COS 511: Foundations of Machine Learning

Rob Schapire — Lecture # 24
Scribe: Forrest Collman — May 1, 2003

1 Markov Decision Processes

A Markov decision process consists of a string of states (S), and at each state there is an action (A) which causes a transition to occur to the next state and for a reward r to be given. A policy π is a mapping from S to A, i.e. an action to be taken for every state. We aim to find a policy which maximizes our reward.

The value of a policy, given that you start at a state s is

$$V^{\pi}(s) = E(r_{t+1} + r_{t+2} + ...|s_t = s) \tag{1}$$

It is more realistic to decay away the value of the reward

$$V^{\pi}(s) = E(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3}...|s_t = s) \tag{2}$$

Last time we reviewed policy reiteration, in which we used V^{π} to calculate a new π and proved convergence to the optimal policy π^*.

Given π how did we evaluate $V^{\pi}(s)$? All the information is there, but how do we get it by experimenting? We must approximate $V^{\pi}(s)$ (call it just V from now on). Note

$$V(s_t) = E(r_{t+1} + \gamma V(s_{t+1})|s_t) \tag{3}$$

The idea of the temporal difference approach is to initialize the value of V and then update it based upon the experience of the most recent reward. Namely to average the old value with the new one

$$\hat{V}' = (1-\alpha)\hat{V} + \alpha(r_{t+1} + \gamma\hat{V}(s_{t+1})) \tag{4}$$

$$\hat{V}' = \hat{V}(s_t) + \alpha\left(r_{t+1} + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)\right) \tag{5}$$

This is called TD(0) because it takes the difference in values of V from the estimate to one point in the future.

THEOREM: Assume all states visited infinite many times.

α_t depends on the # of times s_t has been visited ($n(s_t)$). if $\sum_n \alpha(n) = \infty$ but $\sum_n \alpha(n)^2 < \infty$ (e.g. $\alpha(n) = 1/n$) then

$$\max_s |\hat{V}(s) - V(s)| \to 0 \tag{6}$$

PROOF: define $s = s_t$, $s' = s_{t+1}$, $r = r_{t+1}$, and $n = n(s_t)$.

$$\hat{V}_{n+1}(s) = (1-\alpha)\hat{V}_n(s) + \alpha(r + \gamma\hat{V}_t(s')) \tag{7}$$

$$\bar{V}_{n+1}(s) = (1-\alpha)\bar{V}_n(s) + \alpha(E(r + \gamma\hat{V}_t(s')|s)) \tag{8}$$

Now we can use the triangle inequality to bound the thing we are interested by two parts.

$$|\hat{V}_{n+1}(s) - V(s)| \le |\hat{V}_{n+1}(s) - \bar{V}_{n+1}(s)| + |\bar{V}_{n+1}(s) - V(s)| \tag{9}$$

For the first part (call it D_n)

$$\begin{aligned} |\hat{V}_{n+1}(s) - \bar{V}_{n+1}(s)| &= D_{n+1} \\ = (1-\alpha_n)(\hat{V}_n(s) - \bar{V}_n(s)) + \alpha_n(r + \gamma\hat{V}_n(s') &- E(r + \gamma\hat{V}_n(s')|s)) \end{aligned} \tag{10}$$

define the part that is multiplied by α_n as x_n, and

$$x_n = r + \gamma\hat{V}_n(s') - E(r + \gamma\hat{V}_n(s')|s) \tag{11}$$

We will show that D_n is a convex combination of $x_1, \ldots, x_{n-1}$ and that $E(x_n|x_1, \ldots, x_{n-1}) = 0$, thus showing that the D_n forms a "martingale", and by Azuma's Lemma converges to zero. So with our new definitions

$$D_{n+1} = (1-\alpha_n)D_{n-1} + \alpha_n x_n \tag{12}$$

We will prove that D_n is a convex combination of $x_1 ... x_{n-1}$ by induction. We must initialize $\hat{V}_1(s)$ and $\bar{V}_1(s)$ to 0, and constrain $\alpha_1 = 1$, so that

$$D_2 = (1-\alpha_1)0 + 1x_1 = x_1 \tag{13}$$

Now assuming by induction we assume we can write $D_n = \sum_{i=1}^{n-1} \beta_i x_i$ where $\beta_i > 0\ \forall i$ and $\sum_{i=1}^{n-1} \beta_i = 1$ and show that D_{n+1} is a convex linear combination of $x_n ... x_1$.

