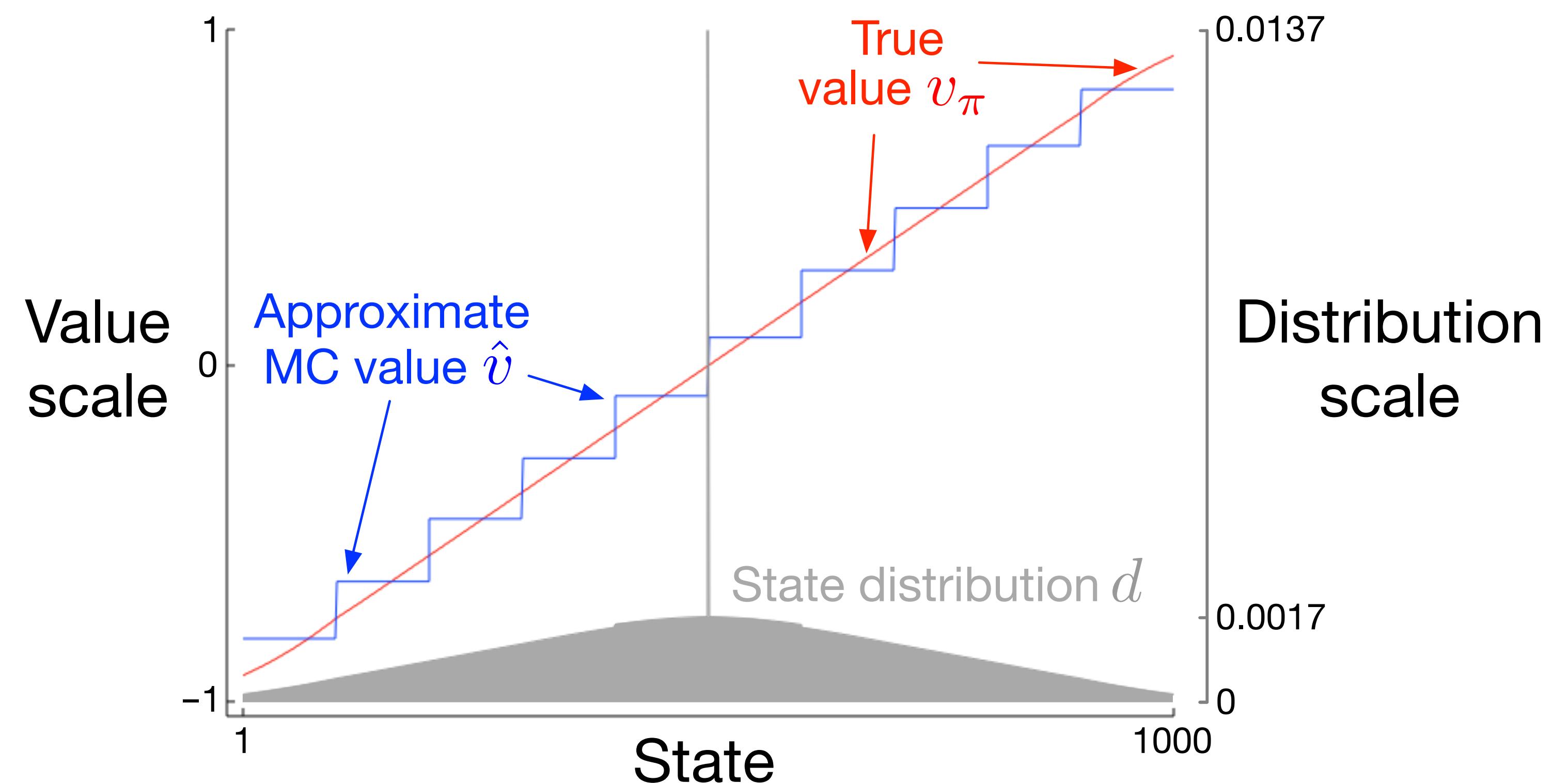
State aggregation into 10 groups of 100



The whole value function over 1000 states will be approximated with 10 numbers!

Gradient MC works well on the 1000-state random walk using *state aggregation*

- 10 groups of 100 states
- after 100,000 episodes
- $\alpha = 2 \times 10^{-5}$
- state distribution affects accuracy



Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated

Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^n \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$

Initialize value-function weights θ arbitrarily (e.g., $\theta = \mathbf{0}$)

Repeat (for each episode):

 Initialize S

 Repeat (for each step of episode):

 Choose $A \sim \pi(\cdot | S)$

 Take action A , observe R, S'

$$\theta \leftarrow \theta + \alpha [R + \gamma \hat{v}(S', \theta) - \hat{v}(S, \theta)] \nabla \hat{v}(S, \theta)$$

$S \leftarrow S'$

 until S' is terminal

TD converges to the TD fixedpoint, θ_{TD} ,
a biased but interesting answer

TD(0) update:

$$\begin{aligned}\theta_{t+1} &\doteq \theta_t + \alpha \left(R_{t+1} + \gamma \theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t \right) \phi_t \\ &= \theta_t + \alpha \left(R_{t+1} \phi_t - \phi_t (\phi_t - \gamma \phi_{t+1})^\top \theta_t \right)\end{aligned}$$

Fixedpoint analysis:

$$\begin{aligned}\mathbf{b} - \mathbf{A}\theta_{TD} &= 0 \\ \Rightarrow \quad \mathbf{b} &= \mathbf{A}\theta_{TD} \\ \Rightarrow \quad \theta_{TD} &\doteq \mathbf{A}^{-1}\mathbf{b}\end{aligned}$$

Least-square method

가

In expectation:

$$\mathbb{E}[\theta_{t+1} | \theta_t] = \theta_t + \alpha(\mathbf{b} - \mathbf{A}\theta_t),$$

where

$$\mathbf{b} \doteq \mathbb{E}[R_{t+1} \phi_t] \in \mathbb{R}^n \quad \text{and} \quad \mathbf{A} \doteq \mathbb{E}[\phi_t (\phi_t - \gamma \phi_{t+1})^\top] \in \mathbb{R}^n \times \mathbb{R}^n$$

