Reinforcement learning

Nathaniel Daw
Gatsby Computational Neuroscience Unit
daw @ gatsby.ucl.ac.uk
http://www.gatsby.ucl.ac.uk/~daw

Mostly adapted from Andrew Moore's tutorials, copyright 2002, 2004 by Andrew Moore. His originals, and many more tutorials are available at: http://www.cs.cmu.edu/~awm/tutorials

The problem

Decision-making in a situation that may be:

1. Sequential
   (like chess, a maze)
2. Stochastic
   (like backgammon, the stock market)

The plan

We discuss:

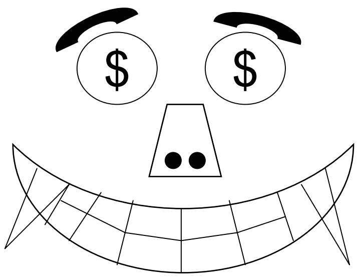
1. How to evaluate long-term payoffs
   - Markov systems
2. How to find optimal decisions
   - Markov decision processes, dynamic programming
3. How to learn these on the fly
   - Reinforcement learning, “semi-supervised” learning
4. Extensions

Discounted rewards

An assistant professor gets paid, say, 20K per year.

How much, in total, will the A.P. earn in their life?

20 + 20 + 20 + 20 + 20 + … = Infinity



What’s wrong with this argument?

Discounted rewards

"A reward (payment) in the future is not worth quite as much as a reward now."

– Because of chance of obliteration

– Because of inflation

Example:

Being promised \$10,000 next year is worth only 90% as much as receiving \$10,000 right now.

Assuming payment *n* years in future is worth only $(0.9)^n$ of payment now, what is the AP's Future Discounted Sum of Rewards ?

Discount factors

People in economics and probabilistic decision-making do this all the time.

The "Discounted sum of future rewards" using discount factor γ is

(reward now) +

γ (reward in 1 time step) +

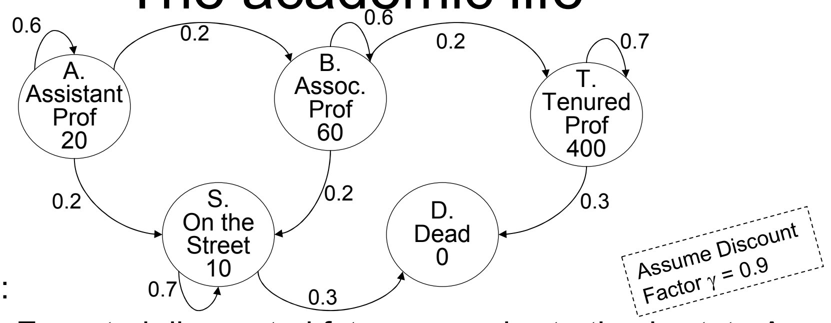
γ^2 (reward in 2 time steps) +

γ^3 (reward in 3 time steps) +

… (infinite sum)

$$= E\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r(\mathbf{s}_t)\right]$$

The academic life



Define:

V_A = Expected discounted future rewards starting in state A

V_B = Expected discounted future rewards starting in state B

V_T = " " " " " " " T

V_S = " " " " " " " S

V_D = " " " " " " " D

How do we compute V_A, V_B, V_T, V_S, V_D ?

A Markov system with rewards

- Has a set of states $\{s_1\ s_2 \cdot\cdot\ s_N\}$
- Has a transition probability matrix:

$$T = \begin{pmatrix} T_{11}\ T_{12} \cdot\cdot\ T_{1N} \\ T_{21} \\ : \\ T_{N1} \quad \cdot\cdot \quad T_{NN} \end{pmatrix} \qquad T_{ij} = \text{Prob}(\text{next state } \mathbf{s}_{t+1} = s_j \mid \text{this state } \mathbf{s}_t = s_i)$$

- Each state has a reward. $\{r_1\ r_2 \cdot\cdot\ r_N\}$
- There's a discount factor γ . $0 < \gamma < 1$

