Chris Hodapp
chodapp3@gatech.edu

2016-05-16, CS-8803-003

- RLDM (Reinforcement Learning & Decision-Making)
  - See RL-1 from CS7641
  - Recall Bellman Equation: $V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s')\right)$
    - We've modified it slightly to use other reward forms.
    - We use V, not U, for utility.
    - $R(s,a)$ is reward for being in state 's' & taking action 'a'
  - We may expand to:

$$V(s_1) = \max_{a_1} \left(R(s_1,a_1) + \gamma \sum_{s_2} T(s_1,a_1,s_2) \underbrace{\max_{a_2} \left(R(s_2,a_2) + \gamma \sum_{s_3} T(s_2,a_2,s_3) \ldots\right)}_{V(s_2)}\right]$$

  - We can see that it repeats in terms of V like above, but it can repeat other ways too:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

  - But why do we need Q rather than V?
    - Q form is more useful in reinforcement learning without requiring transitions & reward function
  - A third form, occasionally useful, is the "continuation" form from Tom Dietrich:

$$C(s,a) = \gamma \sum_{s'} T(s,a,s') \max_{a'} \left(R(s',a') + C(s',a')\right)$$

  - Mostly useful if R is hard to represent
  - We then have 3 Bellman equations:

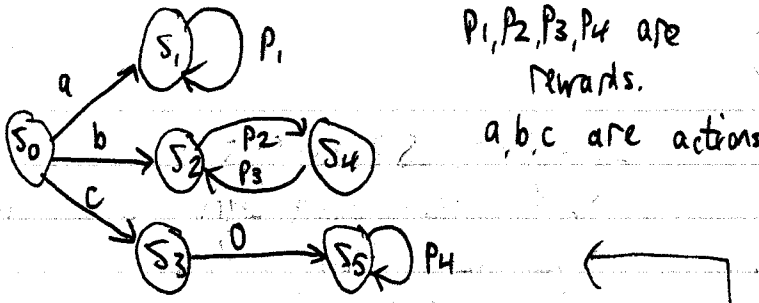
| | V | Q | C |
|---|---|---|---|
| V | $V(s) = V(s)$ | $V(s) = \max_a Q(s,a)$ | $V(s) = \max_a \left(R(s,a) + C(s,a)\right)$ |
| Q | $Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s')$ | $Q(s,a) = Q(s,a)$ | $Q(s,a) = R(s,a) + C(s,a)$ |
| C | $C(s,a) = \gamma \sum_{s'} T(s,a,s') V(s')$ | $C(s,a) = \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$ | $C(s,a) = C(s,a)$ |

1. "Smoov & Curly's Bogus Journey"

★ Get these intuitively!

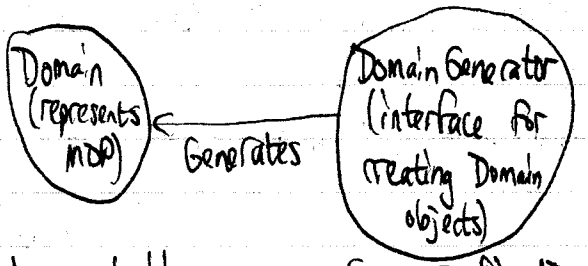
T. Introduction to BURLAP

- Example MDP:




$p_1, p_2, p_3, p_4$ are rewards.
a, b, c are actions

- and probability of ending after each step of $1-\gamma$ (i.e. $\gamma$ discount factor)
- Object-Oriented MDPs in BURLAP:




- We'd probably use GraphDefinedDomain to represent this

2016-05-17

Sutton Ch. 2

- RL, vs. other types of learning, uses training info that evaluates the actions taken, rather than instructs by giving correct actions.
  - Hence, it needs active exploration.
  - This evaluative feedback is also the basis for function optimization.
  - Supervised learning relies on purely instructive feedback, indicating correct action.

Also read: Littman (1996)

2016-05-20

2. Reinforcement Learning Basics

- Agent & environment
  - Environment reveals itself to agent as states 's'
  - Agent interacts w/ environment with actions 'a' & receives rewards 'r'
  - This is like an MDP but more general; agent experiences environment through interactions
- Behavior Structures
  - plan - fixed sequence of actions (isn't appropriate w/ stochasticity)
  - conditional plan - can have if/then in it, in effect (observes context)
  - stationary policy/universal plan - maps states to action

Chris Hodapp
chodapp3@gatech.edu

2016-05-20(6), CS8803-003 (RLDM)

2

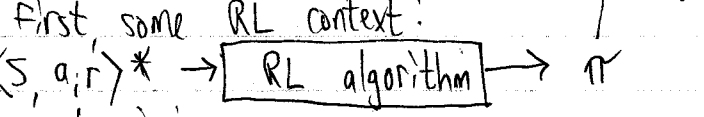
- Stationary plan depends only on state (and as a result can be much 'larger' and harder to convey than a conditional plan)
  - It's less compact, but it can handle all sorts of stochasticity & express an optimal policy.
- But what is an optimal policy?
- Evaluating a policy
  1. State transitions to immediate rewards - $R(s,a)$
  2. Truncate according to horizon - e.g. $T=5$
  3. Summarize sequence - Return: $\sum \gamma^i r_i$ for instance
  4. Summarize over sequences - Average expectation
- Evaluating a Learner - how?
  - Value of returned policy
  - Computational complexity (what time to learn?)
  - Experience complexity (how much data is required to learn?)
- Space complexity matters but we don't really consider it here

Think of MDMC & sample complexity

---

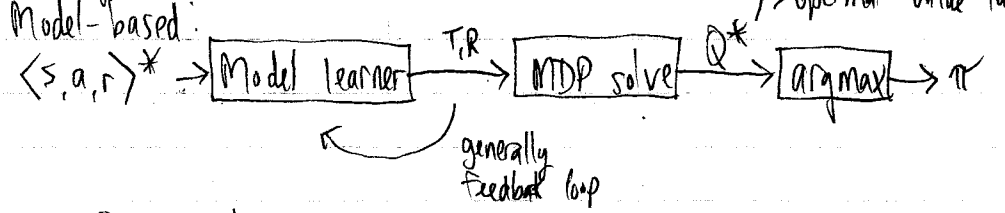
3. TD & Friends

- Temporal Difference Learning
  - Read Sutton (1988), Littman (1996)
  - But first, some RL context:

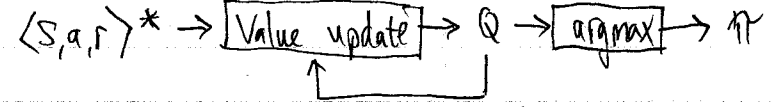



n.b. ⟨s,a,r⟩* = multiple ⟨s,a,r⟩

  - Three categories:
    1. Model-based:




    2. Value-function-based or model-free:




       - Learns value function directly (no model is derived)

3. Policy search: $\langle s, a, r \rangle^* \rightarrow$ [Policy update] $\rightarrow \pi$

- Much more direct - but much harder learning problem.

In general:

Direct learning ↓

1. Model-based
2. Value-function based
3. Policy search

↑ More supervised

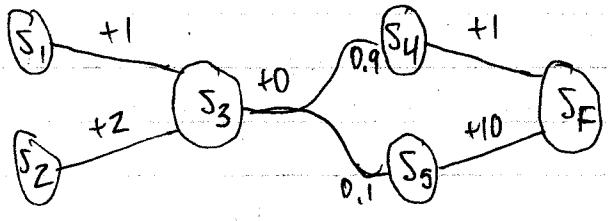
- In model-based, the learner basically solves a supervised learning problem and receives direct feedback, but then has multiple other steps to do to make a policy.
- In policy search, feedback is not directly useful to learning - but result is already a policy.
- So - all have tradeoffs

- TD(1) is from Sutton (Temporal Difference)
  - Learning to predict over time

    $s_0 \xrightarrow{r_0} s_1 \xrightarrow{r_1} s_2 \xrightarrow{r_2} s_F$ (Sequence of states)
  - Each transition has a reward. Can we predict that reward & discounted reward?
  - Example:

    Learn $V(s) = \begin{cases} 0 & \text{if } s = s_F \\ E[r + \gamma V(s')] & \text{otherwise} \end{cases}$ where

$E[..]$ = expected value




If $\gamma = 1$ then $V(s_3) = 1.9$

Easiest way here is to work backwards from $s_F$.

$V(s_F) = 0$, $V(s_4) = 1$, $V(s_5) = 10$, $V(s_3) = 0.1 V(s_5) + 0.9 V(s_4)$

Chris Hodapp
chodapp3@gatech.edu

2016-05-20c, CS8803-003 (RLDM)

3, TD (cont.)

- But suppose we had...

1. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
2. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_5 \xrightarrow{+10} S_F$
3. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
4. $S_1 \xrightarrow{+1} S_3 \xrightarrow{+0} S_4 \xrightarrow{+1} S_F$
5. $S_2 \xrightarrow{+2} S_3 \xrightarrow{+0} S_5 \xrightarrow{+10} S_F$

From 3 samples, estimated $V(s_1) = \frac{15}{3} = 5$

From 4, estimated $V(s) = \frac{17}{4} = 4.25$

- We can incrementally update estimates from more samples
- This still is a bit off from correct $V(s_1) = 1.9$ but it should converge given enough samples.
- Suppose we have $V_3(s_1) = 5$ (3 episodes) and $R_4(s_1) = 2$ (additional episode) - how would we compute $V_4(s_1)$?
  $V_3$ has 3 samples, $R_4$ is just one, so: $\frac{1}{4}(3V_3(s_1) + R_4(s_1)) = \frac{17}{4}$
- In general: $V_{T-1}(s_1)$, $R_T(s_1)$, find $V_T(s_1)$

$$V_T(s_1) = \frac{(T-1)V_{T-1}(s_1) + R_T(s_1)}{T}$$

$$= \frac{T-1}{T} V_{T-1}(s_1) + \frac{1}{T} R_T(s_1) = V_{T-1}(s_1) + \alpha_T \left( R_T(s_1) - V_{T-1}(s_1) \right)$$

where $\alpha_T = 1/T$

Note that it has this temporal difference

- As that difference converges, $V_T$ changes less & less
- Note similarity to perceptron update rule
- General 'learning rate' properties:

$\lim_{T \to \infty} V_T(s) = V(s)$ for any $\alpha_T$ such that: $\sum_T \alpha_T = \infty$, $\sum_T \alpha_T^2 < \infty$ (equals (slide typo))

("learning rate sequence")

$\alpha_T = \frac{1}{T}$ works; so does $\alpha_T = 1/T^{2/3}$. $\alpha_T = \frac{1}{T^2}$ doesn't, neither does $\alpha_T = 1/T^{1/2}$.

(See harmonic series: $\sum_{n=1}^{\infty} \frac{1}{n^p}$ converges for $p > 1$, diverges for $p \leq 1$)

See Sutton & Barto Figure 7.7 (p174)

- Finally, TD(1) update rule:
  - Episode T

Episode & timestep

$e(s)$ = eligibility of state $s$

For all $s$, $e(s) = 0$ at start of episode, $V_T(s) = V_{T-1}(s)$

After $s_{t-1} \xrightarrow{r_t} s_t$ : (step $t$)

$e(s_{t-1}) = e(s_{t-1}) + 1$ - State we just left is more eligible

For all $s$,

$V_T(s) = V_T(s) + \underbrace{\alpha_T (r_t + \gamma V_{T-1}(s_t) - V_{t-1}(s_{t-1}))}\, e(s)$

Apply temporal difference to all states, discounted by $\alpha_T$ and scaled by state's eligibility

$e(s) = \gamma e(s)$

- Note that this step can be done in all states in parallel.

- TD(1) example:

$s_1 \xrightarrow{r_1} s_2 \xrightarrow{r_2} s_3 \xrightarrow{r_3} s_F$

$e =$ 0 0 0

$e =$ 1 " "

First: Transition $s_1 \to s_2$, reward $r_1$, so update $e(s_1)$, and $V$:

$\Delta V_T(s_1) = \alpha (r_1 + \gamma V_{T-1}(s_2) - V_{T-1}(s_1))$

$\Delta V_T(s_2) = 0$

$\Delta V_T(s_3) = 0$

} because $e(s_2) = e(s_3) = 0$ here

and decay $e$ by $\gamma$:

$e =$ $\gamma$ 0 0

$e =$ $\gamma$ 1 0

Then, transition $s_2 \to s_3$, reward $r_2$, update $e(s_2)$ & $V$:

$\Delta V_T(s_1) = \underbrace{\alpha (r_1 + \gamma V_{T-1}(s_2) - V_{T-1}(s_1))}_{\text{Same as } \Delta V_T \text{ above}} + \underbrace{\gamma \alpha (r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2))}_{\text{note } \gamma \text{ factor}}$

$\Delta V_T(s_2) = \alpha (r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2))$

$\Delta V_T(s_3) = 0$ - still $e(s_3) = 0$

But $\Delta V_T(s_1)$ simplifies to: $\alpha (r_1 + \gamma r_2 + \gamma^2 V_{T-1}(s_3) - V_{T-1}(s_3))$

$e =$ $\gamma^2$ $\gamma$ 0

$e =$ $\gamma^2$ $\gamma$ 1

& update $e(s_2)$... Then transition $s_3 \to s_F$, $r_3$ reward and update $e(s_3)$, add $r_3 + \gamma V_{T-1}(s_F) - V_{T-1}(s_3)$ to $\Delta V$...

