Lectures

1.Smoov & Curly's Bogus Journey

MDP
state: S
model: T(s, a, s') = Prob(s' | s, a)
actions: A(s)
reward: R(s), R(s,a), R(s,a,s')
policy: pi(s)->a
optimal policy: argmax(long term reward)

Markovian properties
1. memoryless: model only depends on current state (no condition on anything past current state)
2. stationary: rules (T) doesn't change over time

Rewards
1. delayed reward (e.g. you can lose at first step in chess)
        credit assignment: which action caused delayed reward
2. minor changes matter

Sequences of Rewards (assumptions of this course)
1. infinite horizons (stationary policies): if an agent doesn't live forever, it'll change its policy (max reward in limited steps) and it'll have different policy even if in the same state. without infinite horizon, we lose stationarity in policy
2. utility of sequences (stationary preferences): if U(s0, s1, s2...) > U(s0, s1', s2'...) then U(s1, s2...) > U(s1', s2'...). this is why we do addition over rewards, otherwise this assumption may not hold

Discounted Rewards
U(s0,s1,..) = sum(gamma^t*R(s_t)) [t=0...inf, 0<=gamma<1]

Policies
optimal policy:
pi* = argmax_pi(E[sum_t(gamma^t * R(s_t) | pi]) policy that maximizes long-term expected rewards

true utility of a state:
U_pi(s) = E[sum_t(gamma^t * R(s_t) | pi, s0=s] expected rewards if we follow the <u>optimal</u> policy

pi*(s) = argmax_a(sum_s'[T(s,a,s') * U(s')])
reward:immediate feedback
utility: long-term reward

Bellman equation!
U(s) = R(s) + gamma* max_a (sum_s' [T(s,a,s') * U(s')]) true value of a state

Value Iteration
1. start with arbitrary U
2. update U based on neighbors
3. repeat until converge
4. U_t+1(s) = R(s) + gamma * max_a (sum_s' [T(s,a,s') * U_t(s')])

Policy iteration
1. start with arbitrary policy pi_0
2. evaluate: given pi_t, calculate U_t = U(pi_t) <using VI>
3. improve: pi_t+1 = argmax_a (sum [T(s,a,s') * U_t(s')])
4. U_t(s) = R(s) + gamma * sum_s' [T(s, pi_t(s), s') * U_t(s')]
- can be solved with LP or Bellman operator (fix point). more expensive than VI due to evaluation step

Three forms of Bellman Equation
1. value: V(s) = max_a [R(s,a) + gamma * sum_s' (T(s,a,s') * V(s')]
2. quality: Q(s, a) = R(s,a) + gamma * sum_s' [T(s,a,s') * max Q(s', a)]
3. continuation: C(s,a) = gamma * sum_s' (T(s,a,s') * max_a [R(s',a') + C(s, a)]

when we know Q, we don't need T to get V

Model-based vs Model-free
"model": transition matrix.
model-based: use model predictions of next state & reward to calculate optimal policy; dynamic programming (e.g. policy iteration, value iteration requires learning T)
model-free: purely sample from experience (MC-Control, SARSA, Q-learning)

On-policy vs Off-policy
"policy": optimal actions
on-policy: attempts to evaluate / improve the policy & behave based on the policy (e.g. SARSA updates Q with Q associated with the policy it followed)
off-policy: doesn't behave on evaluated policy (e.g. Q-learning updates Q with maxQ instead of whichever action agent just took)

2. Reinforcement Learning Basics

agent --a -- > env
agent <-- s, r-- env

Behavior Structures
plan: fixed sequence of actions
conditional plan: if... then..
stationary policy/universal plan: map(state) -> actions (pro: always an optimal stationary policy; con: too many ifs, inefficient)

Evaluate a policy
1. state immediate reward
2. truncate horizon
3. summarize sequence
4. summarize over sequences

Evaluate a policy
1. value of returned policy
2. computational complexity (computation time)
3. <u>experience</u> complexity (sample size, data size needed) <important in rl>
4. space complexity

3. TD and Friends

3 types of models
1. model-based
<s,a,r>* -> model_learner <--> T/R -> MDP solver ->Q* -> argmax - > pi

2. value-function-based "model-free" (doesn't know T, learn Q from sar
<s,a,r>* -> value update <--> Q -> argmax -> pi

3. policy search
<s,a,r>* -> policy update <--> pi (difficult, feedback from pi is hard to use to update policy)

- more direct learning 3>2>1
- more supervised 1>2>3

Computing estimates incrementally
$V_T(S_i) = [(T-1)* V_{T-1}(S_i) + R_T(S_i)] / T$ --> (take T as 1/weight_on_reward)
$= V_{T-1}(S_i) + alpha_T* [R_T(S_i) - V_{T-1}(S_i)]$ where $alpha_T = 1/T$

Properties of Learning Rate

for learning function V_T(S) = V_T-1(S) + alpha_T* [R_T(S) - V_T-1(S)]
(notice: R_T is observed reward, V_T-1 is estimation at S)
then V_T(s) <mark>converges</mark> to actual V
limitT->inf [V_T(S)] = V(s) if:
1. sum(alpha_T) = inf   (sum alpha diverges)
2. sum(alpha_T^2) < inf (sum squared alpha converges)

Select learning rate
if p > 1, sum(1/x^p) converges!

TD(1) Rule
for each episode (T)
<mark>for all S</mark>, initialize eligibility=0, V(S) as V(S) at T-1 (last
episode),
after transit, add 1 to eligibility of the state just left (s_t-1)
for all S, update V(S) with TD, learning rate alpha
decay eligibility

TD(1) Rule

Episode T
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)
$e(S_{t-1}) = e(S_{t-1}) + 1$
For all s,
$V_T(s) = V_T(s) + \alpha_T (r_t + \gamma V_{T-1}(S_t) - V_{T-1}(S_{t-1})) e(s)$
$e(s) = \gamma e(s)$

note: in **r_t + gamma * V_T-1(S_t) - V_T-1(S_t-1)** , S_t: new state, S_t-1: last state. this part is
shared by all states in V_T(S) updating!

claim: TD(1) is the same as outcome-based updates (if no repeated states), wait till see all
rewards and compare with est. value at the beginning of episode;

but if there were repeated states:
outcome based learning: ignores what happens during the episode
TD(1) : picks up what it learns the last time it saw the state <EXTRA LEARNING!>

why is TD(1) wrong?
TD(1) not using all data points, only uses individual run. when it gets edge cases, it gets wrong
estimate

TD(0) Rule
if finite data <mark>repeated</mark> indefinitely, TD(0) finds max.likelihood
estimate; without repeats, it doesn't perform well;

$V_T(S_{t-1}) = V_T(S_{t-1}) + \alpha_T (r_t + \gamma V_T(S_t) - V_T(S_{t-1}))$
$V_T(S_{t-1}) = E_{S_t}[r + \gamma V_T(S_t)]$

for TD(1) (outcome-based updates), repeat the sample over n
over doesn't make difference because all data points are already used

<mark>Instead of updating all S, TD(0) only updates S_t-1</mark>

TD(lambda)
x lambda to eligibility each step

TD(λ) Rule

Episode T
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)
$e(S_{t-1}) = e(S_{t-1}) + 1$
For all s,
$V_T(s) = V_T(s) + \alpha_T (r_t + \gamma V_{T-1}(S_t) - V_{T-1}(S_{t-1})) e(s)$
$e(s) = \lambda \gamma e(s)$

TD(1)

Episode T
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)
$e(S_{t-1}) = e(S_{t-1}) + 1$
For all s,
$V_T(s) = V_T(s) + \alpha_T (r_t + \gamma V_{T-1}(S_t) - V_{T-1}(S_{t-1})) e(s)$
$e(s) = \gamma e(s)$

TD(0)

Episode T
For all s, at start of episode, $V_T(s)=V_{T-1}(s)$
After $S_{t-1} \xrightarrow{r_t} S_t$: (step t)
For all $S = S_{t-1}$
$V_T(s) = V_T(s) + \alpha_T (r_t + \gamma V_{T-1}(S_t) - V_{T-1}(S_{t-1}))$

k-step Estimators

TD(0) one step forward looking;
TD(1) all steps forward looking

$$E_1 \quad V(S_t) = V(S_t) + \alpha_T(r_{t_1} + \gamma V(S_{t+1}) - V(S_t)) \quad TD(0)$$

$$E_2 \quad V(S_t) = V(S_t) + \alpha_T(r_{t_1} + \gamma r_{t+2} + \gamma^2 V(S_{t+2}) - V(S_t))$$

$$E_3 \quad V(S_t) = V(S_t) + \alpha_T(r_{t_1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V(S_{t+3}) - V(S_t))$$

$$E_k \quad V(S_t) = V(S_t) + \alpha_T(r_{t_1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V(S_{t+k}) - V(S_t))$$

$$E_\infty \quad V(S_t) = V(S_t) + \alpha_T(r_{t_1} + \cdots + \gamma^{k-1} r_{t+k} + \cdots \quad TD(1)) - V(S_t))$$

TD(lambda) is weighted avg of k-step estimators

sum_(k=1..inf) [lambda ^ (k-1) * (1-lambda)]

TD(lambda) empirical performance: sweet spot is in (0,1) interval

[Reading] MC - sutton rl 5.1

Monte Carlo method: average the returns observed after visits to that state. As more returns are observed, the average should converge to the expected value

estimates each state independently

[Reading] MC control - sutton rl 5.3

control: approximate optimal policy

4. Convergence

Bellman Equation with Control

approximations in Q-learning update:
if we have the model, update with Bellman equation;
else, use TD(0) rule to update Q with Q*

recall <u>property of alpha</u>: sum(alpha) diverge but sum(alpha^square) finite then Q converge