On Each Time Step …

0. Assume your state is $\mathbf{s}_t$
1. You get given reward $r(\mathbf{s}_t)$
2. You randomly move to another state $P(\mathbf{s}_{t+1} = s_j \mid \mathbf{s}_t = s_i) = T_{ij}$
3. All future rewards are discounted by γ

Solving a Markov system

Write $V(\mathbf{s}_t)$ = expected discounted sum of future rewards starting in state $\mathbf{s}_t = s_i$

$$V(\mathbf{s}_t) = r(\mathbf{s}_t) + E\,[\,\gamma\, r(\mathbf{s}_{t+1}) + \gamma^2\, r(\mathbf{s}_{t+1}) + \ldots\,]$$

$$= r(\mathbf{s}_t) + \gamma \cdot \text{(Expected future rewards starting from next state)}$$

$$= r(\mathbf{s}_t) + \gamma \cdot (T_{i1}V(s_1) + T_{i2}V(s_2) + \cdots\, T_{iN}V(s_N))$$

Using vector notation write:

$$V = \begin{pmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_N) \end{pmatrix} \qquad R = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{pmatrix} \qquad T = \begin{pmatrix} T_{11} & T_{12} & \cdots & T_{1N} \\ T_{21} & \ddots & & \\ \vdots & & & \\ T_{N1} & T_{N2} & \cdots & T_{NN} \end{pmatrix}$$

Question: can you invent a closed form expression for V in terms of R, T, and γ ?

Solving a Markov system with matrix inversion

- Upside: You get an exact answer
- Downside:

Solving a Markov system with matrix inversion

- Upside: You get an exact answer
- Downside: If you have 100,000 states you're solving a 100,000 by 100,000 system of equations.

Value iteration: another way to solve a Markov system

Let $\mathbf{s}_t = s_i$. Define:

$V^1(s_i)$ = Expected discounted sum of rewards over the next 1 time step.

$V^2(s_i)$ = Expected discounted sum rewards during next 2 steps

$V^3(s_i)$ = Expected discounted sum rewards during next 3 steps

:

$V^k(s_i)$ = Expected discounted sum rewards during next *k* steps

$V^1(s_i)$ =	(what?)
$V^2(s_i)$ = …	(what?)
$V^{k+1}(s_i)$ =	(what?)

Value iteration: another way to solve a Markov system

Let $\mathbf{s}_t = s_i$. Define:

$V^1(s_i)$ = Expected discounted sum of rewards over the next 1 time step.

$V^2(s_i)$ = Expected discounted sum rewards during next 2 steps

$V^3(s_i)$ = Expected discounted sum rewards during next 3 steps

:

$V^k(s_i)$ = Expected discounted sum rewards during next k steps

N = Number of states

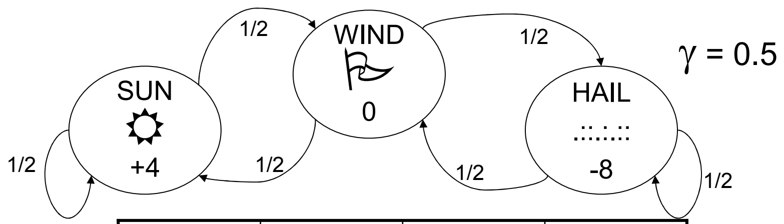
$V^1(s_i) = r(\mathbf{s}_t)$

$V^2(s_i) = r(\mathbf{s}_t) + E[\gamma\, r(\mathbf{s}_{t+1})] = r(s_i) + \gamma \sum_{j=1}^{N} T_{ij} V^1(s_j)$

…

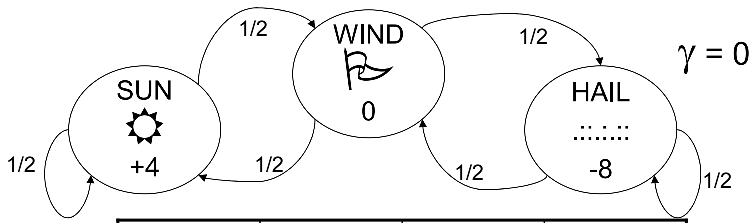
$V^{k+1}(s_i) = r(\mathbf{s}_t) + E[\gamma\, r(\mathbf{s}_{t+1}) + \ldots + \gamma^k r(\mathbf{s}_{t+k})] = r(s_i) + \gamma \sum_{j=1}^{N} T_{ij} V^k(s_j)$

