Joint Decision Space

Markov Decision Process (MDP) Representation:

- **State space:**
  - Joint state $x$ of entire system

- **Action space:**
  - Joint action $a = \{a_1, \ldots, a_n\}$ for all agents

- **Reward function:**
  - Total reward $R(x,a)$
    - sometimes reward can depend on action

- **Transition model:**
  - Dynamics of the entire system $P(x'|x,a)$
Policy

Policy: $\pi(x) = a$

At state $x$, action $a$ for all agents

$\pi(x_0) = \text{both peasants get wood}$

$\pi(x_1) = \text{one peasant builds barrack, other gets gold}$

$\pi(x_2) = \text{peasants get gold, footmen attack}$
Computing the value of a policy

Discounted value of a state:
- value of starting from $x_0$ and continuing with policy $\pi$ from then on

$$V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \ldots]$$

A recursion!

$$V_\pi(x_0) = E_\pi\left[ \sum_{t=0}^{\infty} \gamma^t R(x_t) \right] = E_\pi\left[ R(x_0) + \sum_{t=1}^{\infty} \gamma^{t-1} R(x_t) \right]$$

$$= E_\pi[R(x_0)] + \gamma E_\pi\left[ \sum_{t=1}^{\infty} \gamma^{t-1} R(x_t) \right]$$

$$= R(x_0) + \gamma E_{x_1}\left[ V_\pi(x_1) \right]$$

$$V_\pi \leq \text{simple matrix} \ \mathbb{Q}$$

©2005-2007 Carlos Guestrin
Solving an MDP

Solve Bellman equation

Optimal value \( V^*(x) \)

Optimal policy \( \pi^*(x) \)

\[
V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x'| x, a) V^*(x')
\]

Bellman equation is non-linear!!!

Many algorithms solve the Bellman equations:

- Policy iteration [Howard ‘60, Bellman ‘57]
- Value iteration [Bellman ‘57]
- Linear programming [Manne ‘60]
- …
Value iteration (a.k.a. dynamic programming) – the simplest of all

\[ V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a)V^*(x') \]

- Start with some guess \( V_0 = R \)
- Iteratively say:
  \[ V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a)V_t(x') \]
- Stop when \( \|V_{t+1} - V_t\|_\infty \leq \varepsilon \)
  \( \square \) means that \( \|V^* - V_{t+1}\|_\infty \leq \varepsilon/(1-\gamma) \)

\[ \square \] Greedy Policy \( \Pi_{t+1} \)

\[ \|V^* - V_{\Pi_{t+1}}\|_\infty \leq \frac{\varepsilon \gamma}{(1-\gamma)^2} \]

Converges to \( V^* \)

Can get \( \Pi^* \)
Policy iteration – Another approach for computing $\pi^*$

- Start with some guess for a policy $\pi_0$.
- Iteratively say:
  - **evaluate policy:**
    $$V_t(x) = R(x, a = \pi_t(x)) + \gamma \sum_{x'} P(x' \mid x, a = \pi_t(x)) V_t(x')$$
  - **improve policy:**
    $$V_{t+1} = (I - \gamma P_\pi)^{-1} R$$
    $$\pi_{t+1}(x) = \max_{a} R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) V_t(x')$$

- Stop when
  - **policy stops changing**
    - usually happens in about 10 iterations
  - or $\|V_{t+1} - V_t\|_\infty \leq \varepsilon$
    - means that $\|V^* - V_{t+1}\|_\infty \leq \varepsilon/(1-\gamma)$

Open problem: PI converges in polynomial time?
Policy Iteration & Value Iteration: Which is best ???

