On Policy Prediction With Function Approximation

- first study state value function from on policy data: v_π from π
+ the prediction problem
- not tabular now; weight vector $w \in \mathbb{R}^d$
+ $\hat{v}(s, w) \approx v_\pi(s)$ is approximate value of state s given weight vector w
× w can be weights of features of a linear function
× w can be weights of layers of an ANN
* ANN represents a vast array of functions
× w can be splits in a decision tree
+ dimensionality of w (d) far less than states ($d \ll |S|$)
- powerful in generalization, more difficult to understand
- applicable to partially observable state
+ if $\hat{v}$ does not depend on certain aspects of state, same as if those aspects are unobservable
- function approximation cannot augment state rep with past memories

Value Function Approximation

- focus on "backed-up" value update
- $s \to u$ = s being updated, u is update target that s value shifts towards
+ e.g. MC: $S_t \to G_t$
+ e.g. TD(0): $S_t \to R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t)$
+ e.g. n-step TD: $S_t : G_{t:t+n}$
+ e.g. DP: $s \to \mathbb{E}[R_{t+1} + \gamma \hat{v}(S_{t+1}, w) | S_t = s]$
× DP, s is arbitrary, not real experienced S_t
- no longer focus on single input-output updates
+ now more complex, arbitrary updates for multi state changes
+ mimic input-output examples = $\boxed{\text{supervised learning}}$
× if output is a number u, called $\boxed{\text{function approximation}}$
* estimate value function
- technically, can use any supervised methods
+ however, ANN, decision tree, etc. rely on static training sets
+ must use methods suitable to incremental updates, nonstationary targets, online

The Prediction Objective ($\overline{VE}$)

- till now, tabular methods for problems would converge on known true values
- on policy distribution of episodic tasks
+ $h(s)$ = likelihood of starting in s
+ $\eta(s)$ = time steps spent on avg in s

$$\eta(s) = h(s) + \sum_{\bar{s}} \eta(\bar{s}) \sum_a \pi(a|\bar{s})\, p(s|\bar{s}, a) \quad \forall s \in S$$

+ solve system of equations for $\eta(s)$
+ distribution becomes $\eta(s)$ normalized to one across all states

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$$

× with no discounting
× if $\gamma < 1$, becomes a termination form
- again, separate episodic from continuous for formal analysis
- focus: $\overline{VE}(w) = \sum_{s \in S} \mu(s)\left[v_\pi(s) - \hat{v}(s, w)\right]^2$
  mean square error value
+ is it the best? not sure, all we want is for a better policy, eventually
- first work on finding global optimum w^*: $\overline{VE}(w^*) \le \overline{VE}(w) \ \forall w$
+ easy for linear functions, hard for complex ANN, decision tree
- instead, focus on local optimum in neighborhood of w^*
+ best we can do for nonlinear approximators
× huge number of possible methods, focus on gradient methods

Stochastic-gradient & Semi-gradient Methods

- SGD, great for online learning
  $w = \{w_1, w_2, \dots, w_d\}^T$ (column vector)
- $\hat{v}(s, w)$ differentiable of w $\forall s \in S$
- still update w at discrete t, so w_t
- assume new $S_t \to v_\pi(S_t)$ every t and $v_\pi(S_t)$ is true value
+ limited resolution, no w that allows for exact rep of all states
× also generalize to unseen examples
- assume same μ to minimize $\overline{VE}$
+ minimize error using SGD

$$w_{t+1} = w_t - \frac{1}{2}\alpha \nabla\left[v_\pi(S_t) - \hat{v}(S_t, w_t)\right]^2$$
$$= w_t + \alpha\left[v_\pi(S_t) - \hat{v}(S_t, w_t)\right]\nabla \hat{v}(S_t, w_t)$$

× α positive step size param
× $\nabla f(w)$ for scalar expression $f(w)$ is partial derivatives

$$\nabla f(w) \doteq \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \dots, \frac{\partial f(w)}{\partial w_d}\right)^T$$

