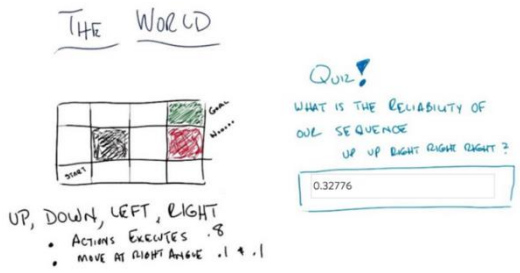
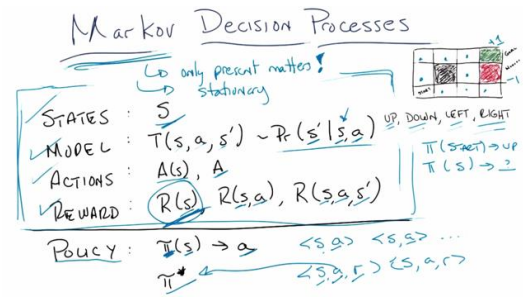
Week 1: Decision Making and Reinforcement Learning





States: set of tokens which describe every state that one could be in,

Model/Transition Function: probability that you'll end up transition to s' given you are in state s and taking action a. This is 'physics' or rules of the world (not changeable).

Action: A(s), things you can do in a given state, a function of state, rule you can play; or A, a set actions not depends regardless of state.

Reward: scalar value that you get for being in a state. Usefulness of entering a state/and taking an action/and ending up into s'

S, T, A and R define the problem, policy is the solution,

Policy: the solution, a function that takes up a state and tells an action that you'll take <s,a> while a is the correct action you want to take to maximize the reward

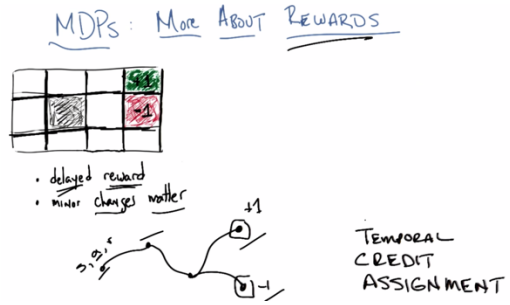
Optimal policy: π, optimized, maximize long term expected reward. (Note, from <s,a,r>, you know s, a and then know the r, based on rewards, find π*)

Recall that in Reinforcement Learning: y = f(x) with z, given x and z, determine function f which can generate y. In Markov process, we know s and r, and we need to learn π to determine a. So, s-->x, r-->z, a-->y, π-->f

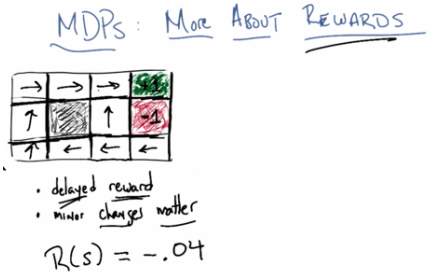
**Markovian property **:

--> Only the present matters; the transition state only depends on the current state

--> Transition model is stationary, rules doesn't change



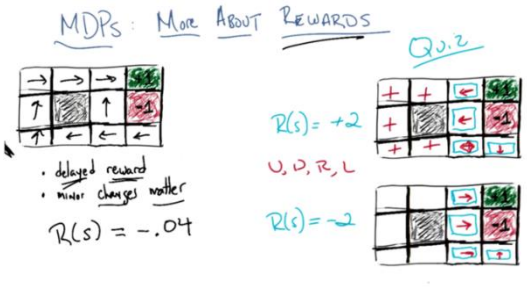
Delayed reward: in each state, we need figure out what action should we take to get the ultimate best rewards at the end.



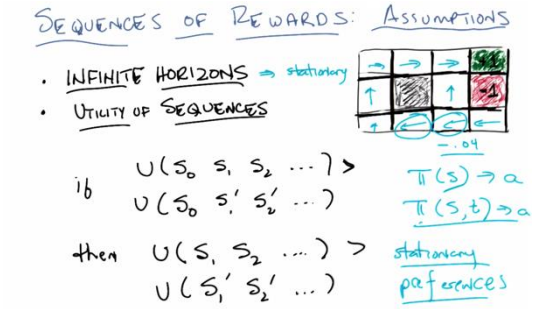
** Minor changes matter: By assigning small negative rewards in each state (except the absorbing state), the learning agent is encouraged to take one of the available actions to leave the current state and pursuit a better outcome.

If R(*s*) is large positive, it encourages you to stay in the game. If R(*s*) is large negative (< -1.6294), it encourages you to leave the world ASAP.

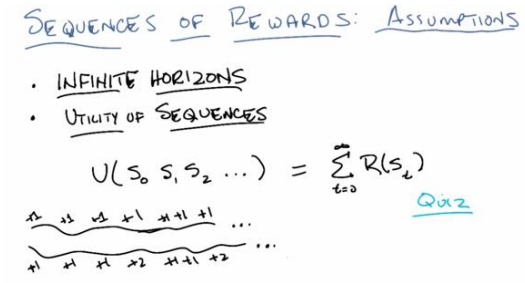
Reward is important to define learning behavior. Rewards depends on domain knowledge. Changing the reward will change the goal of agent.



- Rationale: when the rewards are positive, the goal becomes never leaving the game. When the rewards are negative, the goal becomes leaving the game ASAP.



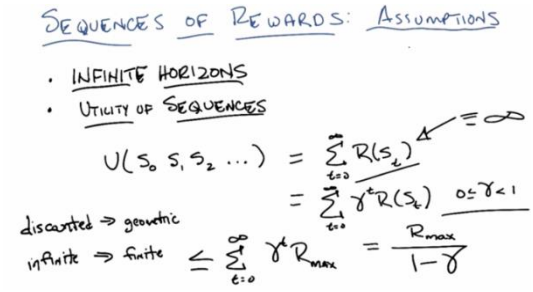
- Infinite horizons: if the time step is finite, the policy might change: π(s,t) --> a, and the Markovian property "π is stationary" will not stand anymore. So, we need to assume we have infinite horizons.



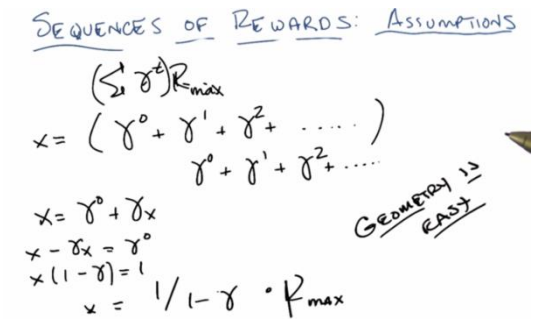
- Utility of Sequences: **stationary preferences**, Utility of all the states is the sum of the rewards of all states.

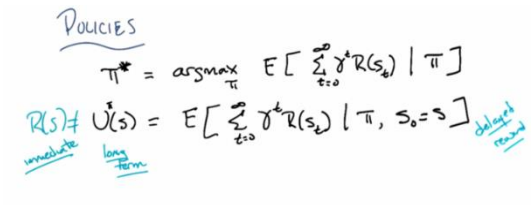
Since there are infinite number of states, the top sequence and the bottom sequence are essentially the same utility ∞ since the reward are all positive if we use the equation above.

So a new equation is introduced below by introducing a discount factor γ:



The equation introduced Discounted factor of Rewards and made the right side of the equation a geometric series so that the utility of sequences is bounded at Rmax/(1-γ), where 0 ≤ γ <1. This way, the infinite horizons is bounded by a finite number.



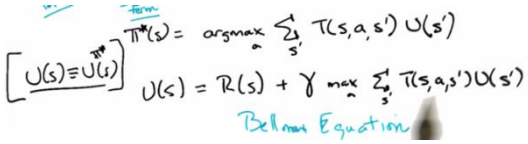


Policy is A mapping from state to action

Optimal policy π*: the policy allow us getting the largest expected discounted rewards when we follow the policy π*

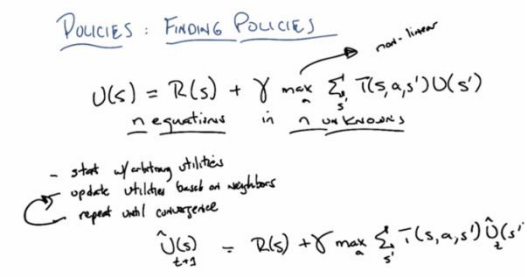
Utility of a state **U(s)** is long term expected discounted rewards from **this point on**; It's delayed reward.

While Reward is immediate satisfaction.



π*(s) is the optimal policy given current state. ** U(s)** is the utility of the state if follow π*. Thus

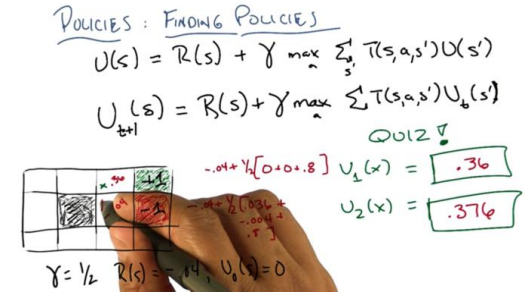
we get the utility of the current state is reward of current state [R(s)] plus the discounted utility from this point on. This is the Bellman Equation.



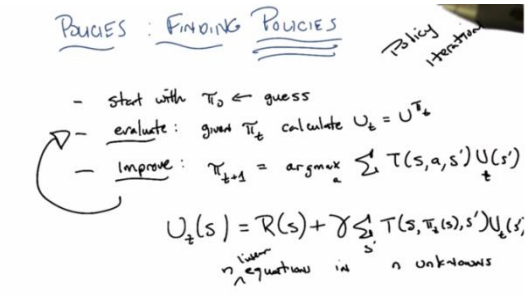
The Bellman equation has *n* equations and *n* unknowns. Because the equations are non-linear (because the max operation), the equations are not easy to solve.

Here list the way to solve the problem: by setting an **arbitrary utility** and update utility based on neighbors, and repeat until converge. This method is called **value iteration**. Neighbors are any state that we can reach.

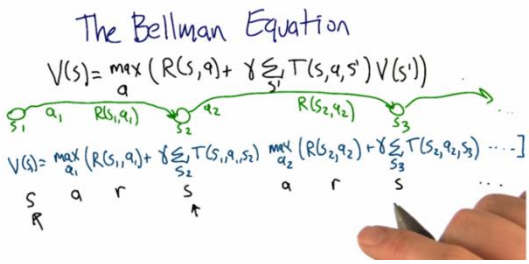
The reason that VI can work is the reward of each state is true and the utility is discounted.



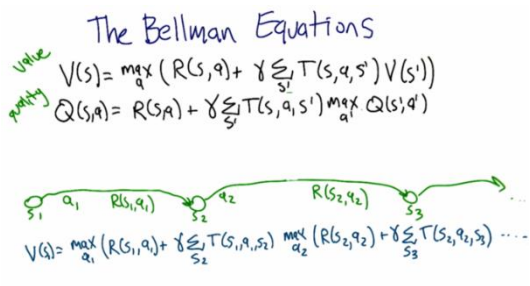
- In this quiz, assuming the initial U of all states but the terminal states is 0. Compute the U at first and second value iteration.



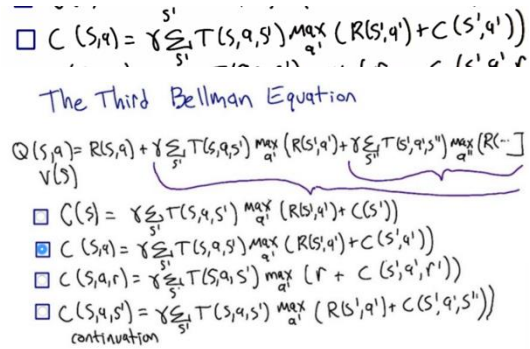
By starting with a guessed policy, the Bellman equation don't have to do Max when calculating the expected utility. So there are n linear equation with n unknowns, and it's fairly easy to get the estimation of utility.



V(s) is infinite sequence with sub V. By regrouping the Value function, we can get Q function as:



And C function as:



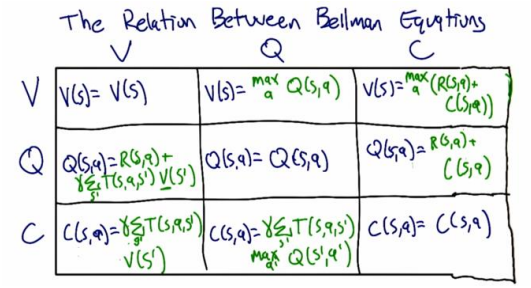
Quiz 5: the correct form of C function

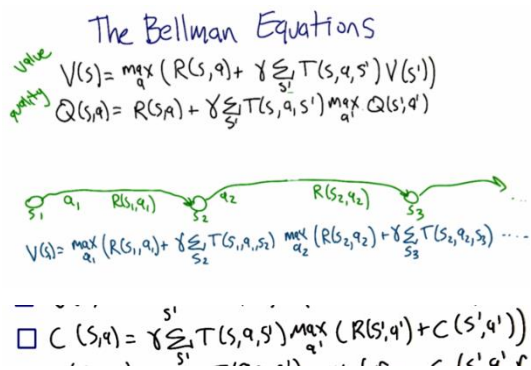
- In the first equation, we do not know which action *a* to use to transition into state *s* ', so we cannot compute the transition function.
- Correct answer. In this formula, all the variables are properly defined, and there is a meaningful relationship between the continuation function and the reward function.

- Notice that there is no meaningful dependency of the max operator on the parameter *r* (which is already passed in to the continuation function). Also, the variable *r* ' is never defined.
- In this equation, the state *s* '' is not defined, so it does not make sense to pass it into the recursive call of the continuation function.

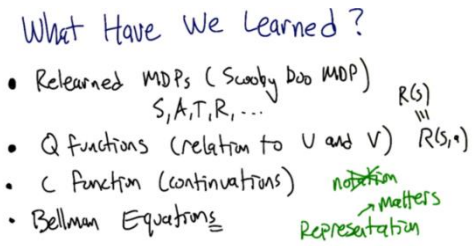
What's the difference between V form and Q form and C form of Bellman equation?

- In value function, V is connected by transition T and reward r. we must know transition and the reward of current state S. V is updated with respect to a state.
- In quality function, we can rely on experience to update Q, this is useful in reinforcement learning when we do not know T and R ahead of time. Q-value is updated with respect to an action.





We don't need reward function when we convert Q to C or V. We can convert C to V and q without knowing the transition function.



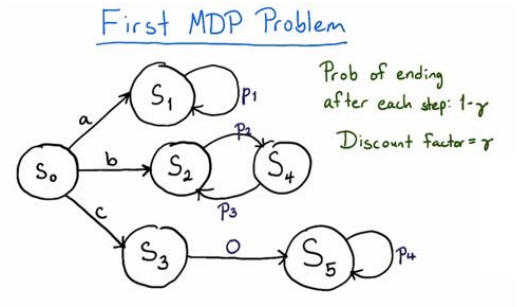
Week 2: Intro to Burlap

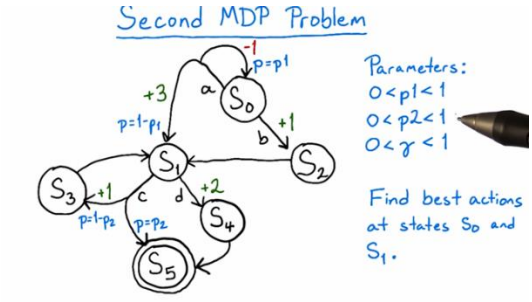
In this introduction lecture to Burlap, I learned how to use BURLAP to setup a MDP.

Steps includes:

- setup domain (using GraphDefinedDomain)
- setup nodes (states)
- setup actions
- initialize state
- setup reward function, termination function, hashFactory
- run Value Iteration
- And get the optimal first action.

See the sample code below and read [BurLAP documentation] to understand the code.

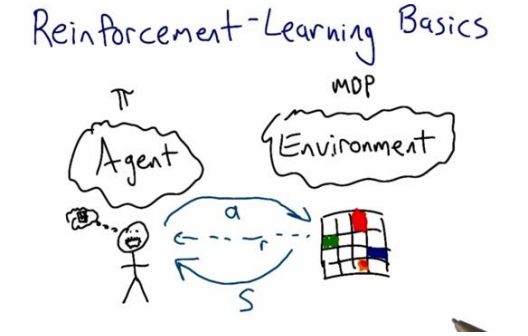




See code: First MDP Problem

The solution of this second problem will not be posted here since it's the quiz of the lecture.

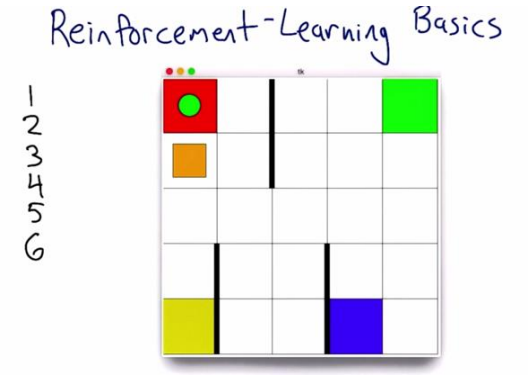
Week 3: Reinforcement Learning Basics



In RL, environment is only available to agent as percepted states (*s*), the agent can interact with the environment by taking action (a) and the environment gives a reward (r) as feedback to tell the agent is the <s a> pair are good or not. The computation is calculated in the agent's head.

The difference of RL and MDP is that in MDP, environment is totally available to the agent, while in RL, Environment is only available through the agent's perception.

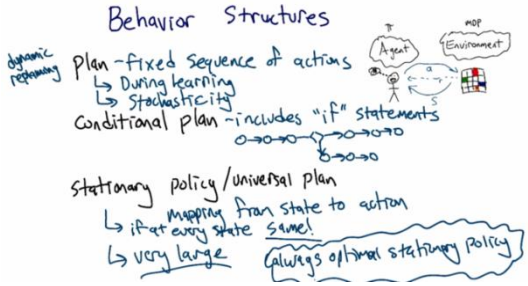
Demo of RL



The small orange square represents the agent, and it can perform 6 actions. The world is the grid with some colored squares and a green dot. The goal what's the game and what are the actions.

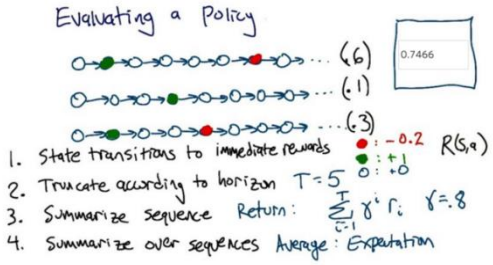
Behavioral structure

The goal is the generate learning algorithm



- **Plan** is a set of fixed actions. Plan won't work during learning or when the environment is only partially known or stochastic.
- **Conditional Plan** includes "if" statements
- **Stationary policy (or Universal Plan)** are mapping from state to action. It can handle stochastic very well but it is very large. **There always is an optimal stationary policy.**

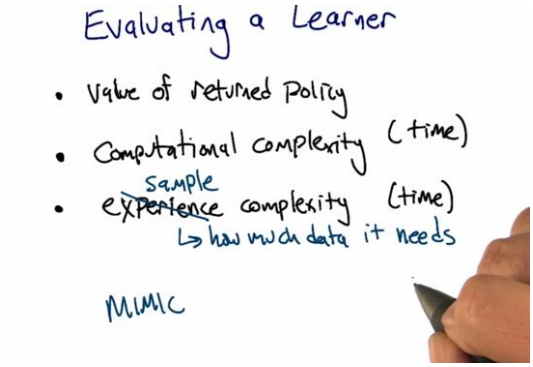
Evaluating a policy



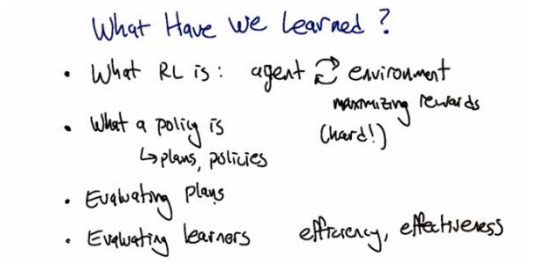
Quiz: evaluating a policy

- The numbers in the parentheses are probabilities of choosing the sequence. R(s,a) is reward function. **Return** is discounted rewards.
- 0.8^1 * 0.6 + 0.8^3 * 0.1 +(0.8^1 +0.8^4*(-0.2))*0.3 = .746624
- This is the expected value of the policy based on the assumption that we index the states in each sequence from left to right starting at **0** on the far left. Interpreting T=5 to mean either truncating at the fifth circle or after the fifth transition (i.e. at the sixth circle) in each row gives the same result.