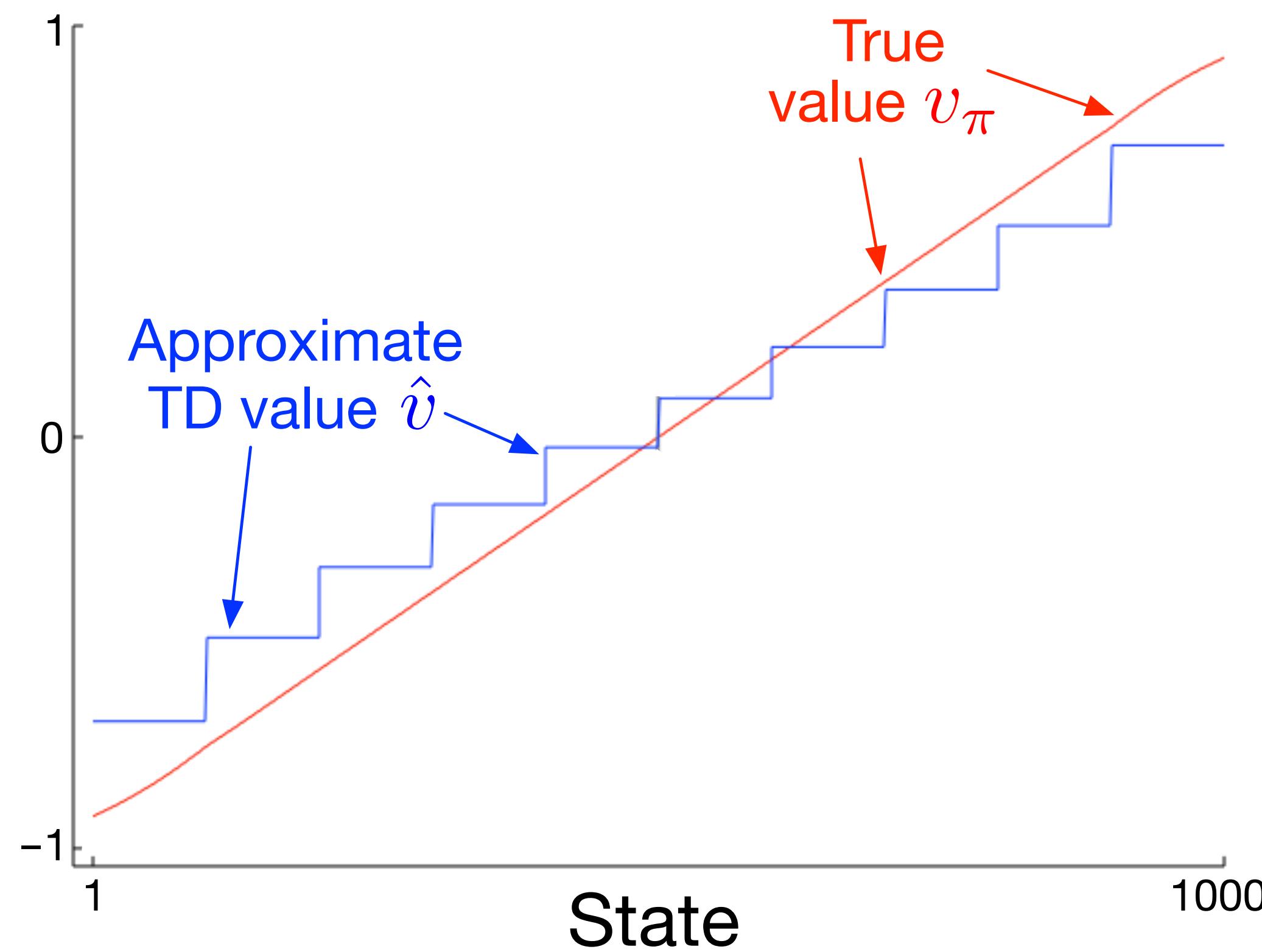
Guarantee:

$$\text{MSVE}(\theta_{TD}) \leq \frac{1}{1-\gamma} \min_{\theta} \text{MSVE}(\theta)$$

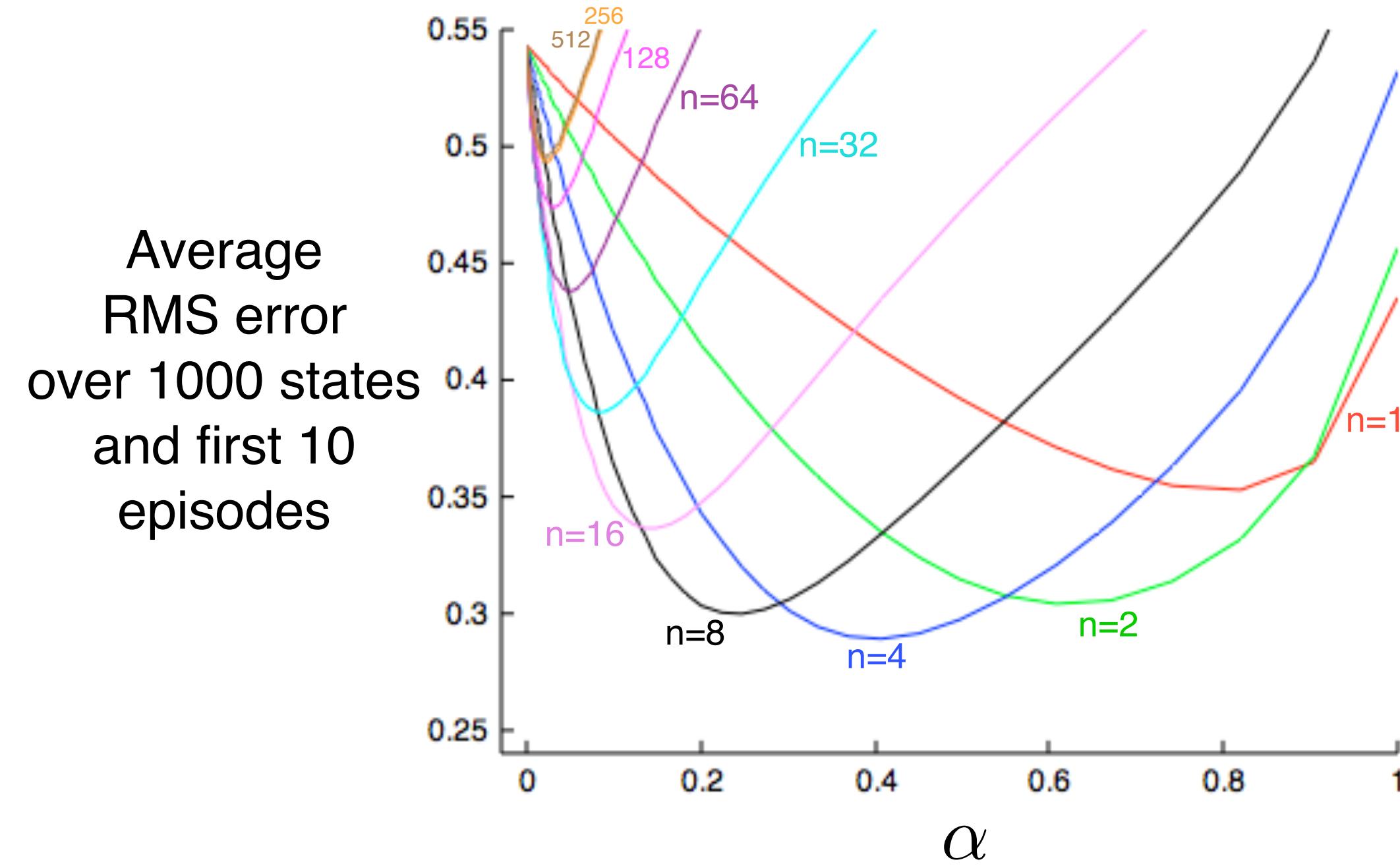
Gradient TD is less accurate than MC on the 1000-state random walk using state aggregation