Let's do Value iteration



k	V^k(SUN)	V^k(WIND)	V^k(HAIL)
1			
2			
3			
4			
5			

Let's do Value iteration



$\gamma = 0.5$

k	V^k(SUN)	V^k(WIND)	V^k(HAIL)
1	4	0	-8
2	5	-1	-10
3	5	-1.25	-10.75
4	4.94	-1.44	-11
5	4.88	-1.52	-11.11

Value iteration for solving Markov systems

- Compute $V^1(s_i)$ for each j
- Compute $V^2(s_i)$ for each j

  :
- Compute $V^k(s_i)$ for each j

As $k \to \infty$ $V^k(s_i) \to V(s_i)$. Why?

When to stop? When

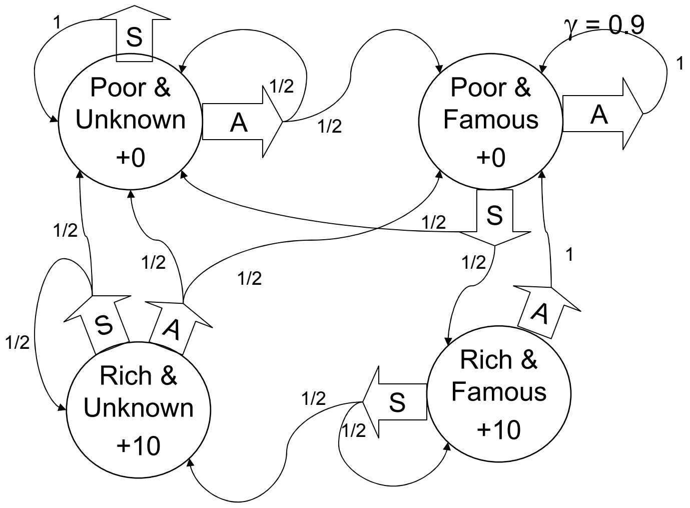
$$\max_i \left| V^{k+1}(s_i) - V^k(s_i) \right| < \xi$$

(This is faster than simple matrix inversion if the transition matrix is sparse)

A Markov decision process

You run a startup company.

In every state you must choose between Saving money or Advertising.



Markov decision processes

An MDP has…

- A set of states $\{s_1 \cdots s_N\}$
- A set of actions $\{a_1 \cdots a_M\}$
- A set of rewards $\{r_1 \cdots r_N\}$ (one for each state)
- A transition probability function

$$T_{ij}^{k} = \mathrm{Prob}\left(\mathbf{s}_{t+1} = s_j \,\middle|\, \mathbf{s}_t = s_i \text{ and } \mathbf{a}_t = a_k\right)$$

On each step:

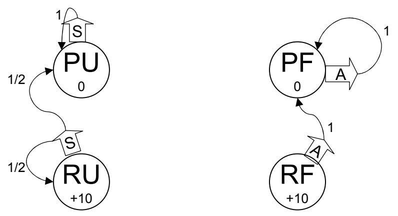
0. Call current state s_i
1. Receive reward r_i
2. Choose action $\in \{a_1 \cdots a_M\}$
3. If you choose action a_k you'll move to state s_j with probability T_{ij}^{k}
4. All future rewards are discounted by γ

A policy

A policy is a mapping from states to actions. Eg:

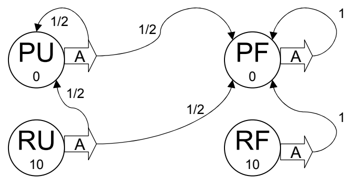
Policy Number 1:

STATE →	ACTION
PU	S
PF	A
RU	S
RF	A



Policy Number 2:

STATE →	ACTION
PU	A
PF	A
RU	A
RF	A



• Which of the above two policies is best?