Bellman Equations with Actions

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

$$\langle s_t, a_{t+1}, r_t, s_t \rangle$$

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

$$Q_t(s,a) = Q_{t+1}(s,a), \text{ otherwise} \quad TD(0)$$

Approximations ① If we knew the model, synchronously update
$$Q_t(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_{t+1}(s',a')$$
② if we knew Q*, sampling asynchronously update
$$Q_t(s_{t+1}, a_{t+1}) = Q_{t+1}(s_{t+1}, a_{t+1}) + \alpha_t(r_t + \gamma \max_{a'} Q^*(s_t, a') - Q_{t+1}(s_{t+1}, a_{t+1}))$$

Bellman Operator

mapping from Q to Q
[BQ](s,a) = r(s,a) + gamma * sum_s' [T(s,a,s') * max_a' (s',a')]
BQ*=Q*: Bellman Equation
Q_t = BQ_t-1: Value iteration

Contraction Mapping

B is an operator,
if for all value function F, G and some 0<=gamma <= 1

||BF-BG||_inf <= gamma * ||F-G||_inf

where ||Q||_inf = max |Q(s,a)|

then B is a contraction mapping (makes things closer)

non-expansion is a kind of contraction mapping

Contraction Mapping Properties
if B is a contraction mapping
1) F* = BF* has a solution and it's unique
2) F_t = BF_t-1 ... F_t -> F* (repeatedly apply contraction mapping, function converges; value iteration converges!)

unique because if there are two then they're not converging together which violates contraction mapping definition

since Bellman operator is contraction mapping, Bellman Equation only has one solution

Bellman Operator Contracts
proof:--->
to remove max in |..| on right work, we need to prove that we can remove max and the diff is still bounded.

Bellman Operator Contracts

Given Q_1, Q_2

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

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

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

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

$\leq \gamma \max_{s',a'} |Q_1(s,a) - Q_2(s',a')| = \gamma \|Q_1 - Q_2\|_\infty$

Max is a non-expansion
proof:

Max is a non-expansion

For all f,g $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$

Proof: WLOG $\max_a f(a) \geq \max_a g(a)$

$|\max_a f(a) - \max_a g(a)| = \max_a f(a) - \max_a g(a)$

$= f(a_1) - g(a_2)$

$\leq f(a_1) - g(a_1)$

$= |f(a_1) - g(a_1)|$

$\leq \max_a |f(a) - g(a)|$

$f(a_1)$
$g(a_2)$
$g(a_1)$

$a_1 = \arg\max_a f(a)$
$a_2 = \arg\max_a g(a)$

.

let B be contraction mapping with fixed point Q*=BQ* (has an unique solution<--property of contraction mapping),
Q0: Q function
update Q_t+1 = [B_t Q_t] Q_t
<u>then Q converges if:</u>

Convergence   Define $\alpha_t(s,a) = 0$ if $s_t \neq s, a_t \neq a$

theorem: Let B be a contraction mapping and $Q^* = BQ^*$ be its fixed point. Let Q_0 be a $\gamma < 1$ Q function and define $Q_{t+1} = [B_t Q_t] Q_t$.

Then, $Q_t \to Q^*$ if:

1. For all u_1, u_2, s, a : $|([B_t u_1] Q^*)(s,a) - ([B_t u_2] Q^*)(s,a)|$
$\leq (1 - \alpha_t(s,a)) |u_1(s,a) - u_2(s,a)|$

2. For all Q, u, s, a :
$|([B_t u] Q^*)(s,a) - ([B_t u] Q)(s,a)|$
$\leq \gamma \alpha_t(s,a) |Q^*(s,a) - Q(s,a)|$

3. $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$

1. [B_t Q_t] is non-expansion function (e.g. max|| in above proof)
2. B_t is a contraction mapping
3. sum(alpha) diverges but sum(alpha^2) is finite
(hidden condition: must visit (s,a) indefinitely often)

ps. in Q-learning, one Q is 1-step look-forward and the other Q is avg.

Convergence Theorem Explained:

Convergence Theorem Explained

$$([B_+ Q] w)(s,a)$$
$$= Q(s,a) + d_+(s,a) \left(r_+ + \gamma \max_{a'} w(s_+, a') - Q(s,a) \right)$$
$$\underbrace{\qquad}_{Q^*}$$

1. For all U_1, U_2, s, a : $\left| \left([B_+ U_1] Q^* \right)(s,a) - \left([B_+ U_2] Q^* \right)(s,a) \right|$
$$\leq \left(1 - d_+(s,a) \right) \left| U_1(s,a) - U_2(s,a) \right|$$

Convergence Theorem Explained

$$([B_+ Q] w)(s,a)$$
$$= Q(s,a) + d_+(s,a) \left(r_+ + \gamma \max_{a'} w(s_+, a') - Q(s,a) \right)$$

2. For all Q, U, s, a : $\left| \left([B_+ U] Q^* \right)(s,a) - \left([B_+ U] Q \right)(s,a) \right|$
$$\leq \gamma d_+(s,a) \left| Q^*(s,a) - Q(s,a) \right|$$

CONTRACTION

use Q function in two ways in convergence theorem:
1. w: acts as if we knew Q*, plays the role of avg (sumT)->avg out stochasticity
2. Q: one-step look-back (in TD formula, prediction made at last state)->value iteration obj

Generalized MDP

Generalized MDPs

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

Max zie...

$$\bigoplus_{s'} g(s') = \sum_{s'} T(s,a,s') g(s') \quad \bigotimes_{a'}^{s'} f(s',a') = \max_{a'} f(s',a')$$ regular MDP

$$\bigoplus_{s'} g(s') = \min_{s' \in \{s' | T(s,a,s') > 0\}} g(s') \quad \bigotimes_{a'}^{s'} f(s',a') = \max_{a'} f(s',a')$$ pessimistic MDP risk averse

$$\bigoplus_{s'} g(s') = \sum_{s'} T(s,a,s') g(s') \quad \bigotimes_{a'}^{s'} f(s',a') = \sum_{a'} p(oracl^*)) f(s,a)$$ exploration-sensitive max, min, mediocre

$$\bigoplus_{s'} g(s') = \sum_{s'} T(s,a,s') g(s') \quad \bigotimes_{a'}^{s'} f(s',a') = \min_{a'} \max f(s',A')$$ zero-sum game

5. AAA (Advanced Algorithm Analysis)

More on VI
1. For some t* < inf, that's polynomial in |S|, |A|, R_max =
max|R(s,a)|, 1/(1-gamma), number of bits of precision that are
used to specify transition prob.
after t* many steps,
pi(S) = argmax_a [Q_t* (s,a)] is optimal

More on VI

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.   optimal   Cramer's rule

2. If $|V_t(s) - V_{t+1}(s)| < \varepsilon$ $\forall s$
$$\max_s |V^{\pi_{V_t}}(s) - V^*(s)| < \frac{2\varepsilon\gamma}{1-\gamma}$$ $H \sim \frac{1}{1-\gamma}$ smartsighted

3. $\|B^K Q_1 - B^K Q_2\| \leq \gamma^K \|Q_1 - Q_2\|_\infty$

"some time before infinity, with value iteration, we'll get a Q that's close enough to optimal Q, such that when we take greedy policy respect to it, it's 100% optimal"
- in infinite time, Q converges to optimal;
- in finite time, Q converges to near-optimal (no optimal guaranteed), but greedy policy respect to Q converges to optimal

* but not polynomial bounded because of 1/(1-gamma) if gamma =1

"greedy policy with respect to VI converges in reasonable time".

2. it two consecutive value estimation is small, value of greedy policy with respect to VI converges to optimal value

both 1&2 encourages us pick small gamma

- horizon = 1/(1-gamma) (small gamma-> shorter future->easier problem -> shortsighted)
- iteration time
hence we still pick large gamma in practice

3. helps us decide how long to run VI

Linear Programming
optimize linear object function under linear constraints

Bellman equation can't directly be solved by LP because of max
e.g. max(-3,7,2,5)=x <==> x>=-3 & x>= 7 & x>=2 & x>=5 (constraints) AND min(x) (obj)

translate max in Bellman equation to linear obj & constraints ("PRIMAL")
obj: min(sum_s(V_s))
constraints:
for any s,a,
V_s >= R(s,a) + gamma * sum_s' [T(s,a,s') * V_s']

solve ^ with LP solver => get optimal V
choose greedy policy => get optimal policy

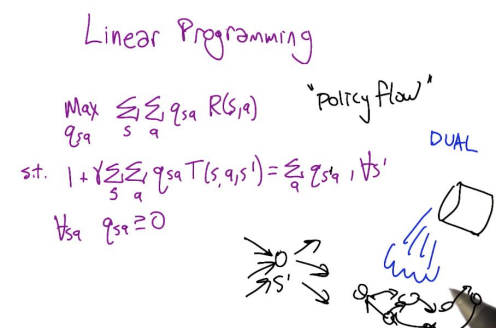**Dual Linear Programming**
add policy flow into <s,a> pair and let the flow go thru so that we get max reward

concentrate more on policy

Policy Iteration
initialize -> policy improvement -> policy evaluation

Linear Programming

$\max_{q_{sa}} \sum_s \sum_a q_{sa} R(s,a)$ "Policy flow"

DUAL

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

$\forall_{sa}\ q_{sa} \geq 0$

Policy Iteration

open question
convergence time
$\geq$ linear, $\leq$ exponential
$|s|$ $\frac{|A|^{|s|}}{}$

Initialize $\forall s, Q_0(s)=0$
policy improvement $\forall s\ \pi_t(s)= \arg\max_a Q_t(s,a)$ (t≥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

tradeoff? computation time
in policy evaluation step, we're doing an entire VI

converge time is
>= linear |S| AND <= exponential |A|^|S|

Domination
pi_1 >= pi_2 iff for all s V_pi1(s) >= V_pi2(s)

strict domination:
exists s that's V_pi1(s) > V_pi2(s)