- gradient descent is proportional negative step along gradient for each weight
- stochastic when on a single example
+ don't fully commit with big step: may result in error in other states
- now suppose target output is not true value
+ $S_t \to U_t$, approximation of $v_\pi(S_t)$
+ now update is

$$w_{t+1} = w_t + \alpha\left[U_t - \hat{v}(S_t, w_t)\right]\nabla \hat{v}(S_t, w_t)$$

+ if U_t is unbiased, i.e. $\mathbb{E}[U_t | S_t = s] = v_\pi(s)$, guaranteed convergence
+ same holds because $U_t = G_t$ if examples are simulation of env
× Bootstrapping does not produce same guarantee
* n-step returns $G_{t:t+n}$ not G_t
* referred to as semi-gradient methods because w actually impacts U_t
- still, bootstrapping converge reliably in specific cases
+ 1) much faster
  2) continual, online, no need for ep end

Gradient Monte Carlo, $\hat{v} \approx v_\pi$

```
input: $\pi$ for evaluation
input: differentiable function $\hat{v}: S \times \mathbb{R}^d \to \mathbb{R}$
algo param: $\alpha$
Init: value function weights $w \in \mathbb{R}^d$ arbitrarily (e.g. $w = 0$)
loop forever:
    gen ep. $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$ using $\pi$
    for each step $t = 0, 1, \dots, T-1$
        $w \leftarrow w + \alpha[G_t - \hat{v}(S_t, w)]\nabla \hat{v}(S_t, w)$
```

Semi-Gradient TD

```
input: $\pi$ for eval
input: differentiable $\hat{v}: S^+ \times \mathbb{R}^d \to \mathbb{R}$
param: $\alpha$
init: $w \in \mathbb{R}^d$ arbitrarily

for each ep:
    init S
    for each step:
        A ← $\pi(S)$
        S', R ← observe(A)
        $w \leftarrow w + \alpha[R + \gamma \hat{v}(S', w) - \hat{v}(S, w)]\nabla \hat{v}(S, w)$
        S ← S'
    until S' terminal
```

- $\boxed{\text{State aggregation}}$ = groups states together and simplify approximation
+ update then only occurs to that group ($\nabla \hat{v}(S_t, w_t)$ is 1 for that, 0 for rest)

Linear Methods

- special case: $\hat{v}(\cdot, w)$ is linear function
+ $\forall s \in S$ there exist vector $x(s) \doteq \{x_1(s), x_2(s), \dots, x_d(s)\}^T$ with the same components of x
- $\hat{v}(s, w) \doteq w^T x(s) \doteq \sum_{i=1}^{d} w_i x_i(s)$
- linear in weights / linear
- x is a feature vector of s
+ $x_i : S \to \mathbb{R}$
× features are basis functions, form linear basis for approximate functions
- $\nabla \hat{v}(s, w) = x(s)$
+ linear SGD update reduces to:

$$w_{t+1} \doteq w_t + \alpha\left[U_t - \hat{v}(S_t, w_t)\right] x(S_t)$$

- extremely simple, effective, favorable
+ esp if one optimum, guaranteed convergence
- semi-gradient TD(0) also converges with linear approximation, but not as simple
+ convergence on local optimum with each update

$$w_{t+1} = w_t + \alpha\left(R_{t+1} + \gamma w_t^T x_{t+1} - w_t^T x_t\right) x_t$$
$$= w_t + \alpha\left(R_{t+1} x_t - x_t (x_t - \gamma x_{t+1})^T w_t\right)$$

$x_t = x(S_t)$ shorthand

+ once in steady state:

$$\mathbb{E}[w_{t+1} | w_t] = w_t + \alpha(b - A w_t)$$
$$b \doteq \mathbb{E}[R_{t+1} x_t] \in \mathbb{R}^d$$
$$A \doteq \mathbb{E}\left[x_t (x_t - \gamma x_{t+1})^T\right] \in \mathbb{R}^{d \times d}$$