It depends.
- Lots of actions? Choose **Policy Iteration**
- Already got a fair policy? **Policy Iteration**
- Few actions, acyclic? **Value Iteration**

**Best of Both Worlds:**
- **Modified Policy Iteration** [Puterman]
  ...a simple mix of value iteration and policy iteration

**3rd Approach**
- **Linear Programming**
LP Solution to MDP

Value computed by linear programming:

\[
\text{minimize: } \sum_{x} V(x) \\
\text{subject to: } \begin{cases} 
V(x) \geq R(x,a) + \gamma \sum_{x'} P(x'|x,a) V(x') \\
\forall x, a
\end{cases}
\]

- One variable \( V(x) \) for each state
- One constraint for each state \( x \) and action \( a \)
- Polynomial time solution

[Manne ‘60]
What you need to know

- What’s a Markov decision process
  - state, actions, transitions, rewards
  - a policy
  - value function for a policy
    - computing $V_\pi$
- Optimal value function and optimal policy
  - Bellman equation
- Solving Bellman equation
  - with value iteration, policy iteration and linear programming
The Reinforcement Learning task

\[ \text{learn } \rightarrow P(x'|x,a) \]

**World:** You are in state 34.
Your immediate reward is 3. You have possible 3 actions.

\[ (x_1 = 34, r_1 = 3, a_1 = 2, x_2 = 77) \]

**Robot:** I’ll take action 2.

**World:** You are in state 77.
Your immediate reward is -7. You have possible 2 actions.

**Robot:** I’ll take action 1.

**World:** You’re in state 34 (again).
Your immediate reward is 3. You have possible 3 actions.
Formalizing the (online) reinforcement learning problem

- Given a set of states $X$ and actions $A$
  - in some versions of the problem size of $X$ and $A$ unknown

- Interact with world at each time step $t$
  - world gives state $x_t$ and reward $r_t$
  - you give next action $a_t$
    - get next state $x_{t+1}$

- **Goal**: (quickly) learn policy that (approximately) maximizes long-term expected discounted reward
The “Credit Assignment” Problem

I’m in state 43,

reward = 0, action = 2

= 0, = 4

= 0, = 1

= 0, = 1

= 0, = 1

= 0, = 2

= 0, = 2

= 100,

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

This is the Credit Assignment problem.
Exploration-Exploitation tradeoff

You have visited part of the state space and found a reward of 100
- is this the best I can hope for???

- **Exploitation**: should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at the risk of missing out on some large reward somewhere

- **Exploration**: should I look for a region with more reward?
  - at the risk of wasting my time or collecting a lot of negative reward
Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment → learn model \( P(x'|x,a) \) and \( R(x,a) \) (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - works quite well in practice when state space is manageable

- Model-free approach:
  - don’t learn a model → learn value function or policy directly
  - leads to weaker theoretical results
  - often works well when state space is large
Rmax – A model-based approach
Given a dataset – learn model

Given data, learn (MDP) Representation:

- Dataset: \( \langle x_1, a_1, r_1, x_2 \rangle \)
  \( \langle x_2, a_2, r_2, x_3 \rangle \)

- Learn reward function:
  - \( R(x,a) \)
    - if I visit state \( x_i, a_i \) and get \( r_i \)
  - \( R(x_i, a_i) \in r_i \)

- Learn transition model:
  - \( P(x'|x,a) \)
    - \( \frac{\text{Count}(x'=i, x=j, a=k)}{\text{Count}(x'=?, x=i, a=k)} \)
    - same as HMMs
Some challenges in model-based RL 1: Planning with insufficient information

- Model-based approach:
  - estimate \( R(x,a) \) & \( P(x'|x,a) \)
  - obtain policy by value or policy iteration, or linear programming
  - No credit assignment problem → learning model, planning algorithm takes care of “assigning” credit

- What do you plug in when you don’t have enough information about a state?
  - don’t reward at a particular state
    - plug in smallest reward (\( R_{\text{min}} \))?  
    - plug in largest reward (\( R_{\text{max}} \))?
      - plug in average reward
  - don’t know a particular transition probability?
Some challenges in model-based RL 2: Exploration-Exploitation tradeoff

- A state may be very hard to reach
  - waste a lot of time trying to learn rewards and transitions for this state
  - after a much effort, state may be useless

- A strong advantage of a model-based approach:
  - you know which states estimate for rewards and transitions are bad
  - can (try) to plan to reach these states
  - have a good estimate of how long it takes to get there
A surprisingly simple approach for model based RL – The Rmax algorithm [Brafman & Tennenholtz]

- **Optimism in the face of uncertainty!!!**
  - heuristic shown to be useful long before theory was done (e.g., Kaelbling '90)

- If you don’t know reward for a particular state-action pair, set it to $R_{\text{max}}$!!

