Chapter 9

approximating v_π state value using on-policy data

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

↳ 9.1 Value Func Approx

all prediction methods are "backups"

↳ updates from value at state to backed up value

$s \rightarrow g$

↑state ↑backed up value

Monte Carlo $S_t \rightarrow G_t$

TD(0) $S_t \rightarrow R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t)$

DP $s \rightarrow E_\pi[R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) | S_t = s]$

↳ backup means that estimated value of s should be more like G

↳ Now allow for arbitrarily complex methods to perform the backup s.t. the estimated values of other states are changed as well.

↳ Machines mimic input-output situations known as supervised learning. When outputs are numbers like g, this is function approximation

View each backup as conventional training example allows for function approximation to give us values.

↳ This includes artificial neural networks, decision trees, other kinds of multivariate regression.

↳ not all of these function approximations are suited for RL

↳ Many functions assume static data set

↳ RL requires Online, real time approximations

↳ non stationarity coming from bootstrapping methods, DP & TD

9.2 Prediction Objective (MSVE)

More states then weights

↳ making one ~~states~~ state's estimate more accurate means making another state's less accurate

Mean Square Value Error

↳ Must specify what states we care most about using weighting distribution $\mu(s) \geq 0$
(how much we care about error in state s)

$$MSVE(w) = \sum_{s \in S} \mu(s) \left[v_\pi(s) - \hat{v}(s,w) \right]^2$$

↳ Objective function is MSVE. Root RMSVE gives how ~~the~~ much the approximate values differ.

↳ $\mu(s)$ is typically chosen based on time spent in state under policy π → on policy distribution

ON POLICY DISTRIBUTION IN EPISODIC TASKS

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

$h(s)$ is probability that an episode ~~of~~ begins in s

Global Optimum
↳ Ideal goal in MSVE is to find w^* (~~a~~ weight vector) s.t. $MSVE(w^*) \leq MSVE(w)$ for all w.

↳ possible to determine w for linear function approx difficult for higher order

Local Optimum
↳ For higher order we consider ~~to~~ ~~w^*~~ a w in the neighborhood of w^* to be optimal

9.3 Stochastic-Gradient and Semi-Gradient Methods

SGD are among most used value function approx methods.

In gradient descent:

$w = [w_1, \dots, w_d]^T$

$\hat{v}(s,w)$ is ~~differential~~ differentiable function of w

update w for each discrete time step

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

$$= w_t + \alpha \left[v_\pi(S_t) - \hat{v}(S_t, w_t) \right] \nabla \hat{v}(S_t, w_t)$$

$$\nabla f(w) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \ldots, \frac{\partial f(w)}{\partial w_n} \right)^T$$

In the case where $S_t \to U_t$ is not the true value $V_\pi(t)$ (off as a result of noise of some sort)

plug in U_T

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

Gradient Monte Carlo Algorithm

Input Policy
Input differentiable functions $v(s, w)$

Initialize weights $(w = 0)$
While True:
  Generate episode using π
  For $t = 0, \ldots, T-1$
   $w = w + \alpha [G_T - \hat{v}(S_t, w)] \nabla \hat{v}(S_t, w)$

Bootstrapping targets do not produce true gradient descents and are therefore known as semi-gradient methods.

Benefits of these methods are that they converge more quickly and online (don't have to wait until EOE)

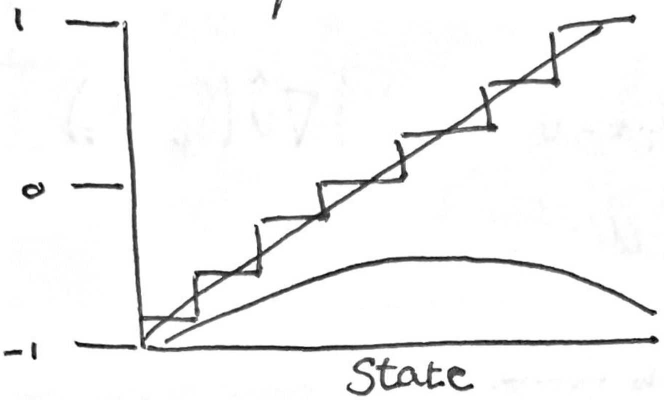
TD(0) Semi-Gradient Descent for Estimating $\hat{v} \approx v_\pi$

Input π
Input differentiable function $\hat{v}$

Initial value function weights w arbitrarily $(w = 0)$
Repeat for each episode:
  Initialize S
  Repeat (each step of episode)
   Choose $A \sim \pi(\cdot|S)$
   Take action A - observe R, S'
   $W \mathrel{+}= \alpha [R + \gamma \hat{v}(S', w) - \hat{v}(S, w)] \nabla \hat{v}(S, w)$
   $S = S'$
  until S' is terminal

Example 9.1 State Aggregation on the 1000-state Random Walk

State Aggregation is a special case of SGD where the Gradient $\nabla v(S_t, w_t)$ is 1 for S_t's group and 0 for all others.