- 10 groups of 100 states
- after 100,000 episodes
- $\alpha = 2 \times 10^{-5}$

Relative values are
still pretty accurate

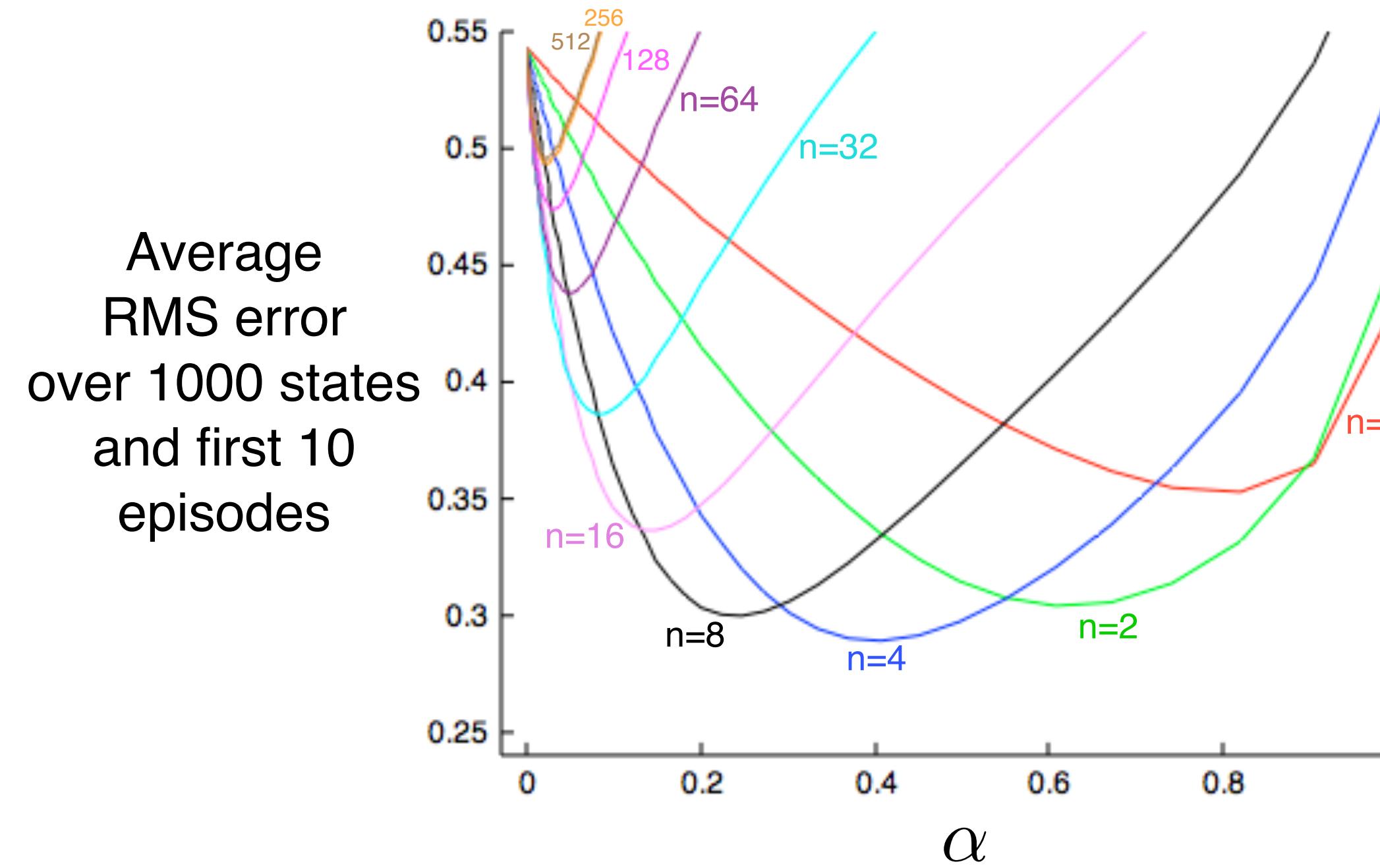


Bootstrapping still greatly speeds learning

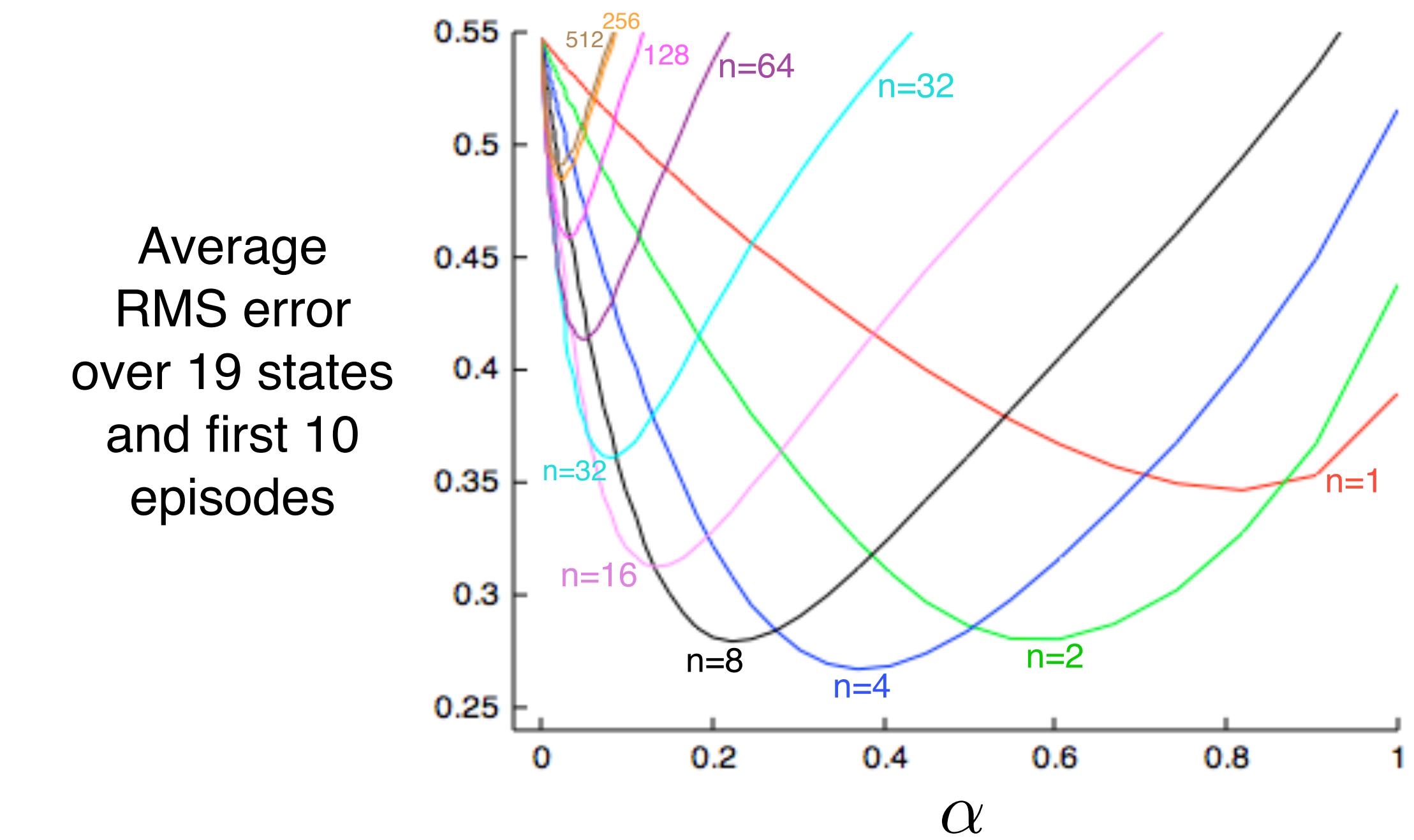


1000 states aggregated
into 20 groups of 50

Bootstrapping still greatly speeds learning very much like the tabular 19-state walk



