Markov Decision Processes (MDPs)

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 $P(x'|x,a)$
Discount Factors

People in economics and probabilistic decision-making do this all the time.

The “Discounted sum of future rewards” using discount factor $\gamma$ is

$\text{discounted sum} = (\text{reward now}) + \gamma (\text{reward in 1 time step}) + \gamma^2 (\text{reward in 2 time steps}) + \gamma^3 (\text{reward in 3 time steps}) + \ldots$ (infinite sum)

Define:

- $V_A$: Expected discounted future rewards starting in state $A$
- $V_B$: Expected discounted future rewards starting in state $B$
- $V_T$: Expected discounted future rewards starting in state $T$
- $V_S$: Expected discounted future rewards starting in state $S$
- $V_D$: Expected discounted future rewards starting in state $D$

Assume Discount Factor $\gamma = 0.9$

How do we compute $V_A$, $V_B$, $V_T$, $V_S$, $V_D$?
Policy

Policy: $\pi(x) = a$

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

- $\pi(x_0) = \text{both peasants get wood}$
- $\pi(x_1) = \text{one peasant builds barrack, other gets gold}$
- $\pi(x_2) = \text{peasants get gold, footmen attack}$

Value of Policy

Value: $V_{\pi}(x)$

Expected long-term reward starting from $x$

$$V_{\pi}(x_0) = E_x[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \gamma^4 R(x_4) + ...]$$

Future rewards discounted by $\gamma \in [0,1)$
Computing the value of a policy

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

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

\[ V_\pi(x_0) = E_\pi [\sum_{t=0}^{\infty} \gamma^t R(x_t)] \]

**A recursion!**

\[ V_\pi(x_0) = \sum_{t=0}^{\infty} \gamma^t R(x_t) \]

Simple approach for computing the value of a policy: Iteratively

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

Can solve using a simple convergent iterative approach:
(\text{a.k.a. dynamic programming})

\[ V_0(x) \quad \text{e.g. } V_0(x) = K(x) \quad \text{e.g. } V_0(x) = 0 \]

- Start with some guess \( V_0 \)
- Iteratively say:
  - \( V_{t+1}(x) \leftarrow R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_t(x') \)
- Stop when \( ||V_{t+1} - V_t||_\infty < \varepsilon \)
  - means that \( ||V_t - V_{t+1}||_\infty < \varepsilon/(1-\gamma) \)
But we want to learn a Policy

- So far, told you how good a policy is...
- But how can we choose the best policy???
- Suppose there was only one time step:
  - world is about to end!!!
  - select action that maximizes reward!

\[ V_{\pi}(x_0) = \max_{a_0} \sum R(x_0, a_0) + \gamma E_{a_0} [ \max_{a_1} R(x_1, a_1) + \gamma^2 E_{a_1} [ \max_{a_2} R(x_2, a_2) + \ldots ] ] \]

Unrolling the recursion

- Choose actions that lead to best value in the long run
- Optimal value policy achieves optimal value \( V^* \)

\[ V^*(x_0) = \max_{a_0} R(x_0, a_0) + \gamma E_{a_0} [ \max_{a_1} R(x_1, a_1) + \gamma^2 E_{a_1} [ \max_{a_2} R(x_2, a_2) + \ldots ] ] \]
Bellman equation

- Evaluating policy $\pi$:
  \[ V^\pi(x) = R(x) + \gamma \sum_{x'} P(x' | x, a = \pi(x)) V^\pi(x') \]

- Computing the optimal value $V^*$ - Bellman equation
  \[ V^*(x) = \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V^*(x') \]

Optimal Long-term Plan

Optimal value function $V^*(x)$ ➔ Optimal Policy: $\pi^*(x)$

Optimal policy:
\[ \pi^*(x) = \arg \max_a R(x, a) + \gamma \sum_{x'} P(x' | x, a) V^*(x') \]
Interesting fact – Unique value

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

- Slightly surprising fact: There is only one \( V^* \) that solves Bellman equation!
  - there may be many optimal policies that achieve \( V^* \)
- Surprising fact: optimal policies are good everywhere!!!

\[ V_{\pi^*}(x) \geq V_{\pi}(x), \forall x, \forall \pi \]

Solving an MDP

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

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

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

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]

\[ \vdots \]
Value iteration (a.k.a. dynamic programming) — the simplest of all

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

- Start with some guess \( V_0 \) \( V^*(x) = \max_{a} R(x, a) \)
- Iteratively say:
  - \( V^{t+1}(x) \leftarrow \max_{a} R(x, a) + \gamma \sum_{x'} P(x' | x, a) V^t(x') \)

- Stop when \( \|V^{t+1} - V^t\|_\infty < \epsilon \)
  - \( \|V^* - V_{t+1}\|_\infty < \epsilon(1 - \gamma) \)

A simple example

You run a startup company.