- when converging, converge to weight vector w_{TD}

$$b - A w_{TD} = 0$$
$$b = A w_{TD}$$
$$w_{TD} \doteq A^{-1} b$$

- TD fixed point
- in continuing case, $\overline{VE}$ bounded by,

$$\overline{VE}(w_{TD}) \le \frac{1}{1 - \gamma} \min_w \overline{VE}(w)$$

+ remember, $\gamma \approx 1$, so expansion of error can be quite large
+ however, TD methods reduce variance a lot compared to MC
- analogous bounds exist for on-policy bootstrapping methods
+ linear semi-gradient DP converge to TD fixed point
× One step semi gradient action value methods (like Sarsa(0)) find their own fixed point
- MUST use on-policy distribution for updates
+ else bootstrap with function approximation may diverge → ∞
- semi gradient n-step TD is easy extension of tabular n-step TD

$$w_{t+n} \doteq w_{t+n-1} + \alpha\left[G_{t:t+n} - \hat{v}(S_t, w_{t+n-1})\right]\nabla \hat{v}(S_t, w_{t+n-1})$$

Feature Construction for Linear Methods

- linear methods are cool
+ convergence guarantees
+ efficient data & computation
- choosing features is critical for RL tasks
+ both with a priori domain knowledge & generalization

Polynomials

- number expressed states, like # of cars in a lot or velocity & position are ~ ~ to approximation is interpolation & regression
+ not always great for other RL problems, still a good, simple, transition
- Suppose $S = s_1 \in \mathbb{R}, s_2 \in \mathbb{R}$
+ Easy tabular form $S = (s_1, s_2)^T$
× does not account for interaction
× both $s_1, s_2 = 0$ makes for lousy value
+ instead, 4 dimension feature vector $x(s) = (1, s_1, s_2, s_1 s_2)^T$
× 1 allows for affine functions in s_1, s_2
+ could even go higher in dimensionality $s_1^2, s_2^2, s_1 s_2^2$
× add quadratic interactions
+ all still linear in weights
- Generally:
  if $S = s_1, s_2, s_3, \dots, s_k$, $s_i \in \mathbb{R}$ then polynomial basis feature x_i is

$$x_i(s) = \prod_{j=1}^{k} s_j^{c_{i,j}}, \quad c_{i,j} \in \{0, 1, \dots, n\}$$

These features make up order-n polynomial basis for dimension k which contains $(n+1)^k$ diff features
- PLUS: allows for more accurate approximation of complicated functions
- NEG: feature explosion, must choose smaller subset

EXERCISE 9.2: Because $(n+1)^k$ is the outcome of applying 9.17 for a certain n, k

EXERCISE 9.3: $k = 2 \Rightarrow s_1, s_2$
$(n+1)^2 = 9$, $n = 2$

$$x_i(s) = s_1^{c_{i,1}} s_2^{c_{i,2}}$$

$c_{i,1}$	$c_{i,2}$	$x_i(s)$
0	0	1
1	0	s_1
0	1	s_2
1	1	$s_1 s_2$
2	0	s_1^2
0	2	s_2^2
1	2	$s_1 s_2^2$
2	1	$s_1^2 s_2$
2	2	$s_1^2 s_2^2$

Fourier Basis

- periodic function, weighted sum of sine & cosine basis functions (features) of diff freqs.
+ f is periodic if $f(x) = f(x + \tau) \ \forall x$ for some period τ
- Fourier series / transforms widely used when function to be approximated is known
+ can exactly with formulas, and given enough basis functions, any function can be approximated
+ RL, function is unknown, still useful?
- approx. aperiodic function over interval = linear combos of sines/cosines limited to τ set to length of interval
+ or, just cosine (since sine = cosine shifted) if τ is twice interval of interest, that approximation is limited to $[0, \tau/2]$
× could do half sine too, but discontinuous about origin structures
- now, $\tau = 2$, interval of $[0, 1]$, order-n Fourier-cosine basis has n+1 features

