CS 7642 Reinforcement Learning and Decision Making Lecture Notes

Jie Wu, Jack

Summer 2022

1 Smoov & Curly's Bogus Journey

1.1 Decision Making & Reinforcement Learning

The differences between the three types of learning are

supervised learning: given x and y, find f such that $y = f(x)$, basically function approximation

unsupervised learning: given x, find f to yield a compact description of x, basically clustering description

reinforcement learning: given x and z, find f and y such that $y = f(x)$

1.2 The World - 2 Quiz

Suppose there is uncertainty associated with the planned direction of motion, with 80% probability of moving in the intended direction and 10% probability of straying to the left/right each. When calculating the probability of reaching the goal, don't forget to include the probability of wrong movements (wrong + wrong = right) which actually ends up in the destination.

1.3 Markov Decision Processes

The basic ingredients are

state: denoted by s

model: also called transition function $T(s, a, s')$ which is basically the probability $\Pr(s'|s, a)$ of starting from state s and ending up in state s' upon taking action a, this expression reflects two properties

Markovian: it means that only current state matters, which is reflected in the expression by including only s, it is not as restrictive as it appears, because you can turn almost anything into a Markovian process by making your current state contain all relevant past information

stationary: the transition function doesn't change over time

action: denoted by a

reward: denoted by r, usually a scalar function of current state s but can also depend on action a and next state s'

These four ingredients along with the Markovian and stationary properties define a Markov decision process (MDP). The solution to a MDP is called a **policy** defined as a map that takes in a state and returns an action $\pi(s) = a$. Among them there is an optimal policy π^* that maximizes the long-term expected reward. Thus in the $y = f(x)$ format π^* corresponds to f, r corresponds to z, y corresponds to a, and x corresponds to s.

1.4 More About Rewards

- It is possible that all you know about is taking a sequence of actions and getting rewards at the very last step, which brings in the problem of what was the action that ultimately led to the final outcome. This reward is called delayed reward and this problem is called credit assignment problem.
- Minor changes to reward function matter, because if the rewards along the way are small negative, then we do not need to be in a hurry to reach the destination. Instead we can detour to a longer but safer path to the destination. In contrast if the rewards along the way are large negative, then we may need to go along the shortest but the riskiest path to the destination.

1.5 Sequences of Rewards

Besides the reward at each step

time horizon: the time horizon also affects the optimal policy, for example if the time is limited it may make sense to take the shortest yet the riskiest path to the destination, therefore policy should really be a function of time remaining as well $\pi(s, t) = a$ which makes it non-stationary, nonetheless this course focuses on infinite time horizon

utility of sequences: there could be reward for going through a sequence of states called **utility**, it has the following property called **stationarity of preference**

$$U(s_0, s_1, s_2, \dots) > U(s_0, s_1', s_2', \dots) \implies U(s_1, s_2, \dots) > U(s_1', s_2', \dots)$$

however there is an issue associated with this definition of utility, because it does not converge if the rewards are always positive, to make it converge we define a discounted reward as follows

$$U(s_0, s_1, s_2, \dots) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \qquad 0 < \gamma < 1$$

1.6 Policies

Define the **optimal policy** as

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi\right]$$

Define the **utility** of a particular state as

$$U^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| \pi, s_0 = s\right]$$

where superscript π indicates that it is policy dependent. Note that it is different from the reward of that state, because reward is immediate whereas utility is long-term. Nevertheless under the optimal

policy they are related by the following recursive relation (from now on if the superscript is missing it refers to the utility under the optimal policy)

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

This is the **Bellman equations** the solution of which defines the optimal policy π^*.

1.7 Finding Policies

If there are n states then the Bellman equations consist of n equations with n unknowns ($U(s)$ for each s). However they are nonlinear due to the max operation. Fortunately they are iteration convergent—start with arbitrary utilities, update the utilities based on neighbors, and then repeat until they converge. This is called **value iteration**. The update is done by

$$\widehat{U}_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')\widehat{U}_t(s')$$

The optimal policy can then be defined in terms of the converged utilities.
Actually we do not care about getting the correct utilities but about getting the correct policy, and there are infinite number of utilities that are consistent with the optimal policy. Hence we can iterate directly on policies rather than values. This can be done as follows:

1. start with an initial policy π_0
2. given the policy π_t calculate the utilities $U_t = U^{\pi_t}$, which can be done by solving the following set of n linear (vs. Bellman) equations

$$U_t(s) = R(s) + \gamma \sum_{s'} T(s, \pi_t(s'), s')U(s')$$

3. improve the policies by finding the actions that maximize the expected utilities

$$\pi_{t+1} = \arg\max_t \sum_{s,s'} T(s, a, s')U_t(s')$$

Iterate step 2 and 3 until the policy doesn't change. This is called **policy iteration**, and the advantage is that we end up with solving a set of linear instead of nonlinear Bellman equations.

1.8 The Bellman Equations

Recall the Bellmans equations read

$$V(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} T(s, a, s')V(s') \right]$$

where we let the reward be dependent on the action as well as the state thus move it inside the max, and we change the notation of utility from U to V to emphasize it is a value function to be distinguished from the quality Q below. Expand the equations as follows

$$V(s_1) = \max_{a_1} \left[R(s_1, a_1) + \gamma \sum_{s_2} T(s, a, s_2) \times \max_{a_2} \left(R(s_2, a_2) + \gamma \sum_{s_3} T(s, a, s_3) \times \cdots \right) \right]$$

where each expression after "×" is the spell-out of the utility of the previous state. Now we group the expansion differently by grouping the expressions inside the "max" as the **quality** Q of the state-action pair instead of the expressions after "×":

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

Thus we have the following two forms of the Bellman equations

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

The motivation of re-writing the Bellman equations in terms of quality Q is that, the quality form is much more useful in the context of reinforcement learning, because we can compute the expectation of the quality based on experience data without knowing the reward function or the transition function. Of course we can also group the expression after "+" which is termed **continuation** C to end up with the third form of the Bellman equations

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

It is useful when the reward function is complicated, in which case it can replace the reward function using the following relations between C and Q.

1.9 Bellman Equations Relations Quiz

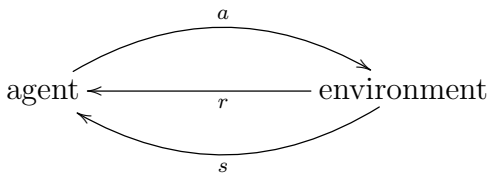
The three forms of Bellman equations are related to each other by

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

2 Reinforcement Learning Basics

2.1 Reinforcement Learning Basics

Reinforcement learning consists of two main ingredients—agent and environment—which interact as follows



The computation occurs on the agent side, and the information about the environment is available only through the interaction in terms of the state s. The agent plays the role of the policy and the environment the role of MDP. However the agent is external to the environment, and the agent's perception of the environment may not coincide with the actual environment. In many cases the agent computes the action a based on its model of the environment.

2.2 Behavior Structures

plan: a fixed sequence of actions, which is not applicable when (1) we are still in the process of learning (2) in stochastic environment

conditional plan: plan that includes "if" statement vs. dynamic replanning = continue with the plan which contains no "if" statement until it breaks down, upon which the plan is revised

stationary policy/universal plan: a map from states to actions e.g. MDP, our focus

2.3 Evaluating a Policy Quiz

The steps are

1. follow the policy from some initial state to generate sequences of (state, action, reward) with definite probabilities
2. for every sequence turn each state transition into immediate reward
3. truncate sequences to finite horizon if they are infinitely long or too long
4. for every sequence compute the return which is in general the sum of discounted rewards
5. sum the returns of all sequences weighted by their probabilities

2.4 Evaluating a Learner

The criteria consist of

- value of the returned policy
- computational complexity (execution time)
- experience/sample complexity (training data)

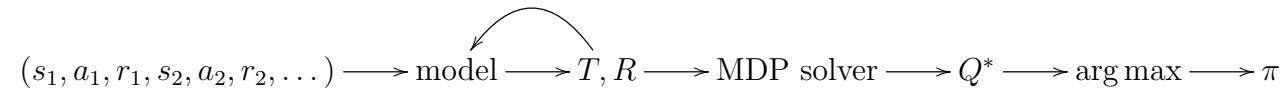
Note that for reinforcement learning we usually hit computational limit before hitting memory limit.

3 TD and Friends

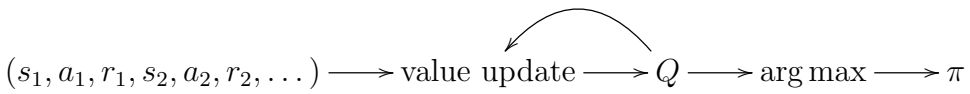
3.1 RL Context

There are three main families of RL algorithms

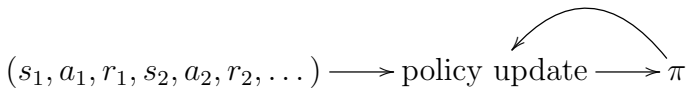
model-based: more supervised learning, creating all the intermediates thus easier to update



model-free or value-based: strikes a balance between simple computation and ease of update



policy-based: more direct learning, more difficult to update because the feedback isn't very useful for updating the policy



3.2 TD(λ)

For example let's try to learn (the value-based approach)

$$V(s) = \begin{cases} 0 & \text{if } s = s_{\text{final}} \\ \mathbb{E}\left[r + \gamma V(s')\right] & \text{otherwise} \end{cases}$$

3.3 Estimating from Data Quiz

Given several episodes starting from s, $V(s)$ can be estimated by taking the average of $r + \gamma V(s')$ for each episode. Moreover since $V(s_{\text{final}}) = 0$ we just need to take the average of the sum of the discounted rewards along the path from s to s_{final} in each episode.

3.4 Computing Estimates Incrementally

Suppose the value estimate up to $T-1$ episodes is $V_{T-1}(s)$, the return or the total discounted reward of the Tth episode is $R_T(s)$, the value estimate up to T episodes $V_T(s)$ is then given by

$$V_T(s) = \frac{(T-1)V_{T-1}(s) + R_T(s)}{T} = V_{T-1}(s) + \alpha_T \left[R_T(s) - V_{T-1}(s)\right]$$

where the learning rate $\alpha_T = 1/T$ diminishes as T increases.

3.5 Properties of Learning Rates

For the learning update rule

$$V_T(s) = V_{T-1}(s) + \alpha_T \left[R_T(s) - V_{T-1}(s)\right]$$

to converge to the true expectation $\lim_{T\to\infty} V_T(s) = V(s)$, the learning rate must satisfy

$\sum_T \alpha_T = \infty$ i.e. the learning rate should be big enough to move towards the true value

$\sum_T \alpha_T^2 < \infty$ i.e. the learning rate should be small enough to damp the noise

3.6 Selecting Learning Rates Quiz

The general rule of finiteness of power series is that $\sum_T 1/T^x < \infty$ whenever $x > 1$ and ∞ otherwise including $x = 1$. Thus we only need to check x where $\alpha_T \sim 1/T^x$.

3.7 TD(1) Rule

The pseudocode of TD(1) rule reads (where e stands for eligibility)

episode T

for all s do initialize

$e(s) = 0$

$V_T(s) = V_{T-1}(s)$

after each step $t : s_{t-1} \xrightarrow{r_t} s_t$ turn on eligibility

$e(s_{t-1}) = e(s_{t-1}) + 1$

for all s do update

$V_T(s) = V_T(s) + \alpha_T \left[r_t + \gamma V_{T-1}(s_t) - V_{T-1}(s_{t-1})\right] e(s)$

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

3.8 TD(1) Example

Apply the above TD(1) rule to the episode $s_1 \xrightarrow{r_1} s_2 \xrightarrow{r_2} s_3 \xrightarrow{r_3} s_f$, we have
after $s_1 \xrightarrow{r_1} s_2$

$$\begin{aligned}\Delta V_T(s_1) &= \alpha\left[r_1 + \gamma V_{T-1}(s_2) - V_{T-1}(s_1)\right]\\ \Delta V_T(s_2) &= 0\\ \Delta V_T(s_3) &= 0\end{aligned}$$

after $s_2 \xrightarrow{r_2} s_3$

$$\begin{aligned}\Delta V_T(s_1) &= \alpha\left[r_1 + \gamma V_{T-1}(s_2) - V_{T-1}(s_1)\right] + \gamma\alpha\left[r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2)\right]\\ &= \alpha\left[r_1 + \gamma r_2 + \gamma^2 V_{T-1}(s_3) - V_{T-1}(s_1)\right]\\ \Delta V_T(s_2) &= \alpha\left[r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2)\right]\\ \Delta V_T(s_3) &= 0\end{aligned}$$

after $s_3 \xrightarrow{r_3} s_f$

$$\begin{aligned}\Delta V_T(s_1) &= \alpha\left[r_1 + \gamma r_2 + \gamma^2 V_{T-1}(s_3) - V_{T-1}(s_1)\right] + \gamma^2\alpha\left[r_3 + \gamma V_{T-1}(s_f) - V_{T-1}(s_3)\right]\\ &= \alpha\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V_{T-1}(s_f) - V_{T-1}(s_3)\right] = \alpha\left[r_1 + \gamma r_2 + \gamma^2 r_3 - V_{T-1}(s_3)\right]\\ \Delta V_T(s_2) &= \alpha\left[r_2 + \gamma V_{T-1}(s_3) - V_{T-1}(s_2)\right] + \gamma\alpha\left[r_3 + \gamma V_{T-1}(s_f) - V_{T-1}(s_3)\right]\\ &= \alpha\left[r_2 + \gamma r_3 + \gamma^2 V_{T-1}(s_f) - V_{T-1}(s_2)\right] = \alpha\left[r_2 + \gamma r_3 - V_{T-1}(s_2)\right]\\ \Delta V_T(s_3) &= \alpha\left[r_3 + \gamma V_{T-1}(s_f) - V_{T-1}(s_3)\right] = \alpha\left[r_3 - V_{T-1}(s_3)\right]\end{aligned}$$

Hence TD(1) is the same as outcome-based updates, if there is no repeated state.
But if we have a repeated state, then we will ignore anything we learned along the way.