In every state you must choose between Saving money or Advertising.

\[ \gamma = 0.9 \]
Let’s compute $V_t(x)$ for our example

$V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_x P(x'|x, a)V_t(x')$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$V_t(\text{PU})$</th>
<th>$V_t(\text{PF})$</th>
<th>$V_t(\text{RU})$</th>
<th>$V_t(\text{RF})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4.5</td>
<td>14.5</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>2.03</td>
<td>6.53</td>
<td>25.08</td>
<td>18.55</td>
</tr>
<tr>
<td>4</td>
<td>3.852</td>
<td>12.20</td>
<td>29.63</td>
<td>19.26</td>
</tr>
<tr>
<td>5</td>
<td>7.22</td>
<td>15.07</td>
<td>32.00</td>
<td>20.40</td>
</tr>
<tr>
<td>6</td>
<td>10.03</td>
<td>17.65</td>
<td>33.58</td>
<td>22.43</td>
</tr>
</tbody>
</table>

In this example

$R(x, a) = R(x)$

$V_{t+1}(x) = \max_a R(x, a) + \gamma \sum_x P(x'|x, a)V_t(x')$
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

Acknowledgment

- This lecture contains some material from Andrew Moore’s excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials
The Reinforcement Learning task

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

**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 $a_t$

- **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
- “ “ “ 39, “ = 0, “ = 4
- “ “ “ 22, “ = 0, “ = 1
- “ “ “ 21, “ = 0, “ = 1
- “ “ “ 21, “ = 0, “ = 1
- “ “ “ 13, “ = 0, “ = 2
- “ “ “ 54, “ = 0, “ = 2
- “ “ “ 26, “ = 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, then 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
TD-Learning and Q-learning – Model-free approaches

Value of Policy

Value: $V_\pi(x)$

Expected long-term reward starting from $x$

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

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

- Estimate $V_\pi(x)$, start several trajectories from $x$.
  - $V_\pi(x)$ 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.
Reusing trajectories

- Value determination:
  \[ V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x') \]

- Expressed as an expectation over next states:
  \[ V_\pi(x) = R(x) + \gamma \mathbb{E}_{x', a} [V_\pi(x') \mid x, a = \pi(x)] \]

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

- Idea 1: Observe a transition: \( x_t \Rightarrow x_{t+1}, r_{t+1} \), approximate expectation with single sample:
  \[ \mathbb{E}[V_\pi(x_{t+1}) \mid x_t, a = \pi(x_t)] \approx V_\pi(x_t) \]

  - unbiased!!
  - but a very bad estimate!!!

Simple fix: Temporal Difference (TD) Learning [Sutton ’84]

- 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

  ✧ exponentially decaying average
TD converges (can take a long time!!!)

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

- **Theorem**: TD converges in the limit (with prob. 1), if:
  - every state is visited infinitely often
  - Learning rate decays just so:
    - \( \sum_{i=1}^{\infty} \alpha_i = 1 \)
    - \( \sum_{i=1}^{\infty} \alpha_i^2 < 1 \)

Another model-free RL approach: Q-learning

- [Watkins & Dayan '92]

- TD is just for one policy...
  - How do we find the optimal policy?

