Announcements

- **Midterm** next Monday (March 6) 8-10pm (more details++ on ed)

---

**Man beats machine at Go in human victory over AI**

Amateur Kellin Pelnirn exploited weakness in systems that have otherwise dominated board game's grandmasters
How to beat a superhuman Go program

White: Kellin Pelrine (~2300)
Black: JBXKata005 (~5200)

9-stone handicap
Q-learning as approximate Q-iteration

- Recall the definition of Q values:
  - \( Q^*(s,a) = \) expected return from doing \( a \) in \( s \) and then behaving optimally
  - \( V(s) = \max_a Q^*(s,a) \) and \( \pi^*(s) = \arg\max_a Q^*(s,a) \)

- Bellman equation for Q values:
  - \( Q^*(s,a) = \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma \max_{a'} Q^*(s',a') ] \)

- Approximate Bellman update for Q values:
  - \( Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') ] \)

- We obtain a policy from learned \( Q(s,a) \), with no model!
  - (No free lunch: \( Q(s,a) \) table is \(|A|\) times bigger than \( V(s) \) table)
Q-Learning

- Learn Q(s,a) values as you go
  - Receive a sample \((s,a,s',r)\)
  - Consider your old estimate: \(Q(s,a)\)
  - Consider your new sample estimate:
    \[
    \text{sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')
    \]
  - Incorporate the new estimate into a running average:
    \[
    Q(s,a) \leftarrow (1-\alpha) \, Q(s,a) + \alpha \cdot \text{[sample]}
    \]
Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if samples are generated from a suboptimal policy!

- This is called **off-policy learning**

- Caveats:
  - You have to explore enough (eventually try every state/action pair infinitely often)
  - You have to decrease the learning rate appropriately
    - Technical requirements: $\sum_t \alpha(t) = \infty$, $\sum_t \alpha^2(t) < \infty$
    - Satisfied by: $\alpha(t) = 1/t$ or (better) $\alpha(t) = K/(K+t)$
Summary

- RL solves MDPs via direct experience of transitions and rewards
- There are several schemes:
  - Learn the MDP model and solve it
  - Learn $V$ directly from sums of rewards, or by TD local adjustments
    - Still need a model to make decisions by lookahead
  - Learn $Q$ by local Q-learning adjustments, use it directly to pick actions
    - (and about 100 other variations)
- Big missing pieces:
  - How to explore without too much regret?
  - How to scale this up to Tetris ($10^{60}$), Go ($10^{172}$), StarCraft ($|A|=10^{26}$)?
CS 188: Artificial Intelligence

Reinforcement Learning II

Instructors: Stuart Russell and Peyrin Kao

University of California, Berkeley
Reminder: Reinforcement Learning

- RL solves MDPs via direct experience of transitions and rewards
- There are several schemes:
  - Learn the MDP model and solve it
  - Learn $V$ directly from sums of rewards, or by TD local adjustments
    - Still need a model to make decisions by lookahead
  - Learn $Q$ by local Q-learning adjustments, use it directly to pick actions
    - (and about 100 other variations)
- Big missing pieces:
  - How to explore without too much regret?
  - How to scale this up to Tetris ($10^{60}$), Go ($10^{172}$), StarCraft ($|A|=10^{26}$)?
Exploration vs. Exploitation
**Exploration vs exploitation**

- **Exploration**: try new things
- **Exploitation**: do what’s best given what you’ve learned so far
- Key point: pure exploitation often gets *stuck in a rut* and never finds an optimal policy!
Exploration method 1: \( \varepsilon \)-greedy

- \( \varepsilon \)-greedy exploration
  - Every time step, flip a biased coin
  - With (small) probability \( \varepsilon \), act randomly
  - With (large) probability \( 1-\varepsilon \), act on current policy

- Properties of \( \varepsilon \)-greedy exploration
  - Every \( s,a \) pair is tried infinitely often
  - Does a lot of stupid things
    - Jumping off a cliff *lots of times* to make sure it hurts
  - Keeps doing stupid things for ever
    - Decay \( \varepsilon \) towards 0
Sensible exploration: Bandits

- Which one-armed bandit to try next?
- Most people would choose C > B > A > D
- Basic intuition: higher mean is better; more uncertainty is better
- Gittins (1979): rank arms by an index that depends only on the arm itself
Exploration Functions

- **Exploration functions** implement this tradeoff
  - Takes a value estimate $u$ and a visit count $n$, and returns an optimistic utility, e.g., $f(u,n) = u + k/\sqrt{n}$

- **Regular Q-update:**
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a Q(s',a)]$

- **Modified Q-update:**
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a f(Q(s',a),n(s',a'))]$

- Note: this propagates the “bonus” back to states that lead to unknown states as well!
Optimality and exploration

- **Fixed $\varepsilon$-greedy**
- **Decay $\varepsilon$-greedy**

Diagram showing:
- Total reward per trial vs. number of trials
- Regret vs. number of trials
- Optimal performance vs. exploration function
Regret measures the total cost of your youthful errors made while exploring and learning instead of behaving optimally.