After much simplifying:

$\Delta V_T(s_1) = \alpha (r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V_{T-1}(s_F) - V_{T-1}(s_1))$

$\Delta V_T(s_2) = \alpha (r_2 + \gamma r_3 + \gamma^2 V_{T-1}(s_F) - V_{T-1}(s_2))$

$\Delta V_T(s_3) = \alpha (r_3 + \gamma V_{T-1}(s_F) - V_{T-1}(s_3))$

Claim: TD(1) is same as outcome-based updates (if no repeated states)

Chris Hodapp
chodapp3@gatech.edu

2016-05-20d, CS8803-003 (RLDM)

3, TD (cont.)

- Which is to say: We didn't have to find the final outcome of an entire 'run'. But, what if states do repeat?
  - Outcome-based learns nothing during the episode. TD(1) does learn during!
  - TD(1) takes advantage of repeats if they occur.

2016-05-21

- See final quiz of this section; I don't quite follow what it asks.
- TD(1) is "wrong" here because it doesn't use very much of the data (compared with max. likelihood estimate). It uses only individual runs that involve the state, and rare events in a run can make a huge bias.
- So... TD(0) Rule
  - Finds ML estimate (if data repeated infinitely often):
    $V_T(s_{t-1}) = V_T(s_{t-1}) + \alpha_T(r_t + \gamma V_T(s_t) - V_T(s_{t-1}))$
  - Note that this is basically TD(1) without $e(s)$.
  - And: $V_T(s_{t-1}) = \underset{s_t}{E}[r_t + \gamma V_T(s_t)]$ because we do this update more often on states that randomly appear more often - so it's approximating $E[\ldots]$.
    This is just the ML estimate.
    It's not the case for the outcome-based TD(1), which uses only the observed reward sequence, not the estimates from elsewhere.
- This generalizes to TD($\lambda$), of which TD(0) & TD(1) are cases. TD($\lambda$) covers both, and others.
- Just replace $e(s) = \gamma e(s)$ with $e(s) = \lambda \gamma e(s)$.
  - For $\lambda = 0$, it "uses" eligibility but only to restrict to states actually visited (exactly as TD(0) does).
  - For $\lambda = 1$ it is just TD(1).

Again, T&T

- To understand TD($\lambda$), first, K-Step Estimators:

$E_1$: $V(s_t) = V(s_t) + \alpha_T(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$

$E_2$: $V(s_t) = V(s_t) + \alpha_T(r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2}) - V(s_t))$

$E_3$: $V(s_t) = V(s_t) + \alpha_T(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V(s_{t+3}) - V(s_t))$

$E_K$: $V(s_t) = V(s_t) + \alpha_T(r_{t+1} + \ldots + \gamma^{K-1} r_{t+K} + \gamma^K V(s_{t+K}) - V(s_t))$

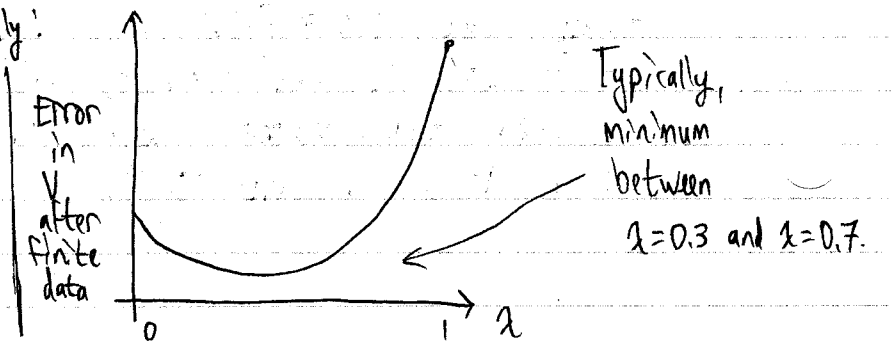
$E_\infty$: $V(s_t) = V(s_t) + \alpha_T(r_{t+1} + \ldots + \gamma^{K-1} r_{t+K} + \ldots - V(s_t))$

- $E_1$ estimates value of state we just left, based on one state ahead. It is just TD(0). $E_2$ extends this to 2 rewards, not 1 - so two steps, then lookahead.
Note that $V(s_{t+\ldots})$ is discounted future estimate.
- $E_\infty$ is just TD(1). All $E_1, E_2, E_3 \ldots$ are different $\lambda$, 0 to 1.
- Weights are:

| | |
|---|---|
| $E_1$: $1-\lambda$ | When we change a state's value, |
| $E_2$: $\lambda(1-\lambda)$ | we do so by all of the estimators |
| $E_3$: $\lambda^2(1-\lambda)$ | (a convex combo of all of them). |
| $E_K$: $\lambda^{K-1}(1-\lambda)$ | Note that $\sum_i^\infty E_i = 1$ for $\lambda \leq 1$. |
| $E_\infty$: $\lambda^\infty$ | |

- $\lambda = 1$ means $E_1 \ldots E_k$ are all weighted by 0, except that $E_\infty = 1$. $\lambda = 0$ means $E_1$ has weight 1, all others 0.
- Intermediate $\lambda$ values, however, put a nonzero weight on all estimators, exponentially decreasing.
- So... TD($\lambda$) can be seen as a weighted combination of every K-Step estimator, with $\lambda$ varying those weights.
- However, this doesn't give a good intuitive sense of what, for instance, $\lambda = 1/2$ means.
- But TD($\lambda$), empirically:

- It usefully combines what occurs at the end of each episode & at each step while being an unbiased estimator



Chris Hodapp
chodapp3@gatech.edu

2016-05-22, CS8803-005 (RLDM)

3, TD (cont.)

- TD(1) is less efficient with data and this tends to increase variance
- TD(0) provides ML estimate
- TD(λ) interpolates & provides cleaner estimates

2016-05-23

- Reading: Littman's thesis (Ch. 1 & 2)
  - Good overview, mostly redundant with things already presented
- Reading: "Learning to Predict by the Methods of Temporal Differences" (Sutton)

2016-05-26

Read: Littman & Szepesvari (1996)
Littman (1994)

4. Convergence

- Convergence: TD with control
  - We can use TD not just to find values of states, but also values of actions (and thus a policy)
  - We covered only Bellman equation without actions. If we try to get actions back in:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

"best" action

and make a TD(0)-like rule:

For transition $\langle s_{t-1}, a_{t-1}, r_t, s_t \rangle$

$$Q_t(s_{t-1},a_{t-1}) = Q_{t-1}(s_{t-1},a_{t-1}) + \alpha_t \left(r_t + \gamma \max_{a'} Q_{t-1}(s_t,a') - Q_{t-1}(s_{t-1},a_{t-1})\right)$$

  - That is: For the action & state we just had.
  - Elsewhere: $Q_t(s,a) = Q_{t-1}(s,a)$ - no change.
  - If we knew the model we could just update everything:

$$Q_t(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_{t-1}(s',a')$$

  - If we knew $Q^*$, we could update... (sort of cheating. Assume we know $Q^*$ at just $s_t$.)

$$Q_t(s_{t-1},a_{t-1}) = Q_{t-1}(s_{t-1},a_{t-1}) + \alpha_t \left(r_t + \gamma \max_{a'} Q^*(s_t,a') - Q_{t-1}(s_{t-1},a_{t-1})\right)$$

- But... why does any of this, including Bellman equation, converge on what we want?
- First: Operator notation. Let B be an operator from value functions to value functions.

→ Bellman operator

$[BQ](s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$

$Q^* = BQ^*$ is just Bellman equation.

$Q_t = BQ_{t-1}$ is just value iteration

Here, $\|Q\|_\infty = \max_{s,a} |Q(s,a)|$ → "infinity norm" "max norm"

- Contraction Mappings
  - B is an operator. $\gamma \in [0,1)$
  - If for all F, G and some $\gamma$, $\|BF - BG\|_\infty \leq \gamma \|F - G\|_\infty$ then B is a <u>contraction mapping</u>
- As an example (simplified): $B(x) = x/2$ and $B(x) = (x+100)(0.9)$ are. $B(x) = x+1$ and $B(x) = x-1$ are not.
- Nice properties of a contraction mapping:
  - For contraction mapping B:
    1. $F^* = BF^*$ has a unique solution.
    2. $F_t = BF_{t-1} \Rightarrow F_t \to F^*$ (value iteration converges in the limit)

    Note that by definition, $\|BF_{t-1} - BF^*\|_\infty \leq \gamma \|F_{t-1} - F^*\|_\infty$ but $BF_{t-1} = F_t$ and $BF^* = F^*$. So:

    $\|F_t - F^*\|_\infty \leq \gamma \|F_{t-1} - F^*\|_\infty$ → We're getting closer to $F^*$

    #1 follows similarly to show uniqueness. If we had another solution, then the above would fail to converge, which violates the definition of a contraction mapping.
- Does Bellman operator contract? Start from $[BQ](s,a)$ above, and with $Q_1$ & $Q_2$.

$\|BQ_1 - BQ_2\|_\infty = \max_{s,a} |[BQ_1](s,a) - [BQ_2](s,a)| =$

$\max_{s,a} \left| \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_1(s,a)\right) - \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_2(s',a')\right) \right|$

R(s,a) cancels, factor $\gamma$, $\Sigma$

$= \max_{s,a} \left| \gamma \sum_{s'} T(s,a,s') \left(\max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a')\right) \right|$

Note that the two a' here ↔ are <u>not</u> the same a' so we can't factor that or combine directly. But, since it is a convex combination over s' (note that T is probabilities), it cannot exceed the maximum over s':

$\leq \gamma \max_{s'} \left| \max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \right|$

Note $\leq$, not $=$. This is worst-case.

Note that $\gamma \|Q_1 - Q_2\|_\infty = \gamma \max_{s',a'} |Q_1(s',a') - Q_2(s',a')|$ - Close.

Chris Hodapp
(chodapp3@gatech.edu)

2016-05-26 (b), CS8803-003 (RLDM)

4, Convergence (cont.)

- It turns out that actually:
  $\gamma \max_{s'} |\max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a')| \leq \gamma \|Q_1 - Q_2\|_\infty$
  because of a "magic step". What we want:
  $\forall f, g, \; |\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$.
  (or: Is max a "non-expansion"?)
  First: Assume that $\max_a f(a) \leq \max_a g(a)$ without loss of generality. (If not true: switch f & g.)
  Then: $|\max_a f(a) - \max_a g(a)| = \max_a f(a) - \max_a g(a)$.
  $= f(a_1) - g(a_2)$ where $a_1 = \text{argmax}_a f(a)$, $a_2 = \text{argmax}_a g(a)$

  $a_2$ maximizes $g$ by definition. Any other value then can't produce a higher value - i.e. $g(a_1) \leq g(a_2)$.
  Thus, $f(a_1) - g(a_2) \leq f(a_1) - g(a_1) = |f(a_1) - g(a_1)|$ (guaranteed $\geq 0$).
  And $|f(a_1) - g(a_1)| \leq \max_a |f(a) - g(a)|$
  - Why: $\max_a$ includes $a_1$ - so $\max_a$ has to be at least equal. It <u>might</u> include some other, higher $a$, hence also $\leq$.
  $\rightarrow |\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$
- After all that: Bellman operator <u>is</u> a contraction mapping! So... It converges to a unique solution.
- Convergence
  - Define: $\alpha_t(s,a) = 0$ if $s_t \neq s$, $a_t \neq a$. (Basically: Leave Q value alone, except for the state-action pair just seen, by setting all other learning rates to 0.)
  - Theorem: Let $B$ be a contraction mapping and $Q^* = BQ^*$ be its fixed point. Let $Q_0$ be a Q function and define $Q_{t+1} = [B_t Q_t] Q_t$. ($B_t$ is a new operator.) Then, $Q_t \rightarrow Q^*$ if:
    1. For all Q-functions $U_1$ & $U_2$, and $s, a$:
       $|([B_t U_1] Q^*)(s,a) - ([B_t U_2] Q^*)(s,a)| \leq (1 - \alpha_t(s,a)) |U_1(s,a) - U_2(s,a)|$
       - This is another non-expansion rule. Basically, $B$ doesn't move things further.