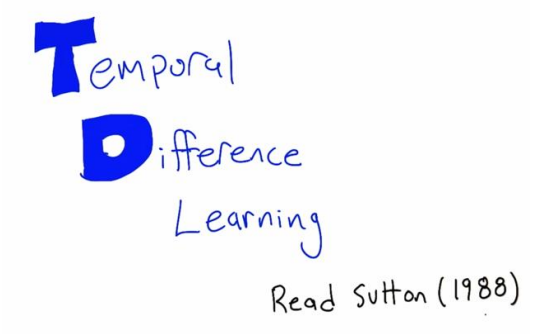
Evaluating a Learner



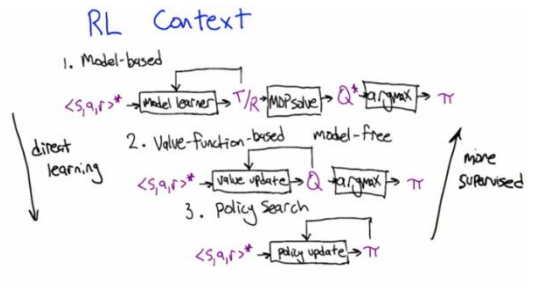
Recap



Week 4: Temporal Difference Learning

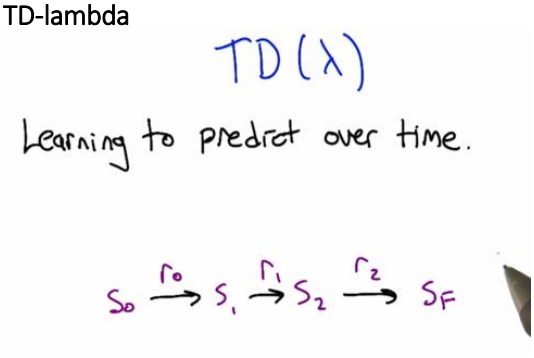


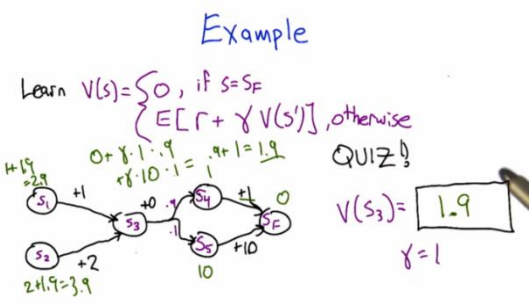
- Read Sutton, Read Sutton, Read Sutton. Because the final project was based on it!



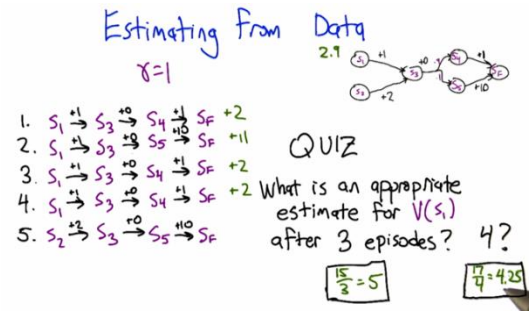
- Model based
- Model free
- Policy search
- Form 1 --> 3: more direct learning
- From 3 --> 1 more supervised

TD-lambda

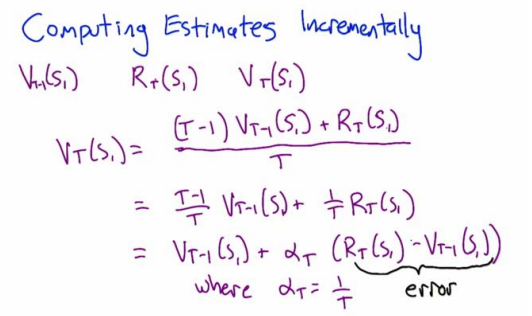




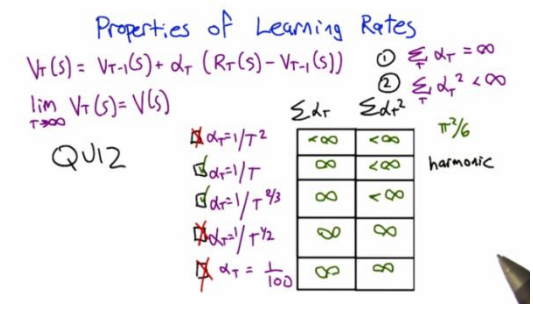
- In this case the model is known, the calculation is easy.



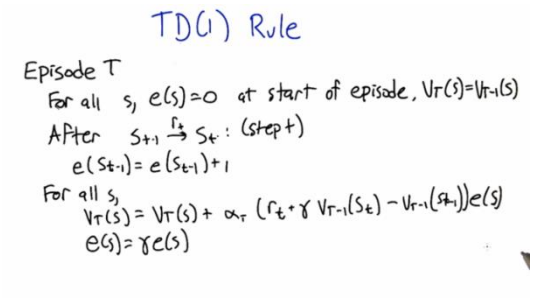
- Remember from the previous lecture, we need to get value from each episode and average over them.



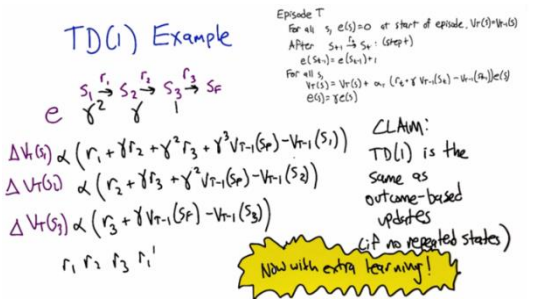
- The rewrite makes the formula looks a lot like neuro-net learning. And alpha is introduced.



Quiz 2: alpha will mke learning converge (tips:if 指数 i 大于 1, 1/(T)ⁱ will be bounded

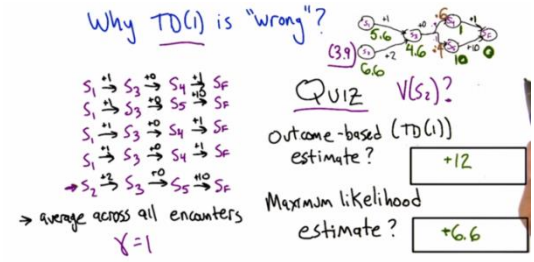


TD (1) rule



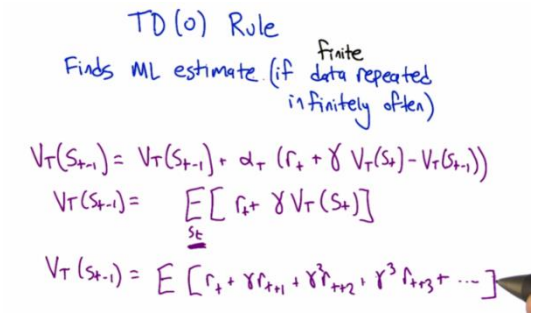
TD(1) with and without repeated states

- When no repeated states, the TD(1) is the same as outcome-based updates (which is see all the rewards in each state and update weights).
- When there is repeated states, extra learning happens.



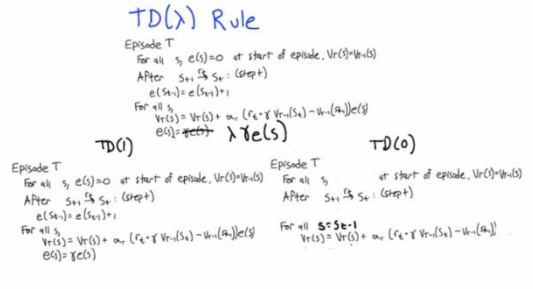
Why TD(1) is "Wrong"

- In case of TD(1) rule, V(s2) can be estimated by average episodes. We only see V(s2) once and the value is 12. Then V(s2) = 12
- In case of Maximum likelihood estimates, we have to kind of learn the transition from data. E.g. for the first 5 episodes, we saw s_3->s_4 3 times and s_3 -> s_5 2 times. So the transition probability can be extracted from data as 0.6 and 0.4 respectively.



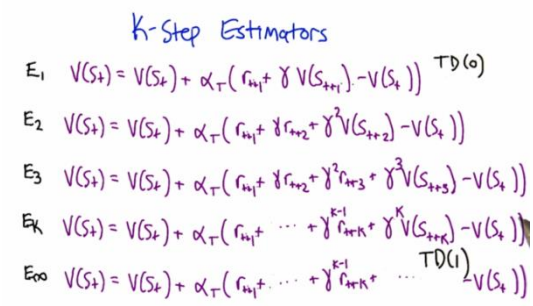
TD(0) Rule

- First of all, if we have infinite data, TD(1) will also do the right thing.
- When we have finite data, we can repeatedly infinitely sample the data to figure out all the ML. This is what TD(0) do.



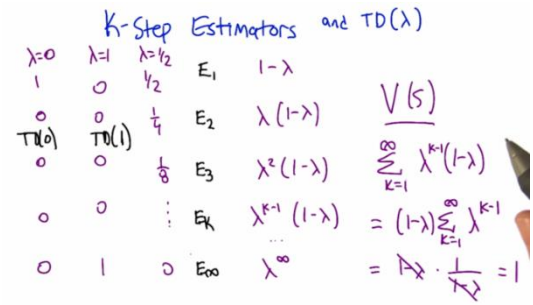
Connecting TD(0) and TD(1)

K-Step Estimators



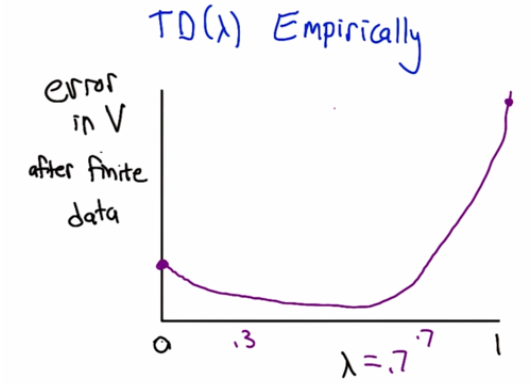
K-Step Estimators

- E1 is one-step estimator (one-step look up) TD(0)
- E2 is two-step estimator, and Ek is k-step lookup.
- When K goes to infinity, we got TD(1)



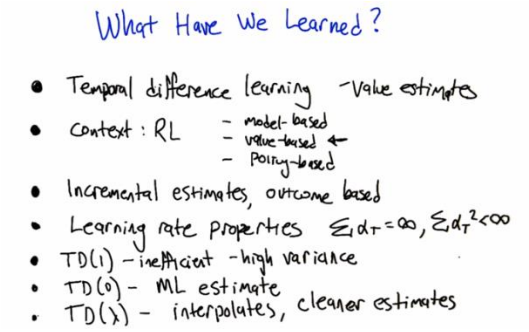
K-step Estimators and TD-lambda

TD-lambda can be seen as weighted combination of K-step estimators. The weight factor are $\lambda^k(1-\lambda)$.

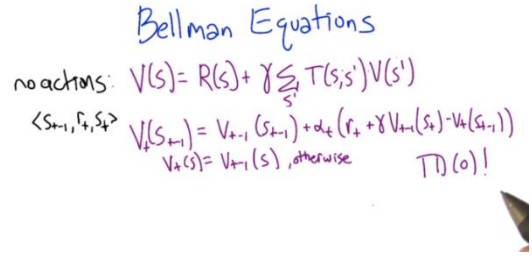


Why use TD-lambda?

The best performed lambda is typically not TD(0), but some λ in between 0 and 1.

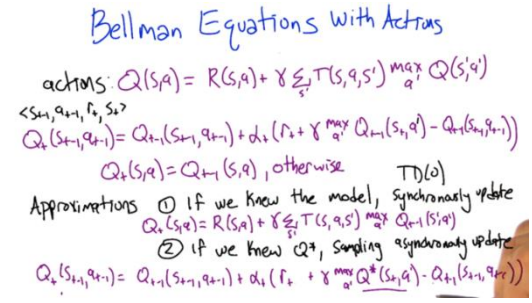


Week 5: Convergence: TD with control



Learning without control:

- no actions by learner. The value function is that value equals reward R(s) plus discounted expected value of the next state. (T(s, s') is transition function.
- The new estimated value function for the state the learner just left is TD(0).

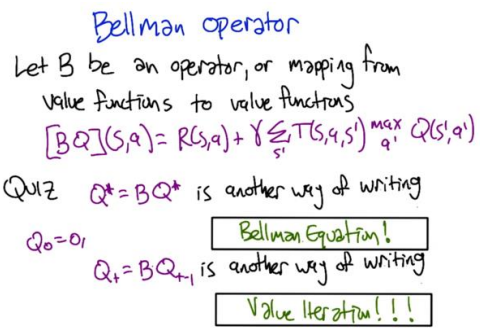


Learning with control/actions.

Q function is used to estimate the value at current state, take a action then get a reward and ended up in a new state $< s_{t-1}, a_{t-1}, r_t, s_t >$. The Q updating rule takes care of two approximations: 1) if model is known, it can be

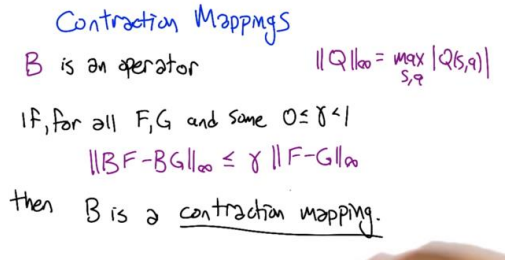
used to update Q; 2) if Q* is known, it can also be used to update Q.

And both will converge.

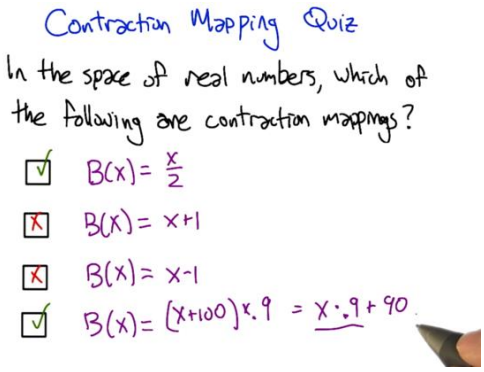


Quiz 1: Bellman Operator

See value iteration here

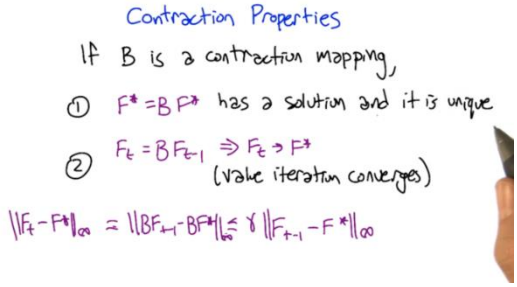


Contraction mapping def: If applying the B operator makes the distance between two functions smaller than the the distance between the original functions.



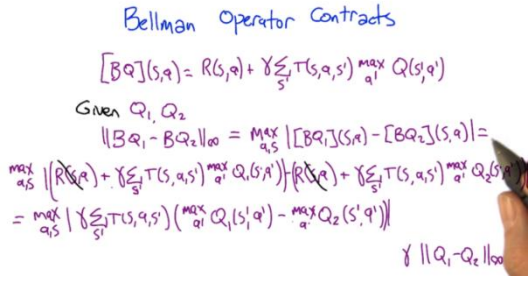
• Practically, if B is multiplying a number in [0,1), B is contraction mapping.

- Take the first option for example: ||B(x) - B(s γ >=1/2 makes ||B(x) - B(y)|| <=γ||x - y||. So B(x) = x/2 is a contraction mapping.

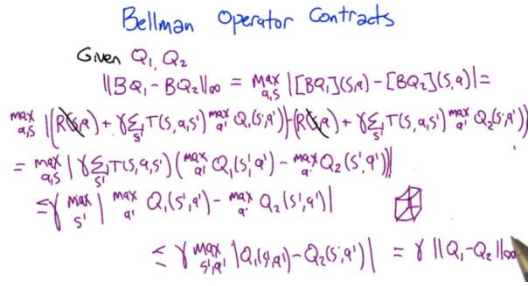


Contraction Properties

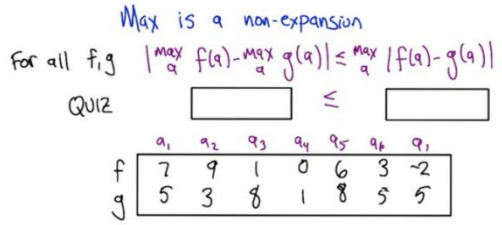
- ① and ② are true, so that $BF_{t-1} = F_t$, F_t will converge at F* through value iteration.
- If there is two fix point, G* and F*, Putting them into the B operator will not change the distance of them because both of them are fixed, and this violates the definition of B.



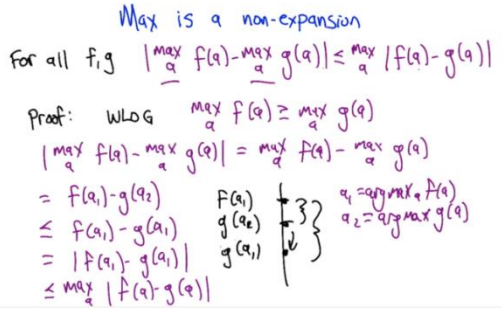
- Applying B operator to Bellman equation and unpacking $||BQ_1 - BQ_2||_\infty$.
- Not combining the *max* a' because the a' are not the same in Q_1 and Q_2



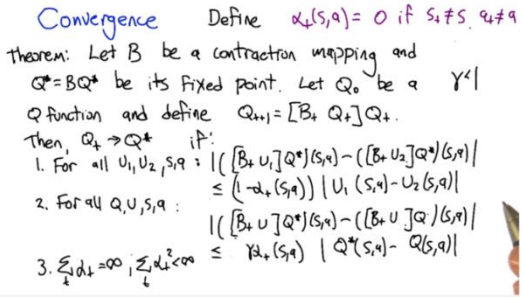
- Stop here, first, will go back. That call this the unfinished proof.



Quiz 3



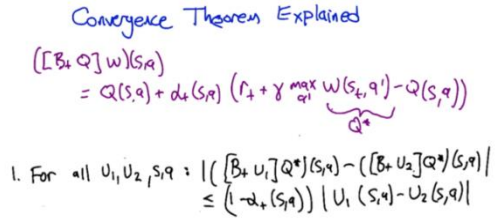
- Max is non-expansion: ||maxf(a) - maxg(a)|| <= max|(f(a) -g(a)|. Using this



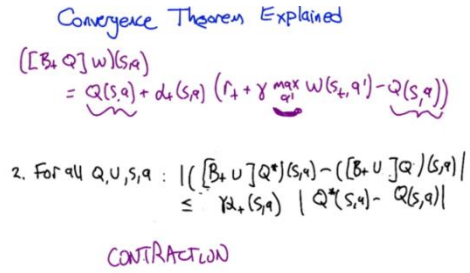
Statement of therom

The three properties need to be true for Q_t to converge to Q*. And they are true

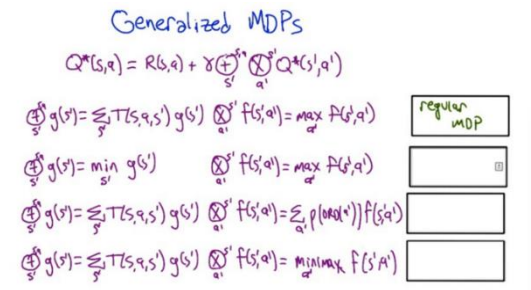
So, Q-learning converges.



