transpose
\n
$$
\overrightarrow{V(s)} \leq v_{\pi}(s) \approx \hat{v}(s, \theta) \approx \theta^{\top} \phi(s) \approx \sum_{i=1}^{n} \theta_{i} \cdot \phi_{i}(s) = 1.71
$$
\ninner product

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

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

Chapter 9: Generalization and Function Approximation

Objectives of this chapter:

- \Box Look at how experience with a limited part of the state set be used to produce good behavior over a much larger part.
- \Box 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 θ : on a veetor or parameters of

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

only the parameters are updated approximated value of state *s* given parameter vector ✓. For example, ˆ*v* might be the $f(x,y)$ and puremeters are up allow

e.g., θ could be the modifiable connection weights and thesholds of a deep neural network $\cos \theta$ and the the wedifield be ign, be could be the mountable tresholds of a deep heural hetw

Adapt Supervised Learning Algorithms

 $Error = (target output - actual output)$ Training example $=$ {input, target output}

Backups as Training Examples watter time step. At time step. At time step. At time step. At time σ

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: the TD update is *Rt*+1 + *Vt*(*St*+1).

$$
\begin{array}{ccc}\n\text{features of } S_t \longrightarrow R_{t+1} + \gamma \ V(S_{t+1}) \\
\uparrow & \uparrow \\
\text{input} & \text{target output}\n\end{array}
$$

Any FA Method?

 \Box In principle, yes:

- **E** artificial neural networks
- **decision trees**
- multivariate regression methods
- \blacksquare etc.
- \Box But RL has some special requirements:
	- usually want to learn while interacting (online)
	- ability to handle nonstationarity
	- \blacksquare other?

Gradient Descent Methods Γ redient Descent Methods particularly welcome internet

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

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

Assume, for now, training examples of this form:

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

Gradient Descent

Let $f(\theta)$ be a function to be minimized, e.g., an error Its gradient with respect to θ 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 Dorformance Moscures $\overline{}$ or the interested over the inputs. In our value prediction problem, the input set of the in

- □ Many are applicable but...
- \Box 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?
- \Box Let us assume that *d* is always the distribution of states at which backups are done. the relative importance of errors in distribution in distribution in distribution is important states. The state of the states of the states. The states of the state $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ is always the distribution of states at all states. After all states at all states. After all states at all, $\begin{bmatrix} 1 & 1 \end{bmatrix}$ t there are generally far more states than the flexibility of \mathcal{L} are components to \mathcal{L}
- **T** The **on-policy distribution**: the distribution created while following the policy being evaluated. Stronger results are available for this distribution.

Gradient Descent Derivation

$$
\theta_{t+1} = \theta_t - \alpha \nabla \text{MSVE}(\theta_t)
$$

= $\theta_t - \alpha \nabla \sum_{s \in \mathcal{S}} d(s) \left[v_\pi(s) - \hat{v}(s, \theta_t)\right]^2$
= $\theta_t - \alpha \sum_{s \in \mathcal{S}} d(s) \nabla \left[v_\pi(s) - \hat{v}(s, \theta_t)\right]^2$
= $\theta_t - 2\alpha \sum_{s \in \mathcal{S}} d(s) \left[v_\pi(s) - \hat{v}(s, \theta_t)\right] \nabla \left[v_\pi(s) - \hat{v}(s, \theta_t)\right]$
= $\theta_t + \alpha \sum_{s \in \mathcal{S}} d(s) \left[v_\pi(s) - \hat{v}(s, \theta_t)\right] \nabla \hat{v}(s, \theta_t)$

(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*.

to converge to a local optimum under the usual stochastic approximation conditions

Suppose we just have targets V_t instead : approximate it by substituting *V^t* in place of *v*⇡(*St*). This yields the following general pose we just have targets V instead : action (or simulated interaction) with the environment using policy ⇡. Let *G^t* denote