$$x_i(s) = \cos(i \pi s) \quad s \in [0, 1]$$
$$i = 0, 1, \dots, n$$

- now, multi-dimensional case:

$$S = (s_1, s_2, s_3, \dots, s_k)^T$$
$$s_i \in [0, 1]$$

then with Fourier basis

$$x_i(s) = \cos(\pi s^T c^i) \quad c^i = (c_1^i, c_2^i, \dots, c_k^i)^T$$
$$j = 1, \dots, k \quad i = 0, \dots, (n+1)^k \quad c_j^i \in \{0, \dots, n\}$$

define feature for each $(n+1)^k$ integer vectors c^i
$s^T c^i$ assigns an integer $\{0 \dots n\}$ to each s
can be shifted & scaled as application demands

e.g. $k = 2$, $S = (s_1, s_2)^T$
then $c^i = (c_1^i, c_2^i)^T$

$c_j^i = 0$ means constant in that state's dimension
$c_j^i \neq 0$ means varies over that with frequency ~ c_j^i
if multiple ≠ 0, state dim interaction determine frequency and ratio of interaction

- suggestion with Fourier series: alter α per feature
+ $\alpha_i = \frac{\alpha}{\sqrt{(c_1^i)^2 + \dots + (c_k^i)^2}}$ if $c_j^i = 0 \ \forall j$ then $\alpha_i = \alpha$
- features of order-n Fourier grows exponentially with states (k)
+ $k \le 5$, all $(n+1)^k$ can be used relatively easily
+ higher dims, must filter out features that are evidently high frequency "noise"
× unfortunately, very good at representing a state's global properties, but not local ones
- do not recommend polynomials for online learning

Coarse Coding

- state representation focusing on features as receptive fields that a state is captured in
+ e.g. 2 dim continuous state represented by overlapping "circles" for which features are on or off
× in this case, binary
- bigger receptive field, more possible generalization
+ fortunately, acuity not lost because final factor in precise learning is # of receptive fields, i.e. n

Tile Coding

- form of coarse coding for multi-dim, continuous spaces
+ flexible, computationally efficient
× extremely practical in the modern age
- use partitions or "tilings"
+ each tiling has square tiles non-overlapping
+ tilings are shifted to create overlaps
- x(s) has a feature for each tile in each tiling
+ 4 tilings × (4×4 tiles) = 64 feature components
- # of features "active" per one state is the same
+ makes α more intuitive: $\alpha = \frac{1}{n}$ for one trial learning
× for $s \Rightarrow v$ training example, $\hat{v}(s, w_t)$ doesn't matter, $\hat{v}(s, w_{t+1}) = v$ if $\alpha = \frac{1}{n}$
× if $\alpha = \frac{1}{10n}$, will be $\frac{1}{10}$th of value target
- prefer asymmetric shift in tilings to better generalization and avoid
- if w = width of tile, n = # of tilings, $\frac{w}{n}$ is a unit and unit representing how shift tiles in uniform sense
- asymmetric displacement should be displaced by first odd integers in $(1, 3, 5, \dots, 2k-1)$ for state space of k dims with n tilings $2^n \ge 4k$
e.g. $k = 2$, $n = 3$ displace = (1, 3)
e.g. $k = 3$, $n = 4$ displace = (1, 3, 5)
so positions are (0, 0, 0), (1, 3, 5), (2, 6, 10), (3, 9, 15)
- how to pick tile shape?
+ must include different styles for full coverage
+ reduce memory footprint using hashing