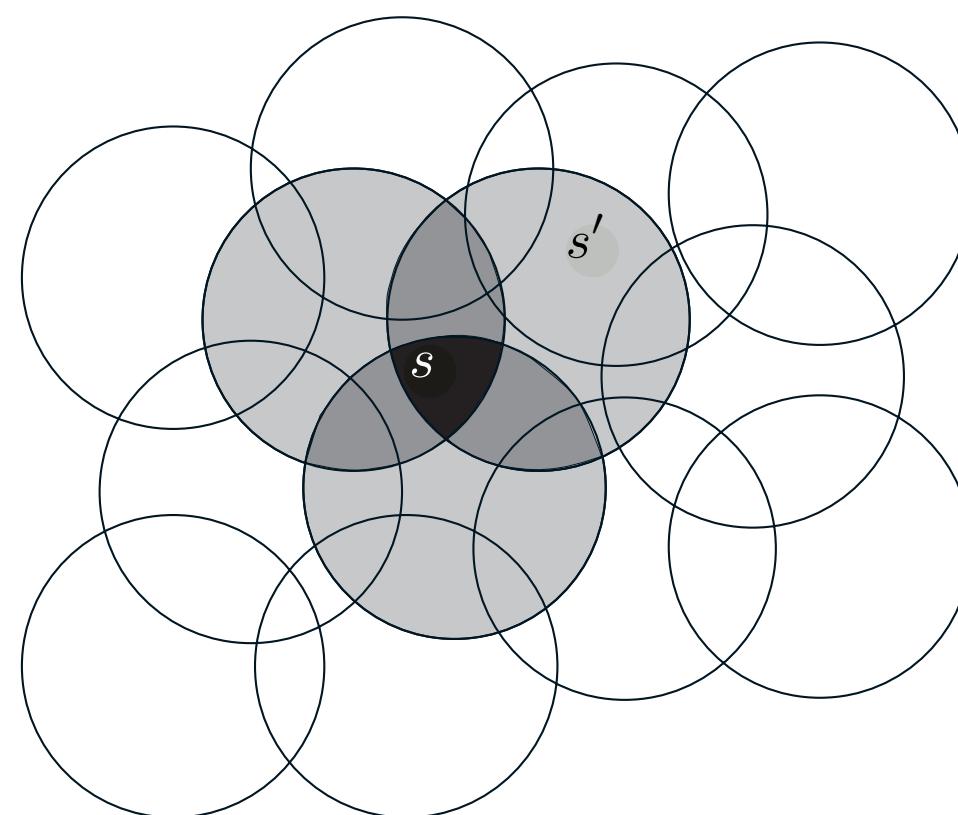
1000 states aggregated
into 20 groups of 50



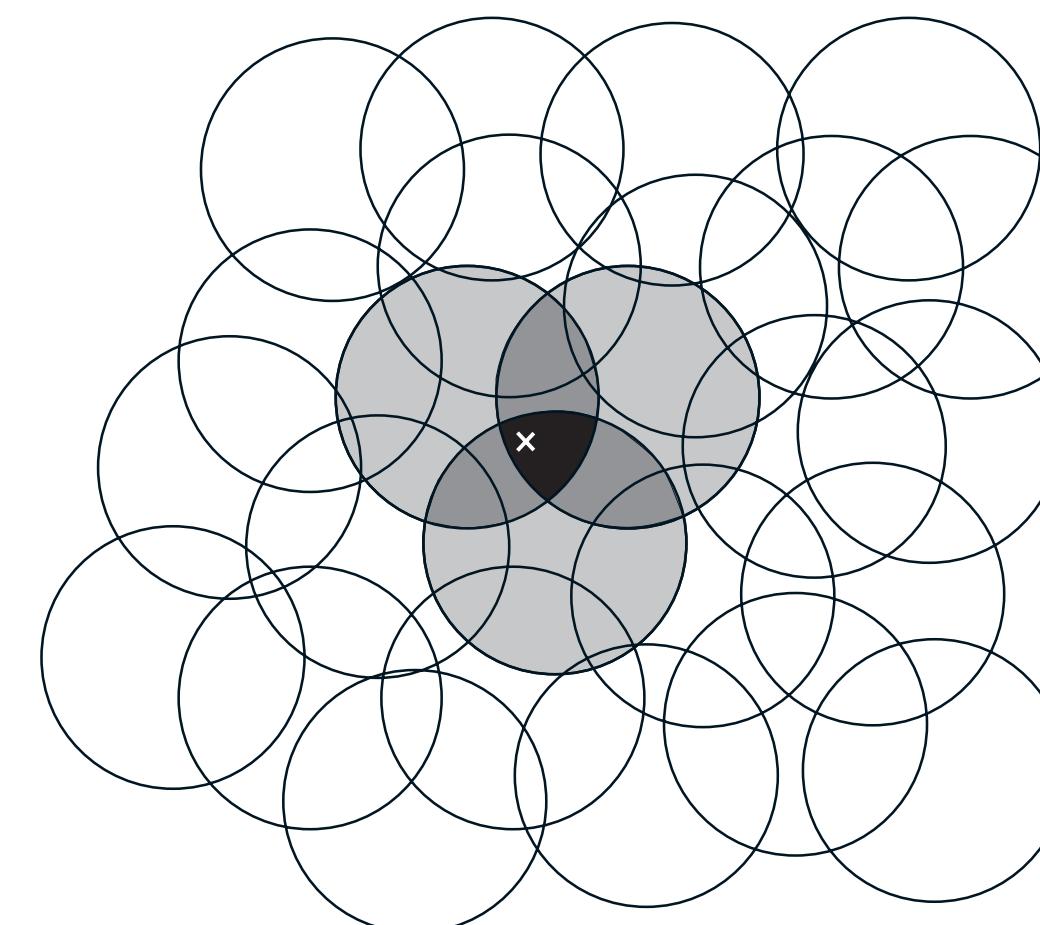
19 states tabular
(from Chapter 7)

that is combination of natural qualities

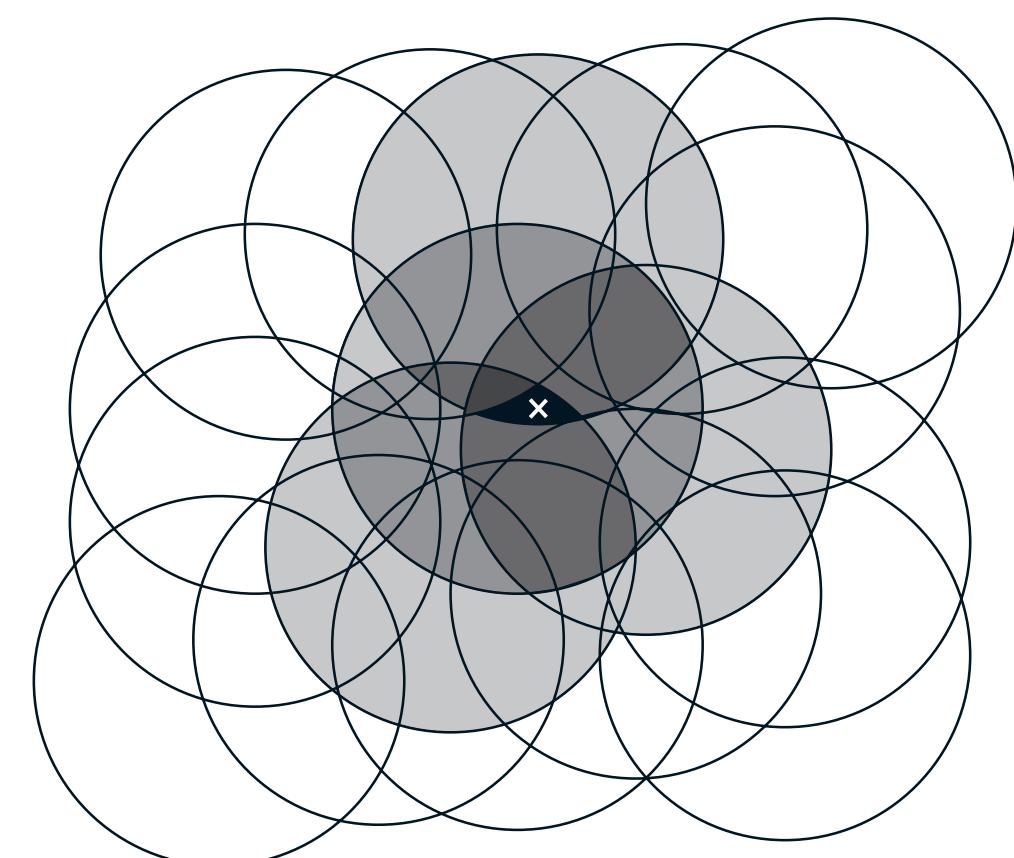
With binary features, a continuous state space
can be coarsely coded, adding *generalization*