- Q-learning:
  - 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:
  \[ V_{t+1}(x) \leftarrow \max_a R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) V_t(x') \]

- Or:
  \[ Q_{t+1}(x, a) \leftarrow R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) V_{t}(x') \]
  \[ V_{t+1}(x) \leftarrow \max_a Q_{t+1}(x, a) \]

- Writing in terms of Q-function:
  \[ Q_{t+1}(x, a) \leftarrow R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) \max_{a'} Q_{t}(x', a') \]

Q-learning

- Observe a transition: \( x_t, a_t \Rightarrow x_{t+1}, r_{t+1} \), approximate expectation by mixture of new sample with old estimate:
  \[ Q_{t+1}(x, a) \leftarrow R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) \max_{a'} Q_t(x', a') \]

- TD learning on \( Q(x, a) \):
  \[ Q_t(x'): \new \text{sample: } r_{t+1} + \gamma \max_{a'} Q_t(x', a') \]

- Transition now from state-action pair to next state and reward
  \[ Q_t(x, a) \leftarrow (1 - \alpha_t) Q_t(x, a) + \alpha_t \left[ r_{t+1} + \gamma \max_{a'} Q_t(x', a') \right] \]

- \( \alpha > 0 \) is learning rate
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: $a_t = \arg \max_a Q_t(x, a)$
    - with prob. $\epsilon$ take an action at (uniformly) random
  - **Boltzmann (softmax) policy:**
    - $P(a_t | x) \propto \exp \left( \frac{Q_t(x, a)}{K} \right)$
    - $K$ – “temperature” parameter, $K \to 0$, as $t \to \infty$

Announcements

- **University Course Assessments**
  - Please, please, please, please, please, please, please, please, please, please...
- **Project:**
  - Poster session: Friday 3-6pm, NSH Atrium
    - please arrive a 15 mins early to set up
  - Paper: December 9th by 10:30pm
    - electronic submission by email to instructors list
    - maximum of 8 pages, NIPS format
    - no late days allowed
- **Final**
  - 12/14/2009, 8:30am-11:30am in UC McConomy
  - Bring a calculator (that can do logs... ☺)
Given a dataset – learn model

Given data, learn (MDP) Representation:

- Dataset: \( X_t, A_t \rightarrow Y_{t+1}, X_{t+1} \)
- Learn reward function:
  - \( R(x, a) \)
  \[ R(x_t, a_t) = y_{t+1} \]
- Learn transition model:
  - \( P(x'|x, a) \)

\[ \text{MLE counts just like in HMMs} \]
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 \( (R_{\text{max}}) \)?
  - 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:**
  1. $X_1, \ldots, X_n$ are i.i.d. Bernoulli trials with prob. $\theta$
  2. $P(|\frac{1}{n} \sum_i X_i - \theta| > \varepsilon) \leq \exp\{-2n\varepsilon^2\}$

Putting it all together

- **Theorem:** With prob. at least $1-\delta$, $R_{\text{max}}$ will reach a $\varepsilon$-optimal policy in time polynomial in: num. states, num. actions, $T$, $1/\varepsilon$, $1/\delta$

- Every $T$ steps:
  1. achieve near optimal reward (great!), or
  2. visit an unknown state-action pair! num. states and actions is finite, so can’t take too long before all states are known
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? $n^m$  
  - How many actions? $K^n$

- **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) \( Q(s,a) \to \mathbb{R} \)
  - A regression problem!
- **Learning a policy**: mapping from states to actions \( \pi(s) \to \{1, \ldots, k\} \)
  - A classification problem!

Use many of the ideas you learned this semester:
- linear regression, SVMs, decision trees, neural networks, Bayes nets, etc.!!!
What you have learned this semester

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

Improving the performance at some task through experience!!! 😊

Before you start any learning task, remember the fundamental questions:

- What is the learning problem?
- From what experience?
- What model?
- What loss function are you optimizing?
- With what optimization algorithm?
- Which learning algorithm?
- With what guarantees?
- How will you evaluate it?

What next?

- Machine Learning Department Seminar: http://calendar.cs.cmu.edu/ml/google_seminar
- 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
  - AllStats: intersection of Statistics and AI
  - Also AAAI, IJCAI and others
- Some MLD 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)
  - 10-725 Optimization (Spring)
  - ...
You have done a lot!!!

- And (hopefully) learned a lot!!!
  - Implemented
    - NB
    - LR
    - Nearest Neighbors
    - Boosting
    - SVM
    - HMMs
    - PCA
    - ...
  - Answered hard questions and proved many interesting results
  - Completed (I am sure) an amazing ML project
  - And did excellently on the final!

Thank You for the Hard Work!!!