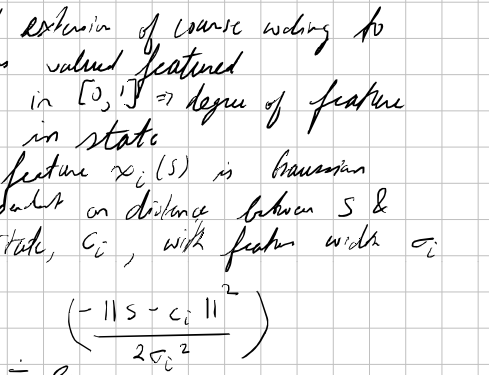
EXERCISE 9.4: Would want large many overlapping rectangles in the important dim, and few rectangles in less important dim

Radial Basis Functions

- natural extension of coarse coding to continuous valued features
+ anything in $[0, 1] \Rightarrow$ degree of feature presence in state
- RBF feature $x_i(s)$ is Gaussian and depends on distance between s & center state, c_i, with feature width σ_i

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

- choose σ with whatever appropriate distance metric



- although always differentiable, not actually practical
+ computationally complex
× can move into nonlinear RBF
- called RBF networks, linear approximation using RBF

Selecting Step Size Parameters Manually

- how to pick??
- theory doesn't help much
- $\alpha = 1$ means total acceptance of update to target
+ usual choice is to learn more slowly
+ use $\alpha = \frac{1}{10}, \frac{1}{100}, \frac{1}{\tau}$ for slower learning
- does not easily apply to function approximation
+ for linear approximators, similar rule can be created
× linear SGD:

$$\alpha \doteq \frac{1}{\tau \mathbb{E}[x^T x]}$$

for learning substantially from the same feature vector in τ experiences
× x must be a feature vector that is in the distribution of input vectors
× ideally $x^T x$ is constant

EXERCISE 9.5:

$$\alpha = \frac{1}{10(98)} = \frac{1}{980} \approx 0.001$$

EXERCISE 9.6:

① $w_{t+1} \doteq w_t + \alpha\left[U_t - \hat{v}(S_t, w_t)\right]\nabla \hat{v}(S_t, w_t)$

$$x(S_t)^T x(S_t) = \mathbb{E}[x^T x]$$

for linear SGD $\hat{v}(S_t, w_t) = w_t^T x(S_t)$
$$\nabla \hat{v}(S_t, w_t) = x(S_t)$$

① $w_t + \frac{1}{x(S_t)^T x(S_t)}\left[U_t - w_t^T x(S_t)\right] x(S_t)$

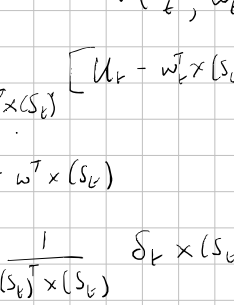
$$\delta_t = U_t - w^T x(S_t)$$
$$w_t + \frac{1}{x(S_t)^T x(S_t)} \delta_t x(S_t)$$

after one update

$$\hat{v}(S_t, w_{t+1}) = w_{t+1}^T x(S_t)$$
$$= \left[w_t + \frac{1}{x(S_t)^T x(S_t)} \delta_t x(S_t)\right]^T x(S_t)$$
$$w_t x(S_t) + \frac{1}{\cancel{x(S_t)^T x(S_t)}} \delta_t \cancel{x(S_t)^T x(S_t)}$$
$$\boxed{w_t x(S_t) + \delta_t}$$

Nonlinear Function Approximation Artificial Neural Networks

- deep learning advancements
- layered: in ⇒ layers ⇒ out
- feed forward unless loop in connection, then recurrent



- inner units compute weighted sum of inputs and feed to activation function
+ typically S shaped, sigmoid activations or simple rectified linear units
- ANN with large enough 2nd layer can approx. any continuous function
+ must have non linear activation for this
- use multiple layers for better abstract reasoning
- ANN natural at automatically learning features rather than specifying them
+ use loss function to train
+ for RL, TD error learns value function or maximize reward like in gradient
- use backpropagation to train weights
- however, decay or explosion of gradient occurs in deep ANN
+ dropout to avoid overfitting
+ batch normalization to improve gradient
+ deep residual learning via residual connection
- Deep CNN do not need to rely on ? due to spatial arrays & architecture
- most RL still uses linear approx methods, but thoughts of nonlinear function approx make strides everywhere