This proof method ends up working for all sorts of other things that are similar to Q-learning

2. For all $Q, U, s, a$:
$$\left|\left([B_t U]\, Q^*\right)(s,a) - \left([B_t U]\, Q\right)(s,a)\right| \le \gamma\, \alpha_t(s,a)\, \left|Q^*(s,a) - Q(s,a)\right|$$
- If we hold $U$ fixed, $Q$ contracts toward $Q^*$.

3. $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$.
- This also gives us the constraint (because we've put $Q$s into $\alpha_t$) that we must visit all state/action pairs infinitely often.

- To explain further:
$$\left([B_t\, Q_w]\, w\right)(s,a) = Q(s,a) + \alpha_t(s,a)\left(r_t + \gamma \max_{a'} w(s_t, a') - Q(s,a)\right)$$
- This is like Q-learning but we've separated $Q_t$ & $w$ - $Q$ is the Q-function for where we just left, and $w$ is the estimate for the state we just entered.
- It satisfies condition 1 above (it's able to average over noise)
- It satisfies condition 2 (it's a contraction), & 3 trivially holds (?)

- Generalized MDPs:
$$Q^*(s,a) = R(s,a) + \gamma \bigoplus_{s'}^{s,a} \bigotimes_{a'}^{s'} Q^*(s', a')$$

If $\bigoplus_{s'}^{s,a} g(s') = \sum_{s'} T(s,a,s')\, g(s')$ & $\bigotimes_{a'}^{s'} f(s', a') = \max_{a'} f(s', a')$, this is just a "regular" MDP.
Other $\oplus$ and $\otimes$ can make for other cases.

- N.B. Order statistics (max, min, for example)
Fixed convex combinations (can't depend on value, e.g. the way Boltzmann does)
→ Those are non-expanding. Whether other convex combos do is sort of an open problem.

Chris Hodapp
(chodapp3@gatech.edu)

2016-05-27, CS8803-003 (RLDM)

5. AAA

- Advanced Algorithmic Analysis
  1. For some $t^* < \infty$, polynomial in $|S|$, $|A|$, $R_{max} = \max_{a,s} |R(s,a)|$, $\frac{1}{1-\gamma}$, bits of precision, $\pi(s) = \arg\max_a Q_{t^*}(s,a)$ is optimal. (Loosely: You can find optimal policy in polynomial time.)
     Note that this isn't precisely convergence. It relates to Kramer's rule.
  2. If $|V_t(s) - V_{t+1}(s)| < \varepsilon$ for all states in MDP,
     $\max_s |V^{\pi_{V_t}}(s) - V^*(s)| < \frac{2\varepsilon\gamma}{1-\gamma}$ where $\pi_{V_t}$ is policy at time $t$
     (If it has stopped changing, we've approached optimal to some limit.)
- These put stronger conditions than just "converges in the limit".
- Intuitively, smaller $\gamma$ gives quicker answers (because for one thing it doesn't look very far into the future. It's myopic, in other words.) Larger $\gamma$ looks ahead more, but takes longer.
  3. $\|B^K Q_1 - B^K Q_2\|_\infty \leq \gamma^K \|Q_1 - Q_2\|_\infty$ (K-step Bellman operator is a contraction mapping.)

- Linear Programming
  - The only known way of solving MDP in polynomial time.
  - But how do we make linear constraints & linear objective function so that we can encode MDP as a linear program?
  - Bellman equation has max though, which is nonlinear...
  - Consider: $x = \max(-3, 7, 2, 5)$. How does $x$ relate to each number?
    $x \geq 3$, $x \geq 7$, $x \geq 2$, $x \geq 5$ - these are all true, but we want the <u>smallest</u> such $x$. This gives some idea...
  - Bellman: $\forall s,\ V_s = \max_a (R(s,a) + \gamma \sum_{s'} T(s,a,s') V_{s'})$
  - As a linear program:
    $\min \sum_{s'} V_{s'}$ (objective function)
    $\forall s, a \quad V_s \geq R(s,a) + \gamma \sum_{s'} T(s,a,s') V_{s'}$ (constraints)
    "Primal" representation of LP for solving MDPs. It works but is not the most efficient representation.

- LP has a neat "dual" property: You can change variables to constraints & constraints to variables, and have an equivalent program.
- This dual:
  - It has a mechanical process for this conversion (not shown here). The result is:

$$\max_{q_{sa}} \sum_s \sum_a q_{sa} R(s,a) \quad \text{- Our objective function.}$$

See Littman (1996) section 2.3.3

$$\text{s.t.} \quad 1 + \gamma \sum_s \sum_a q_{sa} T(s,a,s') = \sum_a q_{sa'} \quad \forall s'$$

$$\forall s,a \quad q_{sa} \geq 0 \quad \text{(policy flow is non-negative)}$$

  - $q_{sa}$ is "policy flow"; how much "agentness" is in each state-action pair?
  - 1st equation is maximizing reward
  - 2nd: "policy flow" through next state should equal number of times state was visited, times transition prob, discounted.
  - Despite this dual being produced mechanically, it still has a meaningful interpretation in terms of value/policy
- Could suggest another way of solving... Policy Iteration.
  - Initialize: $\forall s, Q_0(s) = 0$
  - Policy improvement: $\forall s, \pi_t(s) = \text{argmax}_a Q_t(s,a) \quad (t \geq 0)$
  - Policy evaluation: $Q_{t+1} = Q^{\pi_t}$
  1. $Q_t \to Q^*$
  2. Convergence is exact & complete in finite time
  3. Converges at least as fast as VI
  - Drawback: Improvement step requires basically VI. PI needs fewer iterations - but is more computationally intensive.
    - But this is still an open problem (not known if linear, exponential, or in between).

Chris Hodapp
(chodapp3@gatech.edu)

2016-05-27b, CS8803-003 (RLDM)

5. AAA (continued)

- Domination
  - Why does policy iteration work?
  - $\pi_1 \geq \pi_2$ iff $\forall_{s \in S} \; V^{\pi_1}(s) \geq V^{\pi_2}(s)$ (domination)
  - $\pi_1 > \pi_2$ iff $\forall_{s \in S} \; V^{\pi_1}(s) \geq V^{\pi_2}(s)$ (strict dominance)
    and $\exists_{s \in S} \; V^{\pi_1}(s) > V^{\pi_2}(s)$
  - $\pi$ is $\epsilon$-optimal iff $|V^{\pi}(s) - V^{\pi^*}(s)| \leq \epsilon \quad \forall s \in S$
  - Consider $\pi_1, \pi_2$ (policies), and $B_1$ & $B_2$:
    $[B_1 V](s) = R(s, \pi_1(s)) + \gamma \sum_{s'} T(s, \pi_1(s), s') V(s')$
    $[B_2 V](s) = R(s, \pi_2(s)) + \gamma \sum_{s'} T(s, \pi_2(s), s') V(s')$
    ($B_1$ & $B_2$ are associated with fixed policy)
    - $\pi_1$ dominates
    - Theorem: If $V_1 \geq V_2$, then $B_2 V_1 \geq B_2 V_2$ (monotonicity)
      (order doesn't change). Easily proved (see slides).
    - Also: $Q_1 \leq B_2 Q_1$ (value improvement)
  - Another property:
    - If $\pi_1$ optimal, then greedy policy w.r.t. $Q_1$ cannot be worse.
    - Definition of domination prevents us from being in cycles
      (Think of k-means & proof of convergence)
    - Particularly: Domination on a state-by-state basis.
- VI: Converges in finite steps (greedy policy)
- LP: Primal/dual representation
- PI: Value improvement

2016-06-06

6. Messing With Rewards

- To read: Ng, Harada, Russell (1999), Asmuth, Littman, Zinkov (2008)
- Why might we want to change the reward function for an MDP?
  - Perhaps it's easier to solve
  - Perhaps we don't even have one yet...
- Reward functions form a sort of "programming language" that's turned into the desired behavior

- 'Easier' might mean various things - faster, possible in the first place, more efficient
- Dr. Isbell likes the 'program' analogy because many different ~~reward functions~~ programs might be semantically identical, but get results different ways
- How can we change RF without changing optimal policy?
  - Multiply by positive constant (they're sort of unitless)
  - Shift by constant
  - Less obviously: non-linear potential-based rewards

$Q'(s,a) = c\,Q(s,a)$

- Multiplying by positive scalar (must be positive):
  - Just look at Bellman equation, constant passes through.

$Q'(s,a) = Q(s,a) + \frac{c}{1-\gamma}$

- Likewise for adding a constant
  - Slightly trickier. Start with:

$$Q(s,a) = \underbrace{R(s,a)}_{\text{Replace with } R+c} + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

Note +c at each term

  - Note then that we have extra +c in sum, but $\gamma c$, $\gamma^2 c$, $\gamma^3 c$... added up → geometric series,

$$Q'(s,a) = Q(s,a) + c/1-\gamma$$

  - Easily proven (note that Bellman eq. has unique solution) by seeing if $Q'/R'$ satisfy Bellman equation

- Reward Shaping
  - Think of how we train animals, given that we can't just tell an animal what to do directly.
  - Example in lectures: simplified 'soccer' game: how do we guide it to actually be near the ball, get the ball, & hit into goal?
  - How do we avoid it just sitting near the ball and racking up rewards? If merely hitting the ball gives a reward, how do we avoid it just repeatedly hitting the ball without getting to goal?

Chris Hodapp
chodapp3@gatech.edu

2016-06-06(b), CS8803-003 (RLDM)

6. Messing with Rewards

- How do we avoid 'positive suboptimal loops'?
  - (Or: suboptimal positive-loop)
  - One way is appropriate negative rewards
- Potential-based shaping
  - Certain states give bonus when you achieve, and you lose that bonus when you 'unachieve'
  - State-based bonuses
    - Instead of a constant, cumulative bonus for (e.g.) just being near the ball, have some increment/decrement based on state.
    - More sort of change-in-state-based bonuses
- More formally:

$$Q(s,a) = \sum_{s'} T(s,a,s') \left( R(s,a,s') + \gamma \max_{a'} Q(s',a') \right)$$

$$R'(s,a,s') = R(s,a) - \underbrace{\Psi(s)}_{\text{leaving } s} + \underbrace{\gamma \Psi(s')}_{\text{entering } s'}$$

then $Q'(s,a) = Q(s,a) - \Psi(s)$ - derivation is fairly simple (it's an infinite telescoping sum).

Intuitively, this just says, "True value of an action is the value, minus the potential you'll lose by leaving that state."

- How do we know the optimal policy is the same?
  - $\max_a Q(s,a) - \Psi(s)$ is just $\left(\max_a Q(s,a)\right) - \Psi(s)$; that $\Psi(s)$ has no effect on argmax (as we'd use to find policy from $Q$)
- We can then modify $\Psi(s)$ in various ways and not change the optimal policy - but perhaps help us learn faster.
- Applying with Q-learning: for $\langle s,a,r,s' \rangle$,

$$Q(s,a) = Q(s,a) + \alpha_t \left( r - \Psi(s) + \gamma \Psi(s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Say that hypothetically $\Psi(s) = \max_a Q^*(s,a)$ (i.e. "true" Q)

- Then $Q(s,a) = Q^*(s,a) - \Psi(s) = \max_a Q^*(s,a) - \max_a Q^*(s,a)$ ?
  $= 0$ for "optimal" actions
  $< 0$ otherwise
- Q learning with potential ≡ Q-learning with Q function initialized to that potential ($Q_0(s,a) = \Psi(s)$)
  - The Q function differs, but the convergence at every step is identical, thus policy is identical.
  - But this also says that when you randomly initialize Q, you're making assertions on rewards
- Certain "optimistic" initializations can be shown to put bounds on convergence, but little is known past that.

2016-06-17

7. Exploring Exploration

- Exploration is sometimes seen as fundamentally what separates reinforcement learning & other machine learning
- Topics: Bandits
  Deterministic MDPs
  Stochastic MDPs

Readings:
Fong (1995)
Li, Littman, Walsh (2008)

- k-armed bandits
  - There are k different "one-armed bandits" (i.e. a slot machine). They all have different payoffs.
  - Which arm do we want to pull? Without knowing payoffs, we don't know.
  - Let's say each one has a probability in $[0,1]$ of giving a payoff of 1. So, we may estimate probability but it takes pulls to do so
  - Suppose k=7, bandits a, ..., g

| | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| Observed | $\frac{1}{20}$ | $\frac{2}{20}$ | $\frac{4}{20}$ | $\frac{1}{5}$ | $\frac{2}{10}$ | $\frac{4}{20}$ | $\frac{8}{20}$ |

  - Which has highest expected payoff? Which has highest confidence?