Why Does Policy Iteration Work? (PI Property)
1. Bellman operator is monotonic. proof->
2. if we apply greedy policy one step to Qt, it'll be equal to Qt if not making it better (greedy policy value improvement)
3. PI doesn't get stuck in local optima
^ because of value improvement property, greedy policy must strictly dominate at least one state in each iteration. If not, we must have found the optimal policy.
it's impossible to go back forth between 2 values (stuck) because of non-deprovement

Policy iteration Proof
greedy policy converges to fixed point, optimal policy

Domination

$$\pi_1 \geq \pi_2 \quad \text{iff} \quad \forall_{s \in S} \quad V^{\pi_1}(s) \geq V^{\pi_2}(s)$$

strict domination

$\bar{B_2}$ is Monotonic

$[B_1 \bar{V}](s) = R(s, \pi_1(s)) + \gamma \sum_{s'} T(s, \pi_1(s), s') V(s')$

$[B_2 \bar{V}](s) = R(s, \pi_2(s)) + \gamma \sum_{s'} T(s, \pi_2(s), s') V(s')$

$\exists . \forall_{s \in S}$

Thm: If $V_1 \geq V_2$, then $B_2 V_1 \geq B_2 V_2$
Proof: $(B_2 V_1 - B_2 V_2)(s) = \gamma \sum_{s'} T(s, \pi_2(s), s') \underbrace{(V_1(s') - V_2(s'))}_{\geq 0}$
$\geq 0 \checkmark$

Another Property in Policy Iteration

$\pi_1 \rightarrow B_1 \rightarrow Q_1 = B_1 Q_1$
π_2 greedy policy wrt Q_1
$\downarrow B_2$
$Q_1 \leq B_2 Q_1$

$Q_1(s,a) = R(s,a) + \gamma \sum_{s'} T(s, \pi_1(s), s') Q_1(s', \pi_1(s'))$
$\leq R(s,a) + \gamma \sum_{s'} T(s, \pi_2(s), s') Q_1(s', \pi_2(s'))$

$Q_1 \leq B_2 Q_1$

Policy Iteration Proof

π_1 policy, B_1 operator, Q_1 value function wrt π_1
π_2 greedy wrt Q_1, B_2 operator, Q_2 value function wrt π_2

non-deprovement

$B_2 Q_1 \geq Q_1$ — value improvement

$k \geq 0, \quad B_2^{k+1} Q_1 \geq B_2^k Q_1$ — monotonicity

$a \geq b, b \geq c, \quad \therefore a \geq c$ — transitivity

$\lim_{k \to \infty} B_2^k Q_1 \geq Q_1$

$Q_2 = Q_1$ — fixed point

6. Messing with Reward

Why do we want to change reward function?
1. easier to solve/ repeat and similar to what it would have learned
        - faster - solvable

2. make agent behave the way we want it to through reward
3. efficiency:
      -speed (experience, computation)
      -solvability

How can we change MDP reward function without changing the optimal policy?
1. multiply by positive constant
      if $R'(s,a) = R(s,a)$ then $Q'(s,a) = cQ(s,a)$

2. shift by constant (add)
      if $R'(s,a) = R(s,a)$ then $Q'(s,a) = Q(s,a) + c/(1\text{-gamma})$

3. non-linear potential-based
      potential-based shaping

Reward Shaping
how to train a dolphin to jump through a hoop?
near hoop -> through hoop -> in air -> higher

soccer game reward function?
must <u>give hints</u>

score in goal + 100, near ball +1/distance, hitting ball +10, near goal +1/distance
what would happen? it causes spurious behavior (keep hitting the ball and moving away)

must <u>avoid suboptimal positive loop</u>!

Potential-based Shaping
put bonus on certain states
by changing from state to state, how much more potential does an agent get?

Change-in-State-based Bonuses
keep track of which state agent's in, when in desirable states, + bonus, otherwise - bonus
agent gets the difference in rewards between states, which can't accumulate
score in goal +100 near ball delta(1/distance - 1/last_distance)

Potential-based Shaping
if we subtract leaving-state's potential and add discounted arriving-state's potential when an agent moves from s to s', new $Q'(s,a) = Q(s,a) - \text{psi}(s)$.

Potential-based Shaping

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

$R'(s,a,s') = R(s,a) - \Psi(s) + \gamma \Psi(s')$

$Q'(s,a) = \boxed{Q(s,a) - \Psi(s)}$

infinite telescoping sum

s s' s" s''' s'''' .... $+ \gamma^\infty \Psi(s^\infty)$

$Q = R + R + \gamma R + \gamma R +$ ... $\gamma^{<1}$
$-\Psi(s) + \gamma \Psi(s')$
$\gamma(-\Psi(s') + \gamma \Psi(s'')) + \gamma^2(-\Psi(s'') + \gamma \Psi(s''')) + \cdots$

this can be interpreted as, the value of a state is the expected total reward of the state minus potential loss of leaving the state

Q-learning with Potentials
if set psi = optimal value max_a Q*(s,a), then Q(s,a) = 0
if a is optimal else < 0

strategy: explore and discover < 0 then take optimal action

Q-learning with potential == Q-learning with Q-function
initialized what potential would've been
(action choices, same strategy at each step; but maybe different Q-values along the way)

injecting knowledge (initialize Q with potential instead of randomly), may improve performance
(quicker convergence)

potential-based reward shaping preserves the original MDP's optimal policy
potential-based shaping learning time also depends on potential selection. it could take longer /
stuck in suboptimal for a while before eventually finds optimal

7. Exploring Exploration

Topics
bandits (no state transition, full stochasticity)
deterministic MDP (state transition, no stochasticity)
stochastic MDP (both)

[Bandits] Confidence-based Exploration
two bandits (#payoff/#pull: 0/1, 1/2) what to do?

every time new data comes in, do following & pick:
maximize likelihood->always b (but never visit a)
maximize confidence->always b (pick high denominator, never a)
minimize confidence->alternate a&b (but never get to use the info collected)

better solution: balance b/w maximize likelihood & minimize confidence
"exploration-exploitation dilemma"

Metrics for Bandits
1. identify optimal arm in the limit (can pull arms in order, but may take a long time to learn)
2. maximize (discounted) expected reward (collect reward along the way)
    Gittins index:

success/pull-> map to an index-> choose greedily->optimal

^ opportunity cost: how high would the reward be (expected reward of bandit) if you were to stop trying

combines max likelihood & low-confidence

but difficult to generalize

3. maximize expected reward over finite horizon

computable but expensive

finite=1 == max likelihood

4. identify near-optimal arm (epsilon) with high probability (1-delta) in time T(k, epsilon, delta),
pulls before we identify near-optimal arm is polynomial in k, 1/delta, 1/epsilon
(figure out the optimal arm)

PAC: probably approximately correct

PAO: probably approx optimal

5. nearly maximize reward (epsilon) with high probability (1-delta) in time T'(k, epsilon, delta),
pulls before near-optimal: polynomial in k, 1/delta, 1/epsilon
(collect max reward on the way figuring it out, we can take longer if reward is high enough)

6. pull a non-near-optimal arm (epsilon) no more than T"(k, epsilon, delta) with high probability
(1-delta)

4&5&6 are equivalent! (proofs)

Find Best $\Rightarrow$ Few mistakes

Algorithm that finds ε best with probability
$1-\delta$ in $T(k, \varepsilon, \delta)$ pulls. (A)

Want ε' close arm on $T'(k, \varepsilon', \delta')$ pulls with
probability $1-\delta'$.

$\varepsilon = \varepsilon'$, $\delta = \delta'$, run algorithm A and after
$T(k, \varepsilon, \delta)$ pulls we know ε' close arm.
Pull it forever. $T'(k, \varepsilon', \delta') \leq T(k, \varepsilon, \delta)$

Few Mistakes $\Rightarrow$ Do well

We have an algorithm that pulls ε-suboptimal arms at most $\tau(k,\varepsilon,\delta)$ times with probability $1-\delta$. $\underline{B}$

Want an algorithm $\overset{C}{\text{that}}$, with high probability $1-\delta'$, has a per-step reward ε' close to optimal after $\tau'(k,\varepsilon',\delta')$ steps.

$\varepsilon = \varepsilon'/2$

$\tau(k,\varepsilon,\delta') = m$

$$\underbrace{}_{\text{mistakes}} \quad \overbrace{}^{\varepsilon'/2}$$

$\tau'(k,\varepsilon',\delta') = n$

$\varepsilon' \geq \dfrac{1 \cdot m + \frac{\varepsilon'}{2}(n-m)}{n}$

$n\varepsilon' \geq m + \dfrac{n\varepsilon'}{2} - \dfrac{\varepsilon' m}{2}$

$\dfrac{n}{2}\varepsilon' \geq m - \dfrac{\varepsilon' m/2}{}$

$n \geq m\left(\frac{2}{\varepsilon'} - 1\right)$

Do well $\Rightarrow$ Find Best

We have an algorithm $\overset{C}{\text{that}}$ gets within ε (per step) of optimal after $\tilde{\tau}(k,\varepsilon,\delta)$ pulls with probability $1-\delta$. $=n$

Want is algorithm $\underline{A}$, which, after $\tau'(k,\varepsilon',\delta')$ pulls, returns an arm that is ε' optimal. w.p $1-\delta'$

Run C: per step near optimal payoff.

e_i: suboptimality of the plurality arm $\varepsilon = e_i \cdot \frac{1}{k}$

$e_i \leq \varepsilon'$ $\boxed{\varepsilon = \varepsilon'/k}$ $e_i \leq \varepsilon k$

.