Least Squares TD

- can we do better with more computation
- linear function approx for TD(0) converges to TD fixed point

$$w_{TD} = A^{-1} b$$
$$A \doteq \mathbb{E}\left[x_t (x_t - \gamma x_{t+1})^T\right]$$
$$b \doteq \mathbb{E}[R_{t+1} x_t]$$

- can compute estimate immediately instead of iteratively
- $\boxed{\text{LSTD}}$

$$\hat{A}_t \doteq \sum_{k=0}^{t-1} x_k (x_k - \gamma x_{k+1})^T + \varepsilon I$$
$$\hat{b}_t \doteq \sum_{k=0}^{t-1} R_{k+1} x_k$$

I → identity matrix
- hence $\hat{w}_t \doteq \hat{A}_t^{-1} \hat{b}_t$
- LSTD complexity requires $O(d^2)$ computation due to outer product (matrix update)
+ space also $O(d^2)$
+ inverse even more complex ($O(d^3)$) but specialty of $\hat{A}$ allows for incremental computation

$$\hat{A}_t^{-1} = \left(\hat{A}_{t-1} + x_{t-1}(x_{t-1} - \gamma x_t)^T\right)^{-1}$$
$$= \hat{A}_{t-1}^{-1} - \frac{\hat{A}_{t-1}^{-1} x_{t-1} (x_{t-1} - \gamma x_t)^T \hat{A}_{t-1}^{-1}}{1 + (x_{t-1} - \gamma x_t)^T \hat{A}_{t-1}^{-1} x_{t-1}} \quad ①$$
$$t > 0, \ \hat{A}_0 \doteq \varepsilon I$$

① Sherman-Morrison formula

- is LSTD better though?
+ no α, so everything is remembered, never forgets
× not great for RL, specifically GPI
× usually must combine LSTD with a forgetting mechanism, making no α moot
+ still need ε

LSTD for est. $\hat{v} = w^T x(\cdot) \approx v_\pi$ ($O(d^2)$)

```
input: feature rep. $x: S^+ \to \mathbb{R}^d$, $x(\text{term}) = 0$
param: $\varepsilon > 0$
$\widehat{A^{-1}} \leftarrow \varepsilon^{-1} I$        d×d matrix
$\hat{b} \leftarrow 0$        d-dim vector
for each episode
    initialize S
    x ← x(S)
    for each step of episode:
        A ~ $\pi(\cdot|S)$
        R, S' ← env(S, A)
        x' ← x(S')
        $v \leftarrow \widehat{A^{-1}}^T (x - \gamma x')$
        $\widehat{A^{-1}} \leftarrow \widehat{A^{-1}} - (\widehat{A^{-1}} x) v^T / (1 + v^T x)$
        $\hat{b} \leftarrow \hat{b} + Rx$
        $w \leftarrow \widehat{A^{-1}} \hat{b}$
        S ← S'
        x ← x'
    until S' is terminal
```

Memory Based Function Approximation

- current algos are parametric to approx value functions
+ each training example $s \mapsto g$ changes function params to reduce error
- every state is evaluated at the current parameters
- memory-based doesn't update params
+ saves examples for state, then queries & evaluates values from those examples
+ $\boxed{\text{lazy learning}}$
× technically learn any function and is not limited in form
- uses local neighborhood of queried states
+ nearest neighbor example:
× s is query state, $s' \mapsto g$ is example in memory closest to s, g is value of s
+ weighted avg similar, but obviously by distance
+ locally weighted regression fits a parametric equation
- adv: local, flexible form
- adv: similar to trajectory sampling
- adv: immediate effect
- avoids curse of dimensionality
- disadv: space/time for lookup/calc
+ solved with special hardware, parallel comp, k-d trees (special data structure)