Chris Hodapp
chodapp3@gatech.edu

2016-06-17(b), CS8803-003 (RLDM)

7. (cont.)

- Note that d, e, f, g all have highest expected payoff ($\frac{1}{5}$) but g has highest confidence due to 40 trials, as does c (also 40).
- But if $a = 0/1$, $b = 1/2$ then b has both highest expected value and confidence. If we could only pick one handle, b would be best, but
  - If a & b are randomly chosen, then observing this pattern occurs 30% of the time if a is higher, 70% of the time if b is higher.
    So, picking b always leaves us with a worse option 30% of time.
  - Possibilities:
    maximum likelihood: Continuously update expected values and choose max likelihood each time.
    This is bad here because for $a = 0/1$, $b = 1/2$, we never choose a ($b$ is never $\leq 0$). It'd always be b.
    Really maximum maximum likelihood
    maximum confidence: Pick one with best estimates.
    Problem: Highest confidence pull just gets higher confidence
    minimum confidence: Tends to alternate; over time it gets better estimates, then ignores them.
  - But none of these are always better, compared with just always picking 'b'.
  - Max max likelihood represents exploiting, minimum confidence represents exploring, in simplest form.
  - You want to "eventually" pull the best arm but how do you get there?
- Metrics:
  1. identify optimal arm in the limit.
     By itself not optimal: round-robin does this.

2. maximize (discounted) expected reward.
   - One way to solve this: Gittons index, a function of successes & pulls which gives an ordering.
   - It's based on expected value given never pulling some bandit
   - Combines max max likelihood & min. confidence
   - Gittons index works really well on bandit problems, but has failed to really generalize elsewhere.
3. maximize expected reward over finite horizon.
4. identify near-optimal arm (within $\varepsilon$) with high probability $(1-\delta)$ in time $T(k, \varepsilon, \delta)$, polynomial in $k, 1/\delta, 1/\varepsilon$.
   - Note that we can't ever find true optimal arm exactly in any finite time with 100% probability!
   - This looks like PAC learning (think CS7641), Probably Approximately Correct (even though it's really 'Optimal' not correct)
5. nearly maximize reward ($\varepsilon$) with probability $(1-\delta)$ in time $T'(k, \varepsilon, \delta)$, polynomial in $k, 1/\delta, 1/\varepsilon$
   - Perhaps finding near-optimal arm is not necessary.
   - Perhaps it's even a distraction to reward.
6. Pull a non-near-optimal arm ($\varepsilon$) no more than $T''(k, \varepsilon, \delta)$ times with high probability $(1-\delta)$
   - Sort of mistake-bound rather than PAC.

T here is not transition function. Maybe use $\tau$

- But... 4, 5, and 6 are all basically equivalent!
  - If we can solve #4, we can use it to produce #6.
  - Say algorithm A solves #4 with $\varepsilon, 1-\delta, T(k, \varepsilon, \delta)$ pulls. Then run A and after $T(k, \varepsilon, \delta)$ pulls we have $\varepsilon$-close arm. Pull it forever.
    Then for #6, $T''(k, \varepsilon, \delta) \leq T(k, \varepsilon, \delta)$ - It can't have made more than $T$ mistakes to find that arm, and may have fewer. After $T$, we make no more mistakes (with probability $(1-\delta)$.)

Chris Hodapp
chodapp3@gatech.edu

2016-06-17c, CS8803-003 (RLDM)

7, cont.

- Start with #6 solution: Algorithm B that pulls $\epsilon$-suboptimal arms at most $\mathcal{L}''(k, \epsilon, \delta)$ times w/ prob $1-\delta$.
- We want #5: Algorithm that w/ prob $1-\delta'$ has a per-step reward $\epsilon'$ close to optimal after $\mathcal{L}'(k, \epsilon', \delta')$ steps.
- Suppose algorithm C runs for $\mathcal{L}'(k, \epsilon', \delta')$ steps... Suppose that of those, the first $\mathcal{L}''(k, \epsilon, \delta) < \mathcal{L}'(\ldots)$ steps are running algorithm B, thus at most $\mathcal{L}'' = m$ mistakes. Let $n = \mathcal{L}'(k, \epsilon', \delta')$, then $(n-m)$ steps remain after m. We can guarantee by algorithm B's definition ~~that reward for each step after is $\epsilon$, thus~~ that were $\frac{\epsilon(n-m)}{n}$-optimal for that step, so total: $\epsilon' \geq \frac{m + \epsilon(n-m)}{n}$
  - That is, B needs a tighter $\epsilon$. If we say $\epsilon = \frac{1}{2}\epsilon'$ then $n \geq m\left(\frac{2}{\epsilon'} - 1\right)$ - still polynomial in $\frac{1}{\epsilon'}$, but must run longer.
- Start with #5: Algorithm C that gets within $\epsilon$ (per step) of optimal after $\mathcal{L}'(k, \epsilon, \delta)$ pulls with probability $(1-\delta)$.
  - What's algorithm A, which, after $\mathcal{L}(k, \epsilon', \delta')$ pulls, returns arm that is $\epsilon'$ optimal? (w/ prob $(1-\delta)$.)
  - Run C...
  - Let $e_i$ = Suboptimality of the plurality arm chosen by C (arm chosen as much or more than others) → per-step error
    - It's chosen at least $\frac{t}{k}$ of the time, so $\epsilon \geq e_i \cdot \frac{t}{k} \rightarrow e_i \leq \epsilon k$
    But we want $e_i \leq \epsilon'$, so let $\epsilon = \epsilon'/k$

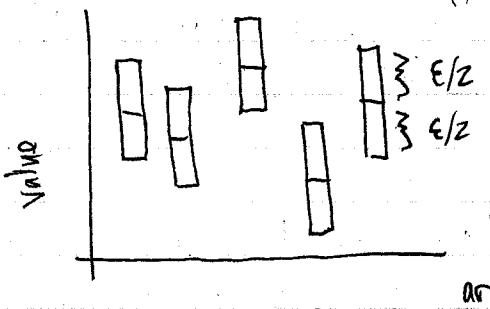
Must be polynomial in $\frac{1}{\epsilon'}$ too

- Find best → Few mistakes → Do well → Find best
  - We've shown all 3 arrows.
  - We need not pick, then.
- But first we need to show that any of these are solvable, without another solution first.

→ Hoeffding, aka. tail-bounds, Chernov

Hoeffding bounds:

- Let $X_1, \ldots X_n$ be i.i.d with mean $\mu$. Let $\hat{\mu}$ be our estimate of $\mu$: $\sum_i x_i/n$. The following is a $100(1-\delta)\%$ confidence interval for $\mu$: $\left[\hat{\mu} - \frac{z_\delta}{\sqrt{n}}, \hat{\mu} + \frac{z_\delta}{\sqrt{n}}\right]$ where $z_\delta = \sqrt{\frac{1}{2} \ln \frac{2}{\delta}}$
  - More data → larger $n$ → smaller bounds
- So... Take $c$ samples of each arm ($c$ is a function of $\delta, \epsilon, k \ldots$) then choose arm with best estimate. We want to be $1-\delta$ sure it's within $\epsilon$. Learn each arm to within $\epsilon/2$ within probability $1-\delta/k$. To visualize:




Those are uncertainty bars for estimate.
Each uncertainty is true with probability $1-\delta/k$.

What is probability that all are correct? $(1-\delta/k)^k$
assuming independence, but this is difficult & risky
Instead say $k \cdot \delta/k$ that at least one is wrong - so $1-\delta$ that all are right, ignoring independence.
"union bound" It's an upper bound that's easier to work with.

- So, how many samples do we need to be $\epsilon/2$ accurate w/ prob $1-\delta/k$?
  Note from Hoeffding bounds that $\frac{z_\delta}{\sqrt{n}} = \frac{\epsilon}{2}$ (see the $\pm$),
  so: $\frac{\sqrt{\frac{1}{2} \ln \frac{2k}{\delta}}}{\sqrt{c}} \leq \frac{\epsilon}{2} \rightarrow c \geq \frac{2}{\epsilon^2} \ln \frac{2k}{\delta}$

Note similarity to agnostic learner & PAC

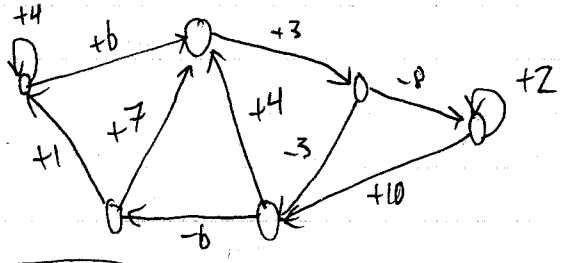
Note how dependence on $\delta$ is very weak (denominator & log), but $\epsilon$ matters much more

Chris Hodapp
chodapp3@gatech.edu

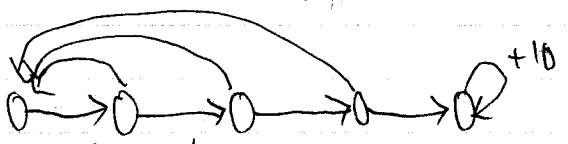
2016-06-17d, CS8803-003 (RLDM)

7. (cont.)

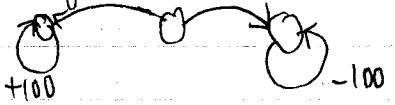
- Exploring deterministic MDPs
  - How do we go about this, knowing only the actions we can take?




- MDP optimization criteria:
  1. Explore randomly:
     problems with things like above




  2. Trap states:



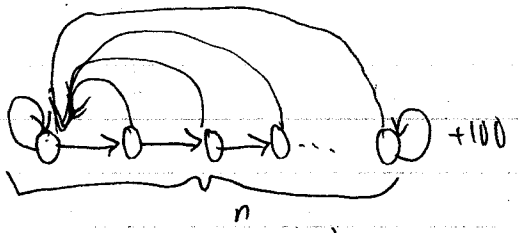

  3. Mistake bound
     - Number of $\varepsilon$-suboptimal action choices is bounded in polynomial on $\frac{1}{\varepsilon}, \frac{1}{\delta}, n, k$ w/ prob $1-\delta$

See similarity to bandits →

- Rmax: Assume every unknown reward is good, loosely.
  1. Keep track of MDP.
  2. Any unknown state-action pair is Rmax.
  3. Solve the MDP.
  4. Take action from $\pi^*$.
- Once all edges known, no more mistakes occur (counting mistakes per-state)
- We may stop visiting unknown states for some $\gamma$ & Rmax
  - But, these aren't mistakes! At least not without discounting. This is contingent on Rmax actually being highest possible value.
- How many mistakes might we make getting to a suboptimal state? It's up to $n$ for $n$ states. How many times can this happen? For $k$ actions, $nk$ - so at most $n^2k$ mistakes (thus polynomial in $n, k, \frac{1}{\varepsilon}, \frac{1}{\delta}$)

- Finally: What about stochastic MDPs with Rmax?

- Lower bound
  - "Combination lock" MDP:




A 'sequential' combo lock. Not very secure.