- B_t is the operator we are going to update Q at *t* time step. *Q(s,a)* is the Q function value of the state we just left and Q function *w* is the Q value of the state we just arrive. In regular Q Learning update, Q and *w* are the same, here we separated them in the theorem.
- So, rule number one is saying if we know the Q* and use the updating rule (in pink) to update the Q function, the expected value of the one-step look ahead (*w*) can be calculated and the stochasticity will be averaged out.



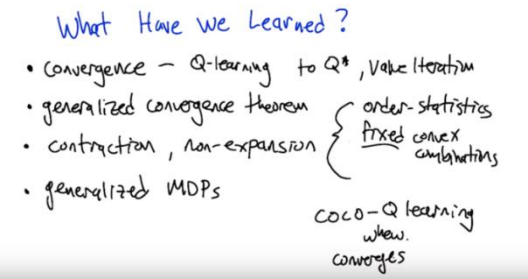
- If we hold the Q(s,a) fixed and only varies the way we calculate the one-step look ahead, the distance between the Q* and Q can only get closer with each update.
- The third condition is the learning rate condition. Which is needed for Bellman equation.



Quiz 4: 1. Decision making on estimated value based on best next action (regular MDP). 2. The environment puts you in the worst possible state and you choose the next best action given the state (risk averse); 3. The state is the expected value but the action to take is based on ranking of the action (exploration-sensitive, min, max, mediocre); 4. (Zero-sum game)

- The take home messages are, these generalized MDPs all converges.

- So the correct answers are: (from top to bottom) worst possible decision, not-optimal decision, and two agent competing decision making (the zero - sum game).

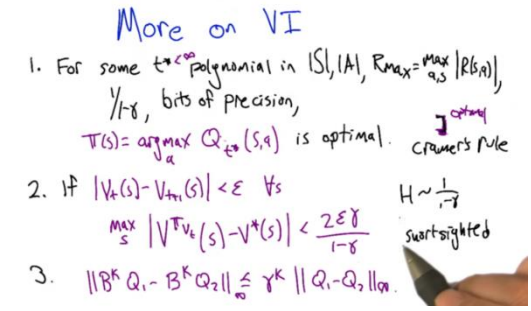


Recap

Generalized MDP can be seen as redefine fix point. Contraction might be something like "收敛" in Chinese. Q-learning converges to Q*. Generalized Convergence theorem uses two Q-functions to prove the convergence of Bellman Equation.

Week 6: Advanced Algorithmic Analysis

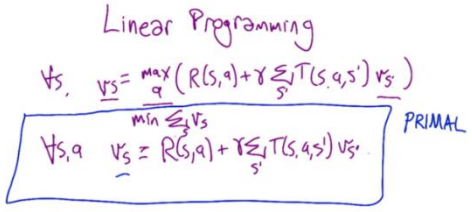
Value iteration



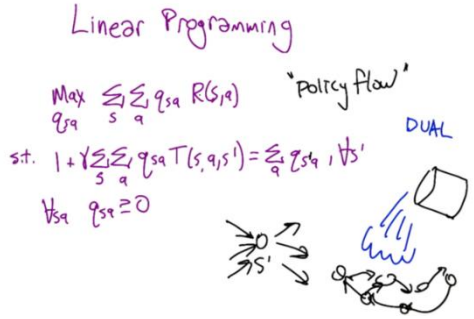
- 1 tells us that VI converges in a reasonable amount of time t* (finite). We know t* exists, but we don't know when that will happen.
- 2 Gives us a bound of difference between the policy now and optimal policy. We can take this bound and test when it is a decent time to stop VI and take the current policy.
- Both 1 and 2 encourage us choosing a small γ. (γ tells us how further in the future we should look). The effective horizon is H ~ 1/(1-γ).
- 3 applying Bellman Operator K times to Q functions will shrink the distance between Q functions.

Linear Programming

- Only one way to solve MDP in polynomial time： solving Bellman equation through Linear Programming. A *linear programming* problem may be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. (Definition extracted from this PDF)
- Bellman equation has one part that is not linear, the *max* function. But it can be expressed by a series of linear function and a objective function `min`.



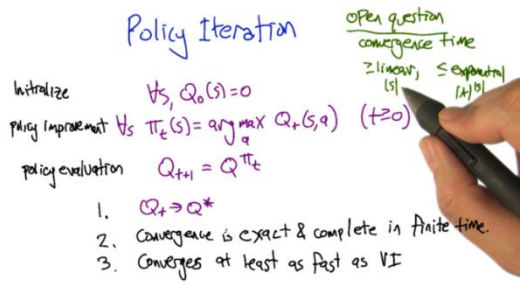
- In the *primal*, the objective function is the minimum of sum of all the Vs;
- In linear programming, we can change constraints to variables and variables to constraints, and the resulting linear program is equivalent to the old one.



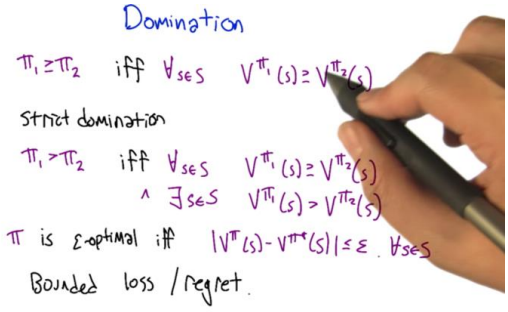
- Dual is a new linear program comes from the old primal version of linear program (没有推导过程）

- q_{sa} is "Policy flow", maximize the expected rewards of all states.
- For each possible next state, we wanted it to be true that the amount of policy flows going through the next state should be equal to the number of the state it has been visited.

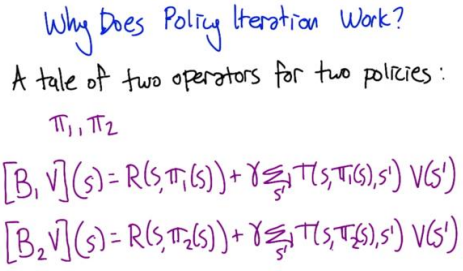
Policy Iteration



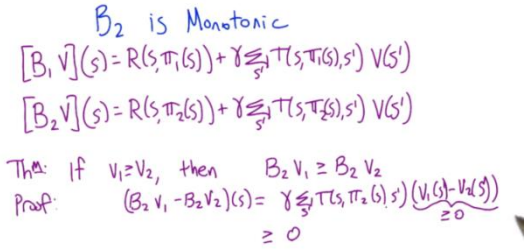
- Initialize the first step Q value of all the states to be 0; improve the policy at time t; Apply the policy to calculate the Q value of t+1 step.
- Convergence time is an open question (but it is finite): >= linear, <= exponential



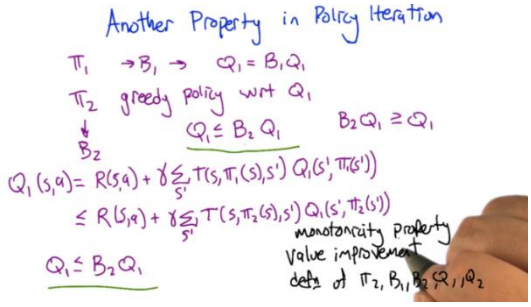
- For every state, if the value of it follows π_1 always equals or is larger than the value when it follows π_2, we say π_1 **dominates** π_2.
- If π_1 **dominates** π_2 and there exits states that $V^{\pi}_1(s) > V^{\pi}_2(s)$, we say π_1 **strict dominates** π_2.
- If for every state, the distance between the value following policy π and following the optimal policy π* is no larger than ε, then π is **ε-optimal**.



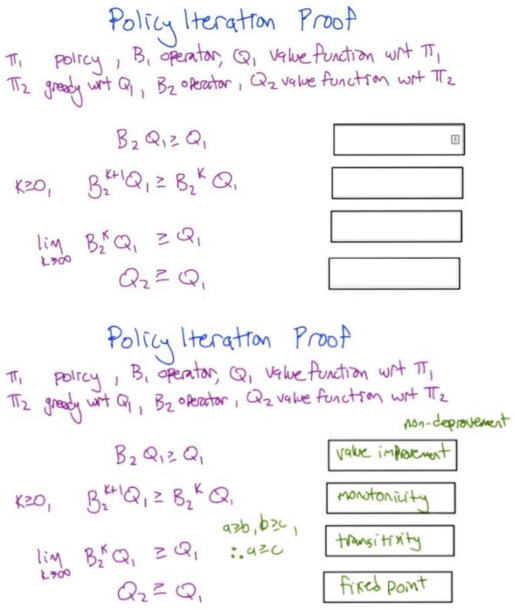
- B_1 makes the update follows π_1 and B_2 makes the update follows π_2
- Applying B operator to two Values will not make them further apart, if the two values are the same, applying B will not make them closer.



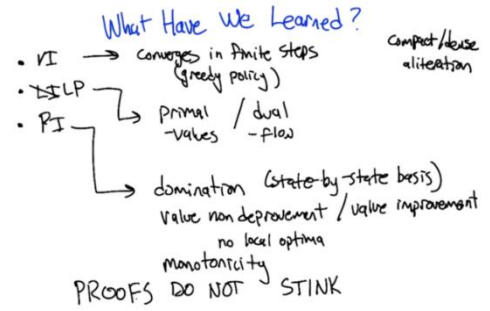
- Theorem: if V_1 dominates V_2, applying B will keep the ordering: $BV_1 >= BV_2$.



- Q_1 is the fix point of B_1;
- π_2 is greedy policy with respect to Q_1
- B_2 is the Bellman operator of π_2
- Applying B_2 (the greedy function wrt Q_1) on Q_1 will add a bound to Q_1, thus getting a better Q_1. This is called **Value Improvement**.



- Value improvement (or value non deprovement): for each state, value will improved until it could not get better anymore.
- Monotonicity: Value can only get better after each iteration.
- Transitivity: a >= b, b >=c, so a >=c.
- Fixed point. If we apply B_2 over and over again on Q1, we will reach the fixed point of B_2, which is Q_2.



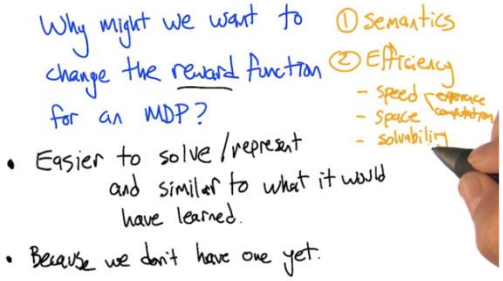
- In Value iteration, greedy policy converges in finite step. This does not necessarily mean value function will converge.

Week 7: messing with reward

This week's tasks

- Watch Reward Shaping.
- Read Ng, Harada, Russell (1999) and* Asmuth, Littman, Zinkov (2008).*
- Office hours on Friday, October 2nd, from 4-5 pm(EST).
- Homework 6.

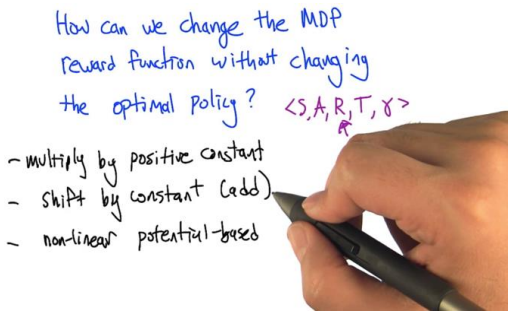
Why changing RF?



Given an MDP, RF can affect the behavior of the learner/agent so it ultimately specifies the behavior (or policy) we want for the MDP. So changing rewards can make the MDP easy to solve and represent

- **Semantics**: what the agent are expected to do at each state;
- **Efficiency**: speed (experience and/or computation needed), space (complexity), and solvability.

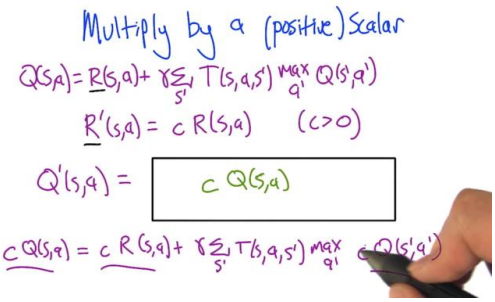
How to change RF without changing optimal policy.



Given an MDP described by <S, A, R, T, γ>, there are three ways to change R without changing optimal solution. (Note, if we know T, then it is not a RL problem anymore, so this part of lecture if for MDP not RL specifically).

- Multiply by a positive constant (non-zero 'cause multiply by 0 will erase all the reward information)
- shift by a constant
- non-linear potential-based

1. Multiply by a positive constant

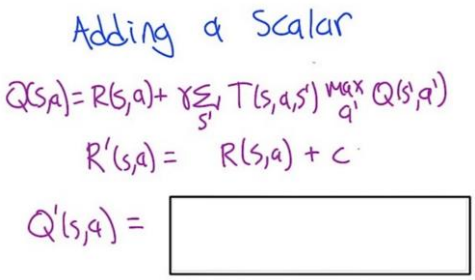


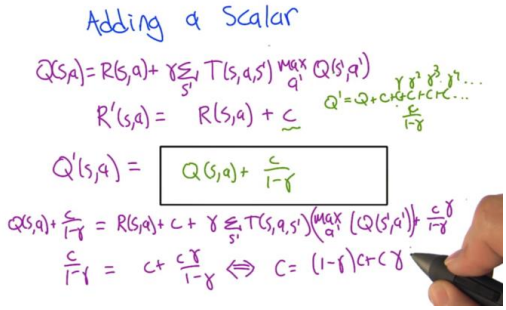
- Q(s,a) is the solution of Bellman function with the old RF R(s,a).
- R'(s,a) is a new RF with is the old RF multiplying by a constant.
- What's the new solution Q'(s,a) with respect to the new RF R'(s,a) and old Q(s,a)?

Here is how to solve the problem:

- $Q = R + \gamma R + \gamma^2 R + \ldots + \gamma^\infty R)$
- $Q' = R' + \gamma R' + \gamma^2 R' + \ldots + \gamma^\infty R'$
- Replace R' with c*R,
  $Q' = (c*R) + \gamma(c*R) + \gamma^2(c*R) + \ldots + \gamma^\infty (c*R)$
  $= c(R + \gamma R + \gamma^2 R + \ldots + \gamma^\infty R)$
  $= c*Q$

2. Shift by a constant

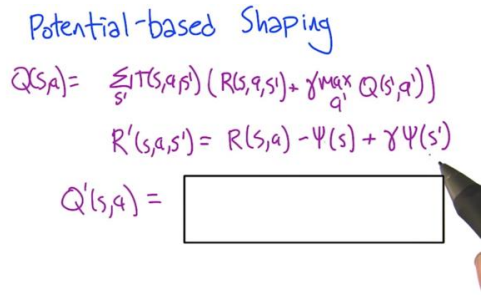




Solution and proof of Quiz 2

- $Q = R + \gamma R + \gamma^2 R + \ldots + \gamma^\infty R)$
- $Q' = R' + \gamma R' + \gamma^2 R' + \ldots + \gamma^\infty R'$
- Replace R' with R+c,
  $Q' = (R+c) + \gamma(R+c) + \gamma^2(R+c) + \ldots + \gamma^\infty (R+c)$
  $= (R + \gamma R + \gamma^2 R + \ldots + \gamma^\infty R) + (c + \gamma c + \gamma^2 c + \ldots + \gamma^\infty c)$
- The first part is Q and the second part is geometric series. So,
  $Q' = Q + c/(1-\gamma)$

3. Nonlinear potential-based reward shaping



Quiz 3: potential-based reward shaping

- $Q = R + \gamma R + \gamma^2 R + \ldots + \gamma^\infty R$)
- $Q' = R' + \gamma R' + \gamma^2 R' + \ldots + \gamma^\infty R'$
- Replace R' with $R-\psi(s) + \gamma\psi(s')$, $Q'=(R-\psi(s) + \gamma\psi(s')) +\gamma(R-\psi(s') + \gamma\psi(s''))+\gamma^2(R-\psi(s'') + \gamma\psi(s'''))) + \ldots + \gamma^\infty (R-\psi(s^\infty) + \gamma\psi(s'^\infty))$

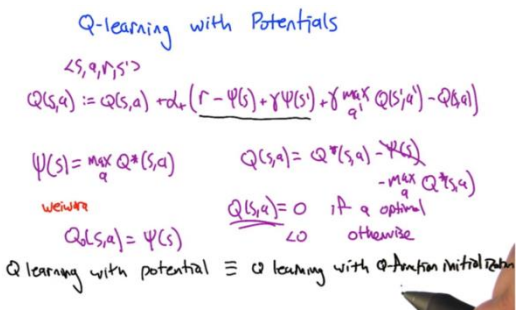
$=(R + \gamma R+\gamma^2 R + \ldots + \gamma^\infty R) + (-\psi(s) + \gamma\psi(s') +\gamma(-\psi(s') + \gamma\psi(s''))+\gamma^2(-\psi(''s) + \gamma\psi(s'''')) + \ldots + \gamma^\infty (-\psi(s^\infty) + \gamma\psi(s'^\infty))$

- The first part is Q. In the second part, most of the elements are cancelling each other out and only has the very first and last elements left. So,
  $Q' = Q + (-\psi(s) + \gamma^\infty \psi(s'^\infty))$
- Given γ is in (0,1), so $\gamma^\infty=0$. Then we have Q':
  $Q' = Q - \psi(s)$

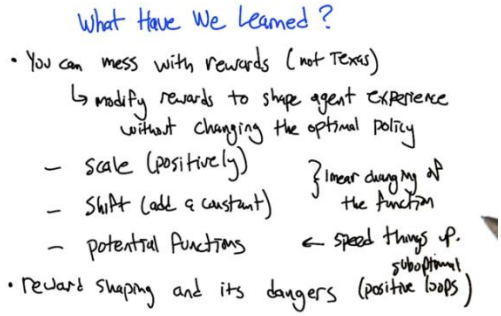
Q-learning with potential

Updating the Q function with the potential based reward shaping,

- Q function will converge at Q*(s,a).
- We know that $Q(s,a) = Q^*(s,a) - \psi(s)$. If we initialize Q(s,a) with zero, then $Q^*(s,a) - \psi(s) = Q^*(s,a) - \max_a Q^*(s,a) = 0$, that means a is optimal.
- So Q-learning with potential is like initializing Q at Q*



What have we learned?



- Potential functions is a way to speed up the process to solve MDP
- Reward shaping might have suboptimal positive loops which will never converge?

Week 8: Exploration

This week

- Watch Exploration.
- The readings were *Fong (1995) *and Li, Littman, Walsh (2008).

Type	state trans-ition	Stoch-astic	solution
Bandits	✘	✔	hoeffding bound to do stochastic decision making
Deterministic MDPs	✔	✘	Rmax to do sequential decision making
Stochastic MDPs	✔	✔	both hoeffding bound and Rmax

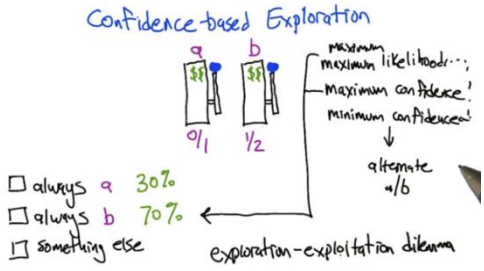
K-armed Bandit Problem

- Each slot has a probability to get certain payoff or no payoff
- Payoff or the associated probabilities are unknown.
- What's the best thing to do? **Exploration**.