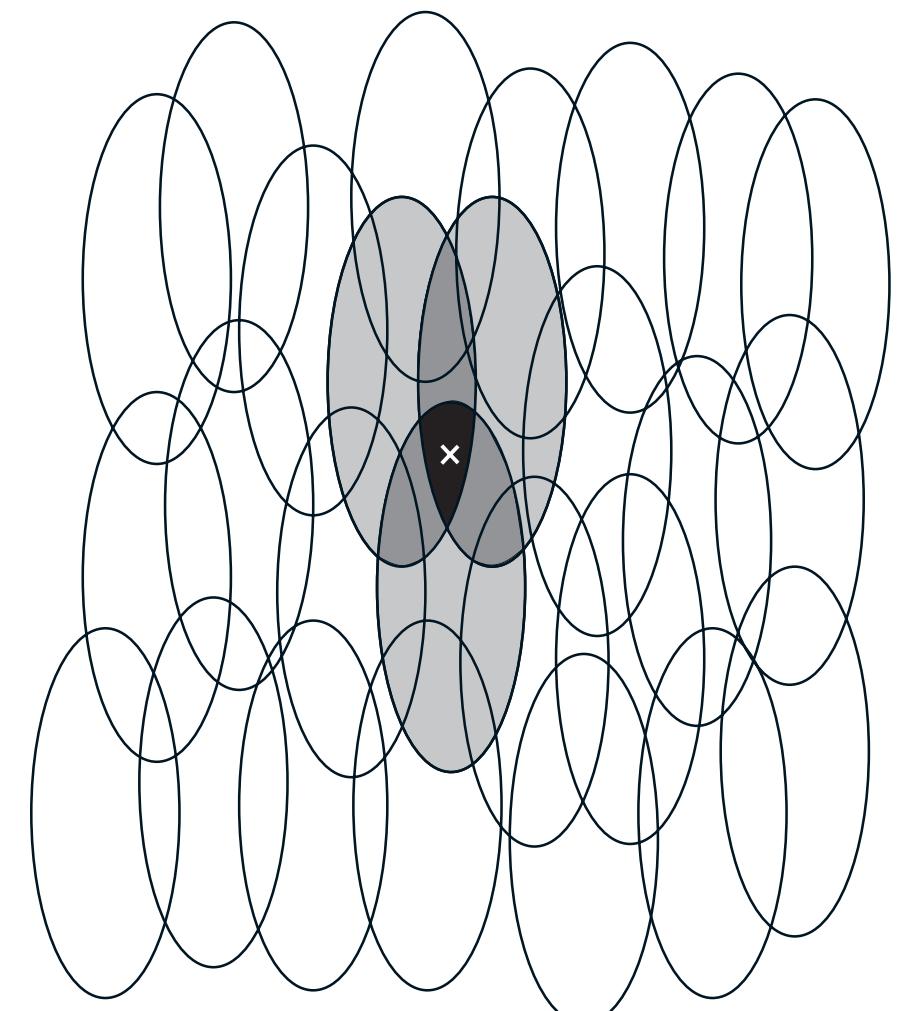
Idea



a) Narrow generalization



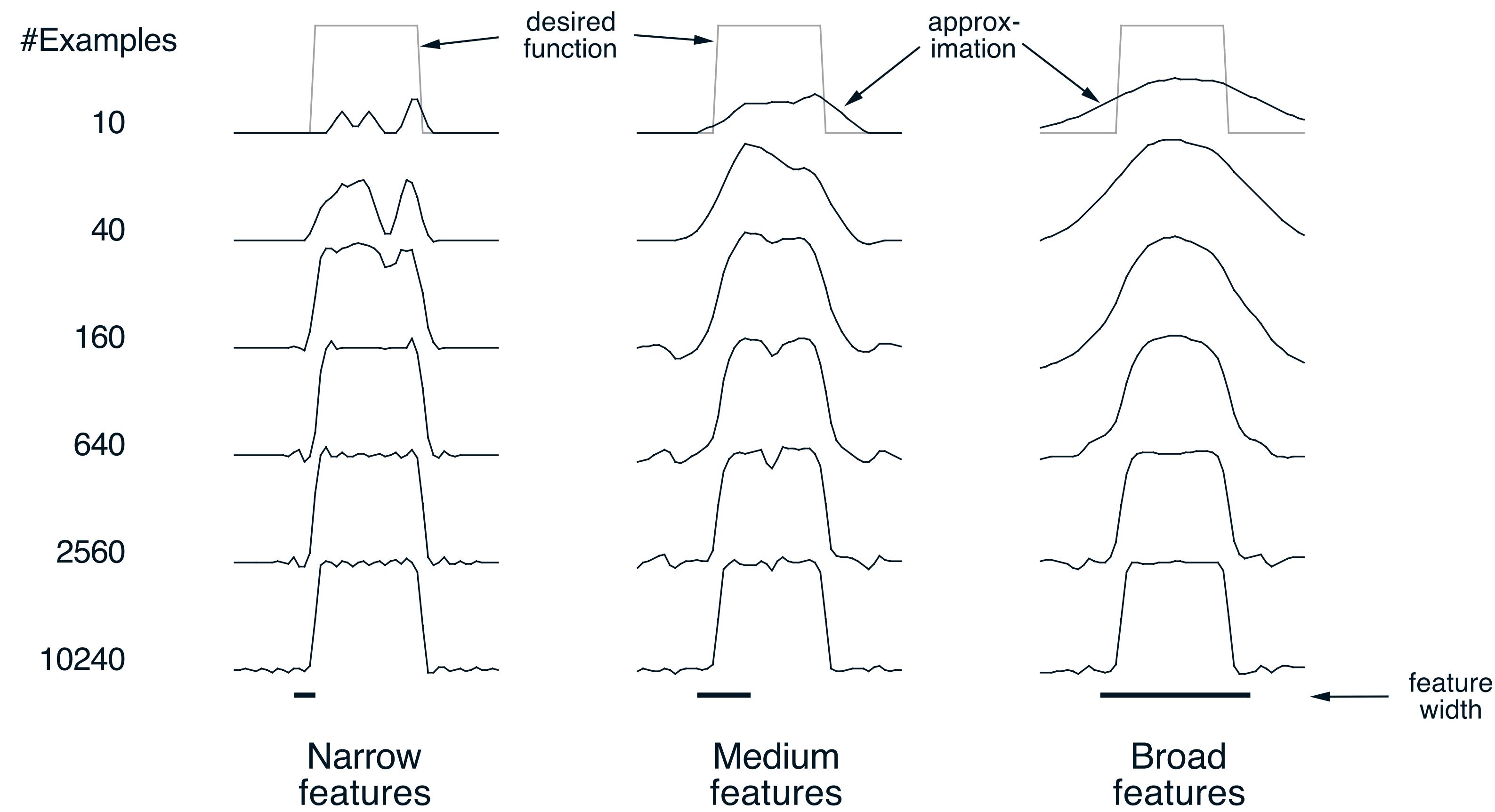
b) Broad generalization



c) Asymmetric generalization

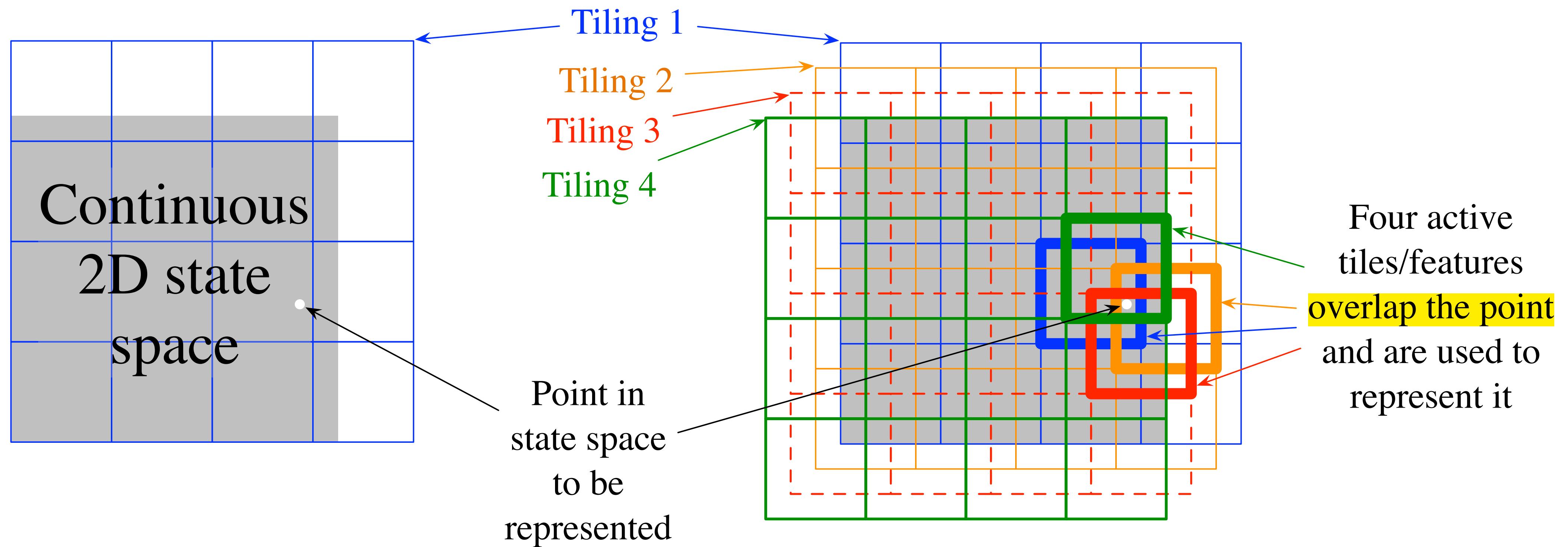
The width of the receptive fields determines breadth of generalization

1D example,
supervised training



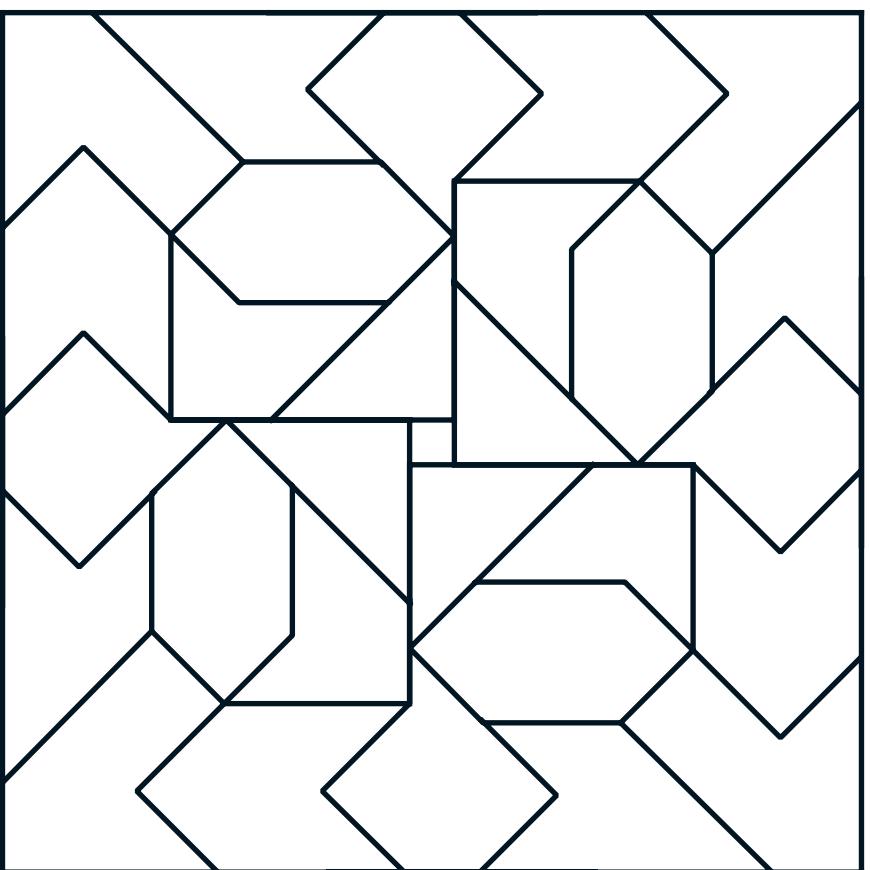
Tile coding is coarse coding for digital computers, with rectangular receptive fields, controlled overlap