    - assume actions are randomized w.r.t. forward/backwards
    - If we're in state $i$, it takes a minimum of $i$ to get there
    - So, $1+2+3+4 \ldots +n = \Omega(n^2)$ worst-case
    - If $k$ actions ($k-1$ go backwards, 1 goes forward) then... $kn^2$

Hoeffding = stochastic
Rmax = sequential

- General stochastic MDP
  - Combine Hoeffding & Rmax.
  - General Rmax: (nearly the same)
    1. Keep track of MDP with T & R
    2. Any unknown state-action pair is Rmax
       - We consider $(s,a)$ unknown if tried less than $c$ times. Past that, use ML estimate.
    3. Solve MDP.
    4. Take action from $\pi^*$
- Simulation Lemma
  - Transitions & rewards off by $\alpha$ or less

Fixed $\pi$

$V_1^\pi(s)$ $\quad$ $V_2^\pi(s)$ $\quad\rightarrow\quad$ $\max_{s,a,s'} |T_1(s,a,s') - T_2(s,a,s')| \leq$

$T_1, R_1$ $\quad$ $T_2, R_2$

$\alpha \max_{s,a} |R_1(s,a) - R_2(s,a)| \leq \alpha$

  - Define $G = \alpha + \gamma n \alpha R_{max}/(1-\alpha)$

$\varepsilon = |V_1^\pi(s) - V_2^\pi(s)| \leq G/(1-\gamma) \rightarrow \alpha = \dfrac{\varepsilon(1-\gamma)}{1+\gamma n \frac{R_{max}}{1-\gamma}}$

Use Hoeffding union bound again

    - But how do we turn $\alpha$ to $c$? (How many $(s,a)$ samples do we need before we're confident?)
- Explore/Exploit Lemma: If all transitions are either accurately estimated or unknown, optimal policy is either near optimal or an unknown state is reached "quickly" (e.g. polynomial bounds)

KWIK framework = know what it knows

Chris Hodapp
chodapp3@gatech.edu

2016-06-21, CS8803-003 (RLDM)

8. Generalization

Readings:
Gordon (1995)
Baird (1995)

- It helps to refer to states like collections of features, not just blobs.
- This can relate to <u>inductive bias</u>. If (for instance) a feature is the distance between a person and a taxicab, then all states where they're 2 units apart look the same
- What are we actually trying to generalize?
  - policy, states → actions
  - Or value function (policies are harder)
  - Maybe we should use similar actions in similar states
  - Maybe estimate: state/action → estimated return
  - Maybe: Estimate model T\R (but in practice these need to work several steps ahead)
  - So usually value functions are the approach
- General form:

$$Q(s,a) = F(w^a, F(s))$$

For $\langle s, a, r, s' \rangle$:

$$\Delta w_i^a = \alpha \underbrace{\left(r + \gamma \max_a Q(s',a')\right)}_{\text{Bellman eq.}} - Q(s,a)) \frac{\partial Q(s,a)}{\partial w_i^a}$$

(Gradient step; TD error)

  - Thus we can view Q-learning as an update rule for underlying parameters of a function approximator.
  - We're <u>bootstrapping</u> here: Using predictions to guide new predictions.
    - Contrast with supervised learning
- Linear value function approximation

$$Q(s,a) = \sum_{i=1}^{n} f_i(s) \cdot w_i^a$$

(Dot product; parameters provide generalization)

Each action gets a set of n weights to represent the Q values for that action. Weights give "importance" of all the features in contributing to an action's value.

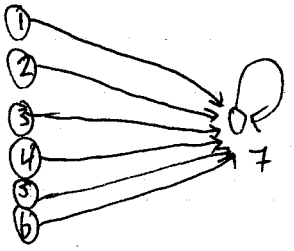
Pollack had a paper too on how backgammon is amenable to genetic algorithms?

8 cont.

- Successes: 3-layer backprop (Tesauro, Wang/Dietterich) - but difficult → Backgammon, Watson & Jeopardy
  CMAC (Sutton) - like neural net but not at 1st layer → Shuttle strategies (NASA)
  Linear nets (Singh, Parr)
  Deep nets (Minh. et al., Atari) - actually used convnets on pixels & learned actions
- Features need to be: Computationally efficient
  Value informative
  - e.g. a feature that just encodes value function is very useful! but not too efficient.

Print & read: Tesauro, Boyan/Moore, Sutton

- Baird's Counterexample




| State | Feature $w_0$ (0) | $w_1$ (1) | $w_2$ (2) | $w_3$ (3) | $w_4$ (4) | $w_5$ (5) | $w_6$ (6) | $w_7$ (7) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | | | | | | 0 |
| 2 | 1 | | 2 | | | | | |
| 3 | 1 | | | 2 | | | | |
| 4 | 1 | | | | 2 | | | |
| 5 | 1 | | | | | 2 | | |
| 6 | 1 | | | | | | 2 | |
| 7 | 7 | 0 | — | | | | | 1 |

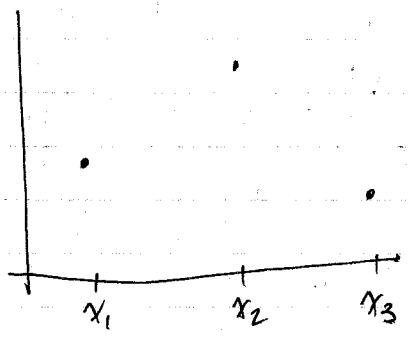
- No rewards, deterministic, no choices
- 8 features in feature vector
- Linear value function approximation
- It's nearly tabular.
- Can we represent the 'real' value function (which is all 0s)? One way is all weights 0.
  - But another way is: $w_0 = -1$ $\quad w_1 \dots w_6 = \frac{1}{2}$
    $w_7 = 7$
    - It's actually infinitely representable.
- Start with Q(s,a) on prior page:
  $V(1) = w_0 + 2w_1$, $V(2) = w_0 + 2w_2$, $V(3) = w_0 + 2w_3$, $V(4) = w_0 + 2w_4$,
  $V(5) = w_0 + 2w_5$, $V(6) = w_0 + 2w_6$, $V(7) = 7w_0 + w_7$
- Assume $w_i > 0$, $\gamma = 0.9$, $\alpha = 0.1$, and $V(7) \gg\gg V(i)$
  Update round-robin: $\langle 1,0,7\rangle$, $\langle 2,0,7\rangle$, $\langle 3,0,7\rangle$, $\langle 4,0,7\rangle$, $\langle 5,0,7\rangle$, $\langle 6,0,7\rangle$, $\langle 7,0,7\rangle$, let $w_i = 1$
  - This increases all weights and this spiral just continues

Chris Hodapp
chodapp3@gatech.edu

2016-06-21, CS8803-003 (RLDM)

8 cont.

- Then, transition $\langle 7, 0, 7 \rangle$ reduces $w_0$, but not enough to reduce what rest of sequence added. The updates never converge
- If $w_i = 0$ then weights all just stay at 0.
- The exact answer is "sticky" (if deterministic) - TD error = 0
- But shared weights can diverge (as they do here)
- Averagers
  - Function approximator introduced by Gordon
  - Consider:




- We want to approximate a function, and we want to do it by setting the values of $x_1, x_2, x_3$ on the left.
- In between $x_1$ & $x_2$, and $x_2$ & $x_3$, we average. (Must be a convex combination of anchor points.)

- Outside of anchor points, we need some defined behavior - but still must be convex combination (thus, it can't ever go "outside")
- $V(s) = \sum_{s_b \in B} \overbrace{w(s, s_b)}^{\text{Predefined weights}} \underbrace{V(s_b)}_{\text{Parameters}}$ s.t. $w(s, s_b) \geq 0 \quad \forall s, s_b$ and $\sum_{s_b \in B} w(s, s_b) = 1 \quad \forall s$

  $\overbrace{\quad}^{\text{Basis/anchor}}$

- Note that k-NN is an averager if distance-weighted
- Connection to MDPs:

$$V(s) = \max_a \left( R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right)$$
$$= \max_a \left( R(s,a) + \gamma \sum_{s'} T(s,a,s') \sum_{s_b \in B} w(s', s_b) \cdot V(s_b) \right)$$
$$= \max_a \left( R(s,a) + \gamma \sum_{s_b \in B} \left( \underbrace{\sum_{s'} T(s,a,s') w(s', s_b)}_{T'(s,a,s_b)} \right) V(s_b) \right)$$
$$= \max_a \left( R(s,a) + \gamma \sum_{s_b \in B} T'(s,a,s_b) V(s_b) \right)$$

$T'(s,a,s_b)$ – Acts like transition ($\geq 0, \sum_{\ldots} = 1$)

LSPI
= least-squares policy iteration

Will need to re-read re-watch...

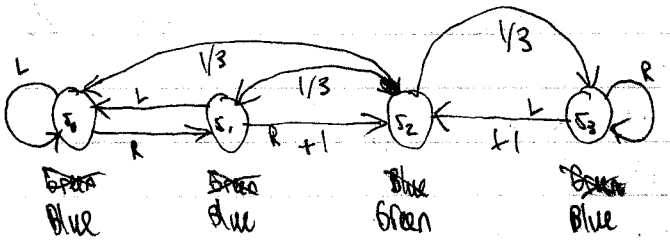
- So this is a sensible approximator, equivalent to a smaller set of basis states. Convex ~~[illegible]~~ combinations look a lot like probabilities
- GTD, different kind of TD using different type of gradient
  - GTD2?
  - Provable convergence
  - Incorporates function approximation itself into loss function
- Go-to algorithm: Q fitted iteration, combining Q-learning & function approx.

2016-06-25

9. Partially Observable MDPs

Readings: Littman (2009)

- POMDP: Way of talking about "non-Markov" environments
  - A normal MDP is "inside", but not observable
    - $A, S, T, R$ - normal MDP
    - $Z$ - observables → Observation function
    - ~~$O(S,Z)$ = probab~~ $O(s,z)$ = probability of seeing $z$ in state $s$ $(z \in Z, s \in S)$
  - O gives a sort of hint at the hidden MDP
  - Agent never really knows current state
- Any MDP has an equivalent POMDP.
  - MDP: $\langle S, A, T, R \rangle$
  - POMDP: $\langle S, A, Z, T, R, O \rangle$
  - } identical $S, A, T, R$, and then $Z = S$ & $O(s,z) = \begin{cases} s=z: 1 \\ \text{else}: 0 \end{cases}$
- POMDP example




Any action from ~~[illegible]~~ green state ($s_2$) transitions randomly to $s_0, s_1$, or $s_3$.
Reaching ~~[illegible]~~ green from blue = +1

Now, suppose we see only green or blue, not the exact state.
If we start at $s_0, s_1$, or $s_2$ with prior probability $\frac{1}{3}$, and we take action L & see blue, then from there take action R and see blue, and from there take L, what is probability that we see blue, and that we see green?

2016-06-25(b), CS8803-003 (RLDM)

9. POMDPs (cont.)

- It's $\frac{7}{8}$ & $\frac{1}{8}$, respectively, but requires some derivation.
  - See lecture video #5
  - It involves some backtracking & revision of prior probabilities. That is, we may eliminate the possibility that we were in some state prior, based on later information.
- Belief state $b(s)$ = probability/belief that we're in state $b$.
  If we have: $b$ (current belief), $a$ (action), $z$ (observation), how do we produce $b'$ (belief of next state)?
  - We can treat reward as not observed - or as observation already giving this information, or $r = f(z)$

$b'(s')$ = probability of being in state $s'$ after $b, a, z$.

$$Pr(s'|b,a,z) = \sum_s \underbrace{Pr(s|b,a,z)}_{\text{Possible start state probability}} \cdot \underbrace{Pr(s'|b,a,z,s)}_{\text{If we were in } s, \text{ what probability of } s'?}$$

$$= \sum_s b(s)\, Pr(s'|a,z,s) = \sum_s b(s) \frac{Pr(z|s',a,s)\, Pr(s'|a,s)}{Pr(z|a,s)}$$

Bayes rule (P(state|observation) → P(observation|state))

But probability that we observe $z$ depends only on $s'$, not $a$ & $s$, $Pr(s'|a,s)$ is just transition model, $Pr(z|a,s)$ is just normalization...

"Belief MDP"

$$= \frac{\sum_s b(s)\, O(s',z)\, T(s,a,s')}{\sum_s \sum_{s''} b(s)\, O(s'',z)\, T(s,a,s'')}$$

- so that's $b'(s')$, and $b'$ as a whole gives a distribution over states

- So, we sort of turned a POMDP into an MDP with an infinite number of states - so our existing algorithms can't handle it.
- Value Iteration in POMDPs:

$V_0(b) = 0$

$V_t(b) = \max_a \left( R(b,a) + \gamma \sum_z Pr(z|b,a) \cdot V_{t-1}(b') \right)$ where $b' = SE(b,a,z)$

(All observations; State Estimation)

↓ but R's not defined this way... it's just 'expected reward' or some such

$R(b,a) = \sum_s Pr(s) \cdot R(s,a) = \sum_s b(s)\, R(s,a)$ - basically dot product.

But, $V_t$ is now defined over an infinite number of belief states

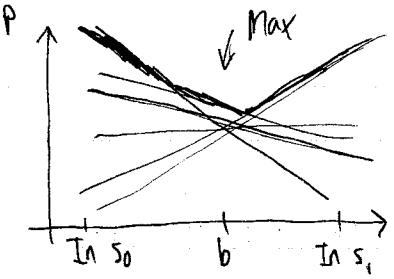
Claim: $\forall t$, $V_t(b) = \max_{\alpha} \alpha \cdot b = \max_{\alpha \in \Gamma_t} \sum_s b(s)\, \alpha(s)$ (over all beliefs $b$)

"piecewise linear and convex"

$\Gamma_t$ is some finite set of vectors

- Consider a belief over states $s_0$ to $s_1$:

See: Sondik's work




← We want max of this, where any given b is a linear combination of those lines.