Confidence based exploration

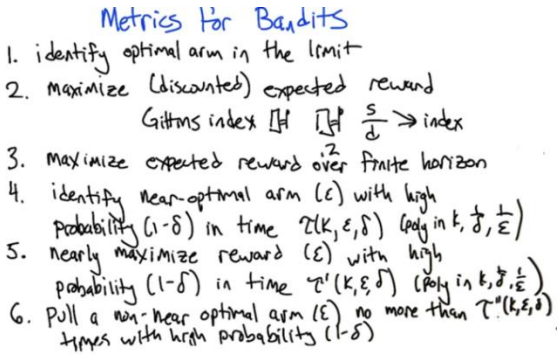
!Quiz 1: K-Armed Bandits, given the observed data, which arm gives the ightest expected payoff and which arm are most comfident](http://upload-images.jianshu.io/upload_images/118382-6578df35f8c00073.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240)

- Arm d gives 1 pay off in the smallest number of try
- Arm g gives most reliable output since it's been pulled the most and gives a good payoff.



- Maximum Likelihood: ① Given current info, figure out the best expected action based on best likelihood -> ② act, and get new info -> repeat ① ② will always choose b
- Maximum Confidence: ① Given current info, figure out the best expected action based on best confidence -> ② act, and get new info -> repeat ① ② will also choose b only
- Minimum Confidence: ① Given current info, figure out the best expected action based on smallest confidence -> ② act, and get new info -> repeat ① ② will also choose to alternating between a and b. But it will not favor the better arm.
- **Exploration-exploitation dilemma**: combine maximum likelihood and minimum confidence. The latter allows to try different actions and the former allows us to choose the best reward.

Metrics for Bandits

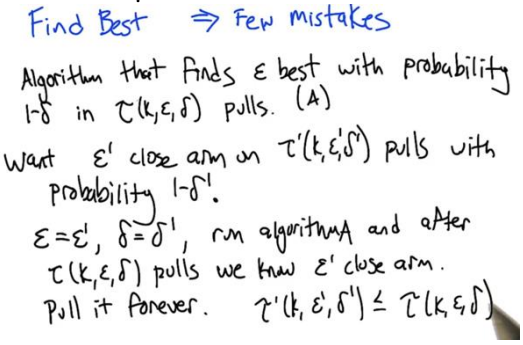


- identify optimal arm in the limit: exploration
- maximize (discounted) expected rewards: exploration + exploitation, but computationally expensive
  - ➢ Gittins index: a real scalar value associated to the state of a stochastic process with a reward function and a probability of termination.
  - ➢ Gittins index works well with Bandits, should not generalize to other RL.
- Maximize expected reward in finite horizon - similar to value iteration
- Identify near-optimal arms (ε) with high probabilities (1- δ) in time τ(k, ε, δ)

(polynomial in k, 1/ε, 1/δ) - The **"get there" goal** is to find near optimal arms. Similar to PAC learning: probably approximately correct.

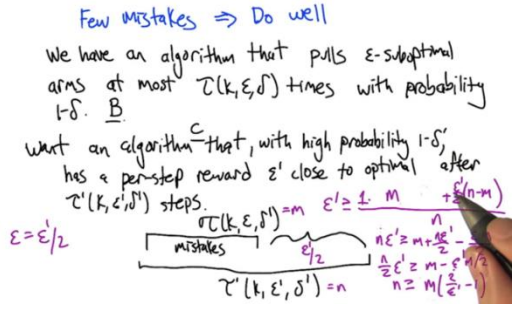
- Nearly Maximize reward (ε) with high probabilities (1- δ) in time τ'(k, ε, δ) (polynomial in k, 1/ε, 1/δ): the **"how to get there" goal** is to accumulate as much more rewards as possible in finite time τ'.
- Pull a non-near optimal arm (ε) no more than τ''(k, ε, δ) times with high probabilities (1- δ): The **"control error" goal** is to minimize mistakes.

Find Best Implies Few Mistakes



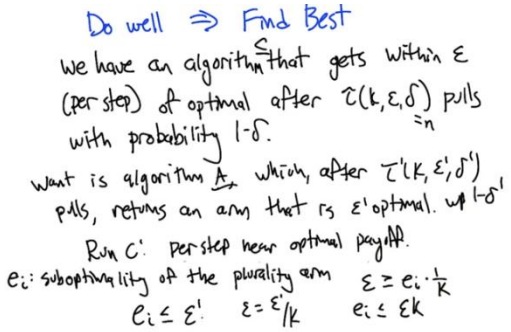
- τ' is the total number of mistakes when we find ε' close arm
- τ' is bounded to τ.

Few Mistakes Implies Do Well



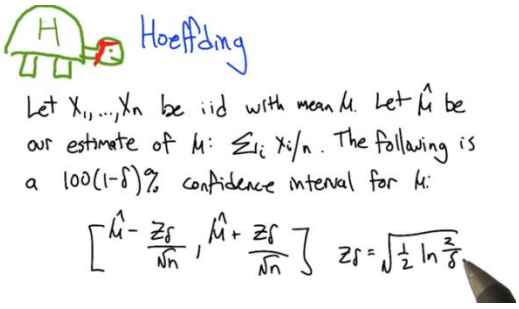
- Is it true that the smaller the *m* is, the smaller the *n* is? not always.
- But it is true that the smaller the *m* is, the small the *n* could be.
- "Do well" means algorithm C can lead to ε' close to optimal arm where ε' < ε.

Do Well Implies Find Best



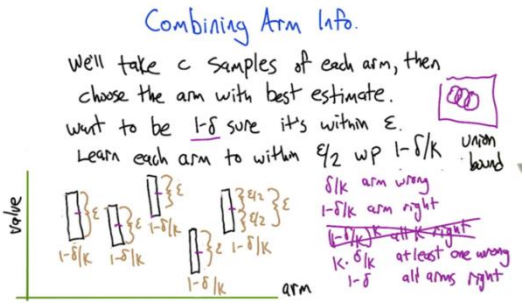
- Given algorithm of one goal, we can derive algorithms to reach the other two.
- So there is no need to pick one.
- Need to know that the three goals are not always solvable (so you might have to pick one?).

Hoeffding



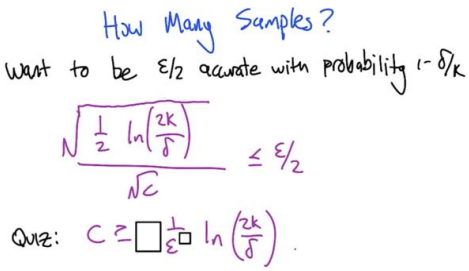
- 0 <= δ <=1. The more confident we are the smaller the δ -> the larger the Z_δ -> the larger the confidence interval
- The larger the *n* is, the smaller the confidence interval

Combining Arm Info



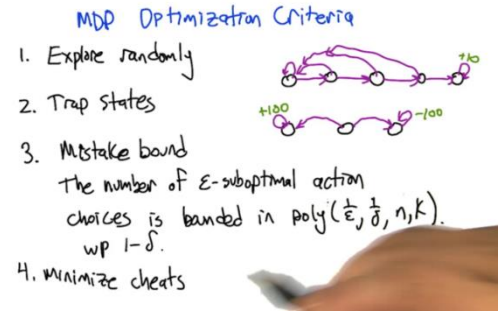
- The length of the bar, ε, is the confidence interval of μ_k.

How Many Samples?

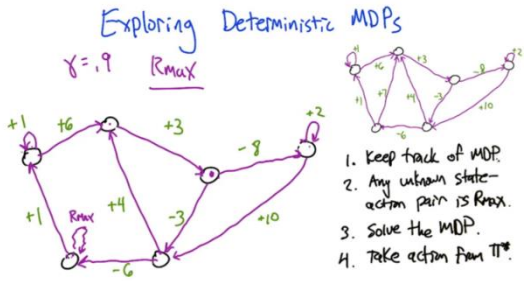


- C depends more on ε and less on δ (δ is in *ln*)
- PAC-style bounds for bandits.

Exploring Deterministic MDPs

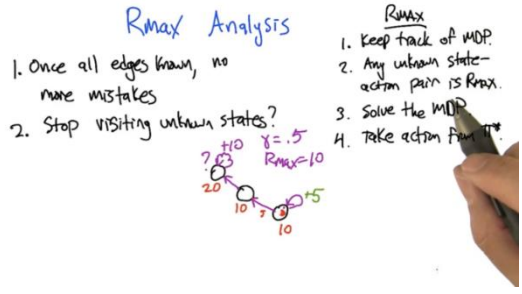


- In MDP, we can only choose action but not state
- We can treat action selection as bandit problem (e.g. *k* action is a *k* arm bandit).

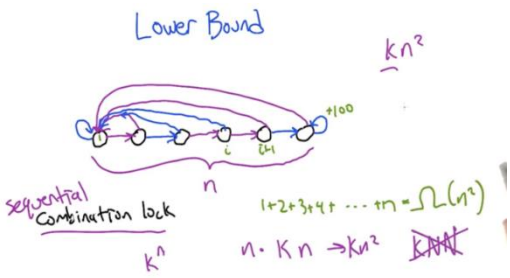


- **Rmax**: assume any state we have not fully explored (unknown state-action pair) has a self-loop of reward Rmax.
- This way, the learner is encouraged to explore the whole MDP.

Rmax Analysis

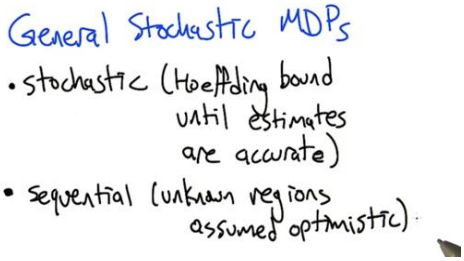


- The example: even with Rmax, there is a possibility that the learning stops visiting unknown states
- Discount factor is important to determine if the agent is going to explore unknown edge.

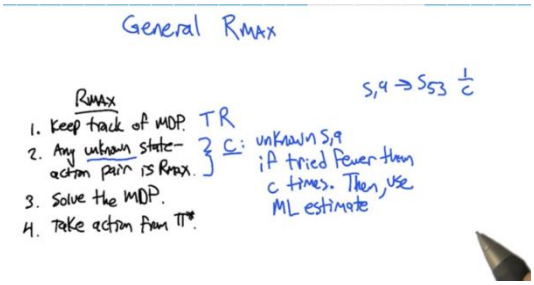


- We have a upper bound saying the algorithm can solve the problem with n^2k steps
- now we are proving that no other algorithms can take less than n^2k steps
- The sequential combination lock has *n* states, in each state the learner can take *k* actions.
- at *i* th state, one action can send the learner to next state (*i + 1*), all other actions will take the learner to state 0, to get back to state *i*, the learner has to take at least *i* steps (the worst case deterministic MDP).
- In the worst case, it might take kn^2 steps to solve a deterministic MDP before stopping make mistakes.

General Rmax



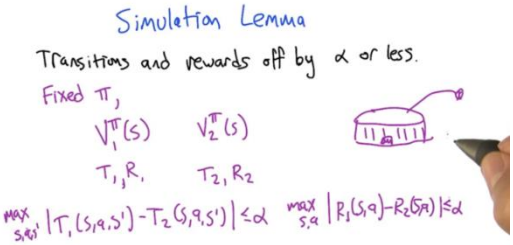
For Stochastic properties, applying hoeffding bound. For sequential deterministic MDP, Rmax can solve the problem. Now applying both, we can solve the stochastic MDP.



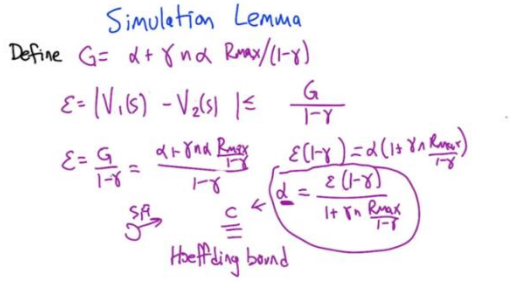
- For Rmax algorithm, define Unknown s, a pair as: if tried fewer than C times. Then use maximum likelihood estimate.
- The C part is hoeffding bound.

Some key ideas for this to work: Simulation Lemma and Explore-or-Exploit Lemma

Simulation Lemma

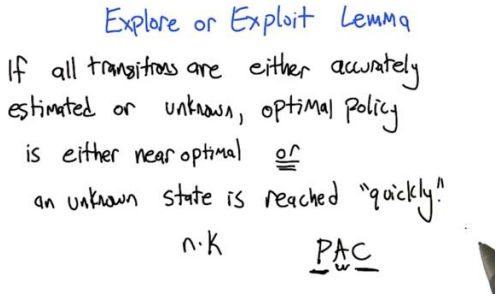


- If we get an estimated MDP that is close to real MDP, then the optimizing the reward based on the estimate will be very close to the optimal policies for the real MDP.
- α is the difference between **estimated** and **real** MDP.
- α could be different if transition possibility and reward are not in the same scale, but we can alway rescale things to that α is between 0 and 1.

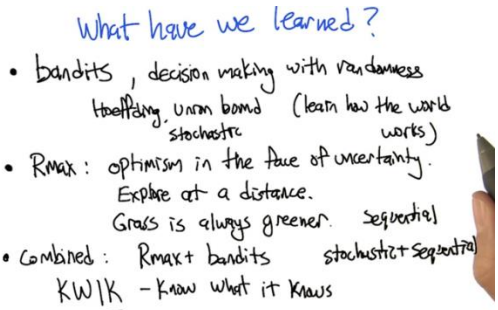


- α is related to ε
- α is used to determine C
- Using **Hoeffding bound** and **union bound** to find a big enough C which can estimate an MDP which is close enough to real MDP.

Explore-or-Exploit Lemma

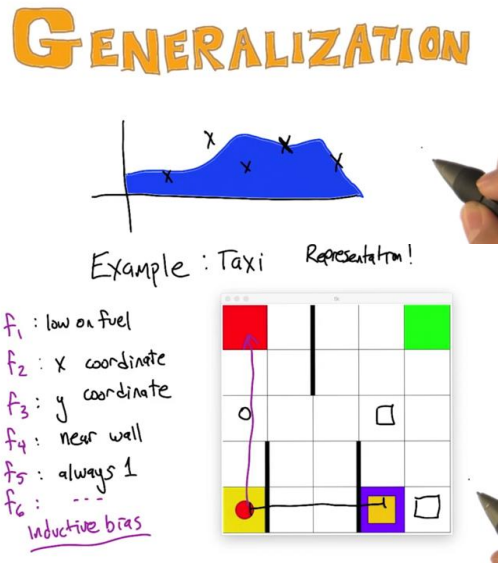


What Have We Learned?

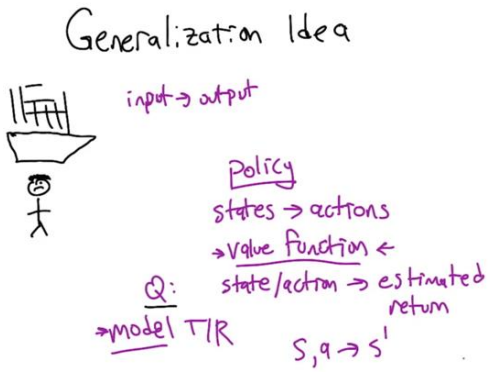


- Hoeffding bound and Union bound let us know how certain we are about the environment
- Rmax makes sure we "visited" things enough to get accurate estimates

Week 9: Generalization



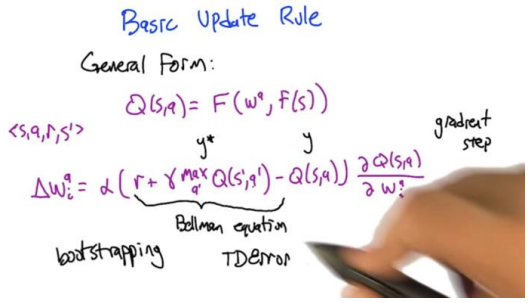
- When there are a lot of states, we can use features to represent states (e.g. grouping states using some common features or using some base states to represent other stats)
- Inductive bias: refer to the way algorithm prefer one solution against another.
- Representation can stay unchanged even when states changed (?)



- In RL, the goal is to learn some input-output mapping.
  - ➢ e.g. Policy maps state -> actions

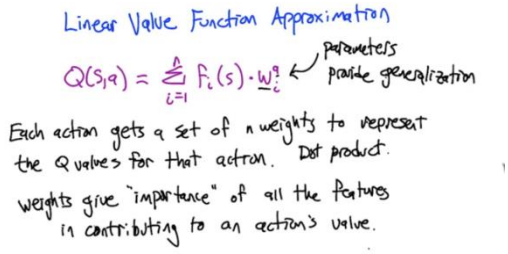
- value function maps state/actions to expected return
- Model maps T/R (transition and reward)
- Generalization can happen on all these levels

- Most researchers are focused on **value function** approximation.

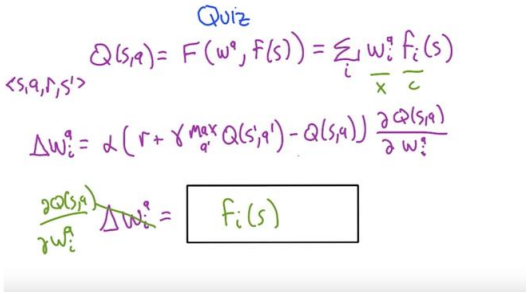


- Q is represented by function approximator $F(w^a, f(s))$. w^a is some kind of weights, f(s) takes a state *s* and generate a series of features.
- Experience tuple <s,a,r,s'> comes in, and do an update.
- TD error= the difference between predicted value (reward + discounted value of next state) and value of current state. TD error tells us the direction to move the parameters *w*. (the prediction is high, low or just right).
- α is learning rate.

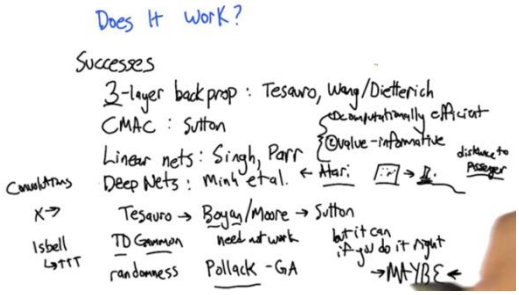
Linear Value Function Approximation



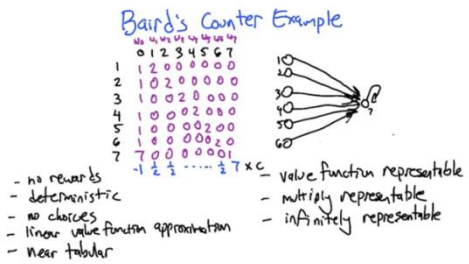
- Q value for an action is represented by features and a set of weights
- w^a_i are weights of features $f_i(s)$, which represent importance of features

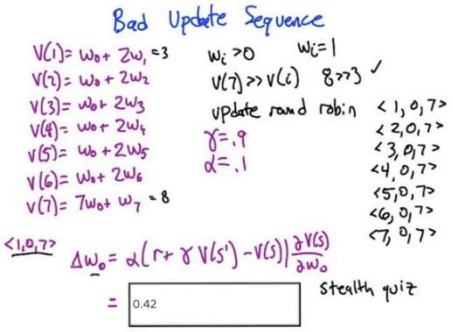


Success and fail stories



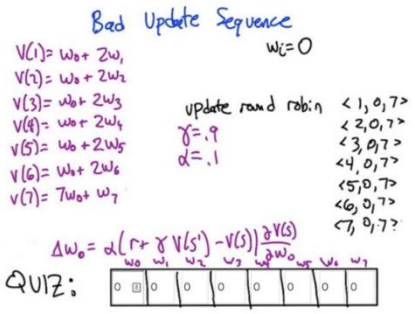
- 3-layer back prop: decision making in Jeopardy
- CMAC(wikipedia) :



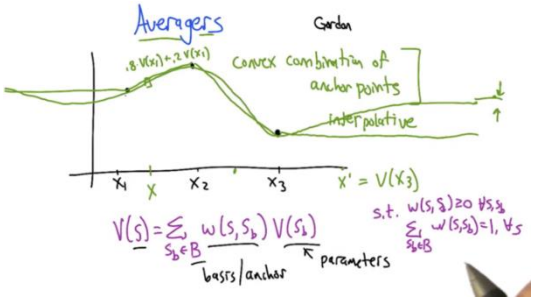


- V(s') = V(7) =8; V(s) = V(1) =3, derivative is w =1; so $\Delta w_0 = 0.1(0 + 0.9 * 8 -3)*1 = 0.42$;

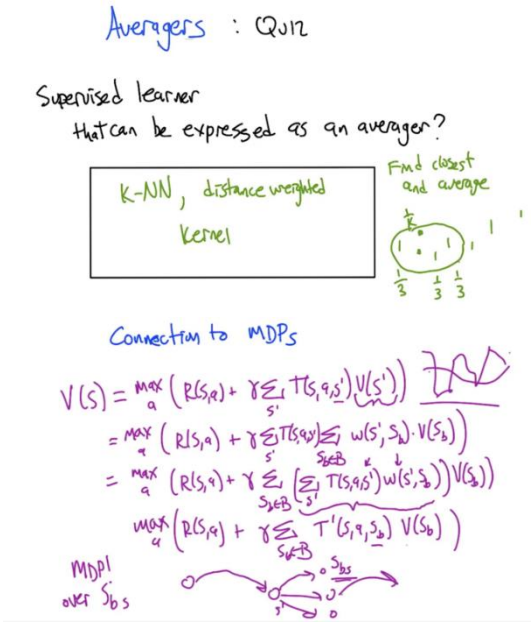
- Δw_0 is 0.42, and Δw_i, i = [0, 6] are all positive, so weight will increase with each update
- Δw_7 is negative when we update <7,0,7>, but w_7 will still increase because it's original value is >> $|\Delta w_7|$
- So, even the algorithm is doing the right thing, the updates will never converge



- In this case, the temporal differences are all zeroes because w_i is 0, which means no updates (changes) are going to happen. **Sticky**
- If at any circumstance, when all values are zero, it could not escape the circumstance unless reward (*r*) is non-zero. **If reward is also zero, we can call the circumstance the solution of the MDP**.
- This holds when MDP is **deterministic**.
- The lesson: Shared weights can "not converge"!