Hoeffding Bound
- maximum likelihood confidence interval
- helps decide # pulls to get confident result
- used in PAC learning

Strategy: combining arm Info
take c samples of each arm, then choose the arm with best estimate
to be 1-delta sure it's within epsilon,

Hoeffding

Let $X_1,\dots,X_n$ be iid with mean μ. Let $\hat{\mu}$ be our estimate of μ: $\sum_i X_i/n$. The following i a $100(1-\delta)\%$ confidence interval for μ:

$\left[\hat{\mu} - \dfrac{z_\delta}{\sqrt{n}}, \hat{\mu} + \dfrac{z_\delta}{\sqrt{n}}\right]$ $z_\delta = \sqrt{\frac{1}{2}\ln\frac{2}{\delta}}$

we need to learn <u>each arm</u> so that its estimate is within <u>epsilon/2</u> with <u>probability 1- delta/k</u>

interpretation:
P(all k-arm right) = (1-delta/k)^k <-assumes independence b/w arms
P(at least 1 arm wrong) = k*delta/k <- union bound, assumes overlap
P(all arms right) = 1-delta
<bonferroni correction>

How many samples?
solve n from confidence interval

How Many Samples?
Want to be $\varepsilon/2$ accurate with probability $1-\delta/k$

$$\frac{\sqrt{\frac{1}{2}\ln\left(\frac{2k}{\delta}\right)}}{\sqrt{c}} \leq \varepsilon/2$$

PAC-style bounds for bandits

Quiz: $c \geq \boxed{2}\frac{1}{\varepsilon^{\boxed{2}}}\ln\left(\frac{2k}{\delta}\right)$

[Deterministic MDP] MDP Optimization Criteria
1. explore randomly
2. trap states
3. mistake bound (works both in bandits and MDPs)
   the number of epsilon-suboptimal actions at a state
is bounded by <mark>polynomial</mark> (1/epsilon, 1/delta, n, k) <n=states, k=actions>
   * epsilon-suboptimal actions: payoff of which is > epsilon from optimal payoff
4. minimize # cheats (Kobayashi Maru test)

Rmax
maximum reward in all transition
for any unknown states, we assume R_unknown = Rmax

1. keep track of MDP
2. any unknown <s,a> is Rmax
3. solve the MDP
4. take action from pi*

Rmax Analysis
1. once visit all <s,a>, no more mistakes will be make (found pi*)
2. What happens if we stop visiting unknown states (get trapped)?
        if we loop w/o learning anything, we still make no mistake (we must be
stuck in optimal policy)
        - we could still do as good without visiting unknown states
        - but discount factor is important here
        - with high discount factor, Rmax forces you to explore around
3. if we execute a path to an unknown <s,a>
        # steps to get to <s,a>: n (total states)
        # time discover>: n*k (total unknown edges)
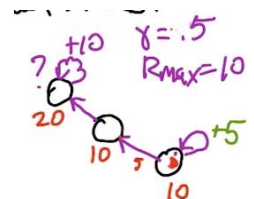
mistake bounded: $n*n*k = O(k*n^2)$
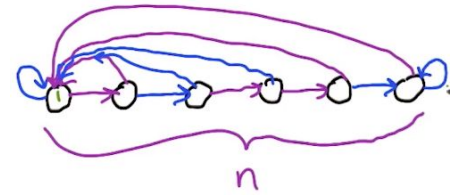
Lower Bound
no algorithms can do better than $O(k*n^2)$
sequential combination lock example (rightmost +100 reward)

for state i, it takes 1+i to set back to 1 & come back
$1+2+..+n = O(n^2)$

if only one action sends forward and k-1 sends back to 1
n*kn again!

[Deterministic Bandit]
no state, deterministic
mistake: $O(K-1)$

[Stochastic MDP]
general stochastic MDPs combine
1. stochastic (hoeffding bound until estimates are accurate)
2. sequential (unknown regions assumed optimistic)

General Rmax
1. keep track of MDP (T, R)
2. **new param:** <u>c -- # tries in an unknown state</u>. if tries fewer than c times, use max likelihood estimate
    (instead of assuming all <s,a>=Rmax)
3. solve the MDP
4. take action from pi*
to make it work...

Simulation Lemma
"if T and R errors are bounded, V is near-optimal"
if we have transitions and rewards off by alpha or less
(for fixed pi, T diff and R diff are both bounded by alpha)
alpha is bounded respect to epsilon
(each bandit must be < epsilon so the overall is epsilon optimal)
-- solve with Hoeffding bound & union bound

Simulation Lemma

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

$\varepsilon = |V_1(s) - V_2(s)| \le \dfrac{G}{1-\gamma}$

$\varepsilon = \dfrac{G}{1-\gamma} = \dfrac{\alpha + \gamma n \alpha \frac{R_{max}}{1-\gamma}}{1-\gamma}$ $\varepsilon(1-\gamma) = \alpha(1 + \gamma n \frac{R_{max}}{1-\gamma})$

$\boxed{\alpha = \dfrac{\varepsilon(1-\gamma)}{1 + \gamma n \frac{R_{max}}{1-\gamma}}}$

$\overset{SR}{\Rightarrow}$ $\dfrac{c}{\equiv}$ $\Leftarrow$

Hoeffding bound, union bound

Explore-or-Exploit Lemma
if all transitions are either accurately estimated or unknown,
optimal policy is

- either near optimal
- or an unknown state is reached quickly (in polynomial time)

^ is true with high probability through Rmax

Rmax doesn't guarantee optimal policy, it obtains near-optimal result

8. Generalization

Generalization Idea
need to generalize: zillions of states will take forever to solve
methods: SL methods (e.g. linear function approximation)

what can we generalize? policy, value function, model
1. **policy**: mapping b/w states -> actions
        generalization: for some similar states, use similar actions
2. **value function**: mapping states/action -> estimated return (Q(s,a)) <- most researches focus
        generalization: similar states have similar return
3. **model**: mapping between s -> s' & r
        s,a->s' : supervised examples
        generalization: predict reward R & transition T.
        challenge: needs model to be able to predict multiple states forward

Basic Update Rule
- create gradient step = learning_rate * y_diff *
gradient_descent
- y_diff = TD error = y* - y_predict
  = (current_Qlearning_pred - Q(s,a))
- y* acts as "made-up label" here
- update current state Q(s,a) towards Q* in Q-learning
- meanwhile update Q* with bootstrapping

Basic Update Rule
General Form:
$$Q(s,a) = F(w^a, f(s))$$
$\langle s,a,r,s'\rangle$ y^* y gradient step
$$\Delta w_i^a = \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \frac{\partial Q(s,a)}{\partial w_i^a}$$
Bellman equation
bootstrapping   TDerror

Linear Value Function Approximation

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

- linear Q function: Q(s,a) = sum_i(feature_function_i(s)*weight(i, a))
- each action gets a set of n weights to represent the Q values for that action.
- dot product weights & feature function
- weights give "importance" of all the features in contributing to an action's value

Successes in research
linear approximation:

3-layer backprop: Tesauro (Backgammon, Jeopardy), Wang/Dietterich (shuttle scheduling, NASA), Charles (tic-tac-toe)
- experience: start to learn well, but bootstrapping aspect is dependent on previous predictions, once learning starts to go south, whole thing falls apart quickly

CMAC: Sutton
- like neural nets but first layer not learned, instead pre-decided how to breakup input space
- generalization of linear

Linear nets: Singh, Parr (cellphone, channel allocation)
- feature selection is the key!!! must be 1) computationally efficient, 2) value-informative
- example: taxi -- distance to passenger is a good feature

aggressive generalization (not so linear):
Deep nets: Minh Etal (Atari)
- massive layers
- convolutions
- trained carefully (to avoid sampling bias, value oscillation, see more in paper notes)

TD gammon (Tesauro) -> sometimes don't work well (Boyan/Moore) -> but you can train it differently & it could work (Sutton)

Failures could happen (Baird's Counter example)
- see paper session "Residual Algorithm"

Star problem:
- no rewards
- deterministic
- no choice
- linear value function approximation
- near tabular
- value function multiply representable (inf solutions)
      1. set all 0 ==> sum 0 ==> w = (0, 0, .. 0)
      2. w0=-1 & solve w_i ==> w = (-1, 1/2, ... 1/2, 7)
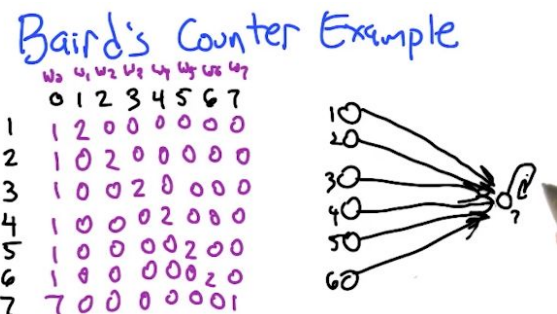
- in 2nd case, w0 gets updated 6 times while other w's update once respectively => w0 gets pumped up quickly, so are other w's except w7 => diverge!!
- in 1st case, w will all get stuck at 0 & never move

- if there are shared weights, can not converge

Averagers function approximator
find convex combination of anchor points (e.g. linear interpolation)