A policy

- Following a policy reduces an MDP to a Markov system.
  - With what transition probabilities?
- How many possible policies in our example?
- (In general, we might also consider stochastic policies)
- How do you compute the optimal policy?

Interesting fact

For every MDP there exists a (at least one) deterministic optimal policy.

It's a policy such that for every possible start state there is no better option than to follow the policy.

(Not proved in this lecture)

More formally

$V^{\pi}(s_i)$: expected discounted future reward following policy π from state s_i

$V^{*}(s_i)$: expected discounted future reward following *optimal* policy π^* from state s_i

$V^{*}(s_i) \geq V^{\pi}(s_i)$ for all states and policies

Computing the optimal policy

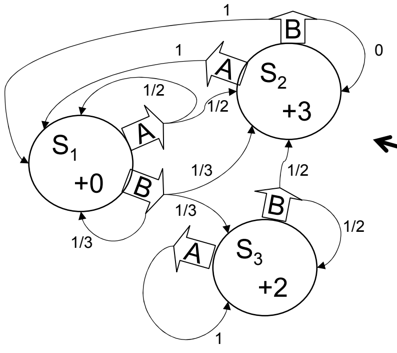
Idea One:

Run through all possible policies.

Select the best.

What's the problem ??

Optimal value function



Question

What (by inspection) is an optimal policy for that MDP?

(assume $\gamma = 0.9$)

What is $V^*(s_1)$?
What is $V^*(s_2)$?
What is $V^*(s_3)$?

Computing the Optimal Value Function with Value Iteration

Define

$V^k(s_i)$ = Maximum possible expected sum of discounted rewards I can get if I start at state S_i and I live for *k* time steps.

Note that $V^1(s_i) = r(s_i)$

Let's compute $V^k(s_i)$ for our example

k	V^k(PU)	V^k(PF)	V^k(RU)	V^k(RF)
1				
2				
3				
4				
5				
6				

Let's compute $V^k(s_i)$ for our example

k	V^k(PU)	V^k(PF)	V^k(RU)	V^k(RF)
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	6.53	25.08	18.55
4	3.852	12.20	29.63	19.26
5	7.22	15.07	32.00	20.40
6	10.03	17.65	33.58	22.43

Bellman Equation

$$V^{n+1}(s_i) = \max_k \left[r(s_i) + \gamma \sum_{j=1}^{N} T_{ij}^k V^n(s_j) \right]$$

Value Iteration for solving MDPs

- Compute $V^1(s_i)$ for all *i*
- Compute $V^2(s_i)$ for all *i*
- …

…..until converged

$$\left[\text{converged when } \max_i \left| J^{n+1}(S_i) - J^n(S_i) \right| \langle \xi \right]$$

…Also known as Dynamic Programming

- Can also update values *asynchronously* (ie in any order)

Finding the optimal policy

Given V*, it is easy to find π*

How?

Finding the optimal policy

Given V*, it is easy to find π*

- Compute V*(s_i) for all i using value iteration
- Define the best action in state s_i as

$$\arg\max_k \left[r_i + \gamma \sum_j \mathrm{T}_{ij}^k \mathrm{V}^*\left(s_j\right) \right]$$

Policy iteration

Another way to compute optimal policies

Algorithm:

Initalize π° = Any randomly chosen policy

Alternate

Policy evaluation:

compute $V^{\pi k}$ (expected rewards using policy π^k)

Policy improvement:

$\pi^{k+1}(s_i) = \arg\max_a \left[r(s_i) + \gamma \sum_j T^a_{ij} V^{\pi k}(s_j) \right]$ (for all i; "greedy" policy under V^k)

… until $\pi^k = \pi^{k+1}$. You now have an optimal policy.

This will converge in a finite number of iterations (why?)