3.9 Why TD(1) is Wrong

Since TD(1) only uses the few episodes that start from the desired state, which do not reflect the true probability of stochastic transitions along its path to the final/terminal state, it would surely go wrong. In contrast maximum likelihood estimate uses all the episodes available to estimate the probability of stochastic transitions along the path. Therefore maximum likelihood estimate is usually more accurate than TD(1) with limited data.

3.10 TD(0) Rule

The pseudocode of TD(0) rule is

episode T
  for all s do initialize
    $V_T(s) = V_{T-1}(s)$
  for all $s = s_{t-1}$ do update
    $V_T(s) = V_T(s) + \alpha_T\left[r_t + \gamma V_{T-1}(s_t) - V_{T-1}(s_{t-1})\right]$

In particular recall the TD(1) update rule is

$$V_T(s) = V_T(s) + \alpha_T\left[r_t + \gamma V_{T-1}(s_t) - V_{T-1}(s_{t-1})\right] e(s)$$

whereas the TD(0) update rule is

$$V_T(s_{t-1}) = V_T(s_{t-1}) + \alpha_T\left[r_t + \gamma V_{T-1}(s_t) - V_{T-1}(s_{t-1})\right]$$

taking the expectation of the right-hand side of the TD(0) update rule yields

$$V_T(s_{t-1}) = \mathbb{E}_{s_t}\left[r_t + \gamma V_T(s_t)\right]$$

in contrast to TD(1), the expectation of which does not change

$$V_T(s_{t-1}) = \mathbb{E}_{s_t}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots\right]$$

It matches the corresponding maximum likelihood estimate if the finite data is repeated infinitely often (⇒ finite data not infinite data, for which TD(1) would yield the same result).

3.11 TD(λ) Rule

The pseudocode of TD(λ) rule is identical to that of TD(1) except the λ factor for eligibility update

episode T
  for all s do initialize
    $e(s) = 0$
    $V_T(s) = V_{T-1}(s)$
  after each step $t : s_{t-1} \xrightarrow{r_t} s_t$ turn on eligibility
    $e(s_{t-1}) = e(s_{t-1}) + 1$
  for all s do update
    $V_T(s) = V_T(s) + \alpha_T\left[r_t + \gamma V_{T-1}(s_t) - V_{T-1}(s_{t-1})\right] e(s)$
    $e(s) = \lambda\gamma e(s)$

When $\lambda = 0$ the eligibility is reset to 0 immediately after the value update for s_{t-1}, which matches the TD(0) rule without eligibility.

3.12 K-step Estimators and TD(λ)

The various estimators are

$$\begin{aligned}
E_1 &: \quad V(s_t) = V(s_t) + \alpha_T\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \\
E_2 &: \quad V(s_t) = V(s_t) + \alpha_T\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 V(s_{t+2}) - V(s_t)\right] \\
E_3 &: \quad V(s_t) = V(s_t) + \alpha_T\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V(s_{t+3}) - V(s_t)\right] \\
E_k &: \quad V(s_t) = V(s_t) + \alpha_T\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V(s_{t+k}) - V(s_t)\right] \\
E_\infty &: \quad V(s_t) = V(s_t) + \alpha_T\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{K-1} r_{t+K} + \cdots - V(s_t)\right]
\end{aligned}$$

Note that E_1 estimator is exactly TD(0) and E_∞ is exactly TD(1).
Consider the following weighted sum of all these estimators

$$\begin{aligned}
E &= (1-\lambda)E_1 + \lambda(1-\lambda)E_2 + \lambda^2(1-\lambda)E_3 + \cdot + \lambda^{k-1}(1-\lambda)E_k + \cdots + \lambda^\infty E_\infty \\
&= \sum_{k=1}^{\infty} \lambda^{k-1}(1-\lambda)E_k
\end{aligned}$$

Again when $\lambda = 0$ we get TD(0), when $\lambda = 1$ we get TD(1).

3.13 TD(λ) Empirical Performance

If we plot the error (compared to the update over infinite data) in value after TD update over finite data against λ, we will typically get a parabolic-like curve where TD(1) gives the worst error and the minimum occurs between $\lambda = 0.3$ and $\lambda = 0.7$. This demonstrates the advantage of TD(λ) that it combines the information from TD(0)—what happens at each step—and that of TD(1)—what happens at the end of the episode.

3.14 What Have We Learned

TD(1) is inefficient in terms of the way it uses data—it cannot use the information from the episodes that start from other states, or in other words it can only use the information at the end of the episodes, resulting in high variance. TD(0) has the nice property that it gives maximum likelihood estimate.

4 Convergence

4.1 Bellman Equations with Actions

The difference between using reinforcement learning for control and using reinforcement learning without control is whether or not there is action that is being chosen by the learner.
Bellman equations without action reads

$$V(s) = R(s) + \gamma \sum_{s'} T(s, s')V(s')$$

with TD(0) update rule

$$\begin{aligned} V_t(s_{t-1}) &= V_{t-1}(s_{t-1}) + \alpha_t \left[r_t + \gamma V_{t-1}(s_t) - V_{t-1}(s_{t-1})\right] \\ V_t(s) &= V_{t-1}(s) \text{ for other } s \end{aligned}$$

Bellman equations with action reads

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

with TD(0) update rule

$$\begin{aligned} Q_t(s_{t-1}, a_{t-1}) &= Q_{t-1}(s_{t-1}, a_{t-1}) + \alpha_t \left[r_t + \gamma \max_{a'} Q_{t-1}(s_t, a') - Q_{t-1}(s_{t-1}, a_{t-1})\right] \\ Q_t(s, a) &= Q_{t-1}(s, a) \text{ for other } s \end{aligned}$$

It actually takes two approximations

- if we knew the model, synchronously update it

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

- if we knew Q^* under the optimal policy, sampling asynchronously update it

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

4.2 Bellman Operator

Let B be an operator that maps value function to value function defined as

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

Therefore $Q^* = BQ^*$ is another way of writing Bellman equations, and $Q_t = BQ_{t-1}$ is another way of writing value iteration.

4.3 Contraction Mappings

If for all value functions F, G and some $0 \leq \gamma < 1$ we have

$$\|BF - BG\|_\infty \leq \gamma \|F - G\|_\infty$$

where $\|\cdot\|_\infty$ is the infinity norm or max norm defined as $\|Q\|_\infty = \max_{s,a} |Q(s,a)|$, then the operator B is a contraction mapping.

4.4 Contraction Properties

If B is a contraction mapping, then it has the following two properties

1. $F = BF$ has a unique solution F^*, the uniqueness can be proved by contradiction—suppose there are two distinct functions F^* and G^* satisfying $F^* = BF^*$ and $G^* = BG^*$ and $\|F^* - G^*\|_\infty > 0$, then we have the following contradiction with the fact that B is a contraction mapping

$$\|F^* - G^*\|_\infty = \|BF^* - BG^*\|_\infty$$

2. applying successively contraction mapping $F_t = BF_{t-1}$ results in F_t converging to F^*, the convergence can be shown by applying the defining property of contraction mapping

$$\|F_t - F^*\|_\infty = \|BF_{t-1} - BF^*\|_\infty \leq \gamma \|F_{t-1} - F^*\|_\infty$$

4.5 The Bellman Operator Contracts

To show the Bellman operator is a contraction mapping, apply the definition of the infinity norm and that of the Bellman operator twice

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

we have

$$\begin{aligned}
\|BQ_1 - BQ_2\| &= \max_{a,s} |BQ_1(s,a) - BQ_2(s,a)| \\
&= \max_{a,s} \left| \left[R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_1(s',a') \right] - \left[R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a''} Q_2(s',a'') \right] \right| \\
&= \max_{a,s} \left| \gamma \sum_{s'} T(s,a,s') \left[\max_{a'} Q_1(s',a') - \max_{a''} Q_2(s',a'') \right] \right| \\
&\leq \gamma \max_{s'} \left| \max_{a'} Q_1(s',a') - \max_{a''} Q_2(s',a'') \right| \leq \gamma \max_{s',a'} |Q_1(s',a') - Q_2(s',a')| = \gamma \|Q_1 - Q_2\|_\infty
\end{aligned}$$

The first inequality takes the maximum of the summands and uses $\sum_{s'} T(s,a,s') = 1$. We don't need to include $\max_{a,s}$ because the resulted expression doesn't include a and s at all.

4.6 Max is a Non-expansion

The second inequality above follows from the fact that for all functions f and g we have

$$\left|\max_{a'} f(a') - \max_{a''} g(a'')\right| \leq \max_{a} |f(a) - g(a)|$$

Intuitively it just says that when one reaches maximum the other usually cannot simultaneously reach minimum, let alone subtracting the maximum of the other.

4.7 Proof that Max is a Non-expansion

The proof of the above inequality regarding the maxima of two functions goes as follows: without loss of generality assume $\max_{a'} f(a') \geq \max_{a''} g(a'')$ so that we can get rid of the absolute value signs

$$\left|\max_{a'} f(a') - \max_{a''} g(a'')\right| = \max_{a'} f(a') - \max_{a''} g(a'') = f(a_1) - g(a_2) \leq f(a_1) - g(a_1)$$

where $a_1 = \arg\max_{a'} f(a')$ and $a_2 = \arg\max_{a''} g(a'')$, and the last inequality follows from the fact that $g(a_2) \geq g(a_1)$. It then follows that

$$f(a_1) - g(a_1) \leq |f(a_1) - g(a_1)| \leq \max_{a} |f(a) - g(a)|$$

Combining the two inequalities we prove that the maximum operation can be taken out of the absolute value signs

$$\left|\max_{a'} f(a') - \max_{a''} g(a'')\right| \leq \max_{a} |f(a) - g(a)|$$

and applying it to Q function

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

we prove that the Bellman operator is a contraction mapping.

4.8 Convergence

Define $\alpha_t(s, a) = 0$ if $s_t \neq s$ or $a_t \neq a$. Let B be a contraction mapping and $Q^* = BQ^*$ be its fixed point. Let Q_0 be a Q function and define $Q_{t+1} = [B_t Q_t] Q_t$, then Q_t converges to Q^* if the following three conditions are satisfied

1. for all Q functions Q_1, Q_2 and state-action pair s, a we have

$$\left|\left([B_t Q_1]Q^*\right)(s, a) - \left([B_t Q_2]Q^*\right)(s, a)\right| \leq \left[1 - \alpha_t(s, a)\right] |Q_1(s, a) - Q_2(s, a)|$$

2. for all Q functions Q, Q' and state-action pair s, a the contraction property holds

$$\left|\left([B_t Q]Q^*\right)(s, a) - \left([B_t Q]Q'\right)(s, a)\right| \leq \gamma \alpha_t(s, a) |Q^*(s, a) - Q'(s, a)|$$

3. the standard condition of learning rate holds

$$\sum_t \alpha_t = \infty \qquad \sum_t \alpha_t^2 < \infty$$

   note that under the assumption $\alpha_t(s, a) = 0$ if $s_t \neq s$ or $a_t \neq a$, the infinity condition $\sum_t \alpha_t = \infty$ implies that we need to visit every state-action pair infinitely often

4.9 Convergence Theorem Explained

Express Q-learning as an update rule in the form of B_t operator

$$[B_t Q]\widetilde{Q}(s,a) = Q(s,a) + \alpha_t(s,a)\left[r_t + \gamma \max_{a'} \widetilde{Q}(s_t,a') - Q(s,a)\right]$$

when $\widetilde{Q} = Q$ we recover the TD(0) update rule for Q function.
On the one hand fix $\widetilde{Q} = Q^*$ but vary Q we have the following condition

$$\left|\left([B_t Q_1]Q^*\right)(s,a) - \left([B_t Q_2]Q^*\right)(s,a)\right| \leq \left[1 - \alpha_t(s,a)\right]\left|Q_1(s,a) - Q_2(s,a)\right|$$

It means that if we knew where we were going, then we could use this update rule to average out the stochasticity associated with the transition function (because we knew the true value Q^*), resulting in Q_1, Q_2 not only converging to Q^* but also getting closer to each other.
On the other hand fix Q and compare the one-step look-ahead update of $\widetilde{Q} = Q^*$ and $\widetilde{Q} = Q'$, the second condition is satisfied

$$\left|\left([B_t Q]Q^*\right)(s,a) - \left([B_t Q]Q'\right)(s,a)\right| \leq \gamma\alpha_t(s,a)\left|Q^*(s,a) - Q'(s,a)\right|$$

which shows $B_t Q$ is a contraction mapping.
In summary by separating the Q-learning TD(0) update rule into two different Q functions and treating them differently, we can prove the different rules played by them—one is averaging out the stochasticity and the other is taking one-step backup value iteration which is a contraction mapping.
In fact not only Q-learning can be phrased into this scheme. The convergence of many other learning algorithms can be proved by being cast into this scheme, as long as the operator contracts and the look-ahead is non-expansion like max.