9.4 Linear Methods

~~Dealing~~ Special case when $\hat{v}(\cdot, w)$ is linear

↳ For every state there is a vector $x = (x_1, \ldots, x_d)$ matching size of w.

↳ State value function given by

$$\hat{v}(s, w) = w^T x(s) = \sum_{i=1}^{d} w_i x_i(s)$$

basis functions

$\nabla \hat{v}(s, w) = x(s)$

Many useful convergence results are linear

Gradient Monte Carlo converges to the global optimum of the MSVE under linear function approximation.

TD fixed point : point at which linear semi-gradient TD(0) converges.

Also provable that ~~MV~~ MSVE is within the a bounded expansion of the lowest possible error

Example 9.2 Bootstrapping on the 1000 step random walk

n-step semi-gradient TD for estimating $\hat{v} \approx v_\pi$

9.5 Feature Construction for Linear Methods

↳ accounting for various features,

↳ some may be related to each other.

9.5.1 Polynomials

$$x_i(s) = \prod_{j=1}^{d} s_j^{c_{i,j}}$$

9.5.2 Fourier Basis

$$x_i(s) = \cos(\pi c^i \cdot s)$$

9.5.3 Course Coding

Example 9.3 Coarseness of course coding
include pics of circles

9.5.4 Tile Coding

Analogue between course coding and tiling is the use of multiples tiles (show pic)

Advantage: overall # of features active at one time is the same for ~~f~~ any state

Another advantage is binary components, either 0 or 1

Tiles can have shapes to promote generalization in a certain dimension

Hashing to reduce memory demands of higher dimensions.

9.5.5 Radial Basis Functions

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

↳ apparently useless

↳ done over binary ~~functions~~ features, produce approximate functions that are differentiable ~~function~~

"no practical" significance - also more computationally expensive

RBF network - linear function approximator using RBF's for its features.

9.6 NonLinear Function Approximation: Artificial Neural Network

Output - input - hidden layers

Units are typically semi-linear - they ~~comput~~ compute a weighted sum of ~~the~~ their ~~of~~ input signals then apply result to activation ~~fn~~ function

Activation functions are typically S-shaped or sigmoid

$$f(x) = \frac{1}{(1 + e^{-x})} \quad \text{or} \quad f(x) = \max(0, x)$$

ANN with hidden layer ~~can~~ having a large enough amount of sigmoid units can approximate any continuous function on a compact region of the network's input space

→ ANN ~~learn~~ learn typically via SGD

→ Back propagation algorithm for calculating partial derivates at each weight.

- ↳ performs well on shallow networks
  - worse for deeper network
- ↳ overfitting is a big concern because of the large number of weights
  - Method for reducing this is dropout method
    - ~~drop weights~~ units are rand- units are randomly removed. "Thinning the network" multiply by the probability that those units are removed.

- Deep Convolutional Network
  - ↳ Specializes in high dimensional data ~~like~~ like images in spatial arrays
  - ↳ Trained on back propagation without the need for improvement methods

Image: "Include Picture"
Alternating convolutional and subsampling layers
followed by several fully connected final layers

9.7 ...

9.7 Least Squares TD

TD Fixed Point was computed iteratively

↳ by estimating A & b, w_{TD} can be computed directly

↳ This algo is more data efficient, but less computationally efficient than semi-gradient TD(0)

LSTD for estimating $\hat{v} \approx v_\pi$

9.8 Memory-based Function Approximation

We have so far discussed parametric means of estimation

↳ lazy learning -

↳ local learning - retrieve set of training examples from memory ~~from~~ based on query state

↳ nearest neighbor method - returns g to s from nearest state in memory $s' \rightarrow g$; Can also include states s'', s'''... and perform weighted average

↳ locally weighted regression similar to weights

9.9 Kernel-based Function Approximations

Kernel function assigns weights to distances in local regressions - indirectly measure degree of generalization

Kernel regression

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

Gaussian Radial Basis Function

9.10 Looking Deeper at On-Policy Learning : Interest and Emphasis

We have treated all states as equally important in these algorithms

↳ updating according to on policy distribution

— distribution of states encountered while following target policy

↳ Introduce new random non negative variable I interest in valuing state

↳ Introduce M, emphasis, emphasizes or de-emphasizes learning done at time t