Where we are

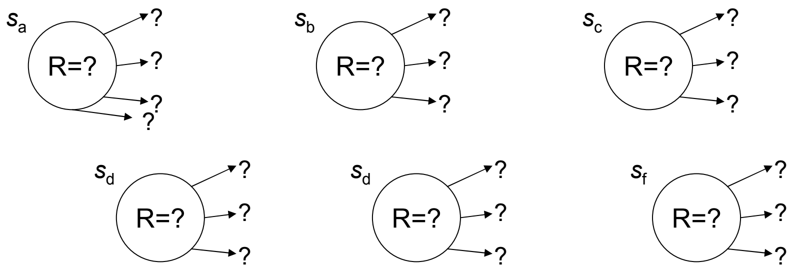
- Formalisms: Markov system and Markov decision process
  - Markov system = MDP + policy
- *Value iteration* for finding expected (optionally optimal) future rewards
  - Optimal values provide optimal policy
- *Policy iteration* for finding optimal policies

Next: *online* versions of these algorithms

Online reinforcement learning

- Imagine you are a robot in a world controlled by an MDP
- You are not given the functions R, T, etc.
- Must learn values and policies from experience with individual states, rewards and actions.
- As before, let's start with the Markov system (no action) case

Learning Delayed Rewards…



All you can see is a series of states and rewards:

s_a (r=0) $\rightarrow s_b$ (r=0) $\rightarrow s_c$ (r=4) $\rightarrow s_b$ (r=0) $\rightarrow s_d$ (r=0) $\rightarrow s_e$ (r=0)

Task: Based on this sequence, estimate $V(s_a), V(s_b) \cdots V(s_f)$

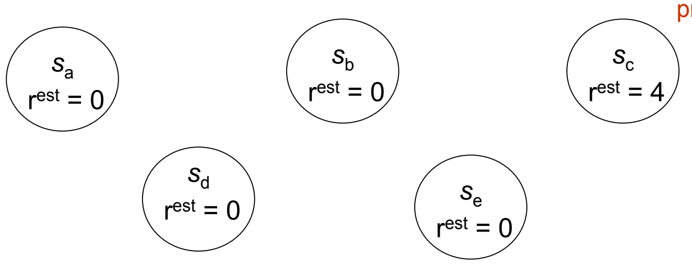
Idea 1: Certainty-Equivalent Learning

Idea: Use your data to estimate the underlying Markov system, then use the previous offline methods to solve it

s_a (r=0) $\rightarrow s_b$ (r=0) $\rightarrow s_c$ (r=4) $\rightarrow s_b$ (r=0) $\rightarrow s_d$ (r=0) $\rightarrow s_e$ (r=0)

Estimated Markov System:

You draw in the transitions + probs



What are the estimated V values?

C.E. for Markov systems

- Estimate T_{ij}, r_i by counting transitions, averaging rewards
- At each step, solve new estimated system with, for instance, value iteration
- (Why do we want new estimates at each step?)
- Slow, memory intensive
- Variations (e.g. prioritized sweeping) minimize computation by taking shortcuts on value iteration step. Can be data-inefficient.

Idea 2: Value sampling

Assume $\gamma=1/2$

Idea: Sample long-term values directly from observed sequence, without estimating T or r

s_a (r=0) → s_b (r=0) → s_c (r=4) → s_b (r=0) → s_d (r=0) → s_e (r=0)

At t=1 we were in state s_a and eventually got a long term discounted reward of $0+\gamma 0+\gamma^2 4+\gamma^3 0+\gamma^4 0\ldots= 1$

At t=2 in state s_b ltdr = 2
At t=3 in state s_c ltdr = 4
At t=4 in state s_b ltdr = 0
At t=5 in state s_d ltdr = 0
At t=6 in state s_e ltdr = 0

State	Observations	Mean LTDR	
s_a	1	1	$=V^{est}(s_a)$
s_b	2 , 0	1	$=V^{est}(s_b)$
s_c	4	4	$=V^{est}(s_c)$
s_d	0	0	$=V^{est}(s_d)$
s_e	0	0	$=V^{est}(s_e)$

(This algorithm is also called Monte Carlo sampling or TD(1))

Idea 3: Temporal Difference learning (Sutton/Barto)

Idea: A sampling version of value iteration

Only maintain a V^{est} array, nothing else

So you've got

$V^{est}(s_1)$, $V^{est}(s_2)$, ··· $V^{est}(s_N)$

and you observe

$$s_i \xrightarrow{r} s_j$$

A transition from i that receives an immediate reward of r and jumps to j

what should you do?