Kernel-Based Function Approximation

- above weighted avg / locally weighted regression assign weights to s' near $s \mapsto g$
- $\boxed{\text{Kernel function}}$ assigns those weights
+ e.g. $k: \mathbb{R} \to \mathbb{R}$ for distance to weight
+ or just some similarity index
× $k: S \times S \to \mathbb{R}$ with $k(s, s')$
- kernel func describes strength of relevance between states
+ e.g. tile coding generalizes as kernel func
- $\boxed{\text{Kernel regression}}$: how to compute
+ D is stored examples $g(s')$ is target for s' then kernel function approximates target function

$$\hat{v}(s, D) = \sum_{s' \in D} k(s, s') g(s')$$

- RBF with kernel function used stored examples for center dist and nonparametric
- every single linear parametric function is really a kernel regression

$$k(s, s') = x(s)^T x(s)$$

- for less computation than linear parametric forms, very useful

Looking Deeper at On-Policy Learning: Interest & Emphasis

- algos update to on-policy dist.
- expand "on-policy dist." def
+ many on-policy dist. differing based on how the trajectory is initiated
- $I_t \Rightarrow$ degree of interest in accurately valuing S_t or (S_t, a_t) ≥ 0
+ set in any way
- μ is the distribution of $\overline{VE}$ defined as the dist of states weighted by interest I_t
+ M_t, emphasis of update at $t \ge 0$

$$w_{t+n} \doteq w_{t+n-1} + \alpha M_t\left[G_{t:t+n} - \hat{v}(S_t, w_{t+n-1})\right] \cdot \nabla \hat{v}(S_t, w_{t+n-1})$$

for n-step learning

$$M_t = I_t + \gamma^n M_{t-n} \quad M_t = 0 \ \ t < 0$$

- still works for MC, $G_{t:t+n} = G_t$
  $n = T - t$, $M_t = I_t$

Summary

- generalization is key in AI/engineering applications
+ use supervised learning function approximation
- best suited are often parameterized function approximation
× policy function uses weight vector w
× $\overline{VE}(w)$ measures error of $v_{\pi_w}(s)$ for on policy dist μ
* $\overline{VE}$ ranks diff. value functions
- SGD is favorable w finder
+ n-step semi gradient TD very applicable
× bootstrap means gradient doesn't use w even though update does = semi-gradient
- works well for linear function approximation (in weights)
+ feature selection is the problem
× several ways, pros and cons for each
* polynomials, coarse coding, Fourier basis, LSTD
* nonlinear, ANN!, in deep reinforcement learning
- linear, semi-gradient n-step TD converges

EXERCISE 9.7:

semi-linear unit: weighted sum with sigmoid:

$$\hat{v}(x, w) = \frac{1}{1 + e^{-w^T x}}$$

good for binary outcomes in games

semi gradient TD(0):

$$w_{t+1} = w_t + \alpha \delta_t \nabla \hat{v}(S_t, w_t)$$
$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) - \hat{v}(S_t, w_t)$$

in supervised learning, update becomes

$$\delta_t = U_t - \hat{v}(x_t, w_t)$$

MSE of loss $= \frac{1}{2}(U_t - \hat{v}(x, w))^2 = \mathcal{L}$

$$z = w^T x \Rightarrow \frac{1}{1 - e^{-z}}$$
$$\nabla_w \hat{v}(z) = \frac{\partial v}{\partial z} = \hat{v}(1 - \hat{v})$$
$$\frac{\partial \mathcal{L}}{\partial w_i} = (\hat{v} - U_t) \cdot \hat{v}(1 - \hat{v}) \cdot x_i$$
$$w_i = w_i - \alpha(\hat{v} - U_t)\hat{v}(1 - \hat{v}) x_i$$

make it a + update

$$w_i = w_i + \alpha(U_t - \hat{v})\hat{v}(1 - \hat{v}) x_i$$