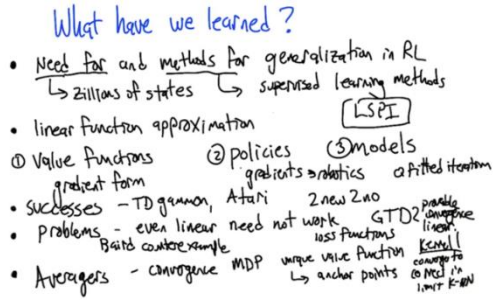


- Problem: given a certain points of value, estimate the function of line. How to estimates the points in-between?
- Naturally, weighted average of neighbors and get convex combinations of anchor points
- For points that are far from anchor points, are good estimate will be the average of the anchor points.
- More anchor points lead to less error in MDP estimation.



- Using base states, we can represent the original MDP with MDP over S_bs

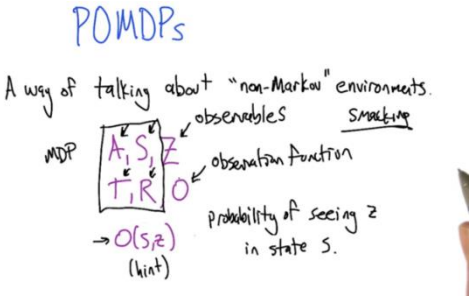
What have we learned?



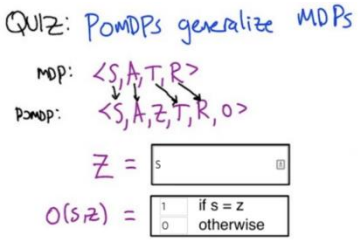
- **Need** for generalization: some problem have Zillions of states; Method: applying supervised learning method.
- Methods: Linear function approximation.
  - value functions 2) policies; 3) models;
- Success: TD gamma, Atari, Fitted Q-iteration (but still have need not work cases)
- Problem cases: even linear need not work (baird counter problem)
- Averagers: can be viewed as MDP itself. Use anchor points to represent the whole MDP.
- **LSPI**: Least Squared policy iteration

Week 10: POMDPs, Partially observable MDPs

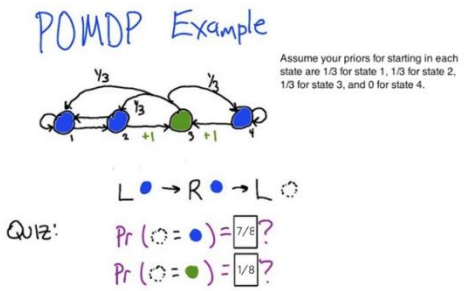
POMDP



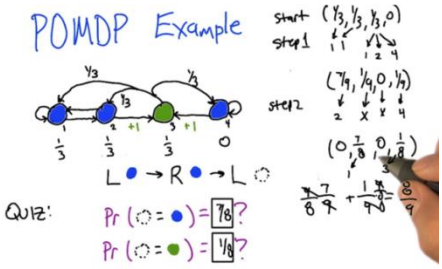
- POMDPs generalizes MDPs.
- In POMDP, MDP (represented by <S,A,T,R>) is not directly observable to the agent. we can only observe the states. Z are observables to the agent, Z is a set of states in S.
- O(S,Z) is the mapping function of Z to <S,A> (e.g. Given Z, what's the probability of S).



- When O(S,Z) = 1, the observable Z is S. So, Z and O together determines the relationship between S and Z in the POMDP.

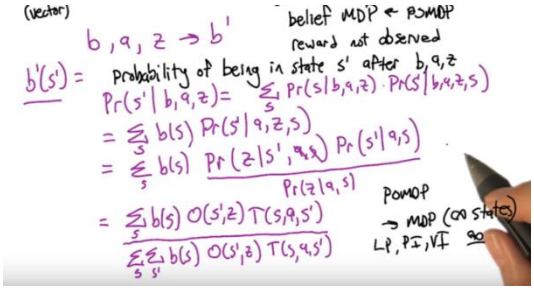


Solution



- Keep track of probability of ending up in each state after each step.
- There is a normalization after each step (not the right thing to do but works.).

State Estimation

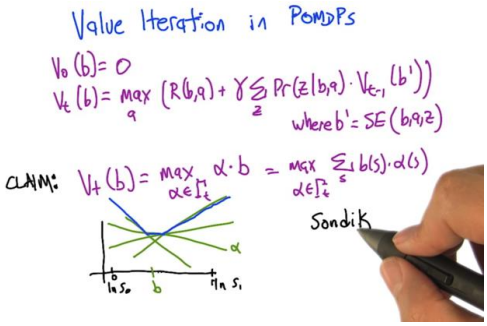


- Belief state b(s) is a probability distribution vector of whether we are in s.
- Using belief state can turn POMDP into belief MDP <b, a, z, b'>

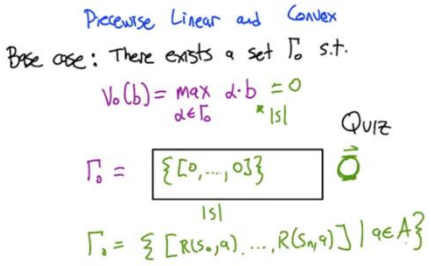
b'(s') = Pr(s'|b, a, z)
= Pr(z|b,a,s') Pr(s'|b,a) / Pr(z|b,a)
= Pr(z|b,a,s') sum Pr(s'|s,b,a) Pr(s|b,a) / Pr(z|b,a)
= O(s',z) sum_s T(s,a,s') b(s) / Pr(z|b,a)

- Note: belief MDP has infinite number of belief states which make VI, LP, PI impossible because they can only deal with finite number of states.

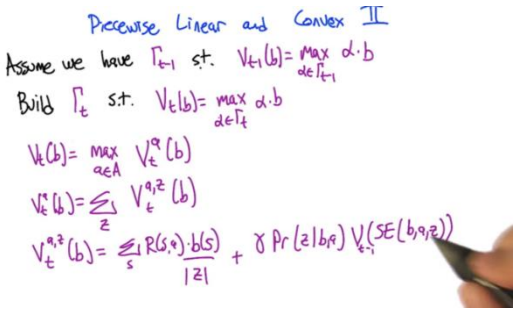
Value Iteration in POMDPs



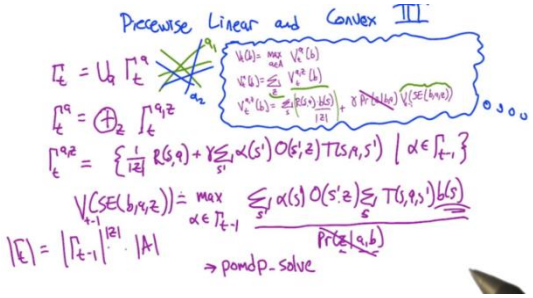
- Initialized $V_0(b)$ as 0
- $V_t(b)$ can be written as something similar to bellman equation
- Claim $V_t(b)$ can be represented by the maximum of a set of linear functions of b and α. α is a finite set of vectors. This is called piecewise linear and convex.



- Γ_0 should be a vector of zeroes.



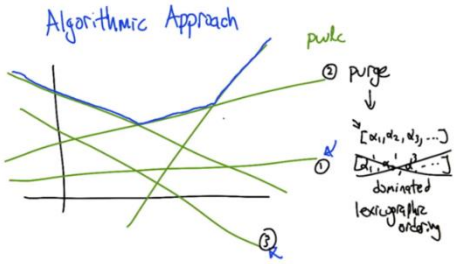
- Piecewise linear means $V_t(b)$ can be represented by $V^a_t(b)$, $V^a_t(b)$ can be represented by $V^{a,z}_t(b)$ and $V^{a,z}_t(b)$ can be represented by something like a Bellman equation



- In a similar sense, Γ_t can be represented by Γ^a_t , $\Gamma^{a,z}_t$ and something like a Bellman equation.
- Thus the belief MDP becomes computable.
- Pomdp-solve is an existing solver for pomdp.
- Guaranteed to be exponential because size of Γ is exponential in respect to the size of observables: $|\Gamma_t| = |\Gamma_{t-1}|^{|z|} \cdot |A|$

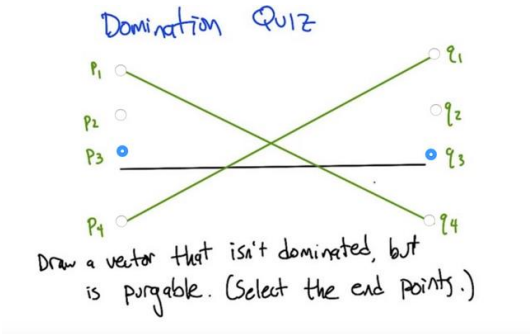
- The calculation could be faster if Q function ($V^a_t(b)$ is the Q function) has a small representation.

Algorithm



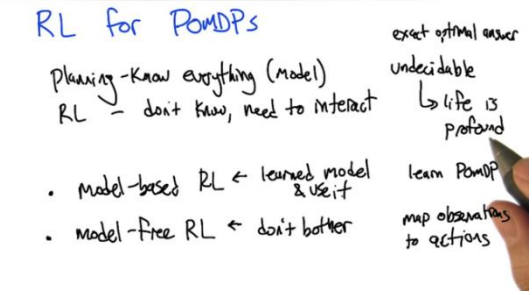
In the figure:

- All the linear functions (vectors) are dominated by the blue lines which involves three linear functions.
- Function ① and ③ can be purged because they have no contribution to the overall vector

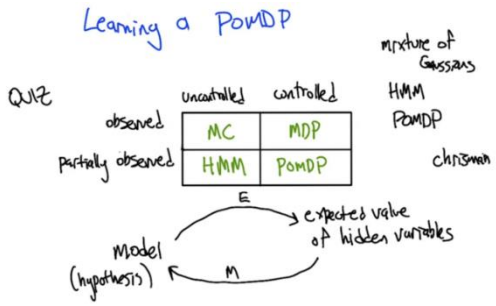


- Domination: if one vector is always less than (in this case) another vector, we say it's dominated by that another vector.
- Vector is purgeable if it does not contribute to the max

Learning POMDP



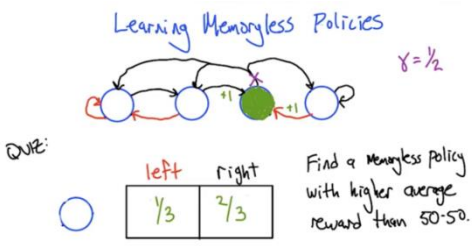
- Planning: if know the model of POMDP, can run value iteration. Since POMDP have infinite number of states, we can only get approximation of optimal policy.
  - VI may not be converge
  - PI is not possible because infinite believe states
- Difference between planning and reinforcement learning:
  - Planning: know the model - planning in POMDP is undecidable.
  - RL: we don't know the model have to interact to learn
- RL has 2 types:
  - Model based RL: learn the model (POMDP) and use it
  - Model free RL: don't learn the model but mapping observables and action.



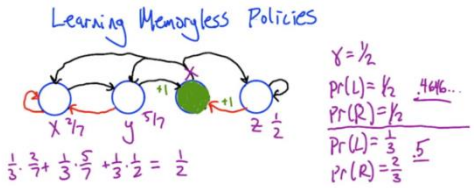
- MC: *Markov Chain*, observed but uncontrolled; HMM: *Hidden Markov Model*, partially observed and uncontrolled; MDP: observed and controlled. POMDP: partially observed and controlled.

- EM: *Expectation Maximization*, an procedure to learn the hidden property in HMM or POMDP.
- Model based Learning.

Learning Memoryless Policies (Model free RL of POMDP)



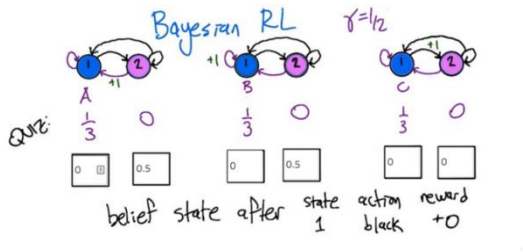
- In the MDP, There 3 blue states, to get to the green state, the agent needs to move to right in 2 of the states and move to left in another state. So the solution for this quiz is 1/3 and 2/3.



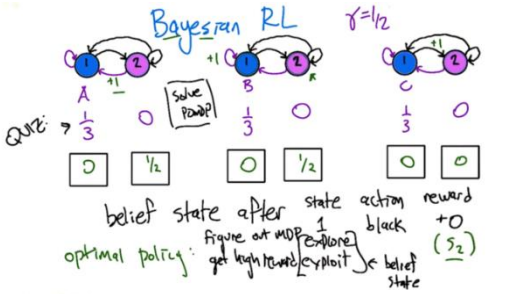
- Solve and compare the value function based on the 50-50 probability (policy) and 1/3 - 2/3 policy.

X = p * (0.5 * X) + (1 -p)(0.5 * Y) => X = 0.5 * (1 - p)Y / (1 - 0.5p)
Y = p * (0.5 * x) + (1 -p) => X = 2 *(Y - 1 + p) / p
Z = p * (1) + (1 -p) (0.5 * Z) =>
V = 1/3 * (X + Y + Z)

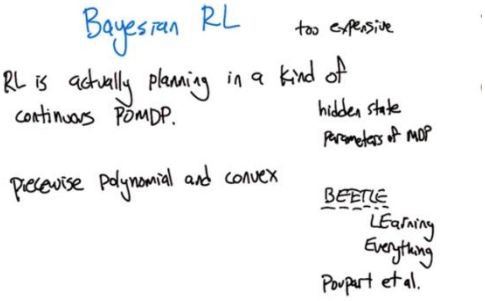
RL as POMDP:



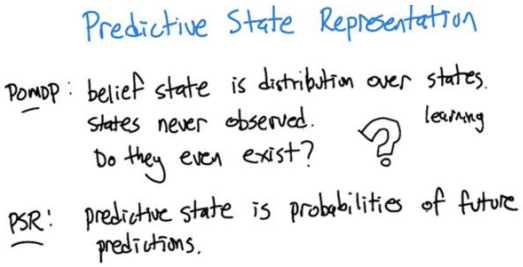
- We can be in MDP A or B but not C, and we will be in state 2.
- Next, we can take a purple action to figure out which MDP we are in.

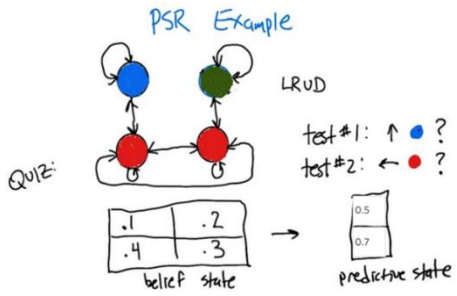


- So the optimal policy is figuring out which MDP we are in and get hight reward by exploiting in the belief state.
- RL itself is like POMDP which requires exploring and exploiting.

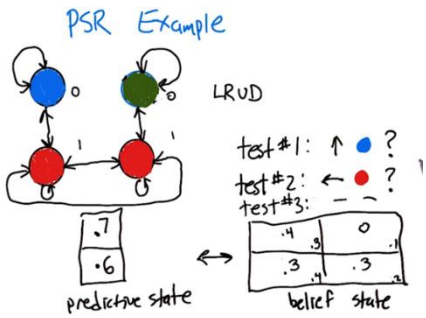


Predictive State Representation

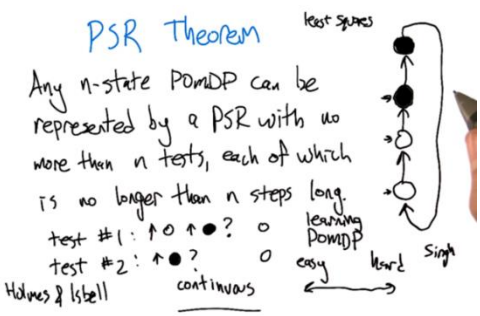




- The POMDP has 4 states but 3 observables represented by colors where the two red states are not distinguishable.
- We can do test to get predictions of states. (e.g, a test can be moving up, and see the probability of ending up at the blue test).

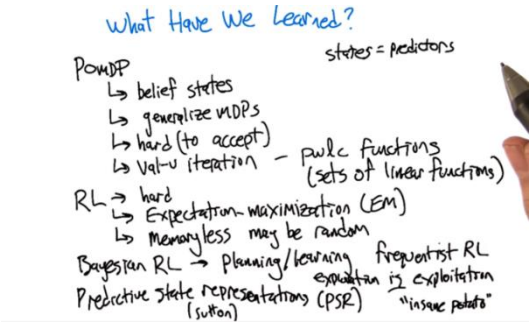


- Belief state and predictive state does not have a one-on-one mapping.
- Increasing number of tests will constrain the mapping.

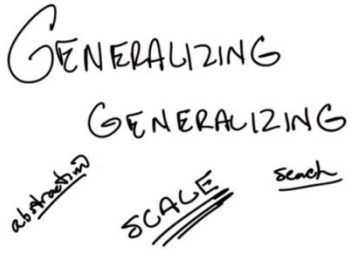


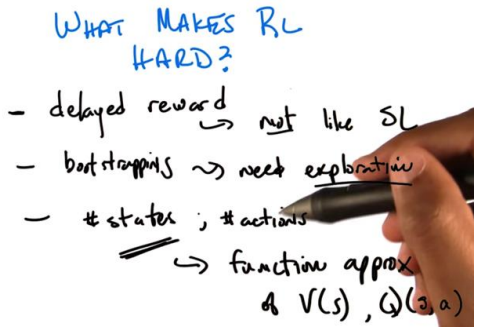
Why go to PSR?