Baird's Counter Example

	0	1	2	3	4	5	6	7
1	1	2	0	0	0	0	0	0
2	1	0	2	0	0	0	0	0
3	1	0	0	2	0	0	0	0
4	1	0	0	0	2	0	0	0
5	1	0	0	0	0	2	0	0
6	1	0	0	0	0	0	2	0
7	7	0	0	0	0	0	0	1

for extrapolation, either or: 1. take the nearest anchor point value 2. between high&low points

requires: 1. pick basis/anchor, 2. pre-defined weights of anchor points, 3. convex comb

$$V(s) = \sum_{s_b \in B} \underbrace{w(s, s_b)}_{\text{basis/anchor}} \underbrace{V(s_b)}_{K \text{ parameters}}$$

$$s.t. \quad w(s, s) \geq 0 \; \forall s, s_b$$
$$\sum_{s_b \in B} w(s, s_b) = 1, \; \forall s$$

SL method can be expressed as averagers? KNN

Averagers' Connection to MDPs
- equivalent to a new transition function T'(s,a,s_b) = T(s,a,s')*w(s',s_b)
- we can form a new MDP with function approximation & solve it with VI & Q-leaning
- although in reality MDP doesn't transit to s_b, we can interpret the new MDP as "after transit to s', add immediate second transition to s_b

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

- not super practical because value functions are usually not formed in a convex way

9. POMDP

consider a "non-markov" environment, where the MDP is NOT observable to agent

MDP: A, S | Z (observables)
       T, R | O (observation function)
z: observables
o: observation function, probability of seeing z in state s, O(s,z)
(don't know where i am, but can observe something)

POMDPs generalize MDPs
can represent MDP as POMDP: O(s,z)=1 if s=z
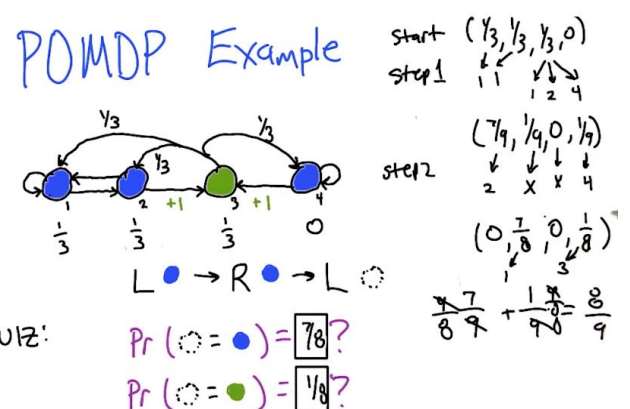<mark>can also represent POMDP as MDP by expanding notion of state</mark> (by exploring & updating T, R & eventually make s "observable")

example: which color are we gonna see?
1. keep track of O(s,z): Prob(s, z)
2. normalize by Prob(z) (e.g. step 2, if we were in s2, go right wouldn't see blue-> remove 1/9 by recalculate average)
3. get Prob(s|z) <Baye's rule!>

State Estimation
we can expand POMDP & make it MDP

POMDP Example

$$\text{start} \; (\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, 0)$$
step 1

$$(\tfrac{7}{9}, \tfrac{1}{9}, 0, \tfrac{1}{9})$$
step 2

$$(0, \tfrac{7}{8}, 0, \tfrac{1}{8})$$

$$\tfrac{\tfrac{1}{3} \cdot \tfrac{7}{9}}{\tfrac{8}{9}} + \tfrac{1 \cdot \tfrac{1}{9}}{\tfrac{8}{9}} \quad \tfrac{8}{9}$$

L ● → R ● → L ○

QUIZ: $Pr(\bigcirc = \bullet) = \boxed{\tfrac{7}{8}}$?

$Pr(\bigcirc = \bullet) = \boxed{\tfrac{1}{8}}$?

1. create belief state b: b(s), probability we're in s
2. turn POMDP to belief MDP where state in bMDP is probability distribution over states of the underlying MDP & POMDP
    belief MDP: b, a, z -> b' (parallel to MDP except b is prob. dist.)
3. actual reward is not observed, can only estimate from observable z
    r = f(z)
4. solve b'(s') = Prob(s' | b,a,z)

    Pr(s'|b,a,z) = Pr(z|b,a,s') Pr(s'|b,a) / Pr(z|b,a)

    = Pr(z|b,a,s') sum_s Pr(s'|s,b,a) Pr(s|b,a) / Pr(z|b,a)

    = O(s',z) sum_s T(s,a,s') b(s) / Pr(z|b,a)

Value Iteration in POMDPs
problem: we turned POMDP to MDP with inf states. if use VI, PI or LP in inf timesteps solve MDP -- must represent in finite way

V0(b)=0
for t> 0, Vt= max(R+gamma * sum(P*V))
in POMDP scheme, V(b) is inf (continuous prob)

(from infinite to finite...)
claim: V(t) is the max of dot product between alpha & believe states where alpha space is finite
interpretation: exists <u>finite</u> number of linear function of believe states (these functions can be infinite) such that we can calculate value of each state by taking the <u>maximum</u> value of all the linear functions

$$V_0(b) = 0$$
$$V_t(b) = \max_a \left(R(b,a) + \gamma \sum_z Pr(z|b,a) \cdot V_{t-1}(b') \right)$$
$$\text{where } b' = SE(b,a,z)$$

CLAIM: $V_t(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b = \max_{\alpha \in \Gamma_t} \sum_s b(s) \cdot \alpha(s)$

Piecewise-linear & Convex
- for each action, take upper surface over belief state (vector), then we can find upper surface over all actions over belief state (union bag of vectors for each actions) (this step convert infinite belief states into finite vector)

- sum(V): sum over bag of vectors -- create bag of vectors for all <a,z> pair & cross-sum

Piecewise Linear and Convex II

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

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

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

- still grow doubly-exponentially, but it's finite

- Q-function can be really big (sum(b), second step)

Algorithmic approach
can get rid of all vectors that are not needed (only keep dominating vectors), shrinks vector space significantly (optimize second step)

RL for POMDPs
VI doesn't necessarily converge!
VI converges in the limit to right value function, doesn't necessarily converge in limit steps; but VI converges to optimal policy in finite steps.
-- not gonna happen to POMDP because infinite # states, can only get near-optimal function approximation with near-optimal policy in finite steps

planning: know everything
RL: don't know, need to interact with environment

RL branches
1. model-based RL: learn model T (learn POMDP)
2. model-free RL: don't bother to learn model (map observations to actions)

Model-based POMDPs
Solve HMM & POMDP:
EM (expectation-maximization)
hypothesis model --**E**xpectation (avg of Gaussian) -->expected value of hidden variables
expected value of hidden variables --(**M**ax.exp.value)--> update hypothesis (model)
... and repeat ...

Is POMDP easy/hard to solve? depends on the POMDP, no absolute answer

Model-free POMDPs
learning memoryless policy: solve LP. may be random
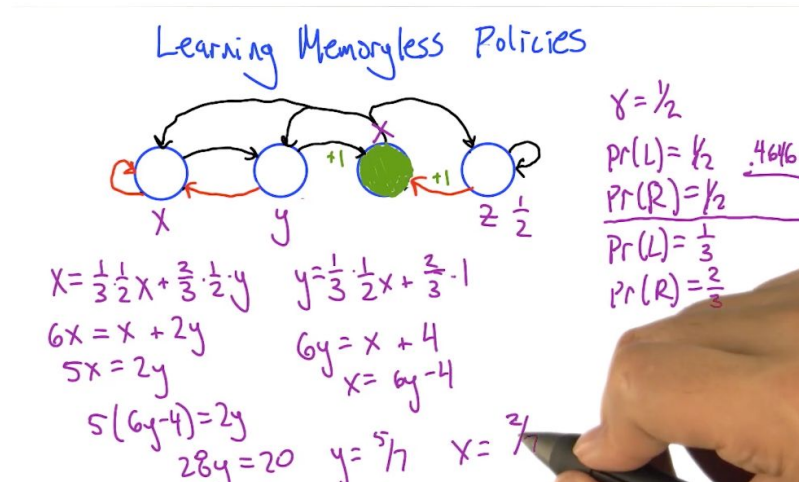
Bayesian RL

track likelihood of possibilities, apply Bayes rule, update likelihood (exploration), maximize reward given learned knowledge (exploitation)
optimal policy: figure out which MDP we're in, get high reward on the way
Bayesian RL blurs dilemma between exploration & exploitation. always exploits in belief space and maximize conditional total reward based on belief

"RL is actually planning in a kind of continuous POMDP, where the hidden states are parameters of MDP we're trying to learn. value function in the continuous POMDP is piecewise polynomial and convex"

algorithms work well: BEETLE (Bayesian exploration exploitation trade-off in learning)

Bayesian RL keeps learned **Bayesian posterior** and use it to **make decisions**
but in "too expensive to use" category, Q-learning still wins

Predictive State Representation (PSR)
motivation:
POMDP: track belief states (distribution of state) which are not observable. but do they even exist?