4.10 Generalized MDPs

The Bellman equations for generalized MDPs read

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

For regular MDPs the $\oplus$ and $\otimes$ operators are defined as

$$\oplus_{s'}^{s,a} g(s') = \sum_{s'} T(s,a,s')g(s') \qquad \otimes_{a'}^{s'} f(s',a') = \max_{a'} f(s',a')$$

Now defining the two operators as

$$\oplus_{s'}^{s,a} g(s') = \min_{s'} g(s') \qquad \otimes_{a'}^{s'} f(s',a') = \max_{a'} f(s',a')$$

gives the pessimistic or risk-averse MDP, because it expects the worst state and takes the best action against it.
Defining the two operators as

$$\oplus_{s'}^{s,a} g(s') = \sum_{s'} T(s,a,s')g(s') \qquad \otimes_{a'}^{s'} f(s',a') = \sum_{a'} \rho(\mathrm{ord}(a'))f(s',a')$$

where ρ is convex, sums to 1, and assigns weight to all actions not just the best action as in $\max_{a'}$ thus generalizes $\max_{a'}$, gives exploration-sensitive MDP, because it allows all possible actions weighted by some order function $\mathrm{ord}(a')$. It is also consistent with the sampling maximum action choice in ϵ-greedy algorithm.
Defining the two operators as

$$\oplus_{s'}^{s,a} g(s') = \sum_{s'} T(s,a,s')g(s') \qquad \otimes_{a'}^{s'} f(s',a') = \sum_{a'} \min \max_{a'} f(s',a')$$

where a' should be understood as a joint action from multiple agents, gives a zero-sum game.

4.11 What Have We Learned

Examples of non-expansive operations include order statistics, fixed convex combinations, and their combinations. In fact they pretty much span all non-expansive operations that we know. It is not true that arbitrary convex combinations result in non-expansion. To achieve non-expansion we need the coefficients to be fixed. Nonetheless there are algorithms that are built on expansive operations and yet still converge.

5 AAA

5.1 More on Value Iteration

We have two practical guidances for the convergence to optimal policy.

1. For some $t^* < \infty$ polynomial in $|S|$, $|A|$, $R_{\max} = \max_{a,s} |R(s,a)|$, $1/(1-\gamma)$, and bits of precision, we have $\pi(s) = \arg\max_a Q_{t^*}(s,a)$ is optimal. It says that there is some finite time steps at which we get a Q function that is close enough so that if we do the greedy policy with respect to it, it is optimal. Thus this problem can be solved within a reasonable amount of time. It is because once you fixed the MDP and the number of bits of precision, then the second best action will be of bounded amount away from optimal i.e. it can't be arbitrarily close to the optimal. Then when we run sufficiently many rounds of value iteration, the distance between the best action and the second best action gets larger than that distance lower bound, so that the greedy policy is going to choose the optimal. This is a consequence of Cramer's rule.

2. if $|V_t(s) - V_{t+1}(s)| < \epsilon$ for all s, then $\max_s |V^{\pi_{V_t}}(s) - V^*(s)| < 2\epsilon\gamma/(1-\gamma)$ where π_{V_t} is the greedy policy of value function V_t. It says that without knowing how close we are from the optimal, we can use the change between two consecutive iterations to bound the distance from the optimal.

Both follow from the contraction property of the Bellman operator $\|B^k Q_1 - B^k Q_2\|_\infty \leq \gamma^k \|Q_1 - Q_2\|_\infty$, and both are better off for small γ. The reason is that $1/(1-\gamma)$ is roughly the horizon or how far into the future do rewards matter, and it is easy to optimize over a short horizon. However a short horizon also implies shortsightedness. Hence there is a tradeoff between easy solution and good planning.

5.2 Linear Programming

Note that $1/(1-\gamma)$ isn't really polynomial in the number of bits to write γ, because you can use very few bits to specify a number that is very close to 1 causing $1/(1-\gamma)$ to explode. Thus value iteration doesn't provide a polynomial time algorithm for solving MDPs. There is only one way that can solve MDPs in polynomial time which is linear programming. However keep in mind that the max operation in the Bellman equations makes them a set of nonlinear equations. To cast them into a linear program that can be solved by linear programming, we need to interpret the max operation as a linear objective function with linear constraints.
The Bellman equations read

$$V(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right]$$

It can be cast into the following linear program

$$\min \sum_s V(s) \text{ subject to the constraint } V(s) \geq R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \text{ for all } s,\ a$$

This is the primal linear program. It has a dual version by the duality of linear programming, which turns constraints into variables and variables into constraints, and leads to a new linear program that is equivalent to the original one.
The dual of the above primal is

$$\max_{q_{sa}} \sum_s \sum_a q_{sa} R(s,a) \text{ so that } 1 + \gamma \sum_s \sum_a q_{sa} T(s,a,s') = \sum_a q_{s'a} \ \forall s' \text{ and } q_{sa} \geq 0 \ \forall s, a$$

where q_{sa} can be interpreted as sort of policy flow, as the equality constraint above can be interpreted as conservation of flows.

5.3 Policy Iteration

The steps consist of

initialization: initialize $Q_0(s) = 0 \ \forall s$

policy improvement: $\pi_t(s) = \arg\max_a Q_t(s,a) \ \forall s$

policy evaluation: $Q_{t+1} = Q^{\pi_t}$

which produces

- Q_t converges to Q^*
- the convergence is exact and completed in finite time, which is also true for value iteration
- it converges at least as fast as value iteration, but at the expense of more complicated computation—mainly the policy evaluation step because it involves solving a system of linear equations, in fact each policy iteration essentially does all the work of value iteration

In fact it is an open question as to how many iterations policy iteration would take. We know for some MDPs the number of iterations policy iteration takes is linear in $|S|$ which suggests a lower bound, and we know it can't be worse than an exponential of $|A|^{|S|}$. But we don't know where it sits in between.

5.4 Domination

In the context of policies

domination: $\pi_1 \geq \pi_2$ if and only if $V^{\pi_1}(s) \geq V^{\pi_2}(s) \ \forall s$

strict domination: $\pi_1 > \pi_2$ if and only if $V^{\pi_1}(s) \geq V^{\pi_2}(s) \ \forall s$ and $\exists s \in S$ such that $V^{\pi_1}(s) > V^{\pi_2}(s)$

ϵ-optimal: π is ϵ-optimal if and only if $|V^{\pi}(s) - V^{\pi^*}(s)| \leq \epsilon \ \forall s$

5.5 Why Does Policy Iteration Work

Define two one-step Bellman operators B_1, B_2 for the two policies π_1, π_2

$$[B_1 V](s) = R(s, \pi_1(s)) + \gamma \sum_{s'} T(s, \pi_1(s), s') V(s')$$
$$[B_2 V](s) = R(s, \pi_2(s)) + \gamma \sum_{s'} T(s, \pi_2(s), s') V(s')$$

5.6 B is Monotonic

If $V_1 \geq V_2$ then $BV_1 \geq BV_2$, because

$$(BV_1 - BV_2)(s) = \gamma \sum_{s'} T(s, \pi(s), s') [V_1(s') - V_2(s')]$$

$$\text{thus } V_1(s') - V_2(s') \geq 0 \ \forall s' \implies BV_1(s) - BV_2(s) \geq 0 \ \forall s$$

5.7 Another Property in Policy Iteration

Consider a policy π_1 and its associated Bellman operator B_1, let $Q_1 = B_1 Q_1$ i.e. Q_1 is the fixed point of B_1. Now take the greedy policy π_2 with respect to Q_1. Let B_2 be the Bellman operator associated with π_2. We then have $Q_1 \leq B_2 Q_1$, because

$$\begin{aligned} Q_1(s,a) &= R(s,a) + \gamma \sum_{s'} T(s, \pi_1(s), s') Q_1(s', \pi_1(s')) \longleftarrow Q_1 = B_1 Q_1 \\ &\leq R(s,a) + \gamma \sum_{s'} T(s, \pi_2(s), s') Q_1(s', \pi_2(s')) \longleftarrow B_2 \text{ is greedy for } Q_1 \end{aligned}$$

Essentially the idea is that, if we do one more update on a value function based on the policy that is greedy with respect to it, then we will improve it—value improvement.

5.8 Policy Iteration Proof

Let π_1 be a policy and B_1 its associated Bellman operator and Q_1 its fixed point. Let π_2 be the greedy policy with respect to Q_1, B_2 its associated Bellman operator and Q_2 its fixed point. We want to prove $Q_2 \geq Q_1$. The proof consists of four steps

value improvement: $B_2 Q_1 \geq Q_1$

monotonicity: $B_2^{k+1} Q_1 \geq B_2^k Q_1$

transitivity: $\lim_{k \to \infty} B_2^k Q_1 \geq Q_1$ since $B_2^\infty Q_1 \geq \cdots \geq B^{k+1} Q_1 \geq B_2^k Q_1 \geq \cdots \geq B_2 Q_1 \geq Q_1$

fixed point: $Q_2 = \lim_{k \to \infty} B_2^k Q_1 \geq Q_1$

5.9 Another Property in Policy Iteration

For the value improvement property, stuck in local minimum cannot happen because we will get strict improvement in at least one of the states if we haven't reached the optimal yet, since for a finite MDP we only have a finite number of action assignments.

5.10 What Have We Learned

value iteration: the value function doesn't converges in finite steps but the greedy policy does

linear programming: primal value $\leftrightarrow$ dual policy flow

policy iteration: the convergence is built on the concept of domination which is defined on a state-by-state basis thus prevents stuck in local minimum, value improvement and monotonicity together proves the convergence

6 Messing with Rewards

6.1 Changing the Reward Function

How can we change the MDP reward function without changing the optimal policy? There are at least three ways

- multiply by positive constant
- shift by a constant
- nonlinear potential-based rewards

6.2 Multiply by A Scalar Quiz

If $Q(s,a)$ is the solution to the Bellman equations

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

then $Q'(s,a) = cQ(s,a)$ is the solution to the following Bellman equations

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

where $R'(s,a) = cR(s,a)$ and $c > 0$.

6.3 Adding A Scalar

If $Q(s,a)$ is the solution to the Bellman equations

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

then $Q'(s,a) = Q(s,a) + c/(1-\gamma)$ is the solution to the following Bellman equations

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

where $R'(s,a) = R(s,a) + c$. Intuitively we can understand the constant $c/(1-\gamma)$ by observing that $Q' = Q + c + \gamma c + \gamma^2 c + \cdots$ since Q' witnesses a constant c in addition to the reward $R(s,a)$ in every iteration.