Under this claim we can ignore the finite number of b, and just deal with a finite number of linear functions (convex & combined piecewise)

- Base case of inductive proof:

1st of them

$\exists$ set $\Gamma_0$ s.t. $V_0(b) = \max_{\alpha \in \Gamma_0} \alpha \cdot b$ — $\Gamma_0 = \{[0, \dots, 0]\}$

note that $V_0(b) = 0$ and $|b| = b$

- Inductive step:

Assume we have $\Gamma_{t-1}$ s.t. $V_{t-1}(b) = \max_{\alpha \in \Gamma_{t-1}} \alpha \cdot b$

Build $\Gamma_{t-1}$ s.t. $V_t(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b$

$$V_t(b) = \max_{a \in A} V_t^a(b) \quad \leftarrow \text{Parts of Bellman update}$$

$$V_t^a(b) = \sum_z V_t^{a,z}(b)$$

$$V_t^{a,z}(b) = \sum_s \left( \frac{R(s,a) \cdot b(s)}{|z|} \right) + \gamma \Pr(z|b,a) V_{t-1}(SE(b,a,z))$$

Can we represent these as a "bag of vectors"?

→ For $V_t(b)$, sort of yes: $\Gamma_t = \cup_a \Gamma_t^a$ (max can be written as an operation over sets of vectors)

↳ Union

For $V_t^a(b)$, if $V_t^{a,z}(b)$ can be represented as a sum over bags of vectors, then so can $\sum_z V_t^{a,z}(b)$ — thus, so can $V_t^a(b)$.

→ $\Gamma_t^a = \oplus_z \Gamma_t^{a,z}$

↳ = "cross sum" (?)

For $V_t^{a,z}(b)$ first expand $V_{t-1}(SE(b,a,z)) = \max_{\alpha \in \Gamma_{t-1}} \frac{\sum_{s'} \alpha(s') O(s',z) \sum_s T(s,a,s') b(s)}{\Pr(z|a,b)}$

$\Pr(z|a,b)$ in denominator is highly nonlinear, but look at $V_t^{a,z}(b)$ formula — we multiply by $\Pr(z|b,a)$ which is the same, so they cancel, and numerator is linear.

Then $V_t^{a,z}(b)$ is just sum of two dot products over b.

Chris Hodapp
chodapp3@gatech.edu

2016-06-26, CS8803-083 (RLDM)

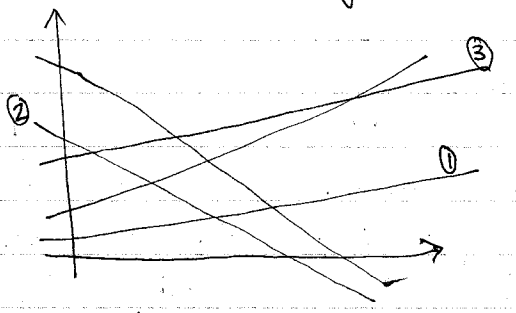
9. POMDPs (cont.)

- Then...

$$\Gamma_t^{a,z} = \left\{ \frac{1}{|z|} R(s,a) + \gamma \sum_{s'} \alpha(s') O(s', z) T(s, a, s') \;\middle|\; \alpha \in \Gamma_{t-1} \right\}$$

$$|\Gamma_t| = |\Gamma_{t-1}|^{|z|} \cdot |A|$$ - exponential, but finite.

- BURLAP has code for this. Also see pomdp_solve from Tony Cassandra.
- $V_t^a(b)$ from last page is actually the Q function
- How do we effectively implement this in an algorithm though?




We want the max over these linear functions to give us a piecewise linear convex function. - the upper surface here.

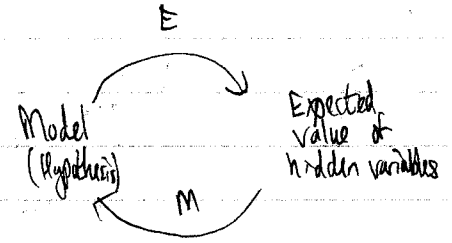
Note that we can ignore vectors that don't form a part of this - a purge operation. (it's a linear programming solution) For instance, vector 3 is always greater than vector 1, so #1 can be purged. This is fast, but not sufficient. Vector 2 is dominated by every other vector at some point, but no single vector at all points, for instance.

(One-step lookahead on value function)

- We covered planning (ish), but what about learning?
  - But note that while VI gives optimal policy in finite iterations even though value function converges only in the limit. With POMDPs, this doesn't hold due to infinite states. We might however get near-optimal VF and from it a policy.
  - Planning in POMDPs is actually formally undecidable, equivalent to halting problem, but that's another matter.
  - Recall model-based & model-free RL - former learns a model, latter does not. That distinction holds in POMDPs too.

# Learning a POMDP

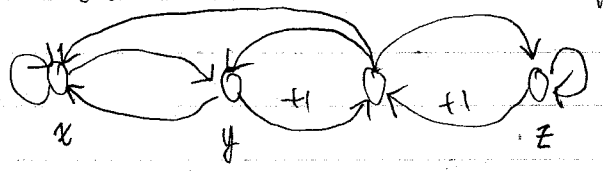
| | Uncontrolled | Controlled |
|---|---|---|
| Observed | Markov chain | MDP |
| Partially Observed | Hidden Markov model | POMDP |




- EM (expectation maximization) can work here - recall
  - We used EM to solve Gaussians, but they also work with HMMs & POMDPs

2016-06-27

: Recall the POMDP from a few pages ago:




- We can't do particular sequences if we are using a <u>memoryless</u> policy (e.g. only taking some action based only on observed color, never past actions)
- Suppose we want optimal policy (can be stochastic) for observing blue, assuming $\gamma = 1/2$, and equal priors on $x|y|z$. Start with $\Pr(L) = 1/2$, $\Pr(R) = 1/2$. Let $x, y, z$ above be the expected rewards from each state.

$$z = \underbrace{\tfrac{1}{2}(0 + \tfrac{1}{2}z)}_{\text{Right} \to \text{no reward, but repeat at } \gamma} + \underbrace{\tfrac{1}{2}(1)}_{\text{Left} \to +1 \text{ \& end}} = \tfrac{1}{4}z + \tfrac{1}{2} \to z = \tfrac{2}{3}$$

$$x = \underbrace{\tfrac{1}{2}\left(\tfrac{1}{2}x\right)}_{\substack{\text{left} \\ \text{(both discounted)}}} + \underbrace{\tfrac{1}{2}\left(\tfrac{1}{2}y\right)}_{\text{right}}, \quad y = \underbrace{\tfrac{1}{2}\left(\tfrac{1}{2}x\right)}_{\substack{\text{left} \\ \text{(discounted)}}} + \underbrace{\tfrac{1}{2}\cdot 1}_{\substack{\text{right} \\ (+1)}} \longrightarrow \begin{matrix} x = 2/11 \\ y = 6/11 \end{matrix}$$

Then, weighting $x, y, z$ for priors on each state:

$$\tfrac{1}{3}\cdot\tfrac{2}{11} + \tfrac{1}{3}\cdot\tfrac{6}{11} + \tfrac{1}{3}\cdot\tfrac{2}{3} = 0.4646\ldots$$

What if we try $\Pr(L) = 1/3$, $\Pr(R) = 2/3$? Reward is then $1/2$.

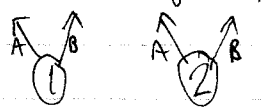
- We can think of RL as itself being an MDP (bayesian RL); we keep a Bayesian posterior over possible MDPs and then we simply <u>plan</u> rather than <u>learn</u>.

Chris Hodapp
chodapp3@gatech.edu

2016-06-27(6), CS8803-003 (RLDM)

9.
POMDPs (cont.)

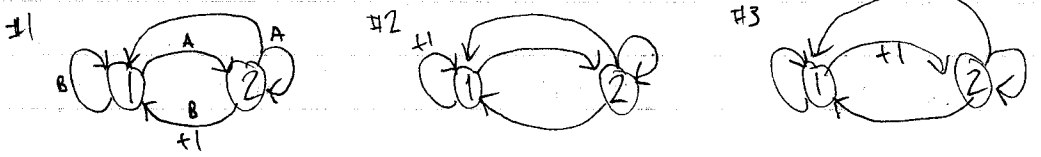
- Consider really simple problem:




$\gamma = \frac{1}{2}$

What action (A or B) should we take?

- This could be 3 MDPs:




Only position of reward changes. If we treat each MDP as equally likely, then we have... (for belief states)

| MDP | 1 | 2 (state) |
|---|---|---|
| #1 | 1/3 | 0 |
| #2 | 1/3 | 0 |
| #3 | 1/3 | 0 |

And suppose we see...
state 1
action A
reward +0

| | | |
|---|---|---|
| #1 | 0 | 1/2 |
| #2 | 0 | 1/2 |
| #3 | 0 | 0 |

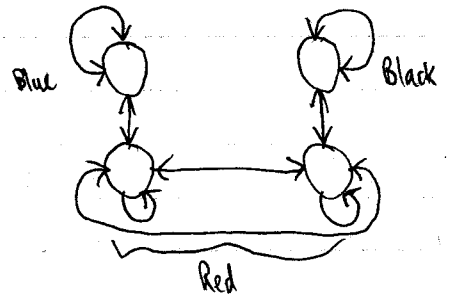
← Then

(0 because the +0 reward eliminates this)

We've eliminated the possibility that we started in MDP #3.

- "optimal policy" tries to balance figuring out which MDP (explore), and getting highest reward (exploit). Bayesian approach is always exploiting in belief space - but this ends up also exploring in the process
- In this sense, RL is actually a sort of planning on a continuous POMDP. The hidden state is the parameters of the MDP, and as this is infinite, the answer is no longer piecewise, linear, convex... but it is piecewise polynomial and convex, though degree of polynomial grows.
- See BEETLE algorithm (approximates polynomial), Poupart et al.
- At the moment, it appears too expensive to be practically useful. It gives, indirectly, a probability distribution over Q functions.

- Predictive State Representation
  - POMDP: Belief state = distribution over states, but states are never observed. Do they even exist?
  - PSR: Predictive state is probabilities of future predictions (instead of these maybe-fictional states).
    - Example:




Observations: Blue, Black, Red
Actions: LRUD (left, right, up, down)

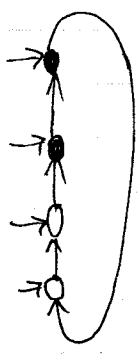
Test #1: If you go up, do you see blue? (and what probability?)
Test #2: If you go left, do you see red?

- In some sense: A distribution over states is replaced with a distribution of outcomes of tests. Importantly, these tests are actually observable.
- Suppose for the above POMDP we have belief state:

| | |
|---|---|
| 0.1 | 0.2 |
| 0.4 | 0.3 |

→ 0.5 ↑ blue, 0.7 ← red } Predictive state

- We can convert belief state to predictive state, but generally don't have a unique answer for the reverse (i.e. multiple belief states might satisfy it). More tests may resolve it...
- PSR Theorem: Any n-state POMDP can be represented by a PSR with no more than n tests, each of which is no longer than n steps long.
- Multiple-step tests, e.g. ↑0 ↑● ? 0
  ↑● ? 0



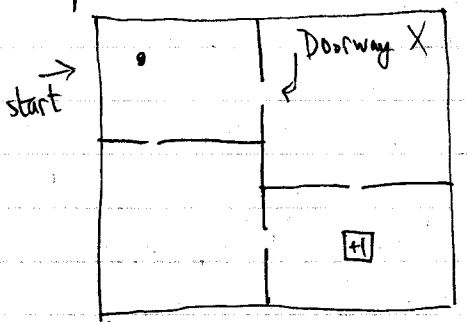
Chris Hodapp
chodapp3@gatech.edu

To read:
Sutton, Precup, Singh (1999)
Jong, Hester, Stone (2008)

2016-07-05, CS8803-003 (RLDM)

10. Options

- What makes RL hard (from an algorithmic perspective)?
  - Delayed reward & indirect learning (unlike SL)
  - Bootstrapping / need for exploration
  - #states, # action
    - This is where function approximation helps (and general idea of learning about nearby states)
- Temporal abstraction




Consider this usual gridworld. Actions are ↕↔ but with chance that you move at a right angle to intended direction.

How would we describe to a person how to navigate this? We might tell them to go to the doorway to the east, then to the south. In effect we've expanded the action space.
This is a form of temporal abstraction.
For doorway X, the result of various starting states is identical once through that doorway.

- Think of Q-learning or value iteration. With the normal actions, the reward at +1 propagates very slowly. We must visit states like the state right in the doorway very often in order to actually propagate data out of it.
- But with these 'abstracted' actions we may propagate information much more readily - basically between 'rooms' and bigger actions, not state-to-state.
- N.B. Function approximation treats many states like one. Temporal abstraction, as here, treats many actions together.