- We cannot observe all the states in POMDP and they might not even exist.
- We can do some test to figure out what state we might be in.
- Learning a PSR sometimes is easier than learning a POMDP.
- Number of test is always going to be no more than the number of states



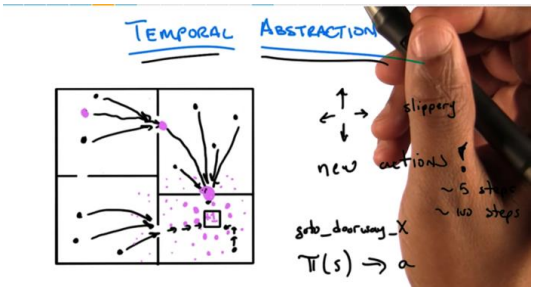
Week 11: Generalizing Generalization





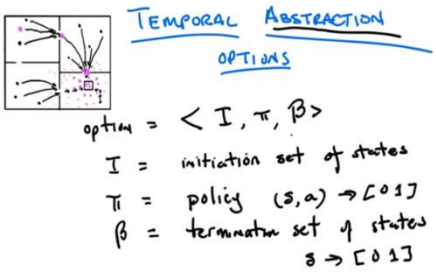
- Delayed reward: agent has weak feedback, the reward is a moving target
- Need exploration to learn the model or action-reward pair for all or a good number of states.
- Computationally, the complexity of RL depends on number of states and # of Actions.
  - Function approximation over value function (V(s), Q(s,a)) is abstraction over states, not actions.
  - The focus of this class is abstraction over actions

Temporal Abstraction



- Original grid world problem: walls separated big square into four rooms, agent could be in any location. Agent's actions are move L,R,U,D and its goal is to reach the small square in the bottom left room.
- A set of new actions (goto_doorway_X, X is orientation) can be generated to represent original actions.
- **Temporal Abstraction** is representing many actions with one or a few actions.(without doing anything to the states)
  - **Temporal Abstraction** align many actions into on and thus makes a lot of states equivalent.
  - **Temporal Abstraction** has computational benefits.

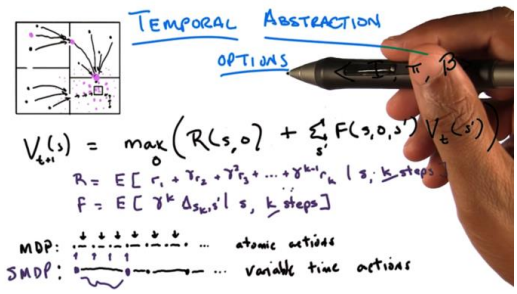
Temporal Abstraction Options



Option is a tuple <I, π, β>

- I is **initiation** set of states
- π here is policy: probability of take action a in states s, (s,a) -> [0 1]
- β is **termination** set of states and it's a set of probability of ending at state s, [0 1].

Temporal Abstraction Option Function

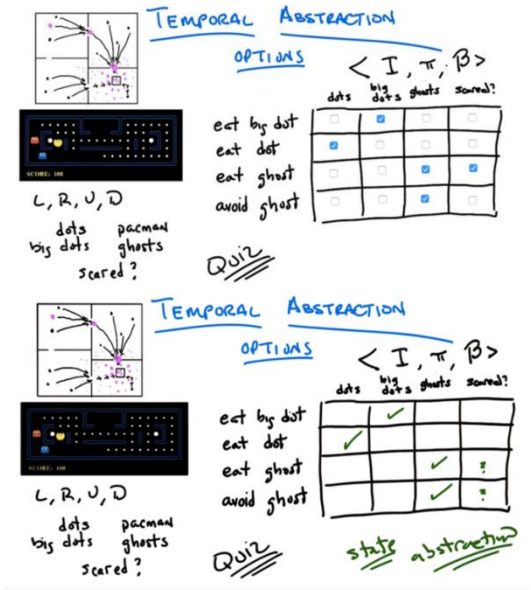


- The function is a rewrite of Bellman function
  - Using "*o*" to replace "*a*" (O is the generalization of A).
  - V(s) (and V(s')) is the value function that needs to update, R(s,o) is reward in s and choose a, F works like a transition function. The discount factor is hidden.
- Using options kind of violates the temporal assumption of MDPs

- MDPs have atomic actions, reward can be easily discounted for each step.
- Using options ends up with variable time actions, discount factor is hidden.

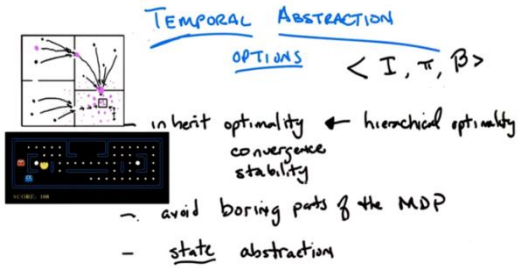
- If o represent k steps, R and F are actually discounted. this is Semi-MDP or SMDP
  - We can turn non-Markovian stuff into Markovian by keep tracking of its history.

Pac-Man Problems



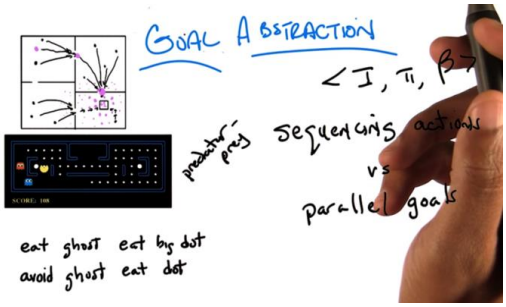
- We can learn two things from the example:
  - If done improperly, temporal Abstraction might not reach optimal policy.
  - Temporal Abstraction might introduce state abstractions (reduce the state space) so the problem can be solved more efficiently.

How It Comes Together

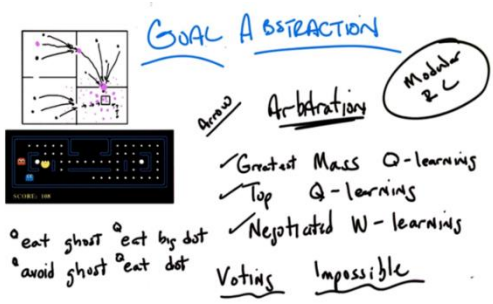


- We can see options and state representations as high level representation. In fact, actions and states are also somewhat made up. Agent's goal is to make decisions with respect to those descriptions of the world, no matter they are action or option, states or abstract states
- If construct options smartly, we might be able to ignore some states (decrease the state space) to reduce computational resource requirement.

Goal Abstraction

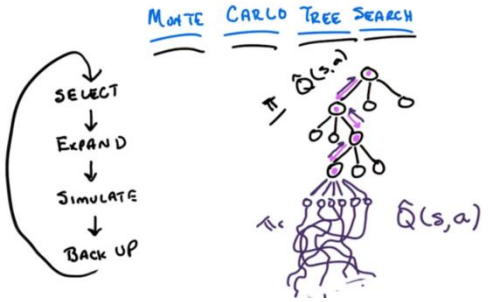


- Agent has multiple parallel goals (predator-prey scenario), at any given time, only one goal is dominating (more important).
- β (the probability of terminated in a state) can be seen the probability of accomplishing a goal or the probability of another goal becomes more important (switching goals).
- Options give ways to think actions as accomplishing parallel goals.
- The goals do not have to be in the same scale.



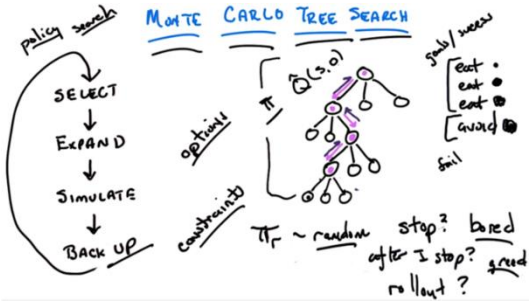
- Modular RL sees options as sub-agents with different goals. There are three ways of choosing goals:
  - Greatest mass Q-learning: each action as a Q function. For each option, we sum up the Q functions of each action in it. The option with largest Q get executed. (Might end up with agent with small Q on every action).
  - Top Q-learnings: choose the action with highest Q. (but the agent might put high Q on many actions)
  - Negotiated W-learning: Minimize loss.
- Modular RL is often impossible because a fair voting system is hard to construct. e.g. the modules might have incompatible goals.

Monte Carlo Tree Search



In the figure above, circles are states, edges are transitions. π =Q^(s,a) is the policy of the known part of the tree. In these states, we know what action to take following π (pink edges). When reach an unknown state, we apply the rollout policy π_r, and simulate actions to take deep in the tree, and then we backup and update π_r and π to figure out what to select at each state, including the unknown state where we started the simulation. π gets expanded as we figure out the policy at unknown state. Then repeat the "Select, Expand, simulate, back up" process.

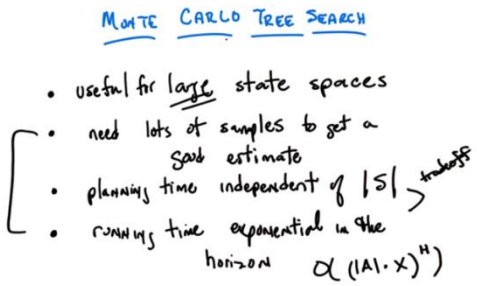
- Initial policy π, we can **SELECT** actions following it. it can be updated at each iteration of tree search.
- Rollout policy π_r, we can **simulate** actions and sample from them starting from the unknown state following it.
- We **back up** after stimulation with the simulated result to update π.
- We figured out what action to take at the unknown state and states above and **expand** π to the previous unknown state.
- Now we can repeat the process to search deeper.



- When to stop? Learn deeper enough before reach the computational resource cap
- Rollup policy π_r can be random. We know the action is good because I get better result by behaviouring randomly from that point on.
- Instead of purely random, on can behave randomly in respect to constraints. (E.g. not eaten by ghost).
  - constraints: defined by failure
  - goals: defined by success
- MCTS is compatible with options to perform the tree search. in this case, π = Q_hat(s,o);
- The Monte Carlo Tree Search can be seen as policy search. When reach a state we are

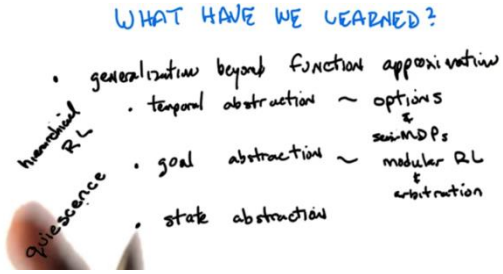
not confident, an inner loop is executed to do some RL.

MCTS Properties

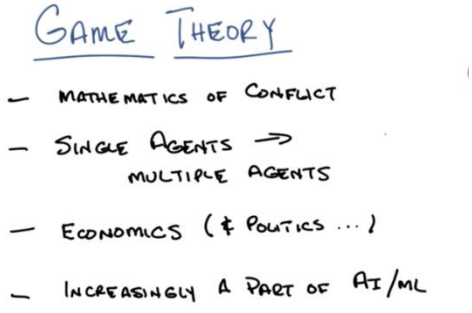


- MCTS is useful for large state spaces and need lots of sample s to get good estimates
- Planning time is independent of |S|
- Running time is O((|A|.X)^{H}).
  - X is how many steps does the search need to go through
  - |A| is the size of the action space
- The tradeoff is between number of states and how far away are we going to search

Recap

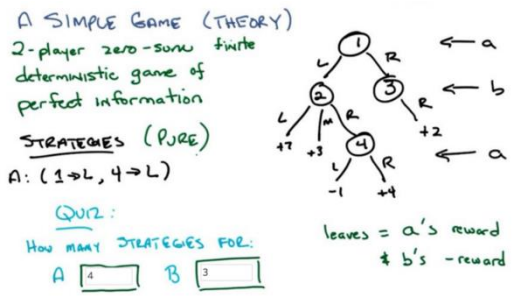


Week 12: Game Theory I

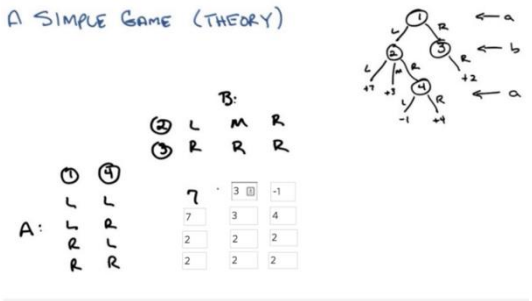


- Game theory is mathematics of conflict of interests.
- It generalizes the RL from single agent to multiple agents.
- It is of the interest of economics, politics, and sociology or even biology since these fields often deal with agents with many agents with conflicts of interests.

Example



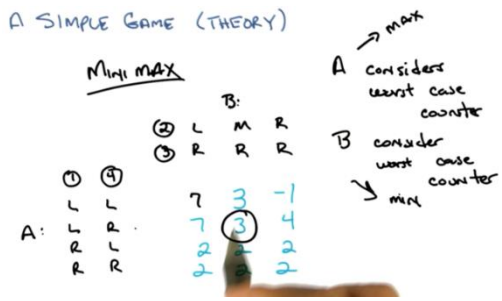
- The example game is represented as a tree, nodes are states, edges are transitions and leafs are rewards.
- The example game has 2 agent (*a* and *b*), all the rewards added up to be a constant (zero-sum), no stochastic transitions. The leafs show the reward that agent *a* can get and *b* gets (-1 * reward).
- Strategies: what action should the agent take at each state it could be in. (equivalent to policy in RL)
- **The agents are assumed to be rational**.



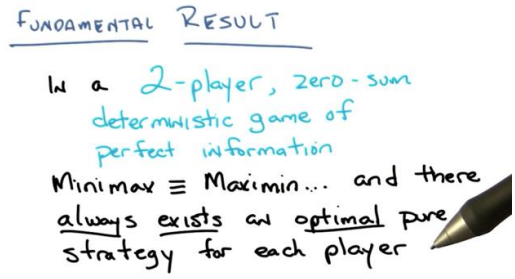
- This is a simple example of 2-agent zero-sum game

- Zero-sum means the reward of A and B will sum to 0 for any strategy.
- A matrix is enough to represent a game.
- Once the matrix is here, the tree doesn't matter now.
- Need to learn to generate the matrix from the tree.

Minimax



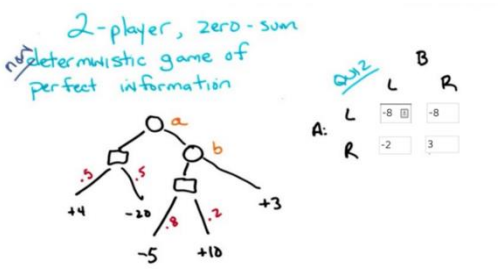
- A and B has the same goal, maximize their own reward (and minimize others reward).
- If A and B behave rationally, they will reach the same strategy.



This is important so I am writing it down:

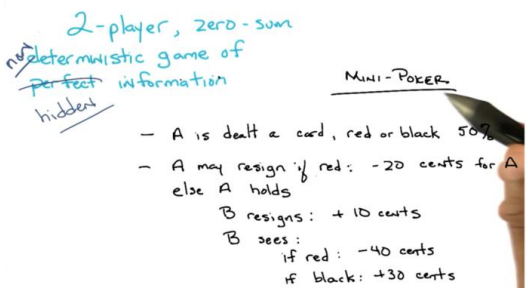
- In a 2-agent, zero-sum, deterministic game of perfect information, Minimax ≡ Maximin,
- And there always exist an **optimal pure strategy** for each agent.
- Based on the strategy matrix, one can always build at least one tree.

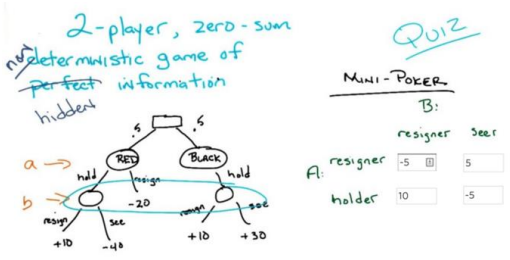
Now, to make the problem a bit more complex, we change the game to be non-deterministic:



- Introduce the **chance box**, so that transition is non-deterministic.
- Construct the strategy matrix based on tree (but could not reconstruct tree based on matrix)
  - Note: from the matrix, we don't know if the tree is deterministic or not.
- The minimax theorem (Von Neumann theorem) still holds
  - Minimax ≡ Maximin
  - Optimal pure strategy exists.

Minipoker

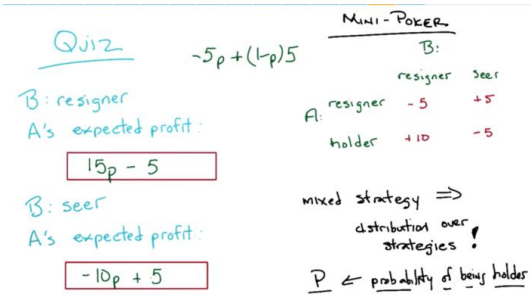




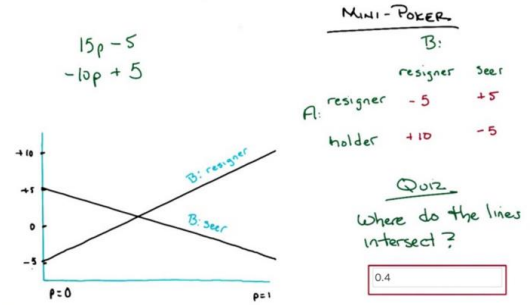
- In the minipoker game, b will not know all the information, so it's a 2-agent, zero-sum, non-deterministic game of **hidden** information

- In this case, Minimax ≠ Maximin and there will be no optimal pure strategy.
- But there will be mixed Strategy, which is a distribution of strategies.

Mixed Strategy

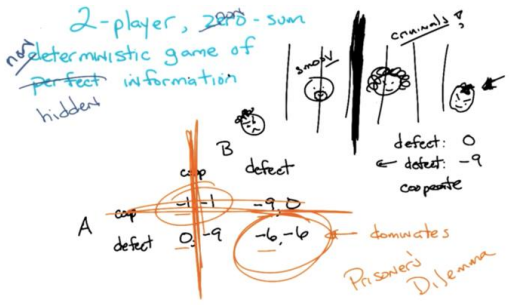


- A's expected strategy are linear equation, which can be represented by lines.



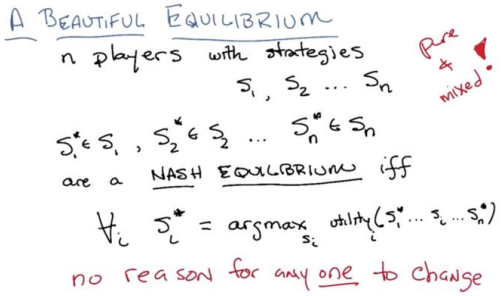
- The mixed strategy should be at the intercept of the two lines in this case.
- If the two lines both have positive slope, the mixed strategy should be at the far right; if negative slope, the strategy should be at the far left.
- The expected value of the game is deterministic still.
- Although the strategy is mixed, there is still an expected value, in this case, the expected value is when p is 0.4, and value is **1**.

Snitch

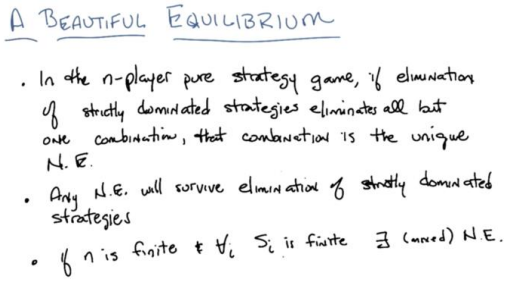


- Now, we are making the game non-zero-sum.
- The prisoner's dilemma is a 2-agent, **non-zero-sum**, non-deterministic game of **hidden** information
- Assume the agents are rational, both of them should defect.

A Beautiful Equilibrium

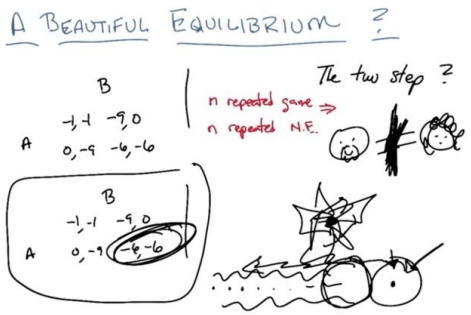


- In practice, if we are in a Nash Equilibrium, any agent will have no good reason to change strategy (pure or mixed).