6.4 Potential-based Shaping in RL

To prevent from creating a suboptimal positive loop, we can introduce a potential function that defines state-based bonuses. Whenever we enter a state we gain the bonus associated with that state. Similarly whenever we leave a state we lose the bonus associated with that state. This can be incorporated into the reward function as follows

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

where $-\psi(s)$ represents the bonus lost as we left state s, while $\gamma\psi(s')$ represents the bonus gained as we enter state s' (leaving is now but entering is in the future thus discounted). Suppose $Q(s,a)$ is the solution to the Bellman equations

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

then $Q'(s,a) = Q(s,a) - \psi(s)$ is the solution to the following Bellman equations

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

which can be guessed from the expression of $R'(s,a,s')$ and notice that $Q(s,a)$ doesn't depend on the future state s' thus the bonus of entering s' is irrelevant for $Q'(s,a)$

$$\begin{aligned} R'(s,a,s') &= R(s,a) - \psi(s) + \gamma\psi(s') \\ Q'(s,a) &= Q(s,a) - \psi(s) \end{aligned}$$

An intuitive way to figure out the bonus $\psi(s)$ lost is to follow the same argument for finding $Q'(s,a)$ when adding a scalar c to $R(s,a)$—consider a state transition $s \to s' \to s'' \to s''' \to \cdots$

$$\begin{aligned} Q' = {} & R + \gamma R + \gamma^2 R + \gamma^3 R + \cdots \\ & + [-\psi(s) + \gamma\psi(s')] + \gamma [-\psi(s') + \gamma\psi(s'')] + \gamma^2 [-\psi(s'') + \gamma\psi(s''')] + \cdots \end{aligned}$$

where the first line gives Q and only $-\psi(s)$ survives the second line if $0 \le \gamma < 1$.
This solution to the Bellman equations defined by $R'(s,a,s')$ can be verified by substituting in both $Q'(s,a)$ and $R'(s,a,s')$ in terms of $Q(s,a)$ and $R(s,a)$

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

where $\sum_{s'} T(s,a,s')\psi(s)$ cancels the $\psi(s)$ on the left-hand side, and $\gamma\psi(s')$ cancels the $\psi(s')$ inside max(). Note that Q' has the same optimal policy as Q since the lost bonus $\psi(s)$ is not action-dependent.

6.5 Q-learning with Potentials

The Q-learning with potentials reads

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

In the limit it should solve the original MDP with just r. Because the choice of ψ doesn't affect the limit, we can tailor it to our advantage. For example let $\psi(s) = \max_a Q^*(s,a)$ where Q^* is the solution to the original MDP so that $Q(s,a) = Q^*(s,a) - \psi(s) = Q^*(s,a) - \max_a Q^*(s,a)$. So defined $Q(s,a) = 0$ if a is the optimal action and < 0 otherwise. Eric showed that the series of updates you get by running Q-learning with potential function is the same as the updates you get by running Q-learning without potential function but with the Q function initialized to what the potential function would have been, same in the sense that it goes through the same series of action choices even though the Q functions are different. This result emphasizes the importance of knowledge injection through initialization.

6.6 What Have We Learned

Potential function can speed up the learning.

7 Exploring Exploration

7.1 Confidence-based Exploration

Suppose we have two bandit arms a and b. We pulled a once and got no payoff. We pulled b twice and got payoff once. We will always choose b based on either maximum likelihood or maximum confidence. If we choose based on minimum confidence we will choose a and b alternatively and are never going to use the information. Maximum likelihood provides more reward whereas minimum confidence provides better estimates as it brings in new information, this is termed the exploration-exploitation dilemma.

7.2 Metrics for Bandits

Possible metrics include

1. identify optimal arm in the limit
2. maximize the (discounted) expected reward, note that there is an index called *Gittins index* which specifies how high a certain payoff should be given in order to make forgoing pulling the bandit arm a wise decision, it combines the notion of maximum likelihood and low confidence, it works very well for bandit problems but is very difficult to generalize to other problems
3. maximize expected reward over finite horizon
4. identify near-optimal arm (ϵ-close) with high probability $(1-\delta)$ in time $\tau(k, \epsilon, \delta)$ polynomial in k, $1/\delta$, $1/\epsilon$ where k is the number of arms
5. nearly maximize reward (ϵ-close) with high probability $(1-\delta)$ in time $\tau'(k, \epsilon, \delta)$ polynomial in k, $1/\delta$, $1/\epsilon$
6. pull a non-near optimal arm (not ϵ-close) no more than $\tau''(k, \epsilon, \delta)$ times with high probability $1-\delta$

It turns out that metric #4, #5, #6 are equivalent in the sense that an algorithm that achieves one gives us an algorithm that achieves the other two.

7.3 Find Best Implies Few Mistakes

Suppose we have an algorithm A that finds the ϵ-close near-optimal arm with probability $(1-\delta)$ in $\tau(k, \epsilon, \delta)$ pulls, and we want an algorithm that choose a non-ϵ'-close suboptimal arm for not more than $\tau'(k, \epsilon', \delta')$ times with probability $(1-\delta')$. We can set $\epsilon = \epsilon'$, $\delta = \delta'$, and run algorithm A for $\tau(k, \epsilon', \delta')$ pulls until we get the ϵ'-close optimal arm. Then we can pull it forever. The number of mistakes it makes over the whole run is just the number of mistakes it made to identify the ϵ'-close optimal arm, beyond which we will not make any non-ϵ'-close mistake. Hence $\tau(k, \epsilon', \delta')$ sets an upper bound on the number of mistakes. So we just need to set $\tau'(k, \epsilon', \delta') \geq \tau(k, \epsilon', \delta')$.

7.4 Few Mistakes Implies Do Well

Suppose we have an algorithm B that chooses a non-ϵ-close suboptimal arm for not more than $\tau(k, \epsilon, \delta)$ times with probability $(1-\delta)$, and we want an algorithm that nearly maximizes per-step reward (ϵ'-close) with high probability $(1-\delta')$ after $\tau'(k, \epsilon', \delta')$ pulls. Again let $\delta = \delta'$ and run algorithm B for $\tau'(k, \epsilon', \delta')$ steps. Because for the first $\tau(k, \epsilon, \delta')$ pulls we always make mistakes i.e. non-ϵ-close suboptimal arm, and for the remaining $\tau'(k, \epsilon', \delta') - \tau(k, \epsilon, \delta')$ steps we still have a probability of ϵ to

not achieve ϵ-close per-step optimal reward. Hence the total number of mistakes over n steps, the ratio of which should be smaller than ϵ', is

$$\epsilon' \geq \frac{\tau(k,\epsilon,\delta') + \epsilon\left(\tau'(k,\epsilon',\delta') - \tau(k,\epsilon,\delta')\right)}{\tau'(k,\epsilon',\delta')}$$

To solve for $\tau'(k,\epsilon',\delta')$ we pick $\epsilon = \epsilon'/2$ i.e. choose a tighter error bound, then we have

$$\frac{\epsilon'}{2}\tau'(k,\epsilon',\delta') \geq \left(1 - \frac{\epsilon'}{2}\right)\tau(k,\epsilon'/2,\delta')$$

$$\implies \tau'(k,\epsilon',\delta') \geq \left(\frac{2}{\epsilon'} - 1\right)\tau(k,\epsilon'/2,\delta')$$

7.5 Do Well Implies Find Best

Suppose we have an algorithm C that nearly maximizes per-step reward (ϵ-close) with high probability $(1-\delta)$ after $\tau(k,\epsilon,\delta)$ pulls, and we want an algorithm that finds the ϵ'-close near-optimal arm with probability $(1-\delta')$ in $\tau'(k,\epsilon',\delta')$ pulls. To find it consider the suboptimality e_i of the plurality arm—arm i that was chosen at least as many times as if not more than any other arms, thus it was chosen at least $1/k$ times. Hence the per-step error $\epsilon \geq e_i \cdot (1/k)$ or equivalently $e_i \leq \epsilon k$. Per requested this plurality arm should be ϵ'-close to optimal arm i.e. $e_i \leq \epsilon'$. Combined we have $\epsilon' = \epsilon k$ and we should set $\tau(k,\epsilon'/k,\delta') = \tau'(k,\epsilon',\delta')$.

7.6 Putting It Together

We have proven

$$\text{find best} \implies \text{few mistakes} \implies \text{do well} \implies \text{find best}$$

which is equivalent to

$$\text{find best} \iff \text{few mistakes} \iff \text{do well}$$

so that we can pick whichever that is most convenient to prove and get all three of them.

7.7 Hoeffding

Let $X_1, \ldots, X_n$ be independent identically distributed random variables with mean μ. Let $\hat{\mu}$ be our estimate of μ say $\sum_i X_i/n$. The following is a $100(1-\delta)\%$ confidence interval for μ

$$\left[\hat{\mu} - \frac{Z_\delta}{\sqrt{n}}, \hat{\mu} - \frac{Z_\delta}{\sqrt{n}}\right] \quad \text{where } Z_\delta = \sqrt{\frac{1}{2}\ln\frac{2}{\delta}}$$

7.8 Combining Arm Info

We will take c samples from each arm, then choose the arm with the best estimate. We want the arm to be ϵ-close to the optimal arm with a probability $(1-\delta)$. To achieve this we need to learn each arm to within $\epsilon/2$ accuracy with a probability $(1-\delta/k)$. Because the probability that at least one of the estimates is wrong i.e. beyond $\epsilon/2$ from the true value is bounded by $\delta/k \cdot k = \delta$. Being ϵ-close to the optimal is taken care of by the $\epsilon/2$ design—we can at most underestimate the best arm (the one we failed to pick) by $\epsilon/2$ and overestimate the second best arm (the one we picked) by $\epsilon/2$.

7.9 How Many Samples

Now that we want to be $\epsilon/2$ accurate with probability $1 - \delta/k$ by taking c samples, we can plug them into the Hoeffding bound above

$$\frac{\sqrt{\frac{1}{2}\ln\left(\frac{2k}{\delta}\right)}}{\sqrt{c}} \leq \frac{\epsilon}{2} \implies c \geq \frac{2}{\epsilon^2}\ln\left(\frac{2k}{\delta}\right)$$

7.10 MDP Optimization Criteria

Exploring randomly can be problematic. It can make the probability of reaching the reward maximizing state exponentially small. Trap states can also stop us from maximizing reward. Thus we advocate mistake bound as the MDP optimization criterion—the number of ϵ-suboptimal action choices is bounded by a polynomial of $1/\epsilon$, $1/\delta$, n, and k.

7.11 Exploring Deterministic MDPs

1. keep track of the MDP we have so far
2. assume any unknown state-action pair is $R_{\max}$
3. solve the MDP
4. take action from π^*

7.12 Rmax Analysis

- once we know all the edges, we would not make any mistake i.e. take suboptimal action
- what if we stop visiting unknown states? if we still don't want to go out of the loop even when assuming that any unknown state-action pair is $R_{\max}$, then we will not make any mistake even if we are looping without learning anything
- the number of times a policy can bring us to an unknown state-action pair is bounded $n \cdot k$, where n is the total number of states and k is the number of actions available, and the number of steps that we can take to get to that state-action pair is bounded by n, therefore the total number of mistakes this algorithm can make is bounded by n^2k which is a polynomial, hence this is an efficient exploration algorithm for deterministic MDPs

7.13 Lower Bound

To show that $O(n^2k)$ is the lower bound for exploring deterministic MDPs, consider the worst case—a sequential combination lock in which for all states there are k but one actions that send it back to the beginning, and the only right action moves towards the most rewarding state at the far end. For this deterministic MDP the exploration takes k actions to figure out the correct one at each of the n states, and if the action is wrong it takes i steps to go back to state i, and the exploration till reaching the final most rewarding state takes n steps. Combined it takes $O(n^2k)$ steps to explore this MDP.

7.14 General Stochastic MDPs

To do efficient exploration in general stochastic MDPs, we need to combine the two ideas that we were talking about

stochastic stateless bandit: repeat to accumulate data until estimates are accurate enough via Hoeffding bound

sequential stateful deterministic MDP: optimistic assumption for unknown state-action pairs

Combining them we can obtain a stochastic version of Rmax analysis.

7.15 General Rmax

It looks pretty much like the Rmax analysis for deterministic MDPs

1. keep track of the MDP we have so far
2. assume any unknown state-action pair is $R_{\max}$
3. solve the MDP
4. take action from π^*

The only difference is the notion of unknown. A state-action pair is unknown if it's been visited fewer than c times. After that we switch over to the maximum likelihood estimate—if $(s, a) \to s'$ occurs only once then the probability of that transition is $1/c$.
As we see, the four steps look almost identical to the deterministic MDP case, whereas the definition of unknown state-action pair is related to the Hoeffding bound estimate in the stochastic bandit case.

7.16 Simulation Lemma

The idea is that if we have a pretty good estimate of the real MDP, then optimizing rewards in that estimate is going to be pretty good in the real MDP. The sense of being good estimate of real MDP can be quantified as follows: suppose the real MDP has transitions T_1 and rewards R_1 and the estimate has transitions T_2 and rewards R_2. We want

$$\max_{s,a,s'} |T_1(s,a,s') - T_2(s,a,s')| \leq \alpha \qquad \max_{s,a} |R_1(s,a) - R_2(s,a)| \leq \alpha$$

In this case for a fixed policy π we have

$$\epsilon = |V_1^\pi(s) - V_2^\pi(s)| \leq \frac{G}{1-\gamma} \qquad \text{where } G = \alpha + \gamma n \alpha R_{\max}/(1-\gamma)$$

To find how accurate our estimate should be for a given ϵ precision, we can solve

$$\epsilon = \frac{\alpha + \gamma n \alpha R_{\max}/(1-\gamma)}{1-\gamma} \implies \alpha = \frac{\epsilon(1-\gamma)}{1 + \gamma n R_{\max}/(1-\gamma)}$$

But from the general Rmax analysis above, what we want is a parameter like c that says how many times do we have to try a state-action pair before we are confident that our estimate of that transition probability is within the α bound.