- It then has a side effect of many states being treated together (e.g. from each 'room' - many states. the action is the same).
- To treat this more formally: Options

Option = $\langle I, \pi, \beta \rangle$

I = initiation set of states - where that 'option' is legal to execute

$\pi$ = policy $(s,a) \to [0,1]$ → probability of being in state $k$ taking action

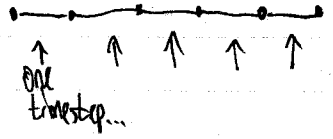
$\beta$ = termination set of states,

$s \to [0,1]$ - probability that option ends in state

$$V_{t+1}(s) = \max_o \left( R(s,o) + \sum_{s'} F(s,o,s') V_t(s') \right)$$

- Basically the Bellman equation, but generalizing actions to options. An action can be trivially turned to an option. Also: no discount factor, and F instead of T. (Sort of. Discount is hidden.)
- R(s,o) also hides a lot of complexity.

Consider an MDP:




We treat actions as atomic/instant, and all timesteps as identical. In options, though, this treatment is rather inaccurate. The 'actions' here all have some timestep that elapses, and the length of the option is variable. To capture this:

$R = E[r_1 + \gamma r_2 + \gamma r_3^2 + \dots \gamma^{k-1} r_k \mid s, k \text{ steps}]$

- That is: average reward expected by following this option, with discounting on true timesteps.
- Applying similarly to F: $F = E[\gamma^k \Delta_{s_k, s'} \mid s, k \text{ steps}]$

→ Expectation of ending up in $s'$ after $k$ steps, from $s$.

$\Delta$ = delta function: =1 if $s_k = s'$, else 0

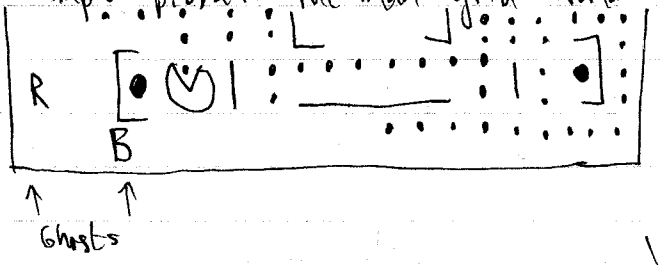
That $\gamma^k$ is to discount the V that is multiplied after.

Chris Hodapp
chodapp3@gatech.edu

2016-07-05b, CS8803-003 (RLDM)

10. Options (cont)

- We've created an SMDP, semi-MDP; in an SMDP, we may have 'jumps' in time. With the given equation though, we can just treat it like an MDP and solve it that way. (Only 'semi' because we must know how much time has passed.)
- Example problem: Pac-Man grid world




  - Normal rules - power pellets (with limited time), normal pellets, walls, ↔ motions
  - What kind of 'options' exist here? What options are effective?
    - Eat pellet
    - Eat power pellet
    - Eat ghost
    - Avoid ghost
- From options we get both high-level thinking and <u>separation of concerns</u>
- State includes: Location of dots, Pac-Man, ghosts, whether ghosts are fleeing
  - However, in line with separation of concerns, note that some parts of state are completely irrelevant depending on the option.
  - If we want to eat the big dot (& treat that as an option), we really just need to know where Pacman is and where dots are, and can sort of ignore the goal of "avoid ghosts that aren't scared so Pacman isn't eaten".

2016-07-06

- Options have to be made in the right way that they still lead near an optimal solution - but also simplify the problem

- Optimality can't necessarily be guaranteed with options, just 'with respect to' options

- Inherit optimality ← hierarchical optimality
  - Convergence — Agent does the best it can,
  - Stability — respecting those options
- Avoid 'boring' parts of the MDP - which helps with exploration
- State abstraction
- Options help us focus on parts of the MDP where decisions actually matter
- Goal abstraction
  - We can conceptualize all options as being in effect simultaneously (almost like processes on a computer)
  - Temporal abstraction in that sense is sort of managing competing goals
- Pac-Man is a sort of predator-prey problem (goal is to eat but while avoiding being eaten)
- β in option, from goal abstraction, is sort of the probability of success, or information on another goal needing to interrupt & take control.
- 'Primitive' actions compared to options don't really correspond to goals
- This becomes a problem of arbitration
  - Area is also called "modular RL" (modules = options)
  - Methods: Greatest Mass Q-learning
    - each option has an associated Q function and this looks for largest ? 'Modules' vote on action

Drawback: this action might not be useful for any one agent

Top Q-learning
- Just pick biggest Q value
- Problem: Agent that gave largest Q value to action might have given nearly same Q value to all others (thus, it doesn't have strong preference)

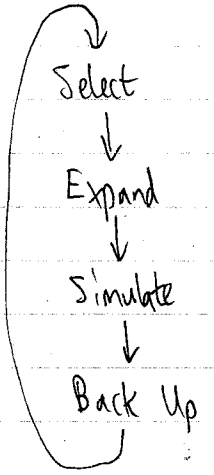
Negotiated W-learning
- More complicated, but loosely, agent with the most to lose decides on policy

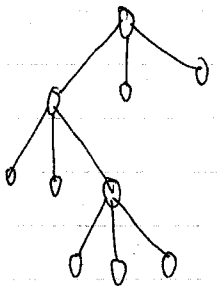
Chris Hodapp
chodapp3@gatech.edu

2016-07-06(b), CS8803-003 (RLDM)

10. Options (cont.)

- Problem with all of these: They're terrible/impossible.
  → Arrow's impossibility result for voting: It's not possible to make a fair system for voting; comparisons aren't ever meaningful between agents.
  Goals aren't necessarily compatible with each other, even if they all are aiming at some larger goal.
- Monte Carlo Tree Search



- Consider this state-action tree...

- Select: Pick an action and proceed, and eventually you'll hit a point where you don't know enough to know how to select.
- Expand: Expand tree at that point.
- Simulate: Do simulation to decide what you ought to be doing in those states
- Back Up: Like in Bellman equation, back those results up the tree, in order to learn what to do in next 'select'

Compare with A*, game search

- The 'monte carlo' part: How do we take the randomness into account? This is where simulation comes in.
- Select step: We pick based on our policy - as far as it takes us in the tree. Expand the tree where needed - perhaps one node & action at a time, simulating to get results.

- For simulation, follow some 'rollout' policy (let's just say it's random) to get results from each state we just expanded. This gives an estimate of being in that state & taking that action - that is, estimates $\hat{Q}(s,a)$.
- We may back this information up the tree. Then we have a Q estimate up there and can simply follow a greedy policy there...
- Followed repeatedly, this gives an ever-expanding tree with more & more confidence. Run infinitely it would converge on proper Q value.
- When do we stop? It may depend on time or other resource limits. We may run this at every timestep, for instance, and take 50 msec, or 5 seconds, or whatever, at every turn - instead of, say, trying to run for 3 ½ months and just solve MDP altogether.
- The 'rollout' policy $\pi_r$ mentioned above: Saying it's random isn't exactly great; how do we choose $\pi_r$?
  - (in Pacman example) Note that for 3 goals - eat ·, eat ●, eat 👻 - you meet them and they end when you succeed. For another goal - avoid 👻 - you can't succeed, you can only fail.
  - That latter category is called constraints.
  - This gives a hint of how we might do better than random. Perhaps we should behave randomly while following constraints.
  - Also, we can use options in that tree search. Following options gets us deeper into that tree, which is more efficient.
- It's a sort of policy search algorithm at this point.
- Some nice properties: (good & bad)
  - Useful for large state spaces.
  - Needs lots of samples for good Q estimate
  - Planning time is independent of $|S|$.
  - Running time exponential in the horizon (how far in the future you must look) - $O((|A| \cdot x)^H)$. $x$ = # steps, $H$ = horizon, $|A|$ = # actions

Chris Hodapp,
chodapp3@gatech.edu

2016-07-06c, CS8803-003 (RLDM)

10. Options (cont.)

- Horizon:
  - Depends on discount factor. Reasonable discount factor requires you to look further out.
+ Generalization past function approximation:
  - temporal abstraction ~ options & semi-MDPs (hierarchical RL)
  - goal abstraction ~ Modular RL & arbitration
  - state abstraction
- Quiescence: When search backtracking has quieted down?
- MCTS (Monte Carlo Tree Search) has been used in many games, for instance.
- Part of why temporal abstraction is useful is because people can help by providing intuitive answers/options to help reduce state space greatly.

11, A: Game Theory III, & IIB

- This is a repeat from CS7641... See RL-3. & RL-4.

- Readings: Greenwald & Hall (2003)

11C: Game Theory Revolutions

- Solution Concepts - A rule for predicting how a game's going to be played
  - Nash - Emphasis on equilibrium (discussed a lot already)
  - (Correlated) - Still equilibrium, but not Nash, & better in some ways
  - Trembling Hand
  - Stackelberg
  - Subgame Perfect
  - Coco
  - Bayes Nash
  - And many others...

We ignore most of these & focus on correlated equilibrium

Readings: Zeibart et al. (2008)
Babes et al. (2011)
Griffith et al. (2013)
Cederborg et al. (2015)
Roberts et al. (2006)
Bhat et al. (2007)

11C (cont.)

- Correlated equilibrium
- First, consider the game of chicken. We can write the matrix:

| | D | C |
|---|---|---|
| D | 0,0 | 7,2 |
| C | 2,7 | 6,6 |

D = dare, C = chicken

- Consider 4 pure strategies:
  - C,C: Not Nash
  - C,D: These are Nash equilibria
  - D,C: These are Nash equilibria
  - D,D: Not Nash
- But neither equilibrium is really "stable". Neither player really wants to chicken out.
- The mixed strategy that is a Nash equilibrium is for both to dare with probability 1/3. Then, expected outcomes: (derivation not shown)
  - C,D: $p = (\frac{1}{3})(\frac{2}{3}) = \frac{2}{9}$, reward (to player 1) = 2
  - D,C: $p = \frac{2}{9}$, reward = 7
  - D,D: $p = (\frac{1}{3})^2 = \frac{1}{9}$, reward = 0
  - C,C: $p = (\frac{2}{3})^2 = \frac{4}{9}$, reward = 6

  so expected reward for player 1 (and 2 since it's symmetric) = $4\frac{2}{3}$.
- That looks like a win, but can we do better?

- What if we have a 3rd party who picks a card with (C,C), (D,C), or (C,D), randomly & uniformly?
  - He then tells player 1 what the card says for player 1, and player 2 for player 2. So, players 1 & 2 both know what cards exist, but neither player is told what the other is.
  - Players 1 & 2 do not have to obey what they're told.
- If player 1 sees 'D' from 3rd party, what should he do?
  - then card can only be (D,C) - player 2 was shown C.
- If player 1 sees 'C', card is (C,C) or (C,D) with 50% probability.
  - Player 2 sees 'C' 50% of time, 'D' 50% of time.
  - If player 2 follows card, then player 1 reward for: C = $\frac{1}{2}(6) + \frac{1}{2}(2) = 4$, D = $\frac{1}{2}(0) + \frac{1}{2}(7) = 3.5$

N.B. Game is symmetric. Good for player 1 = good for player 2. Flipped

Chris Hodapp,
chodapp3@gatech.edu

2016-07-27(b), CS8803-003 (RLDM)

11C (cont.)

- So, player 1 is better off following card, and thus so is player 2!
- Then, knowing that, if player 1 sees 'D', player 2 is better off following 'C' - thus, player 1 is better off following 'D'.
- If player 1 is told one thing he has no incentive to change. The same is true of player 2.
  That looks a lot like... an equilibrium.
- Particularly, it is a correlated equilibrium thanks to that piece of information. (It would not work out the same with the (D,D) card also present.)
- Expected value of this equilibrium $= \overbrace{\frac{1}{3}(2)}^{(C,D)} + \overbrace{\frac{1}{3}(7)}^{(D,C)} + \overbrace{\frac{1}{3}(6)}^{(C,C)} = 5$, and $5 > 4\frac{2}{3}$ (from the stochastic equilibrium already discussed).
- The 3rd party here is a stand-in for things in the world that give some information. Consider when you are driving and come to an intersection with a traffic light, or stop sign. It gives a signal which you can choose to follow or not, and likewise for opposing traffic.
- Facts of C.E.:
  - Can be found in polynomial time. (Nash eq. can't.)
  - All mixed Nash equilibria are correlated, so C.E. exist.
  - All convex combinations of mixed Nash eq. are correlated.

