Reinforcement Learning
Announcements

- Project:
  - Poster session: Friday May 5th 2-5pm, NSH Atrium
    - please arrive a little early to set up
    - posterboards, easels, and pins provided
    - class divided into two shift so you can see other posters

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

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.
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 $\rightarrow$ 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 $\rightarrow$ 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**: $x_1, a_1, r_1 \rightarrow x_2, a_2, r_2 \rightarrow x_3, a_3, r_3$

- **Learn reward function**:
  - $R(x, a) = \text{when I visit } x_1, a_1 \text{ at time } t, \text{ set } R(x, a) = r_t$

- **Learn transition model**:
  - $P(x' | x, a) = \frac{\text{count}(x_1, a, x')}{\text{count}(x, a)}$
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 $\rightarrow$ 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?
    - $P(x'|x,a)$?
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 $1-\delta$, $R_{\text{max}}$ 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 $\rightarrow$ num. states and actions is finite, so can’t take too long before all states are known (almost)
Problems with model-based approach

- If state space is large
  - transition matrix is very large! $|X|^2 \cdot |A|$
  - requires many visits to declare a state as know

- Chernoff is loose

- 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 $V(x)$, start several trajectories from $x \rightarrow V(x)$ 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

Play game from $x$, following $\pi$, $K$ times

Each time $V_i = \sum_{t=0}^{T} x_t$

$V(\pi(x)) \approx \frac{1}{K} \sum_{i=1}^{K} V_i$
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:

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

- Initialize value function (zeros, at random,...) \( V_0 \)

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

  \[ V(x_t) = R(x_t) + \gamma \cdot V(x_{t+1}) \]

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

  high variance!!

  one sample!!
Simple fix: Temporal Difference (TD) Learning [Sutton '84]

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

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

\[ V_{t+1}(x_t) = \alpha (r_t + \gamma V_t(x_{t+1})) + (1-\alpha) V_t(x_t) \]

\( V_t \) is estimate of value function at time \( t \)
\( \alpha > 0 \) is learning rate

\( \alpha \) is a parameter of the algorithm

Trajectory
\[ x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \cdots \]

Updates at \( x_1 \)

Update \( V \) at state \( x_0 \)

Exponentially decaying average:
\[ \hat{x}_t = (1-\alpha) x_t + \alpha \hat{x}_{t-1} \]
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:
  - \[ \sum_{i=1}^{\infty} \alpha_i = \infty \]
  - \[ \sum_{i=1}^{\infty} \alpha_i^2 < \infty \]

As \( t \to \infty \) the value of as policy \( \pi \)
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:
  \[ \pi_{t+1}(x) = \arg\max_a R(x, a) + \gamma \sum_{x'} P(x'| x, a)V_t(x') \]

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

- Writing in terms of Q-function:
  $$Q_{t+1}(x, a) = R(x, a) + \gamma \sum_{x'} P(x'|x, a) \max_{a'} Q_t(x', a')$$
Q-learning

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

- 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
    \[ Q_{t+1}(x_t, a_t) = (1-\alpha) Q_t(x_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max_{a'} Q_t(x_{t+1}, a') \right] \]
  - \( \alpha > 0 \) is learning rate

\[ Q_{t+1}(x, a) = R(x, a) + \gamma \sum_{x'} P(x' \mid x, a) \max_{a'} Q_t(x', a') \]
Q-learning convergence

- Under same conditions as TD, Q-learning converges to optimal value function $Q^*$
  $$\Pi^*(x) = \arg\max_a Q^*(x, a)$$

- Off-policy method:
  - Can run any policy, as long as policy visits every state-action pair infinitely often.
  - Typical policies (non of these address Exploration-Exploitation tradeoff):
    - $\varepsilon$-greedy:
      - with prob. $(1-\varepsilon)$ take greedy action: $a_t = \arg\max_a Q_t(x, a)$
      - with prob. $\varepsilon$ 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 $t \to \infty$
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? \( m^n \)

- **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) \( Q : X \times A \rightarrow \mathbb{R} \)
  - A regression problem!

- **Learning a policy**: mapping from states to actions \( \pi : X \rightarrow a \in \{1, \ldots, K\} \)
  - A classification problem!

Use many of the ideas you learned this semester:

- linear regression, SVMs, decision trees, neural networks, Bayes nets, etc.!!!
What you 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
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
- 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
Improving the performance at some task through 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?


- 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
  - Also AAAI, IJCAI and others

- Some MLD courses:
  - 10-708 Probabilistic Graphical Models (Fall)
  - 10-705 Intermediate Statistics (Fall)
  - 10-702 Statistical Foundations of Machine Learning (Spring)