$$D_{n+1} = (1-\alpha_n)\sum_{i=1}^{n-1} \beta_i x_i + \alpha_n x_n \tag{14}$$

Define $\beta'_i = (1-\alpha_n)\beta_i\ \forall i < n$ and $\beta'_n = \alpha_n$. Thus $D_{n+1} = \sum_{i=1}^{n} \beta'_i x_i$ and

$$\sum_{i=1}^{n} \beta'_i = \left(\sum_{i=1}^{n-1}(1-\alpha_n)\beta_i\right) + \alpha_n = (1-\alpha_n) + \alpha_n = 1 \tag{15}$$

So we have shown that D_n is a convex combination of $x_1 ... x_{n-1}$ for all n.

$E(x_n|x_1, \ldots, x_{n-1}) = 0\ \forall n$ for

$$E(x_n|s) = E(r + \gamma\hat{V}_n(s') - E(r + \gamma\hat{V}_n(s')|s)|s) = E(0) = 0 \tag{16}$$

So D_n forms a martingale, and by Azuma's Lemma (a parallel to Hoeffding's Inequality) it converges to zero.

For the second part

$$|\bar{V}_{t+1}(s) - V(s)| = (1-\alpha)(\bar{V}_t - V(s)) + \alpha(E(r + \gamma\hat{V}_t(s') - (r + \gamma V(s'))|s)) \tag{17}$$

$$\leq (1-\alpha)(\bar{V}_t - V(s)) + \gamma\alpha\Delta \tag{18}$$

if $\Delta \geq \hat{V}_t(s') - V(s')$. So this converges to $\gamma\Delta$. Since $\gamma < 1$ this is smaller than Δ and we can thus redefine $\Delta' = \gamma\Delta$ and reapply the argument over and over again, thus converging the result to zero.

So we have shown both parts go to zero, thus the entire thing must go to zero.

2 TD(λ)

Now in some cases TD(0) is highly inefficient because information about rewards only propagates backward one state at each time step. We can however expand our approximation ($\hat{V}_t(s)$) farther out into the future.

If we define

$$R_t^k = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} + \gamma^k \hat{V}(s_{t+k}) \tag{19}$$

Then our update rule becomes more generally (in parallel to Equation 5)

$$\hat{V}_{n+1}(s) \mathrel{+}= \alpha_n (R_t^k - \hat{V}_n(s)) \tag{20}$$

this would be TD(k). However, this algorithm would require running the process k times into the future before updating. TD(λ) combines all of the TD(k) methods, decaying the weight as k increases

$$R_t^\lambda = (R_t^0 + \lambda R_t^1 + \lambda^2 R_t^2 + \cdots)(1 - \lambda) \tag{21}$$

Our update rule has the same form

$$\hat{V}_{n+1}(s) \mathrel{+}= \alpha_n (R_t^\lambda - \hat{V}_n(s)) \tag{22}$$

let $\hat{V}(s_t) = v_t$ and write out $R_t^\lambda - \hat{V}_n(s)$

$$-v_t + (1 - \lambda)(R_t^1 + \lambda R_t^2 + \lambda^2 R_t^3 ...) \tag{23}$$

$$\begin{aligned} = -v_t + (1-\lambda)(r_{t+1} + \gamma v_{t+1}) \\ +(1-\lambda)\lambda(r_{t+1} + \gamma r_{t+2} + \gamma^2 v_{t+2}) \\ +(1-\lambda)\lambda^2(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 v_{t+3}) \\ ... \end{aligned} \tag{24}$$

Now group the terms by r_{t+k} (doing the infinite sum) and leave the v_{t+k} terms separate

$$\begin{aligned} -v_t \\ +r_{t+1} + \gamma\lambda r_{t+2} + (\gamma\lambda)^2 r_{t+3} ... \\ +(1-\lambda)(\gamma v_{t+1} + \lambda\gamma^2 v_{t+2} + \lambda^2\gamma^3 v_{t+3} + ... \end{aligned} \tag{25}$$