Can You Guess ?

TD learning

Value iteration update: $V^{k+1}(s_i) = E\left[r(s_i) + \gamma V^k(s_j)\right]$

$$\begin{tikzcd} s_i \arrow[r, bend left, "r"] & s_j \end{tikzcd}$$

Use observed r as sample of this

Use observed s_j as sample of this, and $V^{est}(s_j)$ to estimate its value ("bootstrapping")

Learning rule:

nudge $V^{est}(s_i)$ toward sampled value with learning rate α

$$V^{est}(s_i) \leftarrow (1-\alpha)\, V^{est}(s_i) + \alpha \text{ (sampled future reward)}$$
$$= (1-\alpha)\, V^{est}(s_i) + \alpha\, (r + \gamma V^{est}(s_j))$$

(This is actually an algorithm called TD(0))

TD convergence

Dayan (1992) showed that for a more general family of TD rules, as the number of observations goes to infinity, then

$$V^{est}(s_i) \rightarrow V(s_i) \forall i$$

PROVIDED

- All states visited ∞ly often
- Decaying learning rates:

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^{\,2} < \infty$$

This means $\forall k . \exists T . \sum_{t=1}^{T} \alpha_t > k$

This means $\exists k . \forall T . \sum_{t=1}^{T} \alpha_t^2 < k$

Online policy learning

The task:

World:	You are in state 34. Your immediate reward is 3. You have 3 actions.
Robot:	I'll take action 2.
World:	You are in state 77. Your immediate reward is -7. You have 2 actions.
Robot:	I'll take action 1.
World:	You're in state 34 (again). Your immediate reward is 3. You have 3 actions.

The "Credit Assignment" Problem

I'm in state 43,	reward = 0,	action = 2
" " " 39,	" = 0,	" = 4
" " " 22,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 13,	" = 0,	" = 2
" " " 54,	" = 0,	" = 2
" " " 26,	" = 100,	

Yippee! I got to a state with a big reward! But which of my actions along the way actually helped me get there??

This is the Credit Assignment problem.

The MDP machinery we have developed helps address this problem.

Idea 1: Certainty-Equivalent Learning

Idea: Use your data to estimate the underlying MDP, then use the previous offline methods to solve it

Same as before, except now solve estimated MDP using the MDP version of value iteration:

$$V^{est}(s_i) = \max_a \left[r_i^{est} + \gamma \sum_j T_{i,j}^{est,a} V^{est}(s_j) \right]$$

…or policy iteration

The explore/exploit problem

We're in state s_i

We can estimate r_i^{est}, T^{est}, V^{est}

So what action should we choose ?

$$\text{IDEA 1:}\quad a = \arg\max_{a'} \left[r^{est}(s_i) + \gamma \sum_j T_{i,j}^{est,a'} V^{est}(s_j) \right]$$

$$\text{IDEA 2:}\quad a = \text{random}$$

- Any problems with these ideas?
- Any other suggestions?
- Could we be optimal?

Idea 2: Actor/Critic (Sutton)

Idea: An approximate, sampling version of policy iteration, using TD for policy evaluation and stochastic gradient ascent for policy improvement

- Assume stochastic policy, parameterized by *w*:
  Prob(choose action a in state s_i): $\pi(s_i,a) \propto \exp(-\beta w(s_i,a))$
- Observe $s_i \xrightarrow{r,a} s_j$
- Update $V^{est}(s_i)$ with TD
- How do we improve policy *w*?

What policy does V^{est} evaluate?