7.17 Explore-or-Exploit Lemma

If all transitions are either accurately estimated or unknown, then optimal policy is either near optimal or an unknown state is reached quickly (in polynomial time) and something new is learned. It says that stuck in doing something suboptimal for a long time would not happen.

8 Generalization

8.1 Generalization

The idea of generalization is to leverage learning in states where we've been to make predictions about states where we haven't been.

8.2 Generalization Idea

The goal of reinforcement learning is to learn a policy mapping from states to actions. Often we don't know how to learn a policy directly so we learn a value function instead, which maps state-action pairs to estimated returns (Q function). Another map that we can learn in the context of (model-based) reinforcement learning is a model which maps state and action to next state. It is actually a supervised example of transitions. Mostly what researchers have focused on is function approximation of the value function.

8.3 Basic Update Rule

Q function can be written in the general form $Q(s,a) = F(w^a, f(s))$, where the weight w^a is updated according to

$$\Delta w_i^a = \alpha \underbrace{\left(\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{Bellman equations}} - Q(s,a) \right)}_{\text{TD error}} \underbrace{\frac{\partial Q(s,a)}{\partial w_i^a}}_{\text{gradient}}$$

8.4 Linear Value Function Approximation

We approximate Q function as a weighted linear combination of the features that represent the state $Q(s,a) = \sum_{i=1}^{n} f_i(s) w_i^a$ where n is the number of features and the subscript a is the action parameter that provides generalization.

8.5 Does it Work

Recent success include

3-layer backpropagation: Tesauro's work on backgammon, the value function being predicted was approximated by a backpropagating net, Wang & Dietterich's work on predicting the likelihood of success of NASA shuttle scheduling strategies, other people were not able to work nearly as well because using bootstrapping means you are dependent on making predictions based on previous predictions, so when predictions go south it is easy to fall into a death spiral quickly

CMAC: Sutton's CMAC is like a neural net except that the input layer isn't learned

linear nets: Singh & Parr applied linear nets to cell phone channel allocation, the work reveals the importance of feature selection, which should balance between computational efficient (extracted from states) and value-informative

deep nets: Mnih et al. from DeepMind developed strategies for Atari video games mapping screen pixels to joystick actions

Two lessons come out of the stories

- the way of training is important, Tesauro showed that TD works well in backgammon, but Boyan and Moore showed that it works poorly for some very simple examples, later Sutton replied by changing the way of training to make it work on those cases, so there is no guarantee that it has to work or has not to work
- whether adjacent states have values close to each other is the make-or-break for function approximation, Pollack showed that genetic algorithm works well on backgammon as well, because the game contains a lot of randomness that forces it to explore so as to prevent it from falling into a death spiral, on the other hand Isbell in his master thesis work showed that it does not work well on Tic-Tac-Toe, mainly because the game is too deterministic which leaves a substantial portion of the state space unexplored, and nearby states can have drastically different consequences to the game

8.6 Bairds Counterexample

It consists of 6 initial states, all one step away from the 7th terminal state, with no rewards, no choice of actions thus deterministic. Therefore the value function is zero for all states. The feature vectors are tabular, with each of the 7 states having only two features turned on and each of the 8 features turned on in only one state except the 0th feature

$$V(1) = w_0 + 2w_1 \qquad V(2) = w_0 + 2w_2 \qquad V(3) = w_0 + 2w_3 \qquad V(4) = w_0 + 2w_4$$
$$V(5) = w_0 + 2w_5 \qquad V(6) = w_0 + 2w_6 \qquad V(7) = 7w_0 + w_7$$

It turns out that there are infinitely many linear representations for this value function

$$w = c[-1, 1/2, 1/2, \ldots, 1/2, 7]$$

8.7 Bad Update Sequence

Suppose we start with $w_i = 1$ and $V(7) \gg V(i)$. We will update the weights in a round robin fashion: $\langle 1, 0, 7\rangle$, $\langle 2, 0, 7\rangle$, $\ldots$, $\langle 7, 0, 7\rangle$ and then repeat using $\gamma = 0.9$ and $\alpha = 0.1$. After $\langle 6, 0, 7\rangle$ w_0 is updated to 10.4. The last update $\langle 7, 0, 7\rangle$ brings w_0 down to 5.234, which is still larger than its initial value of 1. Hence it will spiral out of control. On the other hand if we start with $w_i = 0$, they would remain 0 (in general in a deterministic setting if the rewards are all zero, then the value functions being zero everywhere is the solution).

8.8 Averages

The value of a state s is given by

$$V(s) = \sum_{s_b \in B} w(s, s_b) V(s_b) \qquad \text{where } w(s, s_b) \geq 0, \sum_{s_b \in B} w(s, s_b) = 1$$

where B is a basis set of states of which the values are known. This is a convex combination approximator.

8.9 Connection to MDPs

Substitute the above convex combination approximator into the Bellman equations

$$\begin{aligned} V(s) &= \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right) \\ &= \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') \sum_{s_b \in B} w(s', s_b) V(s_b) \right) \\ &= \max_a \left(R(s,a) + \gamma \sum_{s_b \in B} \left[\sum_{s'} T(s,a,s') w(s', s_b) \right] V(s_b) \right) \\ &= \max_a \left(R(s,a) + \gamma \sum_{s_b \in B} T'(s,a,s_b) V(s_b) \right) \end{aligned}$$

which is a MDP built only on basis states, since $T'(s,a,s_b)$ retains the convexity of transition functions and it has a unique value solution. As the number of basis states increases, the error in the function approximation goes down and the solution converges to the true Q function.

8.10 What Have We Learned

There is both need for and method to do generalization in reinforcement learning. The need comes from the possibility of having too many states, and the method can be borrowed from supervised learning, in particular linear function approximation.
Although in this lesson we exclusively focused on value function approximation, there are successes in policy function approximation in robotics through policy gradient method.
Even linear approximation may not work in some cases, but incorporating function approximation into the loss function turns out to provide a good solution.

9 Partially Observable MDPs

9.1 POMDPs

The quantities that make up a POMDP consist of

- (A, S, T, R) that make up a MDP inside the POMDP, where S is not directly observable to agents
- observables Z, and observation function $O(s, z)$ which describes the probability of seeing z given that the MDP is in state s

9.2 POMDPs Generalize MDPs Quiz

POMDPs are generalization of MDPs in the sense that for every MDP $\langle S, A, T, R \rangle$ you can build an POMDP $\langle S, A, T, R, Z, O \rangle$ out of it, by simply setting $Z = S$ and $O(s, z) = 1$ if and only if $z = s$.

9.3 POMDP Examples

For a given MDP as depicted, suppose taking a left action ends up in a blue state and then taking a right action ends up in a blue state as well. Then we take a left action again. What is the probability of ending up in a blue state, in a green state?

To solve it suppose we start off from state 1, 2, 3 with equal probability 1/3 (it turns out to be a wrong guess, see below). A left action takes state 1 to itself, state 2 to state 1, and state 3 to state 1,2,4 with equal probability. Hence after a left action the state probabilities are (7/9, 1/9, 0, 1/9). A right action takes state 1 to state 2, state 2 to state 3, and state 4 to itself. Since state 3 is a green state not a blue state, after a right action the state probabilities are (0, 7/9, 0, 1/9), which do not sum to 1 thus need normalization. To avoid this we should have assumed state probability 0 for starting from state 3, because after a left and a right action it returns to itself which is a green state not a blue state.

9.4 State Estimation

Belief state $b(s)$ describes the probability that we are in state s. We use it to build a so-called **belief MDP** where the states are probabilities of being in the states of the POMDP $SE : (b, a, z) \to b'$, where $b'(s')$ is the probability of being in state s' after (b, a, z)

$$\begin{aligned}
\Pr(s'|b,a,z) &= \Pr(z|b,a,s')\Pr(s'|b,a)/\Pr(z|b,a) \\
&= \Pr(z|b,a,s') \sum_s \Pr(s'|s,b,a) \cdot \Pr(s|b,a)/\Pr(z|b,a) \\
&= \Pr(z|s') \sum_s \Pr(s'|s,b,a) \cdot \Pr(s|b,a)/\Pr(z|b,a) \\
&= O(s',z) \sum_s T(s,a,s')b(s)/\Pr(z|b,a)
\end{aligned}$$

This way we have turned a POMDP into a MDP with infinite number of states

9.5 Value Iteration in POMDPs

Let $V_0(b) = 0$. Update $V_t(b)$ by

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

where $R(b,a) = \sum_s b(s)R(s,a)$ and b' is the state estimation of (b,a,z) i.e. $b' = SE(b,a,z)$. To jump from summing over infinite beliefs to summing over finite states, we claim that

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

This max over linear functions of beliefs becomes a convex piecewise linear function with finite pieces.

9.6 Piecewise Linear and Convex

We will first prove by induction that if we can represent the value function at time 0 using a finite set of vectors Γ_0 and the value function at time $t-1$ using a finite set of vectors Γ_{t-1}, then we can represent the value function at time t using a finite set of vectors Γ_t.
For the base case $V_0(b) = \max_{\alpha \in \Gamma_0} \alpha \cdot b = 0$, we can let $\Gamma_0 = [0, \ldots, 0]$ with dimension $|S|$. Alternatively we could let $V_0(b)$ be the immediate reward $R(a)$ and let $\Gamma_0 = [R(s_0,a), \ldots, R(s_n,a)]$.
For the induction step of building Γ_t such that $V_t(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b$ with Γ_{t-1} satisfying $V_{t-1}(b) = \max_{\alpha \in \Gamma_{t-1}} \alpha \cdot b$, we spell out the Bellman equation

$$V_t(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')V_{t-1}(s')$$

as follows

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

where $V_t^{a,z}(b)$ is defined as

$$V_t(b) = \max_{a \in A} V_t^a(b) \qquad \text{where } V_t^a(b) = \sum_z V_t^{a,z}(b)$$

The last piece is to observe that $\Gamma_t = \cup_a \Gamma_t^a$ and $\Gamma_t^a = \oplus_z \Gamma_t^{a,z}$ where

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

Thus if $\Gamma_t^{a,z}$ is a finite set so is Γ_t^a and Γ_t. To show why $\Gamma_t^{a,z}$ can be represented as above and thus is finite, write out $V_{t-1}(SE(b,a,z))$ explicitly in the above Bellman equation spellout

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

where the denominator $\Pr(z|a,b)$ nicely cancels out the $\Pr(z|a,b)$ in the expression of $V_t^{a,z}(b)$

$$V_t^{a,z}(b) = \sum_s R(s,a)\frac{b(s)}{|Z|} + \gamma \sum_{s'} \alpha(s') O(s',z) \sum_s T(s,a,s') b(s)$$

Taking out the common factor $b(s)$ we have $V_t(b) = \max_{\alpha \in \Gamma_t} \alpha \cdot b$ and the expression of $\Gamma_t^{a,z}$ as above, which together with $\Gamma_t = \cup_a \Gamma_t^a$ and $\Gamma_t^a = \oplus_z \Gamma_t^{a,z}$ implies $|\Gamma_t| = |\Gamma_{t-1}|^{|Z|} \cdot |A|$. It shows that $|\Gamma_t|$ is finite if $|\Gamma_{t-1}|$ is.

9.7 Algorithmic Approach

A vector can be purged if it does not participate in the maximum operation, which needs to be solved by linear programming (still polynomial though). A vector is dominated if there is another vector that lies above it throughout the range of independent variables, which can be checked by comparing the corner values.

9.8 RL for POMDPs

The results we have for RL in POMDPs are mostly empirical. In model-based RL we try to learn a POMDP whereas in model-free RL we try to map observations to actions.

9.9 Learning a POMDP

	uncontrolled	controlled
observed	Markov chain (MC)	Markov decision process (MDP)
unobserved	hidden Markov model (HMM)	POMDP

9.10 Learning Memoryless Policies

That is it for model-based POMDP. Now model-free POMDP. Memoryless policies has the form that you can choose to move either to the left or to the right deterministically or probabilistically. But you can't take a sequence because that requires memory.