Now distribute $(1 - \lambda)$ across the last set of terms and separate them into two groups, bringing in the lone v_t

$$\begin{aligned} r_{t+1} + \gamma\lambda r_{t+2} + (\gamma\lambda)^2 r_{t+3} ... \\ +\gamma(v_{t+1} + \lambda\gamma v_{t+2} + (\lambda\gamma)^2 v_{t+3} + .. \\ -(v_t + \lambda\gamma v_{t+1} + (\lambda\gamma)^2 v_{t+2} + ... \end{aligned} \tag{26}$$

Now group the terms with the same power of $\lambda\gamma$

$$\begin{aligned} r_{t+1} + \gamma v_{t+1} - v_t \\ +\lambda\gamma(r_{t+2} + \gamma v_{t+2} - v_{t+1}) \\ +(\lambda\gamma)^2(r_{t+3} + \gamma v_{t+3} - v_{t+2}) \\ ... \end{aligned} \tag{27}$$

So defining $\delta_t = r_{t+1} + \gamma v_{t+1} - v_t$ the update rule is thus

$$\hat{V}(s) \mathrel{+}= \alpha(\delta_0 + \gamma\lambda\delta_1 + (\gamma\lambda)^2\delta_2 + ...) \tag{28}$$

Now ideally the updates would go like this

$$\hat{V}(s_0) \mathrel{+}= \alpha(\delta_0 + (\gamma\lambda)\delta_1 + ...) \tag{29}$$
$$\hat{V}(s_1) \mathrel{+}= \alpha(\delta_1 + (\gamma\lambda)\delta_2 + ...) \tag{30}$$

but if this were the case you would never update because you would never have all the terms to update the first term. So, instead, do the update with the information that you have after every transition.

$$\hat{V}(s_0) \mathrel{+}= \alpha\delta_0 \tag{31}$$
$$-- \tag{32}$$
$$\hat{V}(s_1) \mathrel{+}= \alpha\delta_1 \tag{33}$$
$$\hat{V}(s_0) \mathrel{+}= \alpha\gamma\lambda\delta_1 \tag{34}$$
$$-- \tag{35}$$
$$\hat{V}(s_2) \mathrel{+}= \alpha\delta_2 \tag{36}$$
$$\hat{V}(s_1) \mathrel{+}= \alpha\gamma\lambda\delta_2 \tag{37}$$
$$\hat{V}(s_0) \mathrel{+}= \alpha(\gamma\lambda)^2\delta_2 \tag{38}$$

So after each action and reward you use the information to propagate backward to all the states that came before it.

3 Large State Space

If the number of states is too large for these methods to be effective, neural networks are often used to approximate the value function. If W is the weights of the neural network then we could write $\hat{V}(s) = f(s, W)$. Our update rule

$$\hat{V}(s) \mathrel{+}= \alpha(r + \gamma\hat{V}(s') - \hat{V}(s)) \tag{39}$$

can be framed as performing gradient descent with $dt = \alpha$ on the function

$$\frac{1}{2}\left(r + \gamma\hat{V}(s') - \hat{V}(s)\right)^2 \tag{40}$$

so now considering W we can also perform gradient descent to update W

$$W \mathrel{+}= \alpha(r + \gamma f(s', W) - f(s, W)) \cdot \nabla_W f(s, W) \tag{41}$$

4 TD-gammon

An example application of TD(λ) is backgammon. All the pieces are on the board, so the state of the game is known to both competitors. The rewards can be defined as 1 if black wins -1 if white wins and 0 otherwise. The program chooses the action which maximizes $r(s, a) + \hat{V}(\delta(s, a))$ Where $\delta(s, a)$ is the state of the game reached after taking the action a. Then TD(λ) is used to update $\hat{V}$, encoding $\hat{V}$ in a neural network.

Early versions of the programs played against itself, used an obvious encoding of the board state, and played 300,000 times against itself. This was as good as an above average human.

Later versions, increased the number of games played to 1.5 million, added more specialized features to the encoding of the board state, and chose actions by looking farther into the future. These improvements made it very nearly as good as the best human players.