Minimizing regret goes beyond learning to be optimal – it requires optimally learning to be optimal.
Approximate Q-Learning
Generalizing Across States

- Basic Q-Learning keeps a table of all Q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the Q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - Can we apply some machine learning tools to do this?
Example: Pacman

Let’s say we discover through experience that this state is bad:

In naïve q-learning, we know nothing about this state:

Or even this one!
Describe a state using a vector of **features**

- Features are functions from states to real numbers (often 0/1) that capture important properties of the state.
- Example features:
  - Distance to closest ghost $f_{GST}$
  - Distance to closest dot
  - Number of ghosts
  - $1 / (\text{distance to closest dot}) f_{DOT}$
  - Is Pacman in a tunnel? (0/1)
  - ...... etc.
- Can also describe a q-state $(s, a)$ with features (e.g., action moves closer to food)
Linear Value Functions

- We can express $V$ and $Q$ (approximately) as weighted linear functions of feature values:
  - $V_w(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$
  - $Q_w(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + ... + w_n f_n(s,a)$

- With the wrong features, the best possible approximation may be terrible!
- But in practice we can compress a value function for chess ($10^{43}$ states) down to about 30 weights and get decent play!!!
Updating a linear value function

- Original Q-learning rule tries to reduce prediction error at \( s,a \):
  \[
  Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ]
  \]

- Instead, we update the weights to try to reduce the error at \( s,a \):
  \[
  w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ] \frac{\partial Q_w(s,a)}{\partial w_i}
  = w_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) ] f_i(s,a)
  \]

- Qualitative justification:
  - Pleasant surprise: increase weights on +ve features, decrease on –ve ones
  - Unpleasant surprise: decrease weights on +ve features, increase on –ve ones
Example: Q-Pacman

\[ Q(s, a) = 4.0 \, f_{\text{DOT}}(s, a) - 1.0 \, f_{\text{GST}}(s, a) \]

\[ f_{\text{DOT}}(s, \text{NORTH}) = 0.5 \]
\[ f_{\text{GST}}(s, \text{NORTH}) = 1.0 \]

\[ Q(s, \text{NORTH}) = +1 \]
\[ r + \gamma \, \max_{a'} \, Q(s', a') = -500 + 0 \]

\[ \text{difference} = -501 \]

\[ w_{\text{DOT}} \leftarrow 4.0 + \alpha[-501]0.5 \]
\[ w_{\text{GST}} \leftarrow -1.0 + \alpha[-501]1.0 \]

\[ Q(s', \cdot) = 0 \]

\[ Q(s, \text{NORTH}) = +1 \]
\[ r = -500 \]

\[ Q(s', a') = -500 + 0 \]

\[ Q(s, a) = 3.0 \, f_{\text{DOT}}(s, a) - 3.0 \, f_{\text{GST}}(s, a) \]
Let $V^L$ be the closest linear approximation to $V^*$. 

- TD learning with a linear function approximator converges to some $V$ that is pretty close to $V^L$.
- Q-learning with a linear function approximator may diverge.
- With much more complicated update rules, stronger convergence results can be proved – even for nonlinear function approximators such as neural nets.
Nonlinear function approximators

- We can still use the gradient-based update for any $Q_w$:
  \[ w_i \leftarrow w_i + \alpha \cdot [R(s,a,s') + \gamma \max_a Q(s',a') - Q(s,a)] \frac{\partial Q_w(s,a)}{\partial w_i} \]

- Neural network error back-propagation already does this!

- Maybe we can get much better V or Q approximators using a complicated neural net instead of a linear function
Backgammon
TDGammon

- 4-ply lookahead using $V(s)$ trained from 1,500,000 games of self-play
- 3 hidden layers, ~100 units each
- Input: contents of each location plus several handcrafted features
- Experimental results:
  - Plays approximately at parity with world champion
  - Led to radical changes in the way humans play backgammon
DeepMind DQN

- Used a deep learning network to represent Q:
  - Input is last 4 images (84x84 pixel values) plus score
- 49 Atari games, incl. Breakout, Space Invaders, Seaquest, Enduro
Summary

- Exploration vs. exploitation
  - Exploration guided by unfamiliarity and potential
  - Appropriately designed bonuses tend to minimize regret
- Generalization allows RL to scale up to real problems
  - Represent $V$ or $Q$ with parameterized functions
  - Adjust parameters to reduce sample prediction error