Actor/critic

Policy improvement: $\pi^{k+1}(s_i) = \arg\max_a \left[E\left[\underbrace{r(s_i)}_{\text{sample}} + \gamma \underbrace{V^{\pi k}}_{\text{estimate}} \underbrace{(s_j)}_{\text{sample}} \right] \right]$

Observed: $s_i \xrightarrow{r,a} s_j$

Update rule: For all k,

$w(s_i,a_k) \leftarrow w(s_i,a_k) + \underbrace{\nu}_{\text{learning rate}} (r + \gamma V^{est}(s_j) - V^{est}(s_i)) (\delta_{a_k,a} - \pi(s_i,a_k))$

Kronecker δ: 1 if a_k=a, 0 if $a_k \neq$a

- Performs stochastic gradient ascent on V^{est}
- Increase probability of action a if result better than expected
  i.e. if $r + \gamma V^{est}(s_j) > V^{est}(s_i)$,
- Decrease it otherwise

Actor/critic

Sampling version of policy iteration

- Disadvantages: No convergence guarantees, somewhat flakey in practice
  - Problems with V^{est} not tracking current policy
- Advantages: May be better with function approximation (will return to this)

- Not obvious how to make a sampling version of MDP value iteration for optimal values V*. (Why?)

Idea 3: Q-learning (Watkins)

Idea: TD with a redefined value function, which can be learned independent of exploration policy

Define

$Q^*(s_i,a)$= Expected sum of discounted future rewards if I start in state s_i, if I then take action a, and if I'm subsequently optimal

Questions:

Define $Q^*(s_i,a)$ in terms of V^*

Define $V^*(s_i)$ in terms of Q^*

Q-learning

Q version of Bellman equation:

$$Q^*(s_i, a) = r_i + \gamma \sum_j T^a_{i,j} \max_{a'} Q^*(S_j, a')$$

We maintain Q^{est} instead of V^{est} values

The TD update, on seeing $s_i \xrightarrow[\text{action a}]{\text{reward r}} s_j$, is:

$$Q^{est}(s_i, a) \leftarrow \alpha\left[r + \gamma \max_{a'} Q^{est}(s_j, a')\right] + (1-\alpha)Q^{est}(s_i, a)$$

This is even cleverer than it looks: the Q^{est} values are not biased by any particular exploration policy.

Q-learning is proved to converge.

Q-Learning: Choosing Actions

Same issues as for CE choosing actions

- Optimal action is: $\arg\max_a Q^*(s_i, a)$
- Don't always choose optimally according to Q^{est}
- Don't always be random (otherwise it will take a long time to reach somewhere exciting)
- Boltzmann exploration, as in actor/critic:

$$\text{Prob(choose action a)} \propto \exp\left(-\beta Q^{est}(s,a)\right)$$

Where we are

- Formalism
- Offline algorithms
- Online algorithms for value estimation
  - CE (model based), sampling, TD (model free)
- Online algorithms for policy learning
  - CE, actor/critic, Q-learning

If we had time…

- Avoid lookup tables for value function
  - Use function approximation/regression
  - Convergence guarantees out the window
  - May favor *policy gradient* methods
    - Actor/critic is one example, but policy gradients can also be estimated directly, i.e. without using values
- Optimal exploration/exploitation tradeoffs
  - Gittins indices
  - E^3 (Kearns)

If we had time…

- TD(λ)
  - TD generalized with multistep backups
  - Monte Carlo trajectory sampling is a special case.
- Applications
  - Backgammon
  - Elevator scheduling
  - Neuroscientific modeling

If we had time…

- Partially Observable MDPs
  - RL when state is not observable (like a hidden Markov model)
  - Extremely intractable, some cases proved uncomputable
  - Approximate offline value iteration methods
  - No policy iteration, minimal online methods

For more

- Review paper:
  - Kaelbling et al., “Reinforcement learning: A Survey,” Journal of AI Research, 1996
- Two excellent books:
  - Sutton & Barto, “Reinforcement Learning”
    - Informal and readable
  - Tsitsiklis & Bertsekas, “Neuro-Dynamic Programming”
    - Formal and less readable, full of delightful proofs