For the state transition and reward map given, suppose the discount factor is γ, the probability of left transition is p and that of right transition is $1-p$, we have

$$x = p \cdot \gamma x + (1-p) \cdot \gamma y$$
$$y = p \cdot \gamma x + (1-p) \cdot 1$$
$$z = p \cdot 1 + (1-p)(\gamma z)$$

9.11 Bayesian RL

We will talk about how we can think of RL as a POMDP itself. The idea is that we are going to keep a Bayesian posterior over possible MDPs that we are in, and use it to optimally balance exploration and exploitation.
Suppose there are three two-state MDPs differing only in the location of reward and we don't know which one we are in. We can treat it as a six-state POMDP. Suppose we know we are in state 1, then the belief states are (1/3,0,1/3,0,1/3,0). In this context an optimal policy means both exploration—figure out which MDP we are in—and exploitation—earn high reward. With the Bayesian or POMDP perspective the distinction between exploration and exploitation no longer exist.
Under the Bayesian perspective, RL is actually planning in some kind of continuous POMDP, where the hidden states are the parameters of the MDPs we want to learn. Because there are now infinitely many MDPs we don't get the piecewise linear and convex property any more. But it can be shown that the value function in this continuous space POMDP is actually piecewise polynomial and convex, albeit the degree of the polynomial grows with value iterations.

9.12 Predictive State Representation

Whereas in POMDP a belief state is a probability distribution over states, in predictive state representation (PSR) a predictive state is a probability distribution over future predictions/action outcomes.

9.13 PSR Example

Since each prediction is independent, the probabilities do not need to sum to 1. If we have enough tests then we can establish a bijection between predictive states and belief states.

9.14 PSR Theorem

Any n-state POMDP can be represented by a PSR with no more than n tests, each of which is no longer than n steps long.

9.15 What Have We Learned

Bayesian RL blurs the line between planning and learning.

10 Options

10.1 What Makes RL Hard?

delayed reward: RL receives weak feedback—rewards are not truth as in supervised learning

bootstrapping: estimates based on estimates, thus requires exploration to set up good target

scaling with number of states and actions: RL usually grows super-linearly with the number of states and actions, which is the motivation for generalization over states and generalization over actions, the latter being the focus of this lesson

10.2 Temporal Abstraction

State abstraction is treating a set of states equivalently. Similarly temporal abstraction is treating a set of time steps equivalently. Besides it has the computational benefit of quickly backing up values at remote states.
Action abstraction allows us to treat a set of states similarly, because the optimal actions for them are the same.

10.3 Temporal Abstraction Options

An b is a 3-tuple $\langle I, \pi, \beta \rangle$ consisting of I = initiation set of states, π = policy $[s, a] \mapsto [0, 1]$, β = termination set of states $s \mapsto [0, 1]$.

10.4 Temporal Abstraction Option Function

Option is an generalization of action, because for a given action we can always define I to be the states at which the action is valid, $\pi = 1$ for taking that action and 0 for the other actions, and β to be the states that can be reached via taking that action. Let o denote an option, we have

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

which bears much resemblance with the Bellman equations for MDP. However the formulation in terms of option violates some of the properties of MDPs where all time steps are equal and action is atomic. Because with option the amount of time spent in each option is variable and action is not necessarily atomic. This is reflected in the expression of the reward in terms of option

$$R(s, o) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{k-1} r^k | s, k \text{ steps} \right]$$

where k is the total number of steps taken by the option o. Accordingly the transition function F is

$$F(s, o, s') = \mathbb{E}\left[\gamma^k \Delta_{s_k, s'} | s, k \text{ steps} \right]$$

where $\Delta_{s_k, s'}$ is the Kronecker delta that enforces ending up in state s'. In fact expanding R and F over individual time steps we recover the Bellman equation. This variable-time MDP is called **semi-MDP**.

10.5 Pac-Man Problems

Option allows us to do state abstraction and ignore large parts of the state space.

10.6 How It Comes Together

Semi-MDP inherits the optimality of MDP up to a point as all the convergence proof and stability proof carry over. However with a particular choice of option there is no guarantee that it would end up with an optimal policy. Because as mentioned above option ignores parts of the state space. Hence whether the option pays attention to the relevant parts of the state space affects whether we are going to get the true optimality. What can be guaranteed is the **hierarchical optimality**—it would converge to an

optimal policy with respect to the set of options given. The payoff brought by embracing hierarchical optimality instead of optimality is that we can ignore the boring parts of the state space and focus on the exploration of the relevant parts of the state space, relevant in the sense that we are provided the choices of action by the option there.

10.7 Goal Abstraction

Standing at a higher level option represents competing goals (primitive vs. high-level). In the Pac-man example the competing goals are to eat small dots, eat big dots, and eat/avoid ghosts. This changes our perspective from planning sequential actions to managing competing goals—goal arbitration. It has the name **modular reinforcement learning** because different options define different goals which divide the problem into modules just like the Pac-man example. The three most popular modular reinforcement learning algorithms are

greatest mass Q-learning: the action that maximizes the sum of the Q-functions of all the modules (for the Pac-man example they are $Q_{\text{eat small dots}}$, $Q_{\text{eat big dots}}$, $Q_{\text{eat ghosts}}$, $Q_{\text{avoid ghosts}}$) is selected $\arg\max_a \sum_i Q_i(s,a)$, it is the simplest algorithm among the three, but the action selected might just be a compromise among all the agents, which is not good for any of them

top Q-learning: give the Q-function that has the maximal possible value $Q(s,a) = \max_i Q_i(s,a)$ to select its value maximizing action, but this top Q-function that gets the right to select the action might simply assign nearly the same high value to all actions thus is not discriminating enough, this motivates the third algorithm below

negotiated W-learning: the agent with the most to lose i.e. the maximal difference between the best action and the worst action gets the right to choose action

However because all of them are based on voting, they all suffer Arrow's impossibility theorem, which asserts that it is impossible to construct a fair voting system. In addition all of the above algorithms assume that the Q values of different agents are compatible. If not then we have to make them compatible in some way first before the arbitrator chooses according to whatever selection rule adopted.

10.8 Monte Carlo Tree Search

The tree search consists of looping over four steps

select: decide what action (edge) to take to go to the next state (node) following the learned policy

expand: when in doubt what further action to take (leaf), expand the tree from current node, in case there are too many states that can be expanded to, sampling is conducted

simulate: estimate the values of taking actions from the expanded set of nodes according to some other policy which is usually called *rollout policy* and could simply be a random policy—Monte Carlo

back up: back up the values learned from the simulation to all the selected nodes in the Bellman way, which may trigger revision of the previous selections, but the outcome is more accurate Q value estimates for these visited nodes

After one iteration of the above four steps, the original policy is revised and extended to the expanded nodes (new leaves) of the tree, which is ready for next round of iteration. This iterative search stops when the computational resource is exhausted (time-out) or when all the states and actions have been learned (in case we are working on a finite MDP). After that we can execute the greedy actions

according to the learned policy.
There are at least two ways that can improve the above Monte Carlo tree search (MCTS)

from random simulation to constrained simulation: recall the Pac-man example, out of the four goals the avoiding ghosts is peculiar in the sense that it ends the game only when it fails, thus it behaves more like a constraint rather than a goal, and the random policy can be replaced by a constrained policy in the sense that we want to honor the constraint so as to run the simulation for as long as possible, and we can also select from the expanded nodes based on how long the simulation can go from them without violating the constraint

from action to option: use option throughout the search rather than taking primitive steps, and estimate the Q values defined on options rather than on actions

Note that although MCTS is a kind of tree search, it still qualifies as reinforcement learning. Because it is actually a policy search algorithm, and the back-up in the Bellman way for learning a better policy is the brand of reinforcement learning.

10.9 Monte Carlo Tree Properties

- it is useful for large state spaces, because it only visits the states it needs to visit
- however depending on the randomness of the transition function, it may need lots of samples to get a good estimate
- the planning time is independent of the number of states $|S|$
- however the tradeoff is that the running time is exponential in the horizon H, $O((|A| \cdot k)^H)$, where k is the number of steps the search is to run for, and the horizon H is determined by the discount factor (the smaller the shorter horizon), this is essentially a space-time tradeoff because the more samples we need the further into the future the simulation should go

10.10 What Have We Learned

We discussed several generalizations beyond function approximation, including

temporal abstraction: hierarchical reinforcement learning, option and semi-MDP

goal abstraction: modular reinforcement learning, arbitration

state abstraction: resulted from temporal abstraction

All these generalizations can be applied to MCTS, especially temporal abstraction because the running time of MCTS is exponential in the horizon.

11 Game Theory

11.1 What is Game Theory

What we have been doing with reinforcement learning is mostly pretending that the other agents are just part of the environment and what they do is hidden inside the transition model. Here we are going to move from the reinforcement learning world of single agent to the game theory world of multiple agents.

11.2 A Simple Game

Strategies in the game theory world correspond to policies in the reinforcement learning world.

11.3 Minimax

In this two-player zero-sum finite deterministic game of perfect information, each player must consider the worst case counter-strategy, and end up with a strategy that maximizes the minimum (minimax) or minimizes the maximum (the opponent).

11.4 Fundamental Result

In a two-palyer zero-sum deterministic game of perfect information, one player attempting to maximize the minimum is equivalent to the other attempting to minimize the maximum, and there always exists an optimal pure strategy for each player, assuming that both players aim at maximizing their rewards—rational agents and assume the other is a rational agent.

11.5 Game Tree

The matrix representing the non-deterministic game tree is as follows, where the rows represent the strategies for player A, the columns represent the strategies for player B, and the entries are the rewards for player A and are expectation values for the random nodes

	go left	go right
go left	$0.5 \times (+4) + 0.5 \times (-20) = -8$	$0.5 \times (+4) + 0.5 \times (-20) = -8$
go right	$0.8 \times (-5) + 0.2 \times (+10) = -2$	$+3$

Once we have the matrix and the expected values, we don't care about from which non-deterministic tree that it comes. Actually it even doesn't matter whether it comes from a deterministic or a non-deterministic game tree.

11.6 von Neumann

von Neumann theorem says the above fundamental result also applies to non-deterministic games.

11.7 Minipoker

After relaxing the deterministic constraint, we now relax the perfect information constraint to consider hidden information. The matrix representing the game tree reads

	resign	request to see
resign	$0.5 \times (-20) + 0.5 \times (+10) = -5$	$0.5 \times (-20) + 0.5 \times (+30) = +5$
hold	$0.5 \times (+10) + 0.5 \times (+10) = +10$	$0.5 \times (-40) + 0.5 \times (+30) = -5$

Here the strategies of both player A and player B depend on the card assigned to player A. For example for the top left corner if player A receives a bad card (50% chance) and resigns then whether player B resigns or requests to see doesn't matter; if player A receives a good card then it would never resign and the strategy choice of player B becomes relevant.
Due to the hidden information we cannot figure out the value of the game by minimax, because now minimax is not equal to maximin thus von Neumann theorem fails.

11.8 Mixed Strategy

A mixed strategy simply means some probabilistic distribution over strategies, contrary to a pure strategy in which a player chooses either this or that strategy (either 100% or 0%).

11.9 Center Game

Given the probability P of player A holding the card, if player B resigns then the expected reward for player A is $P \times (+10) + (1 - P) \times (-5) = 15P - 5$; if player B requests to see the card then the expected reward for player A is $P \times (-5) + (1 - P) \times (+5) = -10P + 5$. These two lines meet at $P = 0.4$ at which the expected reward for player A is $+1$ regardless of how player B chooses its strategy. Now what player A is facing is not strategy choice but probability distribution over possible strategies parametrized by P, and given that player B would have exactly the same lines to decide whether to resign or request to see, the best choice of P for player A is $P = 0.4$. Because for all the other P values player B can always choose to make player A's expected reward less than 1. Therefore effectively it is again a maximin problem for player A. But now what player A should maximize is the minimum of the expected reward and the way to maximize it is to choose a probability distribution rather than a specific strategy.

11.10 Snitch

We now relax the zero-sum constraint and consider the prisoner's dilemma. The number of months penalty of cooperating and defecting for the two criminals who are prevented from communicating with each other is tabulated below

	cooperate	defect
cooperate	$(A = -1, B = -1)$	$(A = -9, B = 0)$
defect	$(A = 0, B = -9)$	$(A = -6, B = -6)$

For both criminal A and B the strategy of defecting dominates, even though the best outcome (combined sum) is to cooperate.

11.11 A Beautiful Equilibrium

Suppose there are n players and n strategy pools $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ for them to choose from. A set of strategy choices of the n players $s_1^* \in \mathcal{S}_1$, $s_2^* \in \mathcal{S}_2$, $\ldots$, $s_n^* \in \mathcal{S}_n$ are said to be in a **Nash equilibrium** if and only if given that all players know the strategy choices of all the other players, there is no reason for any one of them to change strategy for better reward. Mathematically it is expressed as

$$s_i^* = \arg\max_{s_i} \text{Utility}_i \left(s_1^*, s_2^*, \ldots, s_i, \ldots, s_n^*\right) \quad \forall i$$

This is the statement for pure strategies. But the concept applies to mixed strategies as well.
The Nash equilibrium for the prisoner's dilemma is $(-6, -6)$, which can be checked against the definition or by focusing on the intersection of the dominant strategies for the two criminals. In some other cases the Nash equilibrium is where the reward is maximal for both players simultaneously. More generally there are three fundamental theorems that assert the existence of Nash equilibrium and identify it.