- If you don’t know the transition probabilities $P(x'|x,a)$ from some state-action pair $x,a$ assume you go to a magic, fairytale new state $x_0$!!
  - $R(x_0,a) = R_{\text{max}}$
  - $P(x_0|x_0,a) = 1$
Understanding $R_{\text{max}}$

With $R_{\text{max}}$ you either:

- **explore** – visit a state-action pair you don’t know much about
  - because it seems to have lots of potential
- **exploit** – spend all your time on known states
  - even if unknown states were amazingly good, it’s not worth it

Note: you never know if you are exploring or exploiting!!!
Implicit Exploration-Exploitation Lemma

- **Lemma**: every $T$ time steps, either:
  - **Exploits**: achieves near-optimal reward for these $T$-steps, or
  - **Explores**: with high probability, the agent visits an unknown state-action pair
    - learns a little about an unknown state
  - $T$ is related to mixing time of Markov chain defined by MDP
    - time it takes to (approximately) forget where you started
The Rmax algorithm

**Initialization:**
- Add state $x_0$ to MDP
- $R(x,a) = R_{\text{max}}, \forall x,a$
- $P(x_0|x,a) = 1, \forall x,a$
- all states (except for $x_0$) are unknown

**Repeat**
- obtain policy for current MDP and Execute policy
- for any visited state-action pair, set reward function to appropriate value
- if visited some state-action pair $x,a$ enough times to estimate $P(x'|x,a)$
  - update transition probs. $P(x'|x,a)$ for $x,a$ using MLE
  - recompute policy
Visit enough times to estimate $P(x'|x,a)$?

- How many times are enough?
  - use Chernoff Bound!

- Chernoff Bound:
  - $X_1,\ldots,X_n$ are i.i.d. Bernoulli trials with prob. $\theta$
  - $P(|1/n \sum_i X_i - \theta| > \varepsilon) \leq \exp\{-2n\varepsilon^2\}$
Putting it all together

**Theorem**: With prob. at least $1-\delta$, Rmax will reach an $\varepsilon$-optimal policy in time polynomial in: num. states, num. actions, $T$, $1/\varepsilon$, $1/\delta$

- Every $T$ steps:
  - achieve near optimal reward (great!), or
  - visit an unknown state-action pair → num. states and actions is finite, so can’t take too long before all states are known
Problems with model-based approach

- If state space is large
  - transition matrix is very large!
  - requires many visits to declare a state as known

- Hard to do “approximate” learning with large state spaces
  - some options exist, though
TD-Learning and Q-learning — Model-free approaches
Value of Policy

Value: $V_\pi(x)$

Expected long-term reward starting from $x$

$$V_\pi(x_0) = E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + \ldots]$$

Future rewards discounted by $\gamma \in [0,1)$
A simple monte-carlo policy evaluation

- Estimate $V_\pi(x)$, start several trajectories from $x$
  $\rightarrow V_\pi(x)$ is average reward from these trajectories
  - Hoeffding’s inequality tells you how many you need
  - discounted reward $\rightarrow$ don’t have to run each trajectory forever to get reward estimate
Problems with monte-carlo approach

- ** Resets**: assumes you can restart process from same state many times

- **Wasteful**: same trajectory can be used to estimate many states

  \[
  \text{also good to estimate } V(x_i) \]

\[ V(x_1) \]
\[ V(x_2) \]
Reusing trajectories

- Value determination:

\[
V_\pi(x) = R(x) + \gamma \sum_{x', a = \pi(x)} P(x' | x, a) V_\pi(x')
\]

- Expressed as an expectation over next states:

\[
V_\pi(x) = R(x) + \gamma E \left[ V_\pi(x') \mid x, a = \pi(x) \right]
\]

- Initialize value function (zeros, at random, …)