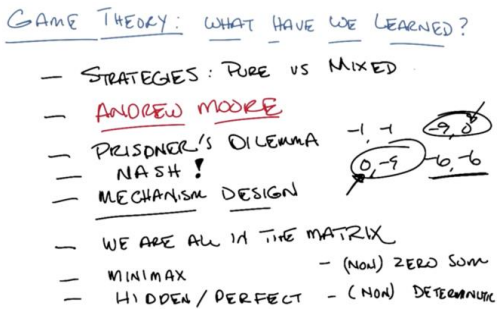
- In the n-player pure strategy game, if elimination of strictly dominated strategies eliminates all but one combination, that combination is the unique NE.

- Any N.E. will survive elimination of strictly dominated strategies
- If n is finite, for each set of finite strategies, then there will be at least one strategy is N.E.



- What if playing the game multiple times?
- Only the last game matters-> the last game will be N.E -> since the last game is known now, the last game moves to the game before it.-> the last game will be N.E. -> repeat.... ->all the game should will be N.E.

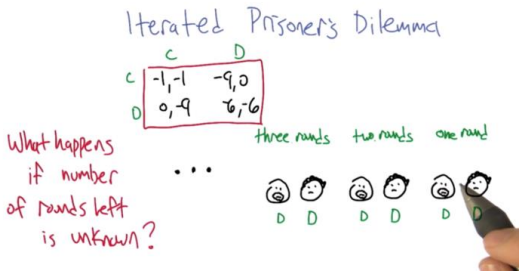
Recap



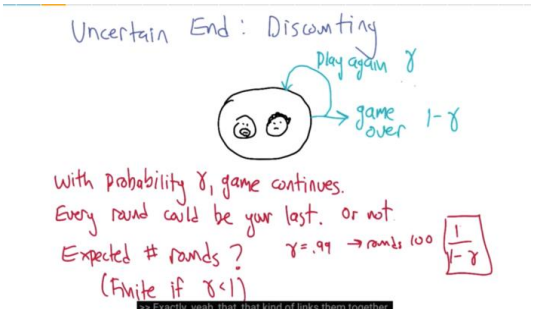
[Andrew Moore's slides on Zero-Sum Games]
[Andrew Moore's slides on Non-Zero-Sum Games]

Week 13: Game Theory II & III

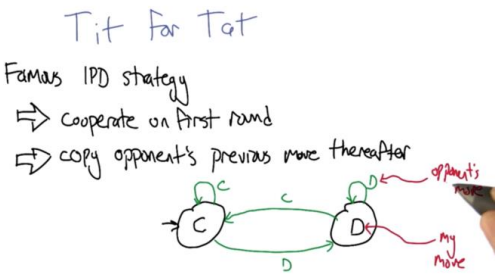
11B Game Theory ReLoaded



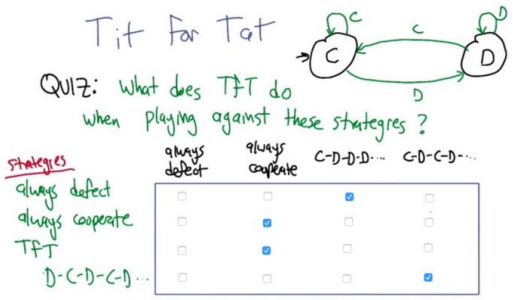
- Prisoner should always defects at the last game (say we have k games), then the kth game is known
  - If you do the above choice, then the last game is known, and the last unknown game becomes to be the one before.
  - Since prisoner should defect in the last unknown game, then it should defect on the (k - 1) th game. And the (k - 1)th game becomes known.
- So prisoners should always defect at every each game
- But what happens when the last a few games are unknown?



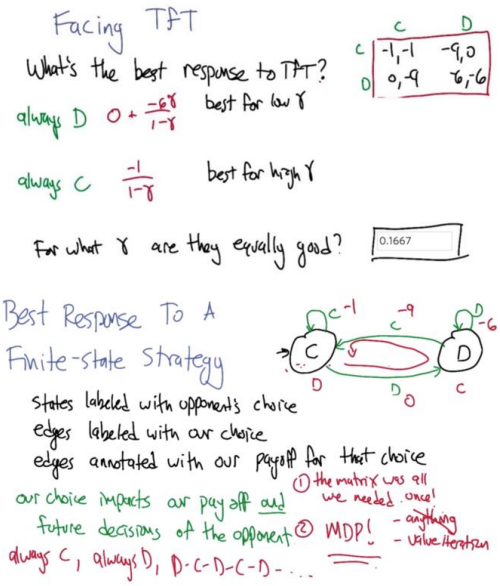
- the probability that the game will continue is γ
- Then the expected number of rounds is 1/(1 - γ).



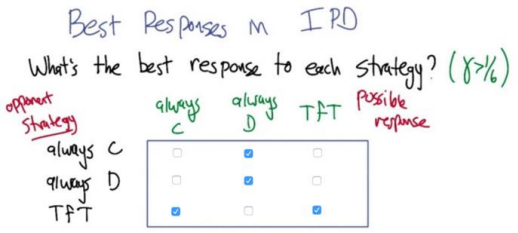
- Finite representation of a unbounded number of rounds
- Copy opponent's previous move thereafter.

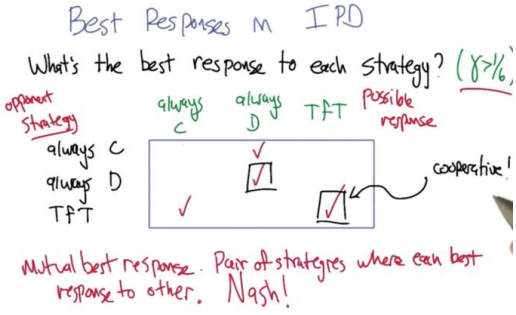


What to do if we Facing TfT?



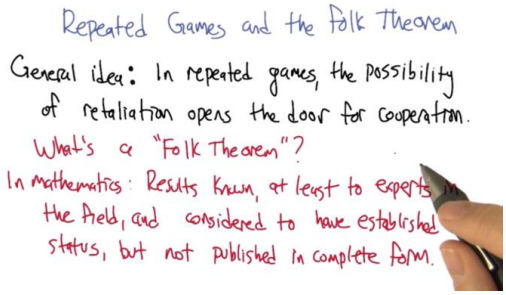
- TfT is like an MDP, we can solve TfT by solving MDP
- If our opponent using TfT, then our decision will affect the opponent's decision.

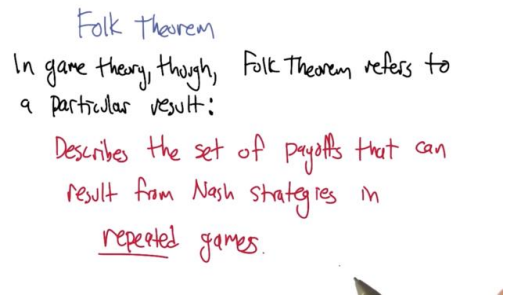




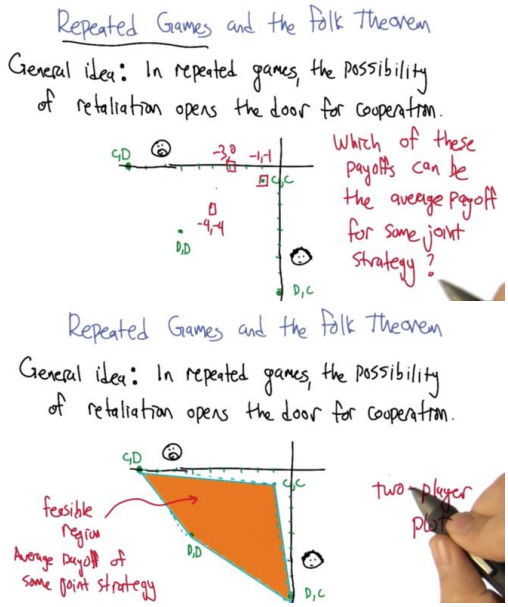
- Playing the game multiple times (unknown times) opens up the possibility of another Nash Equilibrium which is cooperating with each other.
- In this case, both agents playing TfT is a best way to respond to each other.
- But what if γ is not 1/6 here??? Or is γ matter?

Folk Theorem

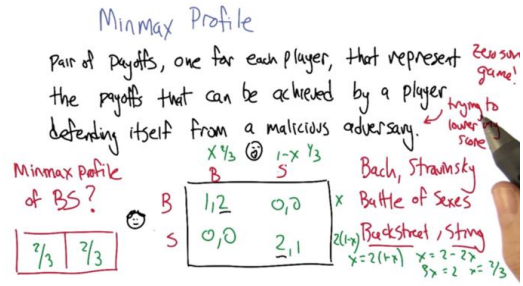




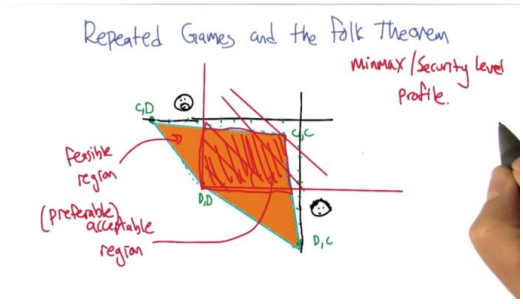
- Describes the set of payoffs that can result from Nash strategies in repeated games.



Minmax Profile and Security level profile

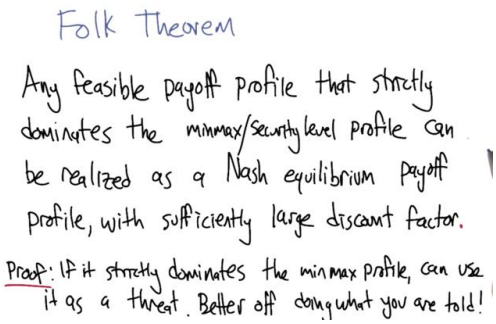


- x is the probability that Smooth chooses "B", Then (1 -x) is the probability that Smooth chooses "S". So Curly will get x if he chooses "B", and 2(1-x) if he chooses "S". Solving x = 2(1-x) will get x.



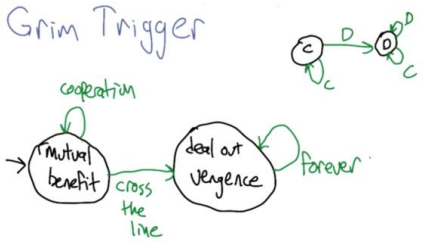
- MinMax Point: DD point
- Feasible payoff: yellow filled region
- Acceptable payoff: any point that to the right-above of MinMax point
- Feasible acceptable payoff: intersection of 1 and 2.

Folk Theorem

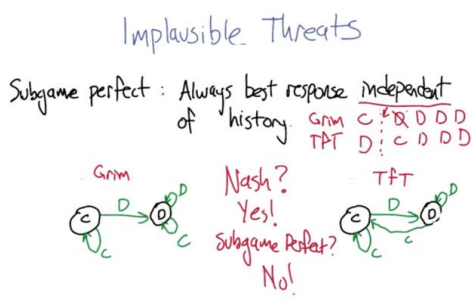


- Any feasible payoff profile that strictly dominates the minmax/security level profile can be realized as a Nash equilibrium payoff profile with sufficiently large discount factor.
- Translation: do what you are told (feasible payoff), or you will be punished (minMax payoff profile). See the next slide for graph representation.

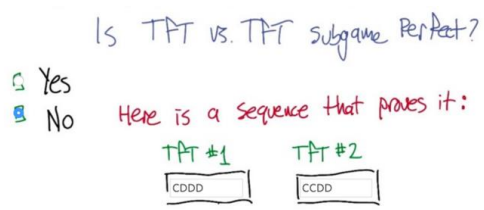
Proof of Folk Theorem



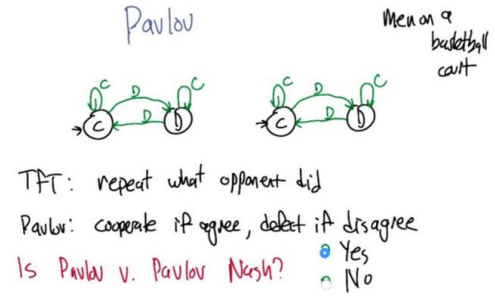
- black color--> actions of one agent
- green color--> another agent
- Grim Trigger: once cross line, will always deal with default.



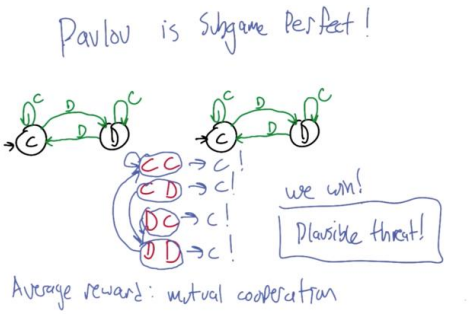
- Subgame perfect equilibrium: means taking test response independent of history.
- If under a threat the machine can reach a Nash equilibrium but cannot reach subgame perfect, then it is an implausible threat (the agent won't carry out the threat)



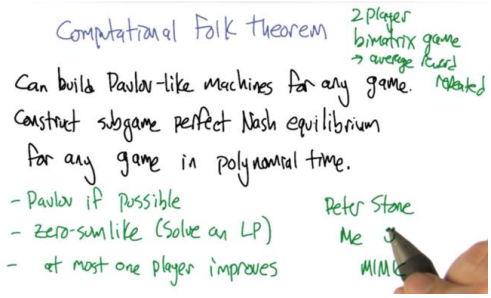
- If #1 is defecting, #2 will get better expected reward if it keeps cooperating instead copying the action of one. SO TfT vs. TfT is NOT subgame perfect.



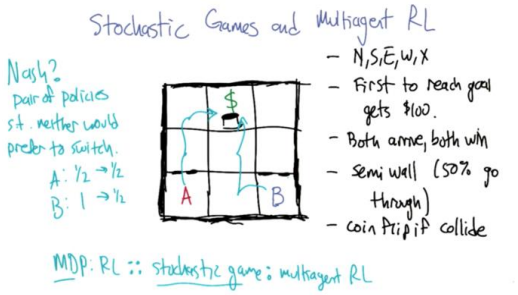
- Pavlov can reach Nash equilibrium if they both choose to cooperate.



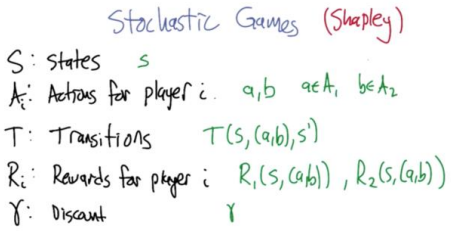
- Pavlov is subgame perfect and reaches NE. So the threat is plausible!



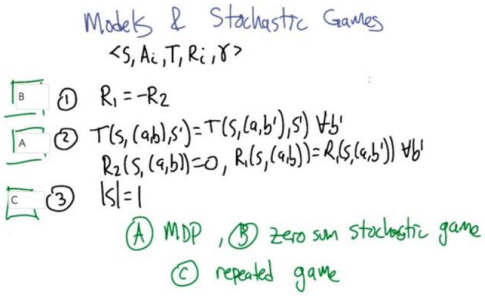
Stochastic Games and Multiagent RL



- Game description: A and B are two agents. If they go north, there is a semi-wall with 50% chance of going through. If they both go to the center, they will flip a coin to decide who will take the block. When one agent reach $, game ends and the agent who reaches $ get 100 reward.
- NE exist? Yes, both go center.

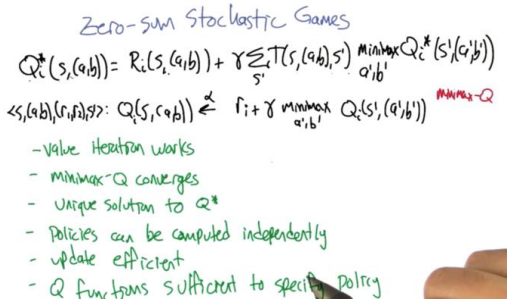


- Stochastic Games is generalization of both MDP and single agent RL.



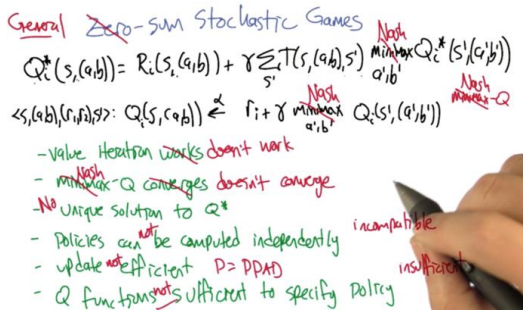
- If put some constraints on the stochastic games, we will get some of the games/mdp etc we learned before.
- Since stochastic games are generalized MDP, so we can try to generalize the method to solved the MDP to solve stochastic games.

Zero-Sum Stochastic Games

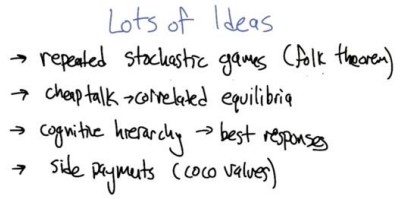


- Minimax can be used in a Bellman-like equation.
- VI works for the situation, minimax-Q converges, and Q* is unique, policies can be computed.

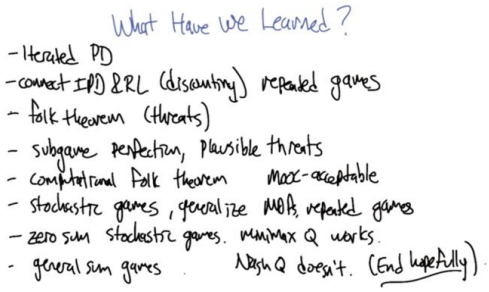
General-Sum Games



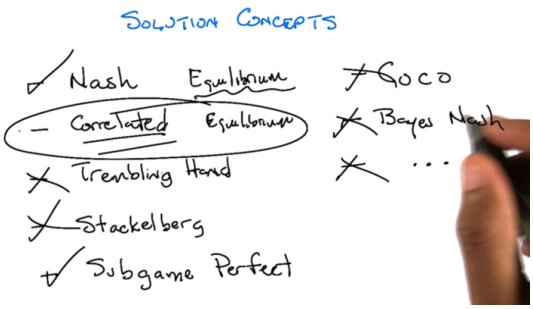
- When we general-sum case, Q type of algorithms will not work.
- The problem is not solved but there are a lot of ideas about =



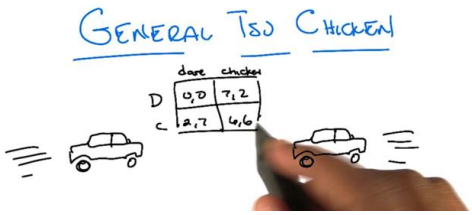
Recap



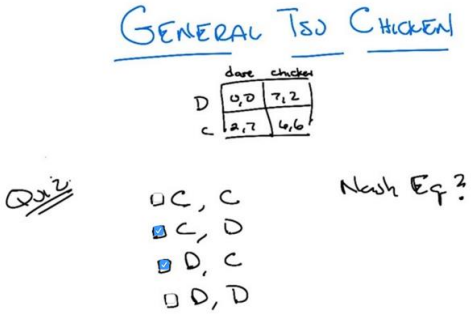
11C Game Theory Revolutions



Correlated equilibrium

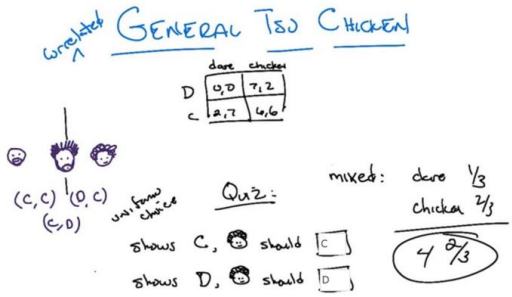


- In the chicken game, if both *Dare*, both get 0 reward (dies), if both chicken, both get 6 rewards. If one dare and the other chicken out, one get 7 and the other get 2 (still alive).