- in an n-player pure strategy game, if the iterative elimination of strictly dominated strategies eliminates all but one strategy combination, then that combination is the unique Nash equilibrium
- any Nash equilibrium will survive the iterative elimination of strictly dominated strategies

- if n is finite and $\mathcal{S}_i$ is finite for any i, then there exists at least one Nash equilibrium, which may involve mixed strategies

In case there is more than one Nash equilibrium, we would not be sure which one the game will end up in. But once it is in an equilibrium it will stay there.

11.12 The Two-Step

Another fundamental theorem of Nash equilibrium states that for n repeated games the solution is n repeated Nash equilibrium. Because all the previous $n-1$ games are sunk costs and the players face the same tradeoff in the nth game as in the first game. After the choice of the nth game is determined the same analysis can be applied to the $(n-1)$th game. Induct backward this way we would end up with the same choice for all the n games.

12 Game Theory Reloaded

12.1 Iterated Prisoner's Dilemma

Even if there are two rounds of game, the prisoners essentially face a one-round game, because what they are going to do in the last round has already been determined. So it is almost as if that round doesn't really matter. This same argument applies to three, four, and many rounds. However if the number of rounds left is unknown then the conclusion is different.

12.2 Uncertain End

Suppose with probability γ game continues. Then every round could be the last round and the expected number of rounds is $1/(1-\gamma)$. Because we can treat the future games as each offering a reward of 1 and then the expected number of rounds is exactly the expected future rewards.

12.3 Tit-for-Tat

The tit-for-tat strategy consists of cooperating on the first round, and copying the opponent's previous move thereafter.

12.4 Facing TfT Quiz

Suppose the payoff for cooperate and defect is the same as before

	cooperate	defect
cooperate	$(A=-1, B=-1)$	$(A=-9, B=0)$
defect	$(A=0, B=-9)$	$(A=-6, B=-6)$

what is the best strategy if the opponent plays tit-for-tat? Consider the two simplest strategy choices (where the γ factor for the always-defect strategy is to account for the fact that the opponent defects starting from the second rather than the first round)

$$\text{always defect:} \quad 0+\gamma\frac{-6}{1-\gamma} \quad \text{best for low } \gamma$$

$$\text{always cooperate:} \quad \frac{-1}{1-\gamma} \quad \text{best for high } \gamma$$

If $\gamma > 1/6$ it is better to always cooperate.

12.5 Finite State Strategy

We have considered only two possible strategies to cope with tit-for-tat. There are many other choices. In fact by labeling states with opponent's choice and edges with our choice and edge annotations with our payoff for that choice, we can turn it into a finite MDP.

12.6 Best Responses in Iterative Prisoner's Dilemma

If the opponent always cooperates then the best strategy is to always defect; if the opponent always defects then the best strategy is to always defect; if the opponent always plays tit-for-tat then the best strategy is either to always cooperate or to always play tit-for-tat because the actions would be the same.
Among them always defect + always defect and always TfT + always TfT are two mutual best responses or Nash equilibrium.

12.7 Folk Theorem

In repeated games the possibility of retaliation opens the door for cooperation, resulting in the folk theorem below. But first we need to define some concepts.

12.8 Repeated Games Quiz

In a two-player plot the one-round game payoffs of the two players are depicted in a Cartesian coordinate. For our prisoner's dilemma the four points are at $(-9, 0)$ for X cooperates but Y defects, $(0, -9)$ for X defects but Y cooperates, $(-1, -1)$ for X and Y both cooperate, and $(-6, -6)$ for X and Y both defect. The possible payoffs that are the average payoffs of some joint strategies are contained in the convex hull established by these four points, the *feasible region*.

12.9 Minimax Profile

A minimax profile is a pair of payoffs, one for each player, that represent the payoffs achievable by a player defending itself from a malicious adversary i.e. getting the lowest score such as in a zero-sum game.
Consider the following possible payoffs

	Y chooses B	Y chooses S
X chooses B	$(1, 2)$	$(0, 0)$
X chooses S	$(0, 0)$	$(2, 1)$

Suppose X only knows Y may choose B with probability p. Then its expected payoff is $1 \times p$ if it chooses B and $2 \times (1 - p)$ if it chooses S. The minimum is reached when the two payoffs are equal (recall the center game in the last lesson) $p = 2(1 - p)$ or $p = 2/3$. This is how Y would choose randomly in order to minimize the score won by X, and the minimum score that X is guaranteed to receive despite this malicious adversary is $1 \times 2/3 = 2/3$.

12.10 Security Level Profile

For the prisoner's dilemma the minimax is located at $(-6, -6)$. The region within the convex hull that is above and to the right of this minimax is the preferable region by both players, because the payoffs there are more than the guaranteed scores assuming the other player is playing malicious adversary.

12.11 Folk Theorem

Any feasible payoff profile that strictly dominates the minimax/security level profile can be realized as a Nash equilibrium payoff profile with sufficiently large discount factor. To prove it we first construct the payoff profile, and then make it a Nash equilibrium by asserting that if it strictly dominates, then the minimax profile can use it as a threat i.e. the player would be better off by doing what it is told to do, otherwise it can be forced down to its minimax by the opponent.

12.12 Grim Trigger

We can construct a Nash equilibrium through a grim trigger, which consists of starting off with cooperation to achieve mutual benefit, but switching to retaliation forever if the opponent crosses the line by failing to cooperate.

12.13 Implausible Threats

Implausible threats occur when the retaliation sworn by the revenger hurts the revenger more so that it doesn't make sense for the revenger to carry out the retaliation. To formally define implausible threats we introduce the notion of **subgame perfect**—a plausible threat corresponds to something that is called a **subgame perfect equilibrium**. Subgame perfect means that each player is always taking a best response independent of the history. Two strategies are not in a subgame perfect equilibrium with each other if there are some history of actions that we could feed to them so that one or both of them can change its behavior to do better. If their responses are the best for all possible histories then they are in a subgame perfect equilibrium. Two strategies that are in Nash equilibrium may not be in subgame perfect equilibrium.

12.14 TfT vs. TfT Quiz

TfT vs. TfT is not subgame perfect. Because if one of them defects, the other should still cooperate rather than copying the opponent in order to maximize long-run expected reward.

12.15 Pavlov

In contrast with TfT which repeats what the opponent did, Pavlov cooperates if both agree (either both cooperates or both defects) but defects if there is disagreement (one cooperates while the other defects).

12.16 Pavlov vs. Pavlov Quiz

Pavlov vs. Pavlov are in Nash equilibrium.

12.17 Pavlov is Subgame Perfect

There are only four possible histories: CC, DC, CD, DD (C = cooperate, D = defect). Applying Pavlov's rule (cooperate if agree, defect if disagree) we have CC $\rightarrow$ CC, DD $\rightarrow$ CC, DC $\rightarrow$ DD $\rightarrow$ CC, CD $\rightarrow$ DD $\rightarrow$ CC. Thus no matter what history is fed into, Pavlov vs. Pavlov would be able to resynchronize and return to the mutually cooperative state. It shows the significance of being subgame perfect, which is that it is worth defecting/punishing because it is not going to cost anything in the long run and it helps stabilize the behavior in the short run.

12.18 Computational Folk Theorem

One can build Pavlov-like machines for any bi-matrix (if not zero-sum then need two matrices) average-reward repeated game to construct subgame perfect Nash equilibrium in polynomial time. And there are three possible forms of Nash equilibrium—if it is possible to construct a Pavlov then its long-run mutual cooperative state is the Nash equilibrium; if not then mutual cooperation is impossible and we have a zero-sum-like game for which the Nash equilibrium can be solved via linear programming; if not solvable then there is at most one player that can improve its behavior and by taking that best response against what the other players do in a zero-sum-like game, then that would be Nash equilibrium.

12.19 Stochastic Games and Multiagent RL

Stochastic games is a generalization of both MDPs and repeated games. It gives a formal model for multi-agent reinforcement learning analogous to MDPs for single-agent reinforcement learning. In this context the strategy is a multi-step policy. As such Nash equilibrium in this setting is defined as a pair of policies so that neither would prefer to switch.

12.20 Stochastic Games

A stochastic game consists of

state S: $s \in S$ depicts the states of all players

actions A_i for player i: $a \in A_1$ and $b \in A_2$ for a two-player game

transitions T: $T(s,(a,b),s')$ depends on current state s, next state s', and joint action (a,b)

rewards R_i for player i: $R_1(s,(a,b))$ and $R_2(s,(a,b))$ both depend on joint action (a,b)

discount factor γ: usually the same for all players

12.21 Models & Stochastic Games

Consider putting the following constraint to a stochastic game $\langle S, A_i, T, R_i, \gamma \rangle$

- $R_1 = -R_2$ gives a zero-sum stochastic game
- $T(s,(a,b),s') = T(s,(a,b'),s')\ \forall b,b'$, $R_1(s,(a,b)) = R_1(s,(a,b'))\ \forall b,b'$ gives a MDP ($R_2(s,(a,b))$ = constant or $R_2 = R_1$)
- $|S| = 1$ gives a repeated game, because although the action impacts the reward it does not affect the transition

12.22 Zero-Sum Stochastic Games

The Bellman equations for zero-sum stochastic games read

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

and the corresponding Q-value update is

$$Q_i(s,(a,b)) \xleftarrow{\alpha} r_i + \gamma \text{minimax}_{a',b'} Q_i(s',(a',b'))$$

The Q-function is sometimes called minimax-Q because of the minimax operation involved.
The properties enjoyed by the above minimax-Q Bellman equations include

- value iteration works
- minimax-Q converges in the same conditions that Q-learning converges
- there is a unique solution Q^* to the minimax-Q Bellman equations
- the optimal policies of the two players can be computed independently, and they will converge to the policies determined by the minimax-Q solution
- the minimax-Q solution Q^* is sufficient to specify the optimal policies of the two players
- the minimax-Q update can be computed efficiently i.e. in polynomial time, because the minimax operation can be computed using linear programming

although it is not true as in the MDP case that we can solve the Bellman equations in polynomial time.

12.23 General-Sum Games

For a general-sum game, we need to replace the minimax operation for zero-sum games with the Nash equilibrium operation, and thus the minimax-Q function becomes the Nash-Q function. We still have the Nash version of the Bellman equations similar to the above, but we lost all the nice properties of the Bellman equations for the minimax-Q function

- value iteration no longer works
- Nash-Q does not converge in the same conditions that Q-learning converges
- there is no unique solution Q^* to the Nash-Q Bellman equations, because there might be more than one Nash equilibrium
- the optimal policies of the two players cannot be computed independently, because Nash equilibrium refers to joint policies and halves of two Nash equilibria do not constitute a Nash equilibrium
- the Nash-Q solution Q^* is not sufficient to specify the optimal policies of the two players
- the Nash-Q update cannot be computed efficiently i.e. in polynomial time, because Nash equilibrium cannot be computed in polynomial time

12.24 Lots of Ideas

Although the general-sum stochastic games haven't been solved, there are lots of ideas that approach a viable solution to it, such as

- repeated stochastic games allows building folk theorem-like ideas at the level of stochastic games
- using communication side-channel (cheap talk, non-binding) allows the computation of correlated equilibrium, which is more efficient to compute than Nash equilibrium and offers a near-optimal solution to the general-sum game
- cognitive hierarchy—instead of solving for an equilibrium assume the other players are constrained by computational resources and taking the best response under these constraints, which is easier to compute because the other players are close to being fixed
- side payments—willing to share some reward with other players for cooperation

12.25 What Have We Learned

We connect iterative prisoner's dilemma (IPD) with reinforcement learning via discounting. The idea behind the connection is that we can encourage cooperation through repeated games via plausible threats, which brings in a whole set of Nash equilibria as dictated by the folk theorem.

13 Game Theory Revolutions

13.1 Solution Concepts

A solution concept is a rule that predicts how a game is going to be played. An equilibrium is a solution in which it is not in individual's best interest to change its behavior.

13.2 General Tso Chicken

The payoffs of the game are

	dare	chicken
dare	(0,0)	(7,2)
chicken	(2,7)	(6,6)

There are two pure Nash equilibria: (2,7) and (7,2). However they are unstable in the sense that both players would prefer dare and persuade the opponent into chickening i.e. settling down in which of the two Nash equilibria is undecided. The usual way to get out of this is to consider mixed strategies and find mixed Nash equilibrium. For this game the mixed Nash equilibrium is both players choosing to dare with a probability of 1/3, so that the expected payoff for each player is $6 \times (4/9) + 2 \times (2/9) + 7 \times (2/9) + 0 \times (1/9) = 14/3$, better than sharing the payoffs in pure Nash equilibria $(2 + 7)/2 = 9/2$.

