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
Given a dataset – learn model

Given data, learn (MDP) Representation:

- Dataset: $x_t, a_t \rightarrow r_t, x_{t+1}$

- Learn reward function:
  
  \[
  R(x_t, a_t) = \gamma
  \]

- Learn transition model:
  
  \[
  P(x_{t+1} | x_t, a_t)
  \]
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 0?
    - 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 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
- $R(x,a) = R_{max}, \forall x,a$
- $P(x_0|x,a) = 1, \forall x,a$
- 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. $\theta$
  - $P(|1/n \sum_i X_i - \theta| > \epsilon) \leq \exp\{-2n\epsilon^2\}$

Putting it all together

- **Theorem**: With prob. at least $1-\delta$, $R_{\text{max}}$ will reach a $\epsilon$-optimal policy in time polynomial in: num. states, num. actions, $T$, $1/\epsilon$, $1/\delta$
  - 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
What you need to know about RL…

- Neither supervised, nor unsupervised learning
- Try to learn to act in the world, as we travel states and get rewards
- Model-based & Model-free approaches
- Rmax, a model based approach:
  - Learn model of rewards and transitions
  - Address exploration-exploitation tradeoff
  - Simple algorithm, great in practice

Closing….
What you have learned this quarter

- Learning is function approximation
- Point estimation
- Regression
- LASSO
- Subgradient
- Stochastic gradient descent
- Coordinate descent
- Discriminative vs. Generative learning
- Naive Bayes
- Logistic regression
- Bias-Variance tradeoff
- Decision trees
- Cross validation
- Boosting
- Instance-based learning
- Perceptron
- SVMs
- Kernel trick
- PAC learning
- Bayes nets
- Representation, parameter and structure learning
- K-means
- EM
- Mixtures of Gaussians
- 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>How will you evaluate it?</td>
</tr>
<tr>
<td>Which learning algorithm?</td>
<td>With what guarantees?</td>
<td></td>
</tr>
</tbody>
</table>
You have done a lot!!

- And (hopefully) learned a lot!!
  - Implemented
    - LASSO
    - LR
    - Perceptron
    - Clustering
    - ...
  - 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!!!