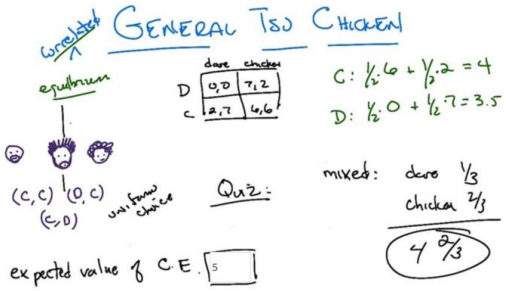


- CC is not NE because if I know you will be C, I will change to D to get more reward.
- CD or DC is NE because C is not willing to change to D, and D is not willing to change to C to get lower rewards.
- DD is not NE because D wants to change to C to get better reward.
- Although CD or DC is NE, but they are not stable because no one wants to always be C to receive a relatively lower reward.

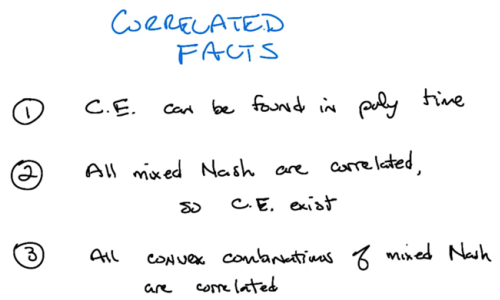
- Mixed Strategy: Dare 1/3 of the time and Chicken out 2/3 of the time. the expected payoff is 14/3.



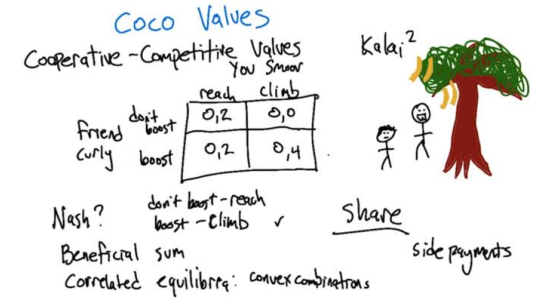
- Both agents knows three cards [C,C],[CD], and [DC]
- A third person Chris choose a card randomly and inform the agent what to do base on the card. Should the agents listen to Chris?
  - If Chris told me to D, then other one must be C. I don't want to change because changing to C will get lower reward.
  - If I was told to C, then the other one has half the chance to be C and half the chance to be D. If I choose C, I will get expected reward as 4. If choose D, will get expected reward of 3.5 (see green words in the next slide). So staying with C is better.
- Yes. And this is called a **correlated equilibrium**(CE)



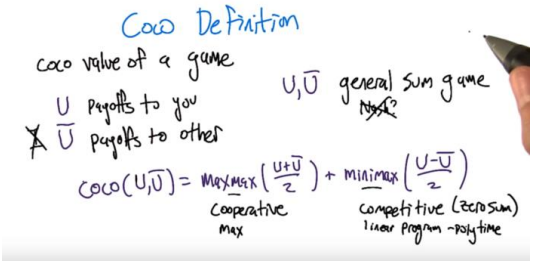
- Given [C,C],[CD], and [DC] are randomly chosen, the expected CE is `1/3 * 7 + 1/3 * 2 + 1/3 *6 =5`, which is better than 14/3.
- Not both chicken out to get reward of 6? **Because [C,C] is not an equilibrium!**
- What are the real life analogy of "Chris"? Traffic lights, stop signs……



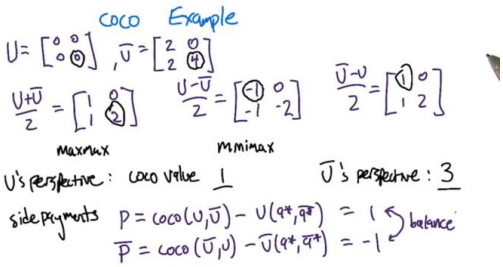
Cooperative-Competitive Values



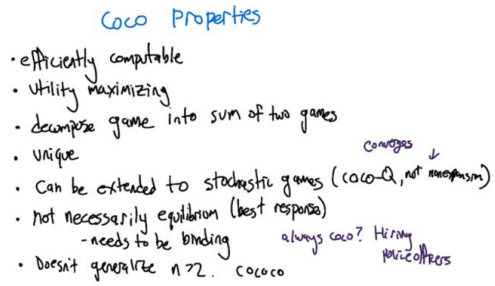
- There are two N.E in the CoCo game.
- Curly are not motivated to do anything because he always gets 0 in the game. How to motivate him? Side payments? Smooth shares sth with him?



- To understand this, need to go back and revisit Maxmax, minimax and linear programming.
- The basic idea is to rewrite CoCo into a cooperative part and a competitive part.

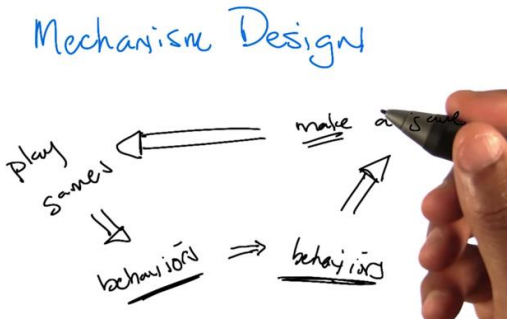


- Given a U and U_bar matrix, compute the coco value and side payments.
- Calculate maximax and minimax. There are minimax form U's and U_bar's perspective. But the two values cannot be values that's outside NE regions.
- Coco value from U and U_bar's perspective are different.
- Side payments can be calculated based on U and U_bar's max value and coco value.

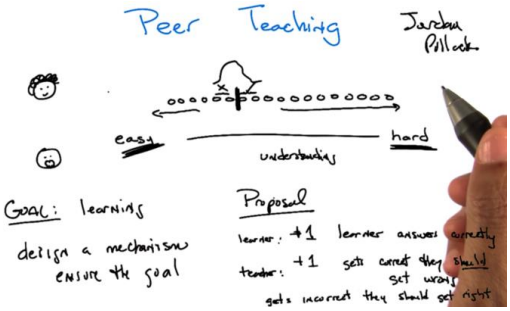


- CoCo-Q converges but it is not non-expansion.

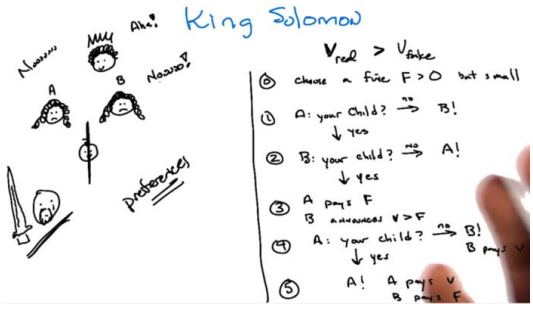
Mechanism Design



- Based on wanted behavior to build/make a game.
- E.G. Tax policy to encourage people buying a house



- Goal: learning.
- Learner get incentives by getting questions correct -> promote learning
- Teacher get incentives by setting the difficulty bar so that if the learning get an easy question wrong and get a hard question correct. -> push student to learn

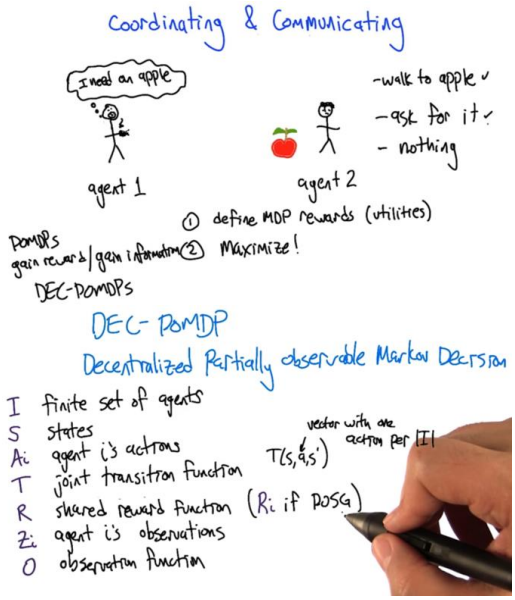


- In the old King Solomon story, A and B will both say "no" and King will not get no real information.
- Given the mechanism on the left, fake mom will be honest.
  - Assumption: the baby's value to real mother and fake mother will be $V_{real} > V_{fake}$
  - Ask both the mother, is this your child, if all say yes, then A pays a fine F.
  - Then B announce a V that is V > F
  - Ask the question the same question to A again, if A say yes, A pays V, B pays F and A keeps the baby. If A says no, B keep the baby and B pays V.

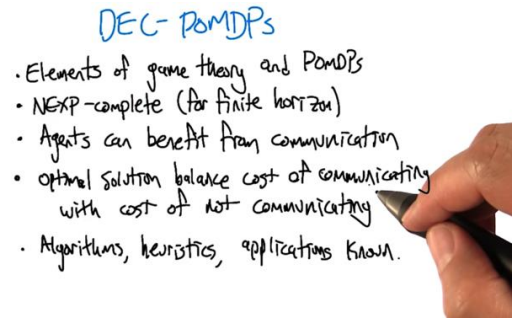
Explanation: if B is fake, she will not announce a value that's larger than V_{fake}, because if A says no, he will get the baby with more than he is willing to pay. So A will get the baby. If B is real mother, she will announce a value V_{real} which is larger than V_{fake}. And A won't say yes because again she will end up with the bay but paying more than she is willing to. So, the real mom will get the baby.

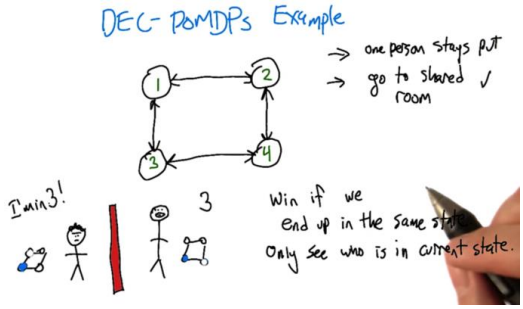
What have we learned?

Week 14: Coordinating Communicating and Coaching

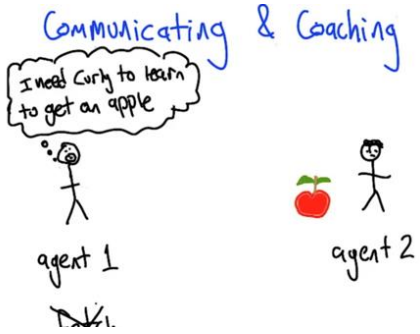


- Dec-POMDP has some perspectives of game theory and MDP
- Multiple agent working on getting a common reward. (if the rewards are separated for all the agents, then it's a POSG *partially observable stochastic game*)



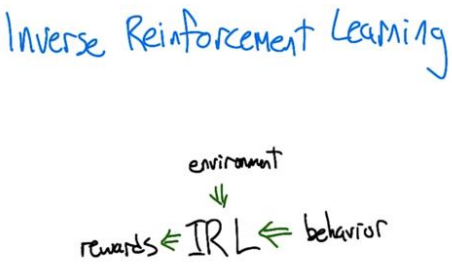


- Two agents, they know where they are but don't know the other's position. When the two are in the same room, they win.
- Strategy: go to a shared room. But my knowledge of my current position could be wrong (partially observable world).

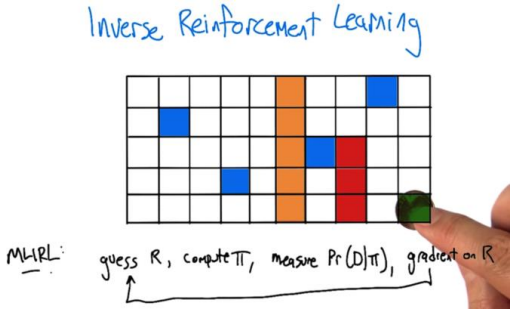


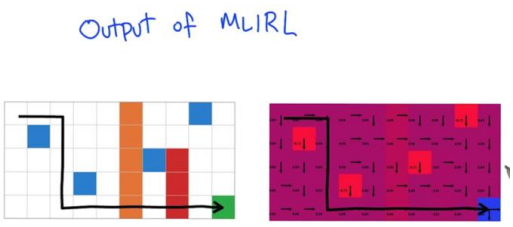
- Agent 1 wants to set up some kind of reward function to move agent to do something (e.g. get the apple for me).

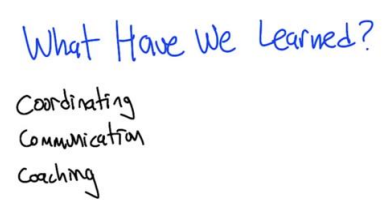
Inverse Reinforcement Learning



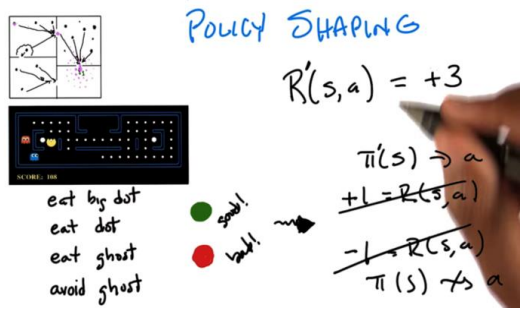
- Inverse Reinforcement Learning: the agent experience the environment and a set a behavior and then generate a reward function based on the inputs.



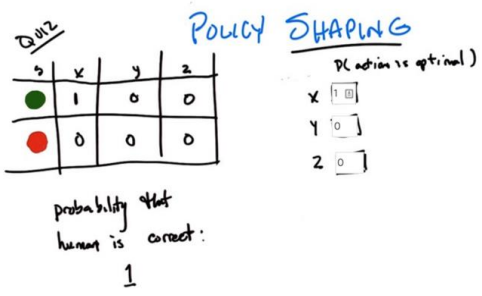




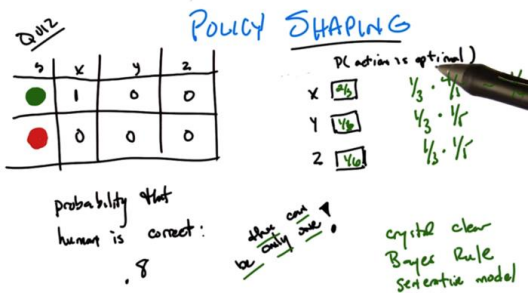
Policy Shaping



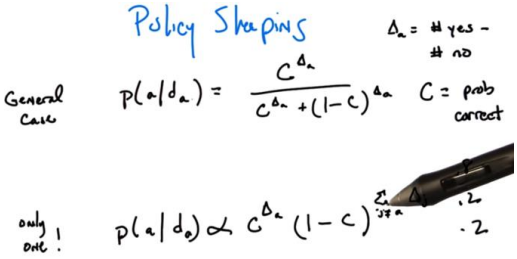
- If a human is giving feedback (commentary) about whether the agent's action is good or bad, s/he is doing policy shaping.
- Policy shaping could be realized by reward shaping which is replace reward of an action with a new reward?
- Agent need a mechanism to learn from the environment and the commentary to decide what policy to take (not just listening to the commentary, because the commentary might not be always right).



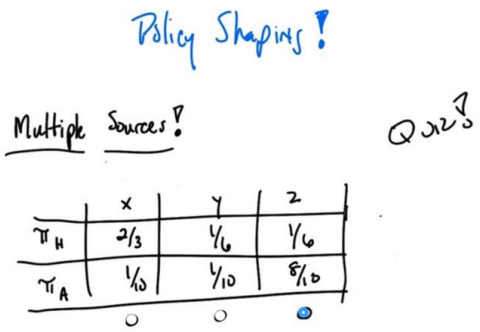
- If human is always correct, given the feedback, what's the probability that the action (x, y, or z) is optimal?
- Answers in the slides above.



- What if human is 0.8 probability of right?
- Counting method:
  - Saying x is optimal is liking saying y and z is not optimal.
  - Since human is 0.8 correct, then x, y, z being optimal is 0.8, 0.2, 0.2.
  - Normalize the numbers above, will get 2/3, 1/6, 1/6.

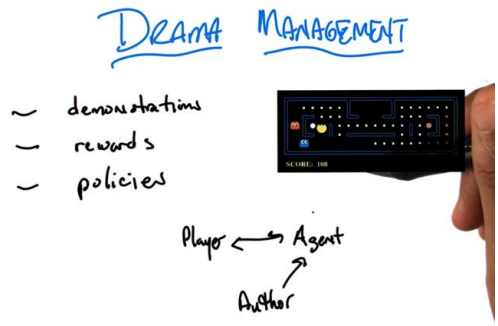


- Δ_a is coming from data of action a (d_a). C is the probability of correct of the people giving commentary.
- The formula above give the method of calculating probability of action a is optimal.
- Note: the final probability will need to be normalized against the probabilities of other actions.

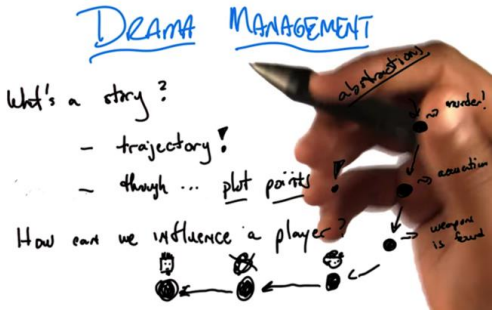


- In the policy shaping case, information are coming from multiple sources.
- E.g. π_a and π_H are policy info from agent exploring the world and human giving feedback.
- Some algorithm decrease the importance of π_H as time goes. One need to know that π_a already incorporated the information of human uncertainty (C).
- The way to combine the two sources is to calculate the probability that the two policy will agree: $a_{opt}=\text{argmax}_a\ p(a|\pi_1) * p(a|\pi_2)$.
  - In the quiz x_{opt} = 1/15, y_{opt}=1/60,a_{opt}=2/15. So we should choose z as optimal.

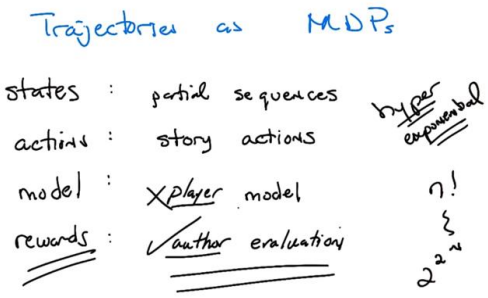
Drama Management



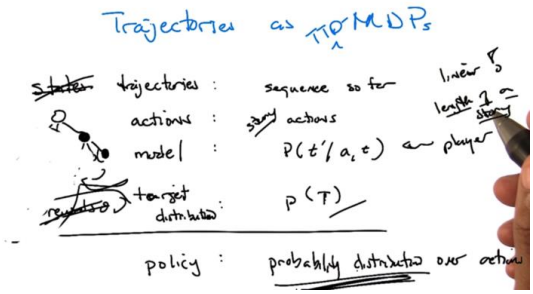
- The way a human can communicate to an agent
  - demonstration: show the agent what's the correct action (inverse RL)
  - reward shaping: giving reward for agent's actions
  - policy shaping: commentary on the agent's actions
- Author convey his intent to the agent so the agent can



- story can be defined as a trajectory through plot points



- Above a some mapping of MDP elements to trajectory MDP elements
- Problems
  - large number of sequence of states (hyper exponential)
  - Since MDP will maximize rewards, treating story as an MDP will only make the author happy and force the player to experience the story.



- p(t'|a,t) is the probability that the player at trajectory t and take action a then ended up in trajectory t'. P(T) is a target distribution.
- The action is not player's action but the story action
- The optimal policy is the policy that will lead to the targeted trajectory distribution P(T)
- The calculation time is linear and dependent on the length of the story.

What have we learned?