PSR: predictive state is probabilities of future predictions
calculate prediction (probability of outcome) based on belief state
can we estimate belief based on prediction? -- need more experiments

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

nice thing about the test model: there are pred. prob = 1/0 in some circumstances, can think tests as "what do i need to do in order to figure out where i am" -- find set of tests to find exactly where i am (w. prob ->1 => turn POMDP to MDP)

PSR work: Holmes & Isbelle
also recent work uses least square instead of probabilistic, can learn PSR efficiently

10. Options

options: abstract actions which are "small goals" for a set of states

Temporal Abstraction Options (generalizations)
options = <I, PI, Beta>
I = initiation set of states (options / "super actions" are legal to be executed)
PI = policy (s,a)->[0,1] but in probability form: P(in state s and take a)
Beta = set of states where option terminates, but in probability form: P(s will end in b), mapping s-> [0, 1]
Temporal Abstraction Option Function

example: pac man game, options: eat dot, eat big dot, eat ghost, avoid ghost

make option that's just an action:
I: the state where agent can take this action
PI: for the state, p(a|s)=1 and all other p(a'|s) = 0
Beta: 1 everywhere (once action is executed, option is terminated)

R: discounted expected reward of a set of rewards
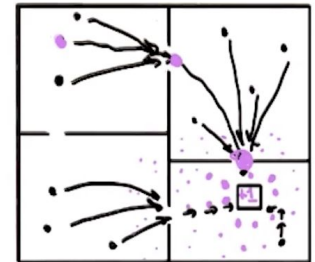F: expectation of ending up in s' after k steps

create Semi-MDP, a generalization of MDP that inherits bellman related properties (semi because it depends on timestep k and hence not really markov)

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

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

$$F = E[\, \gamma^k \Delta_{s_k, s'} \mid s, \text{ k steps}]$$

MDP: atomic actions

SMDP: variable time actions

once create SMDP and bellman operator, can pretend it as MDP

but need to be careful because timestep k of options is variable

it's ok to think SMDP as MDP because:
1. can turn non-markovian to markovian by keeping track of enough history

2. that is what option does. it collects r & s along the way. within one option, it's not markovian because don't know k yet; but once transit out of the option, this "big step" is markovian when look back

Notes on Options
1. inherit optimality, convergence and stability

    given the set of options we have, we can converge to the optimal policy with respect to options (hierarchical optimality)

    BUT, learning with respect to options won't guarantee optimal policy (with respect to actual actions)

    it's ok because actions are relative. we are limited by options of our choice, but were also limited by actions too

2. avoids "boring part" of MDP, skip irrelevant state changes and focus on places where we can make decisions

Goal Abstraction
options can be interpreted as accomplishing goals
1. sequencing actions
2. parallel goals
things can happen in parallel. temporal abstraction is really about managing competing goals
e.g. pac-man, eat small dot, eat big dot, eat ghost, avoid ghost.. all happen at the same time
predator-prey scenario: go around to find prey, but predators are out there to look out for

Beta (termination distribution) can be interpreted as
1. i've accomplished a goal or
2. there's a more important goal i need to focus on and interrupt what i was paying attention to

Goal Arbitration
"modular RL"
different modules try to accomplish some goals, how to arbitrate?
1. greatest mass Q-learning: pick action that has the most mass (chosen by more Qs) (vote on actions)
- but could end up picking least favorable actions because no other actions get enough votes
2. top Q-learnings: pick action that contribute the most value max Q(s,a) (negotiation)
- but one action may benefit a bunch of agents
3. negotiated W-learning: agent with most to lose gets to pick which action to take (min loss)

voting's problem: Arrow's impossibility theorem
it's impossible to construct a fair voting system due to individual different utility units which is impossible to compare

Monte Carlo Tree Search (MCTS) - policy search algorithm
SELECT: decide what action to take

EXPAND: when you don't know enough to make choices, explore available actions that we could take. collect concrete states we could end up in
SIMULATE: simulate different scenarios (behave randomly for a while) (random policy, rollout policy) how far to go? depends on horizon & discount factor. and get an estimation of Q function
BACKUP: back up information we've learned all the way up top with bellman equation
... and repeat....

initial pi: from estimation of Q -> updating Q each loop -> repeat many times -> confident to expand tree -> find optimal policy ->...

after figuring out policy, act greedily & make a move

how to be smarter about it?
- options instead of actions
- instead of random, apply constraints (goals on avoiding damage)

MCTS Properties
1. useful for large state sace
2. need lots of samples to get good estimate
3. planning time is independent of |S| ("voting survey")
4. running time is exponential in the horizon O((|A|*X)^H)
        H: how far in the future to look, depends on gamma
        X: num steps to run
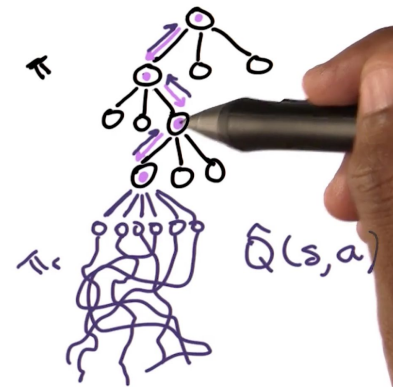
11. Game Theory

2-player zero-sum finite deterministic game of perfect information
zero-sum: means constant sum in all scenarios. not necessarily 0
strategy: minimax
can be solved with LP

Fundamental Result
in a 2-player, zero-sum deterministic game of perfect information (e.g. game tree):
1. minimax == maximin
2. always exists an optimal pure strategy for each player

Von Neumann Theorem
in a 2-player, zero-sum **non-deterministic** game of perfect information (e.g. game tree):
1 & 2 above still hold

in a 2-player, zero-sum **non-deterministic** game of **hidden** information (e.g. minipoker):
1. minimax != maximin anymore!!

2. pure strategy doesn't work any more!!
Von Neumann theorem doesn't hold any more
mixed strategy: prob. distribution over actions
pure strategy: prob(action_pure) = 1

non-zero-sum, **non-deterministic** game of **hidden** information (e.g. prisoners' dilemma)

Nash Equilibrium
n players with strategies s1,...sn
s1*, s2*...sn* are a NE iff exists si* = argmax_si [Utility(s1*, s2*...sn*)]
"no reason for anyone to change their strategy"
can be mixed or pure

Nash Equilibrium Theorems
1. in the n-player **pure strategy** game, if elimination of strictly dominated strategies eliminates all but one combination, that combination is the **unique N.E.**

2. any N.E. will survive elimination of **strictly dominated** strategies

3. if n is finite and exists an i such that s_i is finite, then (mixed) N.E. exists -- ==there exists a N.E. for any finite games==

n-repeated game
when we play game prisoners' dilemma n-times, we'll get n-repeated N.E. (repeating game won't change NE)

12. Game Theory II

Iterated Prisoners' Dilemma (IPT)
what happens if # of round is unknown?
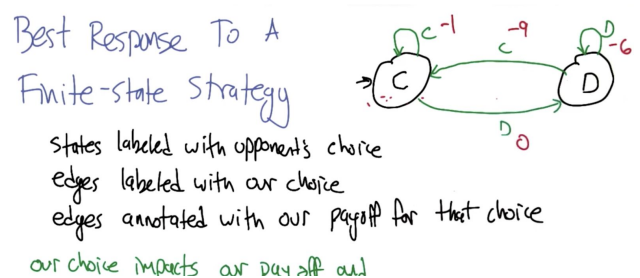uncertain end with discounted factor gamma -> expected # rounds = 1/(1-gamma)

Tit for Tat (TfT)
famous IPD strategy
1. cooperate in first round
2. copy opponent's previous move thereafter

what's the bes response to TfT?
for high gamma, always cooperate; for low gamma, always defect (can be calculated with geometric formula)

NOT subgame perfect

Best Response To A
Finite-state Strategy

states labeled with opponent's choice
edges labeled with our choice
edges annotated with our payoff for that choice

our choice impacts our payoff and

Finite State Strategy

our choice impact our payoff and opponents' future decision
- "the matrix is all you need" is not applied any more (only true for 1-step games)
- it becomes a MDP to us. solve MDP-> VI

mutual best response: pair of strategies where each is the best response to others. also a N.E.!
  （e.g. TfT vs TfT)

Folk Theorem Idea

in <mark>repeated</mark> games, the possibility of <mark>retaliation</mark> opens the door for <mark>cooperation</mark>

(different from definition in math) in game theory, Folk Theorem describes the set of payoffs that
can result from **Nash strategies in repeated games**

Feasible Region

the convex region created by N.E.s == average payoff of some
point strategy == nash strategies

Minimax Profile

a pair of payoffs, one for each player, that represent the payoffs
that can be achieved by a player defending himself from a
malicious adversary
- assuming opponent's goal is to minimize my payoff
- find minimize payoff in respect to each of my action, then pick the strategy which maximizes it

minimax profile: pure strategy
security level: mixed strategy

Security Profile

overlap between feasible region & acceptable region
acceptable region: within acceptable region, the area that has
higher payoff for both players

Folk Theorem

Any feasible payoff profile that <mark>strictly dominates the minimax /</mark>
<mark>security level profile</mark> can be realized as a **Nash Equilibrium**
payoff profile, with sufficiently large discount factor.
- security profile is N.E. with sufficiently large gamma
- reason: if it strictly dominates the minimax profile, opponent can use it as a **threat**
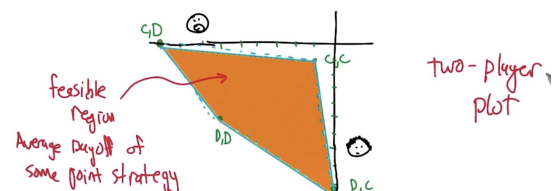
Grim Trigger

"once opponent defects, I defect forever"
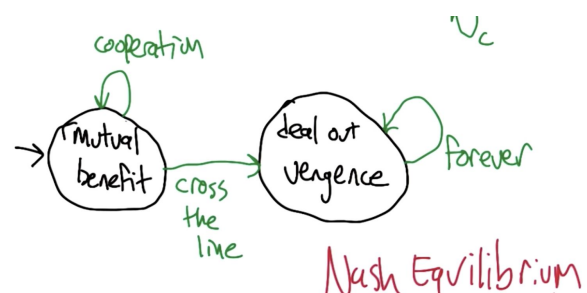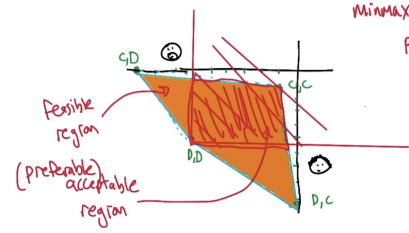has N.E.
but it's an implausible threat

not subgame perfect
(Grim Trigger is supposed to do CCCCC...
but if we alter history & insert D
player will do DDDDD...
changed behavior => not subgame perfect

Implausible Threats
after giving threat, payoff(threat happens) << payoff(don't do it)
e.g. robber with dynamite: "give me $ or i blow both of us up" -> payoff_robber(blow up) <<
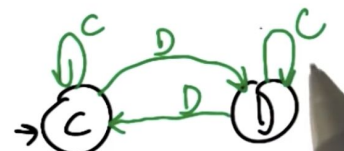payoff_robber(leave without money)

Subgame Perfect
- always best response independent of history
- if alter history at some point, do players have better action? if not, subgame perfect

Grim Trigger: NOT subgame perfect
TfT: NOT subgame perfect

Pavlov
- take advantage of opponent until they pull a trigger
- cooperate if agree, defect if disagree

Pavlov vs Pavlov is Nash! and subgame perfect! < -- plausible
threat

Computational Folk Theorem
For any 2-player bi-matrix average-reward repeated game, can build a Pavlov-like machine for
any game, and use it to construct subgame perfect N.E. for any game in ==polynomial time==

how does it work?
1. if mutual beneficial, Pavlov is possible -> N.E.
2. if not, zero-sum like -> solve an LP -> N.E.
3. if neither works, at most one player improves his behavior (zero-sum like) -> N.E.