13.3 Correlated GTC

Introduce a moderator, who is going to choose uniformly randomly among (C,C), (D,C), and (C,D) where C = chicken and D = dare, and advise both players to choose according to but without disclosing the random selection. No matter what option the moderate advises, the player should follow because

- if the player is advised to choose D, then it can be inferred that the other player is advised to choose C because (D,D) is not in the moderator's pool, and in this case the player should choose D as advised because the reward would be lower, 6 instead of 7, if it chooses not to follow the moderator's advice
- if the player is advised to choose C, then there are 50% chances that the other player is advised to choose C and 50% chances that the other player is advised to choose D, thus the expected payoff for following the moderator's advice is $6 \times (1/2) + 2 \times (1/2) = 4$, better than choosing not to follow since the expected payoff for disobedience is $7 \times (1/2) + 0 \times (1/2) = 7/2$

Following the moderator's advice without incentive to change strategy is called **correlated equilibrium**. The expected payoff for following the moderator's advice is $4 \times (2/3) + 7 \times (1/3) = 5$ which is better than 14/3 of the mixed Nash equilibrium. Note that it is worse than (C,C) where both players get 6 which however is not a equilibrium.
This game actually happens everyday in our life whenever we drive to the intersection, where the traffic light plays the role of the moderator, green light represents dare, and red light represents chicken. This also naturally explains why (D,D) is not in the pool of selection.

13.4 Correlated Facts

There are three

- correlated equilibria can be found in polynomial time
- all mixed Nash equilibria are correlated, thus correlated equilibria do exist
- all convex combinations of mixed Nash equilibria are correlated

13.5 Coco Definition

Coco stands for cooperative-competitive. Let U denote the payoffs to you/the player of interest and $\bar{U}$ to the other. U and $\bar{U}$ together define a general-sum game for which a Nash-like equilibrium is not easy to find. The coco value of this game is defined as

$$\text{coco}(U, \bar{U}) = \underbrace{\max\max\left(\frac{U+\bar{U}}{2}\right)}_{\text{cooperative payoffs}} + \underbrace{\text{mini}\max\left(\frac{U-\bar{U}}{2}\right)}_{\text{competitive payoffs}}$$

The cooperative equilibrium can be solved by taking the maximum cell of the payoff matrix, while the competitive game is a zero-sum game and its equilibrium can be solved by linear programming. Both can be solved in polynomial time.

13.6 Coco Example

For the banana game we have

$$U = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \qquad \bar{U} = \begin{pmatrix} 2 & 0 \\ 2 & 4 \end{pmatrix}$$

$$\implies \underbrace{\frac{U+\bar{U}}{2} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}}_{\max\max} \quad \underbrace{\frac{U-\bar{U}}{2} = \begin{pmatrix} -1 & 0 \\ -1 & -2 \end{pmatrix}}_{\text{mini}\max} \quad \underbrace{\frac{\bar{U}-U}{2} = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}}_{\text{maxi}\min}$$

The $\max\max$ value is 2 while the $\text{mini}\max$ value is -1 and the $\text{maxi}\min$ value is 1. Thus from U's perspective the coco value is 1 while from $\bar{U}$'s perspective it is 3, and the side payments are

$$P = \text{coco}(U, \bar{U}) - \max\max U = +1$$
$$\bar{P} = \text{coco}(U, \bar{U}) - \max\max \bar{U} = -1$$

Hence it is $\bar{U}$ who pays 1 to U as side payment.

13.7 Coco Properties

The properties are

- the coco value can be efficiently computable, because it decomposes the game into a sum of two games which involves simple maximization and linear programming
- the solution is unique and utility maximizing for the players (which is then split between them)
- the solution may not necessarily be the equilibrium, there exists better response unless we make the side payments binding
- it can be extended to stochastic games using coco-Q value, which converges despite being non-expansive
- it cannot be extended to three or more players

13.8 Mechanism Design

Mechanism design is creating games to elicit certain kind of behavior (behavior $\rightarrow$ game). Once the design is completed we play the game and get the expected behavior (game $\rightarrow$ behavior).

13.9 Peer Teaching

For simplicity assume that all the assessment questions can be unambiguously sorted in terms of their levels of difficulty. The student should be incentivized to learn through receiving credit for answering correctly any assigned question. The teacher should be incentivized to assess the student's level of understanding correctly manifested by identifying the center line of a bell-shaped distribution along the spectrum of the assessment question difficulty through receiving credit for the student answering wrongly the questions that lie to the left of the center line (questions the student is expected to answer correctly) and answering correctly the questions that lie to the right of the center line (questions the student is expected to answer wrongly).

13.10 King Solomon

Let V_{real} be the value of the baby to the real mother and V_{fake} the value of the baby to the fake mother. Suppose $V_{\text{real}} > V_{\text{fake}}$. The following procedure would force the fake mother to confess the truth no matter whether it is A or B without threatening to cut the baby into half, which in fact defines a subgame perfect equilibrium.

0. choose a fine F that is smaller than V_{fake}
1. ask A whether it is her baby, proceed if her answer is yes, give the baby to B if her answer is no
2. ask B whether it is her baby, proceed if her answer is yes, give the baby to A if her answer is no
3. force A to pay the fine F, and let B announce a value $V > F$ that is required to get the baby
4. ask A again whether it is her baby, if her answer is yes then A pays V to get the baby and B has to pay the fine F, if her answer is no then B pays V to get the baby

If B is the fake mother, then B would announce $V = V_{\text{fake}}$. Since $V_{\text{fake}} < V_{\text{real}}$ A would answer yes and pay V to get the baby with B ending up paying the fine. Hence B would confess the truth in step 2. If A is the fake mother, the B would announce $V = V_{\text{real}}$. Since $V_{\text{fake}} < V_{\text{real}}$ A would not answer yes and pay V to get the baby. Hence A would confess the truth in step 1.

13.11 What Have We Learned

It is the shared source of randomization that makes correlated equilibrium works, because it allows players to correlate possible outcomes. Coco value makes a stronger assumption of the existence of binding side payments, which incentivizes the players to behave in such a way to maximize the total reward. In fact all these solution concepts can be treated as mechanism designs.

14 CCC

14.1 CCC

CCC stands for coordinating, communicating, and coaching.

14.2 DEC-POMDP

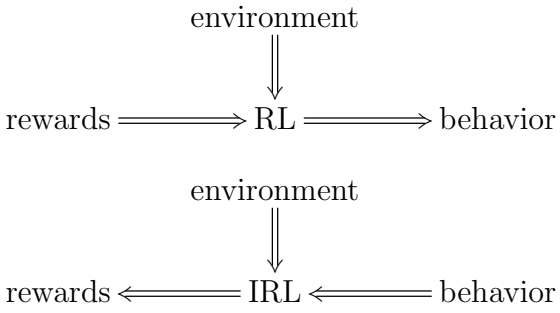
DEC-POMDP, which stands for decentralized partially observable Markov decision process, is defined by the following quantities

- I = finite set of agents
- S = states
- A_i = agent i's actions
- T = joint transition function
- R = shared reward function
- Z_i = agent i's observations
- O = observation function

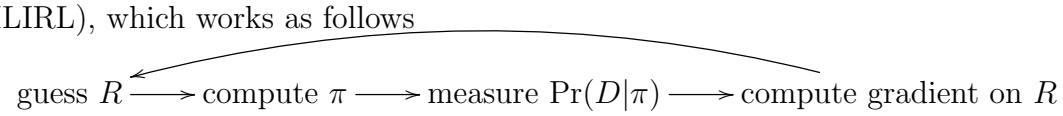
It contains elements of both game theory and POMDPs. DEC-POMDPs are not computable since POMDPs are not. But in finite horizon POMDPs are P-space complete while DEC-POMDPs are known to be non-deterministic exponential time complete or NEXT-complete i.e. solvable between single exponential time and double exponential time. Despite this computational challenge DEC-POMDPs are useful because just like POMDPs allow actions to gain information DEC-POMDPs allow actions to communicate.

14.3 Inverse Reinforcement Learning

The following diagram contrasts reinforcement learning and inverse reinforcement learning



One algorithm for solving inverse reinforcement learning is maximum likelihood inverse reinforcement learning (MLIRL), which works as follows



where D is the data of the behavior trajectory we saw given the policy.

14.4 Policy Shaping

For the general case in which multiple optimal policies are allowed, the probability of an action a being optimal given the sequence of human judgments d_a along the trajectory for action a—the human says 'yes' and 'no' to a being an optimal action, is given by

$$\Pr(a|d_a) = \frac{C^{\Delta_a}}{C^{\Delta_a} + (1-C)^{\Delta_a}}$$

where C is the probability that the human's judgment is correct, and Δ_a is the number of times the human says 'yes' to a being an optimal action minus the number of times the human says 'no' to a being an optimal action, which is the sufficient statistic that summarizes the entire sequence of the 'yes' and 'no' human judgments. This probability formula follows from the binomial distribution, because knowing one action being optimal does not convey any information about the other actions since there can be multiple optimal actions.
When there can be only one policy, the probability of an action a being optimal given the data d_a along the trajectory of action a becomes

$$\Pr(a|d_a) \propto C^{\Delta_a} \cdot (1-C)^{\sum_{j \neq a} \Delta_j}$$

where the complicated normalization factor is omitted, because we only care about which action gives the maximal probability (the optimal action) not the actual probability. It is no longer binomial but multinomial because we can infer from a being the optimal action that the other actions are not optimal since there can be only one optimal action.
Note that in both cases we simplify the probabilities C and $1-C$ of the human judgment. In general there should be four probabilities, namely true positive C, true negative $1-C$, false positive C', and false negative $1-C'$. Also notice that C is reflected from the probability distribution in the sense that the smaller C is the more uniform the probability distribution is, because the less certain which action is optimal the less confident. Moreover as d becomes longer and longer C will rise towards 1, and the probability distribution will peak with probability close to 1 at the optimal action, because the more experience a judge has the more confident it is about which action is optimal.

14.5 Multiple Sources

Given two generative models from two sources that provide two sets of likelihoods about which action is the optimal, the action we should choose is the one that maximizes the product of these two probability distributions

$$a_{opt} = \arg\max_a \Pr(a|d_1)\Pr(a|d_2)$$

In fact what this probability product captures is the probability that the judgments of the two sources would agree. Again as d_1 and d_2 become longer the two probability distributions will converge to each other, because both judges will become more certain about which action is optimal and their judgments will be more likely to be correct.

14.6 Drama Management

We've discussed three ways a human player can communicate to an agent

demonstration: give a behavioral trajectory and do inverse reinforcement learning

reward: give rewards to current action, reward shaping

policy: give the action decision for current state, policy shaping

In all three cases there are two entities—human player and agent. Now we introduce another human—author. The goal of the author is to create an experience for the player that is consistent with the author's intent. The author never communicates to the player, but can only communicate to the agent. It can express its intent through statements like the player can pass a level only after it failed for a certain number of times and learned from the failures. The play designed by the author can be represented by a trajectory of plot points, and there are good and bad trajectories according to the author's intent. A good trajectory makes the player experience the story that the author intends to deliver to the player.

14.7 Trajectories as MDPs

We can turn this drama management into a MDP as follows

state: partial sequences of plot points

action: story action the author and the system can make (not the player action)

model: player model

reward: author evaluation

But there are two problems associated with this formulation:

- because the states are partial sequences of plot points, the number of states is hyperexponential in the sense that the set of partial sequences is the power set of $n!$ where n is the number of plot points thus the number of partial sequences is in the order of 2^{2^n}
- since the reward is for the author not the player, the player may not get its desired experience but is forced to experience the story that the author designs for it

It turns out that we can relax the need to find the best story for the player and solve the hyperexponential problem at the same time by defining the following TTDMDP where TTD stands for targeted trajectory distribution:

trajectory: the partial plot sequence experienced so far

action: story action the author and the system can make (not the player action)

model: transition function $\Pr(t'|a,t)$ i.e. the probability of going to trajectory t' with action a taken from trajectory t, the uncertainty comes from the player action

target distribution: $\Pr(T)$ where T stands for final trajectory i.e. the complete story, the probability is higher for desired stories

Replacing the goal of designing the best story with the target of reaching the desired probability distribution of stories results in a probabilistic policy—a map from a state to a probability distribution of actions rather than an action, and the optimal policy is the one that leads to matching the target distribution of final trajectories. By replacing the hard constraint of designing the best story with the soft constraint of reaching the desired probability distribution of stories we can solve TTDMDP in linear time, linear in the length of final trajectories. This linear claim is plausible because from the tree search perspective once we know which node we are in, we don't need to care about the branches of the tree that do not involve the current node. The hard part is how to derive the targeted distribution of the intermediate nodes from the targeted distribution of final trajectories, which relies on the use of KL divergence to match two probability distributions.