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

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). \int *P* each *V* is an unbiased estimate of *v* (S) , \int $\begin{bmatrix} u & v \end{bmatrix}$ $(e_i, E \n\{V_t\} = V_\pi(S_t)$, then gradient descent convergence. α each V_t is an unbiased estimate of $v_\pi(S_t)$, e. $E\{V\} = v(S)$, then gradient descent converges $\begin{pmatrix} i & j & k+i \end{pmatrix}$ Similarly, we can use *n*-step TD returns and their averages for *Vt*. For example,

e.g., the Monte Carlo target $V_t = G_t$ (unbiased): $\frac{d}{dt}$ Monte $\frac{C}{dt}$ for a $V - C$ (unbiased)[.] and return target $v_t = v_t$ (anonasca). 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 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 $\lambda < 1$ $\frac{1}{\sqrt{2}}$

⇒ standard gradient descent results don't apply

But we do it anyway! bootstrapping methods can be quite e↵ective, and other performance guarantees are all we do it allyway!

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

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** *<u>tient</u>* = ✓*^t* + ↵*t*e*t,* (9.5) \mathbf{t} ient-hased $\mathbf{T} \mathbf{D}(\lambda)$ hackwards view C reading the seed $T\mathbf{D}(t)$, he eliminate of \mathbf{C} σ faulent-daseu t $\bm{\nu}(\wedge)$, dackwafus view

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

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

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

On-Line Gradient-Descent TD(λ**)**

9.3. LINEAR METHODS 207

Initialize θ as appropriate for the problem, e.g., $\theta = 0$ Repeat (for each episode): $e = 0$ $S \leftarrow$ 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' $\delta \leftarrow R + \gamma \hat{v}(S', \boldsymbol{\theta}) - \hat{v}(S, \boldsymbol{\theta})$ $\mathbf{e} \leftarrow \gamma \lambda \mathbf{e} + \nabla \hat{v}(S, \theta)$ $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta \mathbf{e}$ $S \leftarrow S'$ until S' is terminal

Linear Methods below. However the features are constructed, the approximate state-value function **the contract of the contract** is that in which the approximate function, ˆ*v*, is a linear function of the parameter

0*ⁿ*¹*G*(*n*)

vector, ✓. Corresponding to every state *^s*, there is a vector of features (*s*) *.*

z s. ∂ Represent states as feature vectors: for each $s \in \mathcal{S}$: *n*=1 resent states as feature vectors: $\text{rank }_{\Omega} \subset \mathcal{C}$ $\mathbf{b} \subset \mathbf{0}$.

G

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

$$
\nabla \hat{v}(s,{\bm{\theta}}) = -2
$$

^t + 0*^T t*¹*G^t* ⁼ *^G*(1)

Linear Methods below. However the features are constructed, the approximate state-value function **the contract of the contract** is that in which the approximate function, ˆ*v*, is a linear function of the parameter

0*ⁿ*¹*G*(*n*)

vector, ✓. Corresponding to every state *^s*, there is a vector of features (*s*) *.*

z s. ∂ Represent states as feature vectors: for each $s \in \mathcal{S}$: *n*=1 resent states as feature vectors: $\text{rank }_{\Omega} \subset \mathcal{C}$ $\mathbf{b} \subset \mathbf{0}$.

G

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

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

^t + 0*^T t*¹*G^t* ⁼ *^G*(1)

Nice Properties of Linear FA Methods simply *linear*. It is natural to use gradient-descent updates with linear function approximation. 208*CHAPTER 9. ON-POLICY PREDICTION WITH LINEAR APPROXIMATION* \mathbf{N} ies, \mathbf{D} percyclics of \mathbf{I} in equal \mathbf{A} \mathbf{M} othod guaranteed to converge to or near a local optimum is automatically guaranteed to converge to