Stochastic Games and Multi-agent RL
generalization of repeating game
MDP: RL :: Stochastic game: Multi-agent RL

Zero-sum Stochastic Games
replace max in Q-learning with minimax

with additional agent,
√ value iteration works (but unclear if it can
be solved in polynomial time)

Zero-sum Stochastic Games

$$Q_i^*(s,(a,b)) = R_i(s,(a,b)) + \gamma \sum_{s'} T(s,(a,b),s') \min_{a',b'}\max Q_i^*(s',(a',b'))$$

$$\langle s,(a,b),(r_1,r_2),s' \rangle: \quad Q_i(s,(a,b)) \overset{\alpha}{\leftarrow} r_i + \gamma \min_{a',b'}\max Q_i(s',(a',b'))$$

minimax-Q

√ minimax-Q converges
√ unique solution to Q
√ policy can be computed independently
√ update efficiently: updating Q step can be computed in polynomial time (with LP)
√ Q-function sufficient to specify policy

General-sum Stochastic Games
replace minimax above with Nash

everything above breaks

Ideas
1. repeated stochastic game: folk theorem
2. cheap talk -> seek cooperation -> correlated equilibrium
3. cognitive hierarchy (assume other players have more limited resources)-> best responses
4. side payment (coco value)

General Zero-sum Stochastic Games

$$Q_i^*(s,(a,b)) = R_i(s,(a,b)) + \gamma \sum_{s'} T(s,(a,b),s') \underset{a',b'}{\text{minmax}} \underset{\text{Nash}}{Q_i^*}(s',(a',b'))$$

$$\langle s,(a,b),(r_1,r_2),s' \rangle: \ Q_i(s,(a,b)) \xleftarrow{\alpha} r_i + \gamma \underset{a',b'}{\underset{\text{Nash}}{\text{minmax}}} Q_i(s',(a',b'))$$

Nash minmax-Q

- Value iteration ~~works~~ doesn't work
- ~~minmax~~ Nash-Q ~~converges~~ doesn't converge
- ~~No~~ unique solution to Q*
- Policies ~~can~~ be computed independently   incompatible
- update ~~not~~ efficient   P = PPAD
- Q functions ~~not~~ sufficient to specify Policy   insufficient

13. Game Theory Reloaded

Correlated Equilibria (CE)
rely on shared source of information (correlated distribution)

C.E. Facts
1. C.E. can be found in ==polynomial== time
2. all mixed Nash are correlated (so C.E. exists)
3. all convex combination of mixed Nash are correlated

CoCo Values (Cooperative-competitive values)
"split extra payoff by creating side-payment"
coco value of a game:
U: payoff to player, U': payoff to other players
U,U': general sum game
==CoCo(U, U') = maxmax((U+U')/2) + minimax((U-U')/2)==
1. maxmax: cooperative minimax: competition
2. two pair <U,U'> may be different -> computed separately

CoCo Properties
1. efficiently computable
2. utility maximizing
3. decompose game into sum of two games

4. unique solution
5. can be extended to stochastic games (coco-Q, **NOT NON-EXPANSION**)
6. not necessarily equilibrium -- needs to be binding to reach equilibrium
7. doesn't generalize for n> 2

Mechanism Design
design payoff so that players do what we want them to do
e.g. peer-coaching: student +1 when answer a question correctly; teacher +1 when student gets what shouldn't get right right or what shouldn't get wrong wrong

e.g. King Solomon
similar to auction design, make people reveal their true v values

14. Game Theory Revolution

Coordinating Communicating Coaching (CCC)
in MDP world: define MDP rewards (in game theory: utilities)
in MDP: maximize exp. rewards (in game theory: maximize utilities)
POMDP: strike balance b/w gain reward & gain information (will spend time to gain information if it helps me gain more rewards)
DEC-POMDP: coordinating communicating similar with POMDP, agents decide what to do based on what action max. utility

DEC-POMDPs (decentralized POMDPs)
I: finite set of agents
S: states
A_i: agent i's action
T: joint transition function
R: shared reward function
Z_i: agent i's observation
O: observation function

==all agents act cooperatively== (one shared reward function)
if split R_i => POSG (partially observable stochastic game)

DEC-POMDPs Properties
1. DEC-POMDP has all elements of game theory & POMDPs.
    both POMDP & DEC-POMDP are not computable.
2. POMDP piece-base complete, DEC-POMDP NEXP (non-deteministic exponential complete) - between exponential & double-exponential time

3. agents can benefit from computation

4. optimal solution balances cost of communication with cost of not communicating
5. algorithms, heuristics and applications are known to all agents

Inverse RL
rewards => RL => behavior (in environment, rewards comes in and behaviors come out)
rewards <= RL <= behavior (agent experience external behavior in environment, from which it infers rewards)

Policy Shaping
human tries to communicate with policy directly instead of communicating through rewards
reward shaping: replace reward function with new rewards (give agent new reward)
policy shaping: replace agent's policy with their own command (tell agent what to do)

this is how human communicate with agent-- but human can be wrong

General Calculation
1. each action i.i.d
2. only one action is correct

Combine Multiple Sources of Information
calculate max. likelihood
=max. P(multiple sources agree)
= argmax_a P(a|pi1)*P(a|pi2)...

Policy Shaping

$\Delta_a = $ # yes - # no

General Case $\quad P(a|d_a) = \dfrac{c^{\Delta_a}}{c^{\Delta_a} + (1-c)^{\Delta_a}}$

$C = $ prob correct

only one!	X	Y	Z
π_H	2/3	1/6	1/6
π_A	1/10	1/10	8/10
	O	O	●

Drama Management
3 ways humans communicate with agents:
1. demonstrations (demonstration, policy shaping)
2. rewards (reward shaping)
3. policies (policy shaping, give critique on what is good policy by giving feedback on actions)
4. drama management (human - agent - another human)

author (human) ---design ---> agent (game)
agent ---give experience ---> player (human, controls what goes on in world, creates T)
- agent creates experience to player the way author wants
- how does the author communicate with the agent?

Story: trajectory of plots -> MDP
states: partial trajectory / plot sequences
actions: story action
model: player model (player doing something -> uncertainty the agent faces)
rewards: author evaluation

- since reward comes from author, agent (game)'s goal is to make author happy, not the player
- # states: $O(2^{(2^n)})$ where n=sequence length (hyper exponential)

TTD-MDPs (targeted-trajectory-distribution)
state->trajectories: sequence/story so far (t)
       actions: actions (story/system's action)
       model: P(t'|a,t) (uncertainty from player)
       target distribution: P(T) where T is final trajectory (ending)
       policy: prob. distribution over action
       -- the goal is to match target distribution

can see it as a giant tree -- follow one path of the tree. once choose one action, other branches don't matter

RL Sutton Barto Reading Notes

Sarsa

consider action-value function rather than state-value
1. on-policy TD control
2. transition from <s,a> to <s',a'> and learn V(s,a)
3. on-policy methods: continually estimate q_pi for the behavior pi, and at the same time change pi toward greediness with respect to q_pi (update behavior action while learning value)
4. sarsa <mark>converges with probability 1</mark> to an optimal policy and action-value function as long as all <s,a> pairs are <mark>visited an infinite # times</mark>; policy converges to greedy policy in limits

sidenote:
on-policy TD control convergence requirement: **GLIE** (greedy limit in exploration)
GLIE has following 2 properties:
1. If a state is visited infinitely often, then each action in that state is chosen infinitely often (with probability 1). <explore indefinitely>

2. In the limit (as t → ∞), the learning policy is greedy with respect to the learned Q-function (with probability 1). <policy converges to greedy>

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

Q-learning

1. off-policy TD control
2. the learned action-value function Q directly approximates q* (optimal action-value function), ==independent of policy being followed==
3. difference b/w sarsa & q-learning: whether behavior action depends on learned q (sarsa Y q-learning N); both learns optimal action-value function q*
4. converges with probability 1 to q*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Expected sarsa

instead of maximum over next state-action pairs (Q-learning), take into account how likely each action is under the current policy

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big]$$
$$\leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t) \Big],$$

Papers

Knows What It Knows, Li Littman Walsh

Rmax
- estimates are made separately
- guarantees polynomial bound on timestamps in which it has non-near-optimal policy