- Idea 1: Observe a transition: \(x_t \rightarrow x_{t+1}, r_{t+1}\), approximate expec. with single sample:

\[
V_\pi(x_t) = r_{t+1} + \gamma V_\pi(x_{t+1})
\]

- unbiased!!
- but a very bad estimate!!
Simple fix: Temporal Difference (TD) Learning [Sutton '84]

\[ V_\pi(x) = R(x) + \gamma E[V_\pi(x') | x, a = \pi(x)] \]

Idea 2: Observe a transition: \( x_t \rightarrow x_{t+1}, r_{t+1} \), approximate expectation by mixture of new sample with old estimate:

\[ V_\pi(x_t) \leftarrow \alpha \left[ r_{t+1} + \gamma V_\pi(x_{t+1}) \right] + (1 - \alpha) V_\pi(x_t) \]

\( \alpha > 0 \) is learning rate
TD converges (can take a long time!!!)

\[ V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V_\pi(x') \]

**Theorem:** TD converges in the limit (with prob. 1), if:

- every state is visited infinitely often
- Learning rate decays just so:
  - \( \sum_{i=1}^{\infty} \alpha_i = \infty \)
  - \( \sum_{i=1}^{\infty} \alpha_i^2 < \infty \)
Using TD for Control

- TD converges to value of current policy $\pi_t$
  \[ V_t(x) = R(x,a = \pi_t(x)) + \gamma \sum_{x'} P(x'|x,a = \pi_t(x))V_t(x') \]

- Policy improvement:
  \[ \pi_{t+1}(x) = \max_a R(x,a) + \gamma \sum_{x'} P(x'|x,a)V_t(x') \]

- TD for control:
  - run $T$ steps of TD
  - compute a policy improvement step
Problems with TD

- How can we do the policy improvement step if we don’t have the model?

\[ \pi_{t+1}(x) = \max_{a} R(x, a) + \gamma \sum_{x'} P(x' | x, a)V_t(x') \]

- TD is an **on-policy** approach: execute policy \( \pi_t \) trying to learn \( V_t \)
  - must visit all states infinitely often
  - What if policy doesn’t visit some states???
Another model-free RL approach: Q-learning [Watkins & Dayan '92]

- Simple modification to TD

- Learns optimal value function (and policy), not just value of fixed policy

- Solution (almost) independent of policy you execute!
Recall Value Iteration

- Value iteration:
  \[ V_{t+1}(x) = \max_{a} R(x, a) + \gamma \sum_{x'} P(x'|x, a)V_t(x') \]

- Or:
  \[ Q_{t+1}(x, a) = R(x, a) + \gamma \sum_{x'} P(x'|x, a)V_t(x') \]

\[ V_{t+1}(x) = \max_{a} Q_{t+1}(x, a) \]

If I have Q, \[ \Pi_{t+1}(x) = \arg \max_{a} Q_{t+1}(x, a) \]

- Writing in terms of Q-function:
  \[ Q_{t+1}(x, a) = R(x, a) + \gamma \sum_{x'} P(x'|x, a) \max_{a'} Q_t(x', a') \]
  Equivalent to V.I.
Q-learning

Q_{t+1}(x, a) = R(x, a) + \gamma \sum_{x'} P(x' | x, a) \max_{a'} Q_t(x', a')

- Observe a transition: x_t, a_t \rightarrow x_{t+1}, r_{t+1}, approximate expectation by mixture of new sample with old estimate:
  - transition now from state-action pair to next state and reward
    \[ Q(x_t, a_t) \leftarrow (1-\alpha) \quad Q(x_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(x_{t+1}, a)] \]
  - \( \alpha > 0 \) is learning rate