- The gradient is very simple: $\nabla \hat{v}(s, \theta) = \phi(s)$
- \Box For MSE, the error surface is simple: quadratic surface with a single minimum.
- \Box Linear gradient descent TD(λ) converges:
	- **Step size decreases appropriately** ely
	- On-line sampling (states sampled from the on-policy distribution) and the minimum-error parameter vector, \mathbf{I} \mathbf{P} is a reduced over the usual conditions (2.7).
	- Converges to weight vector θ_{∞} with property:

$$
\text{MSVE}(\boldsymbol{\theta}_{\infty}) \leq \frac{1 - \gamma \lambda}{1 - \gamma} \text{MSVE}(\boldsymbol{\theta}^*)
$$
\n(Tsitsiklis & Van Roy, 1997)

\nbest weight vector

Learning and Coarse Coding

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction 22

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
- \Box Easy to compute indices of the features present

Shape of tiles \Rightarrow Generalization

 $#Tilings$ \Rightarrow Resolution of final approximation

Tile Coding Cont.

Coarse Coding

Shaping Generalization in Coarse Coding

a) Narrow generalization

b) Broad generalization

c) Asymmetric generalization

Can you beat the "curse of dimensionality"?

- \Box Can you keep the number of features from going up exponentially with the dimension?
- \Box Function complexity, not dimensionality, is the problem.
- \Box 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

The extension to action-value prediction-value prediction-value prediction-value prediction is straightforward. In this case σ

Learning state-action values \Box The general gradient-descent rule: □ Gradient-descent Sarsa(λ) (backward view): $\{ \text{description of } (S_t, A_t), Q_t \}$ Training examples of the form: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha$ $\sqrt{ }$ $Q_t - \hat{q}(S_t, A_t, \mathbf{w}_t)$ $\overline{}$ $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[Q_t - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla_{\mathbf{w}_t} \hat{q}(S_t, A_t, \mathbf{w}_t).$ aining examples of the form: \overline{a} $\{G_t, A_t\}, \mathcal{Q}_t\}$ ϵ general gradient descent rate. aming state-action values arilling state-action values $\frac{d}{dx}$ $\int d\rho$ conjustigate of $(\rho_4 \wedge \rho)$ $\{$ description of (S_t, A_t) , $Q_t\}$ α α α α α ^t, α ^t α ^t α ^t α ^t α ^t α ^t α ϵ general gradient-descent rule. \overline{a} $Q_t - q(\mathcal{S}_t, A_t, \mathbf{W}_t)$ i $\mathbf{w}_{\mathbf{w}_t} q(\mathcal{S}_t, A_t, \mathbf{w}_t)$

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

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) L ilitat vitaultiil-dastu salsa (\wedge)

Let θ and θ 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 θ as appropriate for the problem, e.g., $\theta = 0$ Repeat (for each episode):

 $e = 0$ $S, A \leftarrow$ initial state and action of episode (e.g., ε -greedy) $\mathcal{F}_A \leftarrow$ set of features present in *S*, *A* Repeat (for each step of episode): For all $i \in \mathcal{F}_A$:
 $e_i \leftarrow e_i + 1$ $e_i \leftarrow e_i + 1$ (accumulating traces)
or $e_i \leftarrow 1$ (replacing traces) $(replacing traces)$ 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., ε -greedy) $\delta \leftarrow \delta + \gamma Q_{A}$ $\theta \leftarrow \theta + \alpha \delta e$ $\mathbf{e} \leftarrow \gamma \lambda \mathbf{e}$ $S \leftarrow S'$ $A \leftarrow A'$

Approx Value Functions on Mountain-Car Task

R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction 30

Radial Basis Functions (RBFs)

e.g., Gaussians

Mountain Car with Radial Basis Functions

Mountain-Car Results

Should We Bootstrap?

Summary

O Generalization

- \Box Adapting supervised-learning function approximation methods
- □ Gradient-descent methods
- \Box Linear gradient-descent methods
	- Radial basis functions
	- Tile coding
	- Kanerva coding