KWIK (knows what it knows)
- make only accurate predictions
- has polynomial bound on idk response
- mistake bounded by epsilon, probability of failed run bounded by delta
- total number of steps on idk bounded by Binomial(epsilon, delta), ideally polynomial in 1/epsilon, 1/delta and parameters defining target function H

comparisons:
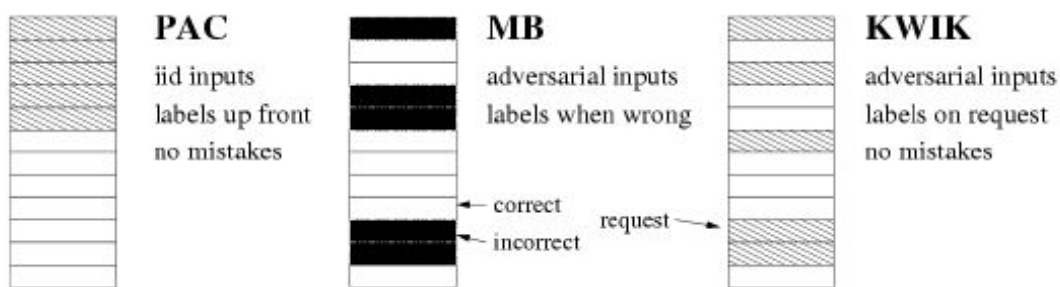
PAC (prob. approx. correct):
- provide label for initial sequence of inputs (correct labels), after which the learner make accurate outputs

MB (mistake bounded):
- provide labels when learner makes mistake, inputs are selected adversarially
- no bound on when the last mistake might be made
- total # mistake is small, incorrect/correct -->0

KWIK (knows what it knows)
- not allowed to make mistakes
- bound on # label requests ("idk")

KWIK learnable classes
1. memorization and enumeration
        rule out all other possibilities
2. real-value functions
3. noisy observations

Learn to Predict by the Methods of Temporal Differences, Sutton88

 learn to predict. example: predict weather
1. TD method: compare predictions with that made the following day
2. advantages:
- more incremental hence easier to compute (conventional: wait till label date & save during training vs. TD: update prediction with one extra observation
- TD converges faster & produces etter predictions
3. single-step & multistep:
- single-step problems: data comes out in observation-outcome pair (TD == SL)
- multi-step problems: next year economy, election result, chess game - TD, adjust prediction with more observations
4. theroem1: on multi-step prediction problems, linear TD(1) procedure produces same per-sequence weight changes as Windrow-Hoff procedure

$\Delta w_t = \alpha(z - P_t)\nabla_w P_t$ SL (back propagation)

$z - P_t = \sum_{k=t}^{m}(P_{k+1} - P_k) \quad \text{where} \quad P_{m+1} \stackrel{\text{def}}{=} z.$ TD definition

$\Delta w_t = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t}\nabla_w P_k$ TD(1)

5. TD(lambda) exponential weighting with recency

$\Delta w_t = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t}\lambda^{t-k}\nabla_w P_k$ 0<=lambda<=1

6. TD(1)==SL; TD(0) weight change only determined by its effect on the prediction associated with the most recent observation
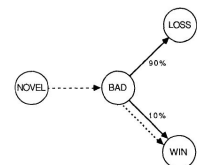7. game-play example:
if novel state aways lead to win:
SL: novel is good (obs-outcome pair b/w novel & wins)
TD: nobel is bad (tracks novel-bad-win & uses knowledge about bad- 90%loss)

if novel->bad->do bad thing
SL: novel is bad
TD: novel has 90% chance losing

if previous evaluation is in error: TD will be affected

if bad state is followed by defeats except when its preceded by novel state (victory):
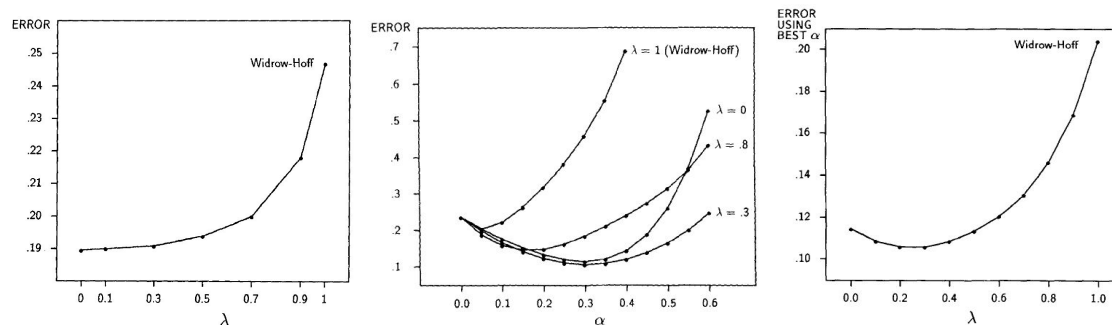TD may perform worse than SL.
<mark>"TD method try to take advantage of the information provided by the temporal sequence of states, whereas SL ignore it"</mark>

8. random-walk example
experiment1: update weight after presenting one train_set + repeated presentation until converge (SL)
experiment2: update weight after each sequence + no repeated presentation

(fig 3,4,5) 3,4 exp1; 5 exp2

- TD(0) in exp1 works best (shocking), because TD(1) Widrow-Hoff procedure minimizes error in training but not necessarily testing. TD(0) converges to optimal estimation
- TD with lambda <1 in exp2 perform better than SL (exp1)
- in exp2, TD(0) is not as good. because it's slowest at back propagating. with repeated presentation it's no problem since the change works back with an additional step in each rep. But single presentation means slow learning. this handicap can be avoided by working backwards thru the sequence (works for offline training)

Residual Algorithm, Baird

1. value iteration form: V(x)<--alpha-- R+ gamma* V(x')
2. if V(x) is separate entry in a lookup table, then ^ is equivalent to 3 RL algorithms:
        TD(0), Q-learning, advantage learning
3.1. direct algorithms
        - when each possible transition is experienced an infinite number of times during learning, then <mark>lookup table</mark> is guaranteed to converge to optimal function as learning rate decays to 0 at appropriate rate

- weight change (exactly TD(0)):

$$\Delta w = \alpha\left(R + \gamma V(x') - V(x)\right)\frac{\partial V(x)}{\partial w}$$

- doesn't converge for a general <mark>function-approximation</mark> system. e.g. star problem. if w's initialized with non-zeroes, w0 will increase every fast
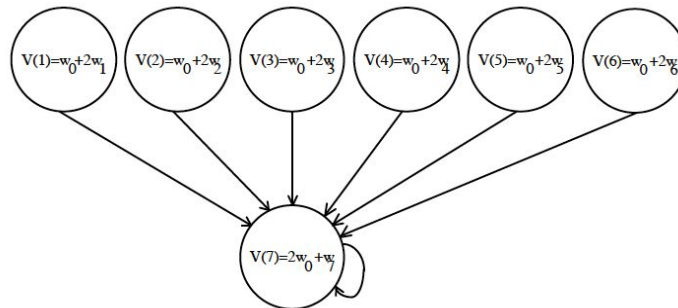

Figure 1. The star problem

3.2. residual gradient algorithms
- residual gradient algorithm: performing gradient descent on bellman residual
- where bellman residual:

$$E = \tfrac{1}{n}\sum_x \left[\langle R + \gamma V(x')\rangle - V(x)\right]^2$$

$$\Delta w = -\alpha\left[R + \gamma V(x') - V(x)\right]$$
$$\left[\tfrac{\partial}{\partial w}\gamma V(x') - \tfrac{\partial}{\partial w}V(x)\right]$$

- if bellman residual != 0 then suboptimal. but suboptimal reinforcement can be bounded although <mark>residual gradient algorithms</mark> have guaranteed convergence
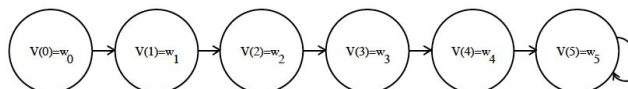does not necessarily learn as quickly as direct algorithms

Figure 2. The hall problem.

- in the hall problem, set w5=0 w4=10, if use direct method, w5->0 w4->0; if use residual gradient, decrease both w5 & w4. info flows both ways & learn slow

3.3. residual algorithms
- direct algorithms: fast but unstable
  residual gradient algorithms: stable but slow
- solution? weighted average! (residual algorithms)

$$\Delta \mathbf{W}_r = (1-\phi)\Delta\mathbf{W}_d + \phi\Delta\mathbf{W}_{rg} \quad \text{where w\_d: direct; w\_rg: residual gradient}$$

$$\Delta\mathbf{W}_d = -\alpha\sum_x\left[R + \gamma V(x') - V(x)\right]\left[-\nabla_{\mathbf{w}}V(x)\right]$$

$$\Delta\mathbf{W}_{rg} = -\alpha\sum_x\left[R + \gamma V(x') - V(x)\right]$$
$$\left[\nabla_{\mathbf{w}}\gamma V(x') - \nabla_{\mathbf{w}}V(x)\right]$$

- guaranteed to converge for appropriate phi

$$\Delta w = -\alpha \left[R + \gamma V(x') - V(x) \right]$$
$$\left[\phi \gamma \tfrac{\partial}{\partial w} V(x') - \tfrac{\partial}{\partial w} V(x) \right]$$

- residual algorithm identical to residual gradient with one term *phi
- how to choose phi?

      on dot line, perpendicular to gradient, no change;

      on the right, result in increase of E, difficult to predict convergence

      closer to delta_w_d, learns quicker

      best: on left of dot line but as close to delta_w_d as possible

Figure 3.   Epoch-wise weight-change vectors for direct and residual gradient algorithms
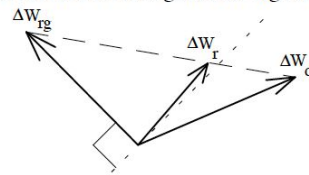
Figure 4.   Weight-change vectors for direct, residual gradient, and residual algorithms.