"shared source of randomness"

- Coco Values (Cooperative-Competitive Values, see Kalai-Kalai)
  - Suppose a game with short & tall player. Short player can't reach bananas in tree by himself. Tall player can reach 2. If short player stands on tall then they can reach 4. In matrix:

| Short \ Tall | Reach | Climb |
|---|---|---|
| Don't boost | 0, 2 | 0, 0 |
| Boost | 0, 2 | 0, 4 |

What's Nash equilibrium?
- (Don't boost, Reach) is
- (Boost, Climb) also is - and is 'better' for tall player

11 C (cont.)

- But short player never sees reward, so it's all the same to him. He has no incentive to help out tall player.
→ Share bananas (i.e. side payments) – split the extra benefit.
- Coco definition
  - Coco value of a game: $u$ = payoffs to you (general sum game, $u, \bar{u}$); $\bar{u}$ = payoffs to other

$$\text{coco}(u, \bar{u}) = \underbrace{\max_{a} \max_{a} \left(\frac{u+\bar{u}}{2}\right)}_{\text{Cooperative}} + \underbrace{\text{minimax}\left(\frac{u-\bar{u}}{2}\right)}_{\text{Competitive (0-sum)}}$$

→ just max → linear programming

- Applying to the banana game above:

$$u = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \bar{u} = \begin{bmatrix} 2 & 0 \\ 2 & 4 \end{bmatrix} \quad \text{(for short player)}$$

$$\frac{u+\bar{u}}{2} = \begin{bmatrix} 1 & 0 \\ 1 & \textcircled{2} \end{bmatrix} \qquad \frac{u-\bar{u}}{2} = \begin{bmatrix} \textcircled{-1} & 0 \\ -1 & -2 \end{bmatrix} \qquad \frac{\bar{u}-u}{2} = \begin{bmatrix} \boxed{1} & 0 \\ 1 & 2 \end{bmatrix}$$

maximax minimax

Coco value for $u$'s perspective: 1 ; from $\bar{u}$: 3

Side payments: $p = \text{coco}(u, \bar{u}) - \underbrace{u(a^*, \bar{a}^*)}_{\text{Utility maximizing joint action}} = 1$

$$\bar{p} = \text{coco}(\bar{u}, u) - \bar{u}(a^*, \bar{a}^*) = -1$$

- So, $u$ player receives 1, $\bar{u}$ gives 1.

- Coco is: (or does)
  - Efficiently computable
  - Utility maximizing
  - Decomposes game into sum of 2 games
  - Unique (in solution)
  - Can be extended to stochastic games (coco-Q, though not non-expansion it still converges)
  - Not necessarily equilibrium (needs to be binding – must agree to side payments in advance)
  - Doesn't generalize past 2 players. It's an open problem how to do this.

Chris Hodapp
chodapp3@gatech.edu

2016-07-27c, CS8803-003 (RLDM)

1/C (cont.)

- Mechanism Design - art, science, & engineering of games to elicit specific behavior
  - In a sense, it's sort of 'reverse' RL.
- Example: Peer teaching
  - Student & teacher, and a bunch of questions from easy to hard. Goal: What incentives can we give to both student & teacher in order to get teacher to give the questions that help the student learn?
- Proposal: +1 to learner for every correct answer
  +1 also to <u>teacher</u> for every correct answer.
  - Problem: This just incentivizes easy questions.
- 2nd proposal: Teacher gets -1 for every correct answer.
  - Problem: Teacher has incentive only to go to hard questions before student can answer them.
- Aim might be to present questions right about at the level of understanding (and perhaps at this level there is maybe 50% chance of student getting things right).
  - If we then reward based on keeping that 'correct' ratio at 50%, then teacher could just pick a mix of very easy & very hard questions to set that ratio.
  - But... What if teacher can't choose questions, but can only report the student's supposed understanding level and then questions are independently picked based on this?
    - Then: teacher gets a +1 when student gets right what they should get wrong - <u>and</u> vice versa. Teacher gets no points for student getting right what they should, and getting wrong what they should ('should' being based on the reported understanding).

See: Jordan Pollack

11C. (cont.)

- King Solomon (recall the story from the Bible)
  - This is mechanism design - of a sort. It was a game designed to elicit a specific response.
  - But... this is not quite same from a game theory standpoint. The best action for both women is to stop the baby being cut in half. It's a lot like chicken..
  - Or... value of baby being alive is much higher to 'real' mother than to 'fake' mother, and Solomon knew this.
  - Another method:
    - ⓪ Choose a fine $F > 0$ but small
    - ① A: your child? no → B! ↓ yes
    - ② B: your child? no → A! ↓ yes
    - ③ A pays F
      B announces $V > F$
    - ④ A: your child? no → B! & B pays V
      ↓ yes
    - ⑤ A! A pays V, B pays F
  - Why does this do a better job?
    - Assume: $V_{real} > V_{fake}$, and one of A & B is real mother while one is fake. A & B know all steps above.
    - If A real, B fake: B will choose $V = V_{fake}$. $V > V_{fake}$ is out because B doesn't want to risk paying extra (yes, paying for a baby... whatever) at step 4. $V < V_{fake}$ is out because A presumably will pay up to $V_{real}$ ($V_{real} > V_{fake}$) and B wants as much as possible. But because A values it higher, A always says yes at step 5 and pays $V_{fake}$ for child.
      Really, B has no reason to ever let this proceed past step 2 (B knows it will lose baby & pay fine F - so why not just avoid paying F?)

Chris Hodapp
chodapp3@gatech.edu

2016-07-27d, CS8803-003 (RLDM)

11C (cont.)

- If A fake, B real:
  - B announces $V_{real}$. A is unwilling to pay this ($> V_{fake}$). So just would have to pay F.
    A has no reason to say yes at step 1: It will lose baby regardless, but can avoid fine F.
- Basically, telling the truth at steps 1 & 2 is the best strategy. The steps <u>after</u> accomplish this, but just by their presence and not by being used.
  - Auctions are a similar idea (trying to elicit 'real' price of item), but we don't pursue this further.
- Things like CE & binding side payments can be seen as a form of mechanism design.

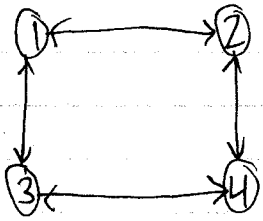
12. CCC

- Coordinating, Communicating, Coaching
  - MDPs are very 'solipsistic' - You're the only thing in the universe
  - How can Markov framework extend to coordinating, communicating, & coaching between agents?
- Imagine 2 agents. Agent 1 needs an apple. An apple is nearby, but closer to agent 2. What might agent 1 do?
- DEC-POMDP - Decentralized, partially-observable MDP
  - $I$, finite set of agents
  - $S$, states
  - $A_i$, agent $i$'s actions (actions are done simultaneously by all agents)
  - $T$, <u>joint</u> transition function (function of <u>all</u> actions by agents)
  - $R$, <u>shared</u> reward function ($R_i$ if POSG, partially-observable stochastic game)
  - $Z_i$, agent $i$'s observations (distributed over each agent)
  - $O$, observation function

This is a <u>cooperative</u> environment, but still is handled in game theory (as a "common-interest game")

12 (cont.)

- Like POMDPs, DEC-POMDPs are undecidable.
- For finite horizon they are NEXP-complete (nondeterministic exponential)
- Agents <u>can</u> benefit from communication
- Optimal solutions balance cost of communicating w/ cost of <u>not</u> communicating
- Example:




2 agents; they win if they end up in same state (room). They can only see who is in current state. They may move N, S, E, W, or stay put.

  - One strategy: One agent stays put, other circles
  - Another: Go to a common room from all locations.
  - But what if MDP had a lot of stochasticity, or room couldn't be fully observed.
- Back to older example: How could agent 1 coach agent 2 into getting the apple?
- Reinforcement learning:

environment ⇓
rewards ⇒ RL ⇒ behavior

- Inverse RL:

environment ⇓
rewards ⇐ IRL ⇐ behavior

  - Agent is experiencing environment through behavior, & needs to learn reward function
  - The paths that were taken, <u>and</u> not taken, when other options existed, can provide a lot of information on rewards
  - MLIRL = max. likelihood inverse RL

    guess R, compute $\pi$, measure $Pr(D|\pi)$, gradient on R

    ($Pr(D|\pi)$ = probability of data given policy)
  - BURLAP has MLIRL implementation (see 9th video in lecture)

IRL is itself a sort of reward shaping (let humans give hints on behavior)

Littman worked on this but many others exist

Chris Hodapp
chodapp3@gatech.edu

28

2016-07-27, CS8803-083 (RLDM)

12 (cont.)

- Recall Monte Carlo Tree Search...
  - and the sorts of things humans give us as constraints
- Policy Shaping
  - What if we give people a green & red button for "good" and "bad" and have them follow along with some test PacMan policy? How would we use this information?
  - Perhaps +1/-1 hints in the reward
  - But perhaps humans are trying to tell you about <u>policy</u>, not rewards (and studies show this is the case).
    - e.g. trying to say that $\pi(s)$ should or should not be some 'a'.
  - How do we actually incorporate this into the policy? How do we do this in the face of noise & error?
  - We must define things like: how many correct answers?
  - Suppose human reports that state $x$ is good once and zero times for $y$ & $z$, and human is correct 80% of time. What's probability that actions $x$, $y$, & $z$ are optimal, assuming only one can be?
    - Assume uniform priors.

normalize →

$P(x) = 1/3 \cdot 4/5 \propto = 2/3$
$P(y) = 1/3 \cdot 1/5 \propto = 1/6$
$P(z) = 1/3 \cdot 1/5 \propto = 1/6$

↑ Priors ↑ Probability human would report that x, y, or z was good when x was actual answer. Note this <u>doesn't</u> sum to 1!

It is sort of like if the human gave 0.8 votes of "good" for $x$, and 0.2 for $y$ and $z$.

- As a general rule:

Evidence = $d_a$ (Labels of yes/no for a being optimal), $C$ = probability human is correct, $\Delta_a$ = #yes − #no

↳ N.B. yes & no just cancel! But, we want mean, not variance

General case: $p(a|d_a) = \frac{C^{\Delta_a}}{C^{\Delta_a} + (1-C)^{\Delta_a}}$

Only one (policy): $p(a|d_a) \propto C^{\Delta_a} (1-C)^{\sum_{j \neq a} \Delta_j}$ ← exponent: evidence for all <u>except</u> a

(needs normalization still)

12 (cont.)

- What if probability of False positives & False negatives differs?
  C assumes the same. We don't really address that here...
- How do we incorporate multiple sources of information?
  e.g. agent itself has information on policy too

| | x | y | z |
|---|---|---|---|
| $\pi_H$ | 2/3 | 1/6 | 1/6 |
| $\pi_A$ | 1/3 | 1/3 | 1/3 |

- Human's policy
- Agent's policy

recall from last page that C = confidence in human

How do we use $\pi_H$ and $\pi_A$?
We already know how much to weight them: $\pi_H$ has already incorporated C! The 'real world' eventually drives $\pi_A$ to what it should be - eventually disregarding $\pi_H$.

- What if they disagree?
  - It's not really even relevant that one is human & one is agent.

$a_{opt} = \underset{a}{\text{argmax}}\ P(a|\pi_1)P(a|\pi_2)$ in general (where do they agree most?) (or, what's probability they do)

- "Drama Management"
  - Demonstrations
  - Rewards
  - Policies
  - Story:
    - Trajectory!
    - Through... plot points!

How can we influence a player?
Think about the structure of the plot in, say, a murder mystery.
- and how this changes versus a horror movie - and how the only real difference might be in which characters know what.

- Trajectories as MDPs
  - States: Partial sequences
  - Actions: Story actions
  - Model: Player model
  - Rewards: Author evaluation

Chris Hodapp
chodapp3@gatech.edu

2016-07-28b, CS8803-003 (RLDM)

12 (cont.)

- One problem: Sequences (as we used in states) are hyper exponential (however we can exploit structure here)
- We might make certain that the player, regardless of action, gets the best experience (even if it means 'negative' results in spots)
- Extension: TTD-MDP (targeted trajectory distribution)
  - trajectories: sequences so far
    actions: actions (story actions, not really player actions)
    model: $P(t'|a,t)$ - next trajectory given start & action
    target distribution: $P(T)$ - $T$ = final trajectory (end of story)
    policy: Probability distribution over actions, leading to matching target
  All the uncertainty comes from the player, not system.
  Going to probability distribution rather than hard constraint, TTD-MDP is solvable in linear time (see paper! What paper, they don't say)
- It uses KL-divergence (somehow) - mostly to match distributions

Introduction

- Why BURLAP (instead of UCI database & Weka)?
  - UCI & Weka don't cover RL really at all.
  - BURLAP is trying to provide lots of algorithms, and the closest thing to data that's really feasible here (data in RL is really hard) - hence, it provides lots of simulations & interactivity.
  - Places RL has been used successfully in "real world":
    - Elevator control
    - Robotics (e.g. Stanford work with flying model helicopter tricks)
    - Games like backgammon (where the simulator sort of is the real world)
      - Video games are an interesting case here (humans interact with same 'simulator' - it's the game)
        - eg. see DeepMind & deep RL, and the work with learning Atari games