2D example

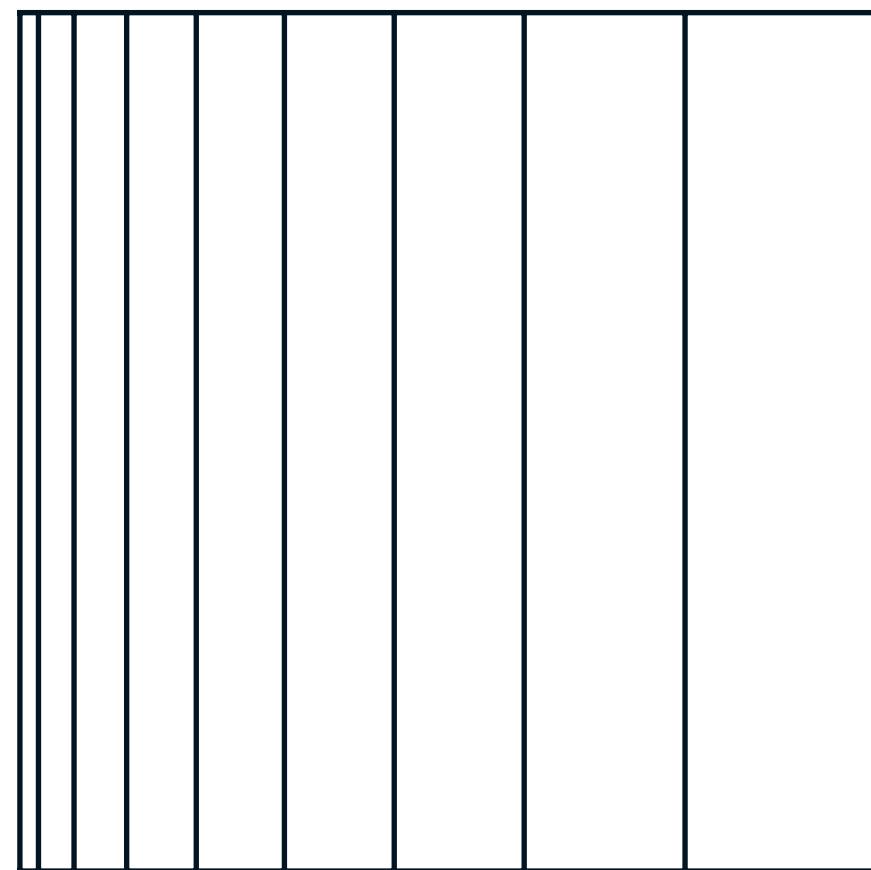


Nevertheless, tile coding is very flexible

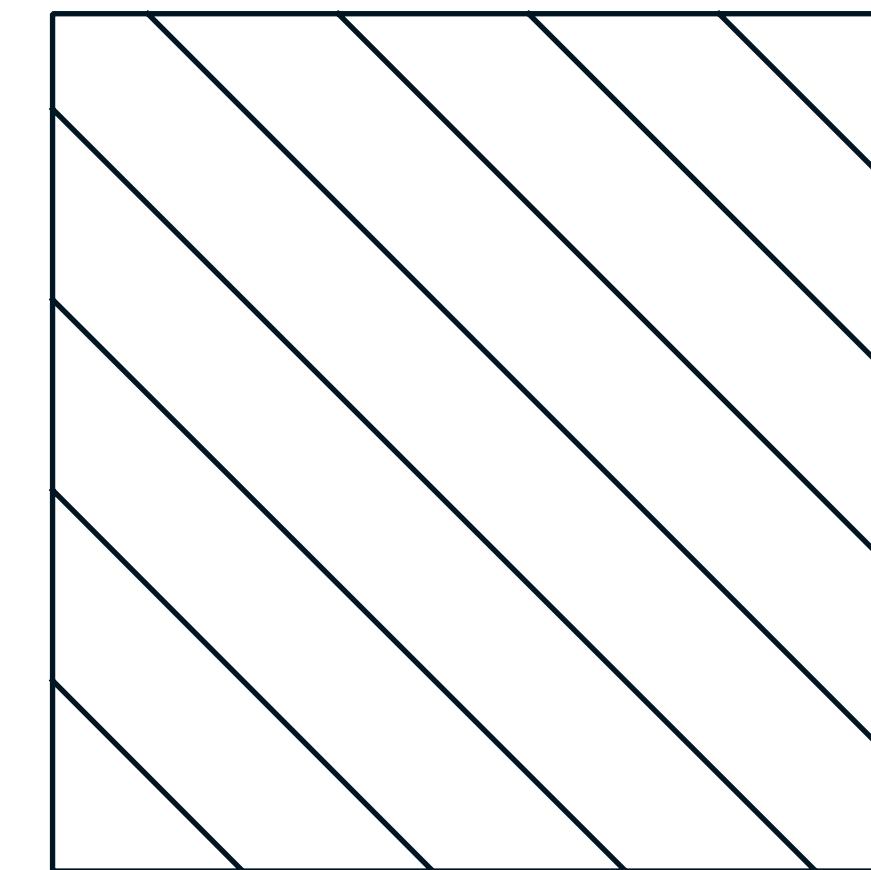
Non-traditional tilings:



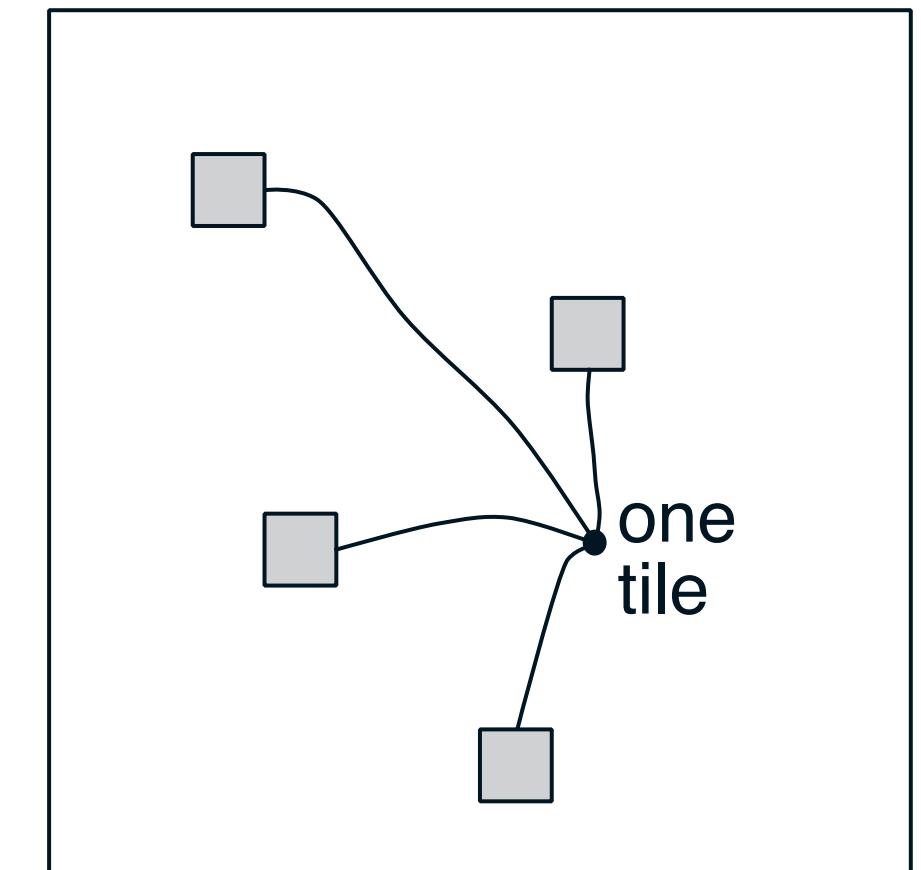
a) Irregular



b) Log stripes

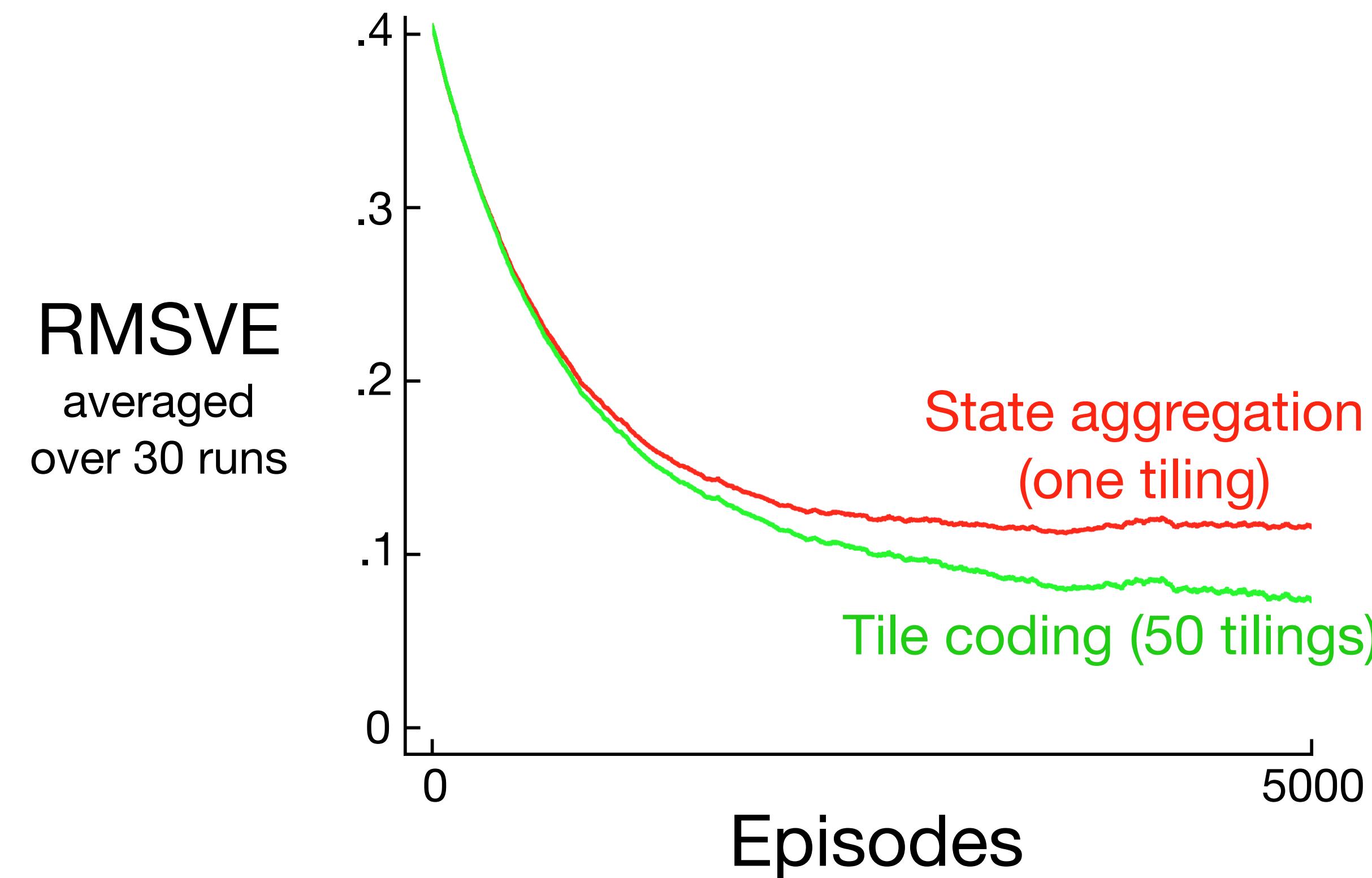


c) Diagonal stripes



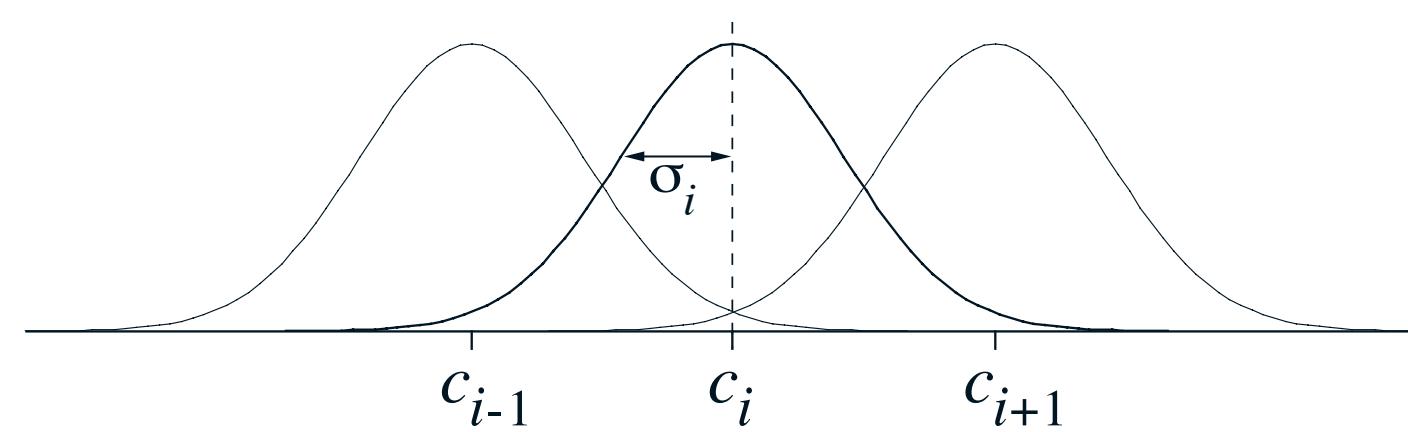
Hashing

Tile coding works better than state aggregation on the 1000-state random walk



- groups/tiles of 200 states
- α set so that initial learning rate is the same for both methods

Smooth-edged receptive fields are little different but increase computational complexity

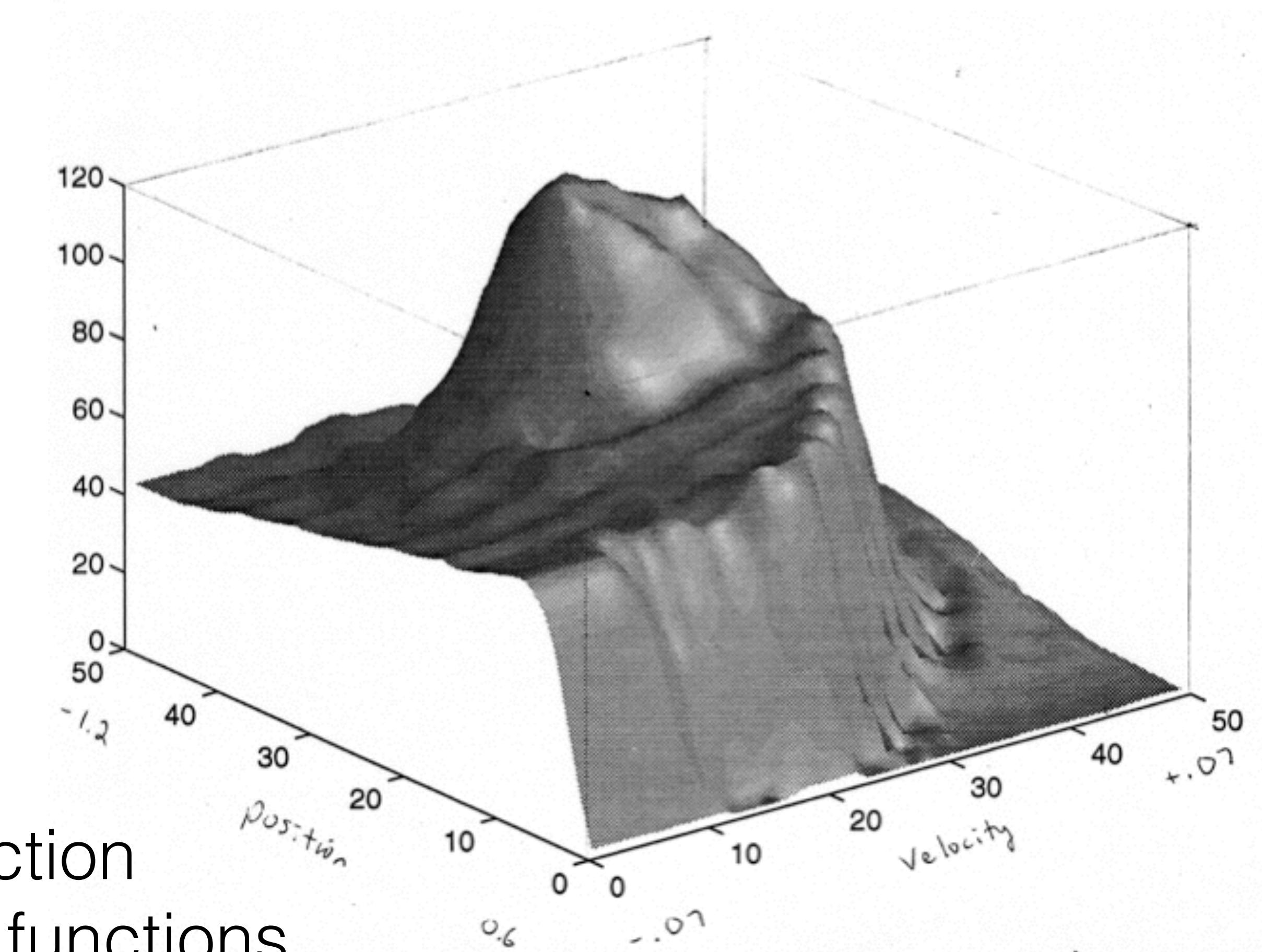


1D radial basis functions

Possible Feature

- RBF(radial basis function)
 - Complex
 - Polynomials
 - Fourier Basis

A 2D approx. value function
learned with 2D radial basis functions



Matt Kretchmar, 1995

Conclusions

linear value function 가

NN non-linear function approximation

Section 9.6

NN

non-linear

가

- Value-function approximation by stochastic gradient descent enables RL to be applied to arbitrarily large state spaces
- Most algorithms just carry over the Targets from the tabular case
- With bootstrapping (TD), we don't get true gradient descent methods
 - this complicates the analysis
 - but the linear, on-policy case is still guaranteed convergent
 - and learning is still *much faster*
- For continuous state spaces, coarse/tile coding is a good strategy
- For ambitious AI, artificial neural networks are an interesting strategy