©2005-2007 Carlos Guestrin
Q-learning convergence

- Under same conditions as TD, Q-learning converges to optimal value function $Q^*$
- Can run any policy, as long as policy visits every state-action pair infinitely often
- Typical policies (non of these address Exploration-Exploitation tradeoff)
  - $\epsilon$-greedy:
    - with prob. $(1-\epsilon)$ take greedy action:
    - with prob. $\epsilon$ take an action at (uniformly) random
  - Boltzmann (softmax) policy:
    - $a_t = \arg \max_a Q_t(x, a)$
    - $K$ – “temperature” parameter, $K \to 0$, as $t \to \infty$
    - $P(a_t | x) \propto \exp\left\{ \frac{Q_t(x, a)}{K} \right\}$
The curse of dimensionality:
A significant challenge in MDPs and RL

- MDPs and RL are polynomial in number of states and actions
- Consider a game with $n$ units (e.g., peasants, footmen, etc.)
  - How many states? $k^n$
  - How many actions? $3^n$

- Complexity is exponential in the number of variables used to define state!!!
Addressing the curse!

Some solutions for the curse of dimensionality:

- **Learning the value function**: mapping from state-action pairs to values (real numbers)
  - A regression problem!
- **Learning a policy**: mapping from states to actions
  - A classification problem!

Use many of the ideas you learned this semester:

- linear regression, SVMs, decision trees, neural networks, Bayes nets, etc.!!!

TD gammon: TD + approx V.F. using neuralnet.
What you need to know about RL

A model-based approach:
- address exploration-exploitation tradeoff and credit assignment problem
- the R-max algorithm

A model-free approach:
- never needs to learn transition model and reward function
- TD-learning
- Q-learning
Closing…..

Machine Learning – 10701/15781
Carlos Guestrin
Carnegie Mellon University

May 2\textsuperscript{nd}, 2007
Announcements

- **Project:**
  - Poster session: Friday May 4\textsuperscript{th} 2-5pm, NSH Atrium
    - please arrive a 15mins early to set up
  - Paper: Thursday May 10\textsuperscript{th} by 2pm
    - electronic submission by email to instructors list
    - maximum of 8 pages, NIPS format
    - no late days allowed

- **FCEs!!!!**
  - Please, please, please, please, please, please, please give us your feedback, it helps us improve the class! 😊
    - [http://www.cmu.edu/fce](http://www.cmu.edu/fce)

Recitation tomorrow about RL

Next week review session TBS,
What you have learned this semester

- Learning is function approximation
- Point estimation
- Regression
- Discriminative v. Generative learning
- Naive Bayes
- Logistic regression
- Bias-Variance tradeoff
- Neural nets
- Decision trees
- Cross validation
- Boosting
- Instance-based learning
- SVMs
- Kernel trick
- PAC learning
- VC dimension
- Margin bounds
- Bayes nets
  - representation, inference, parameter and structure learning
- HMMs
  - representation, inference, learning
- K-means
- EM
- Semi-supervised learning
- Feature selection, dimensionality reduction, PCA
- MDPs
- Reinforcement learning
**BIG PICTURE**

- Improving the performance at some task though experience!!! 😊

  - before you start any learning task, remember the fundamental questions:

<table>
<thead>
<tr>
<th>What is the learning problem?</th>
<th>From what experience?</th>
<th>What model?</th>
</tr>
</thead>
<tbody>
<tr>
<td>What loss function are you optimizing?</td>
<td>With what optimization algorithm?</td>
<td></td>
</tr>
<tr>
<td>Which learning algorithm?</td>
<td>With what guarantees?</td>
<td>How will you evaluate it?</td>
</tr>
</tbody>
</table>
What next?

- Intelligence Seminars: http://www.cs.cmu.edu/~iseminar/

- Journal:
  - JMLR – Journal of Machine Learning Research (free, on the web)

- Conferences:
  - ICML: International Conference on Machine Learning
  - NIPS: Neural Information Processing Systems
  - COLT: Computational Learning Theory
  - UAI: Uncertainty in AI
  - AIStats: intersection of Statistics and AI
  - Also AAAI, IJCAI and others

- Some MLe courses:
  - 10-708 Probabilistic Graphical Models (Fall)
  - 10-705 Intermediate Statistics (Fall)
  - 11-762 Language and Statistics II (Fall)
  - 10-702 Statistical Foundations of Machine Learning (Spring)
  - 10-70? Optimization (Spring)
  - ...