Lecture Overview

Finish Reinforcement learning

• Exploration vs. Exploitation
• On-policy Learning (SARSA)
• Scalability
$Q(s, A)$

$s_1 \ldots s \ldots s_n$

$Q(s_i, a) = r + \gamma \max_{a'} Q(s_i', a')$

$t^{-1} \Rightarrow a$

$sars'$

$Q^t(s, A)$

$Q(s_i, a) = Q^t(s, A) + \alpha_k \left( r + \gamma \max_{a'} Q^t(s_i', a') - Q^t(s, A) \right)$

$TD$

$A^t = A^{t-1} + \alpha_k \left( r + \gamma \max_{a'} Q^t(s_i', a') - Q^t(s, A) \right)$
Clarification on the $\alpha_{k_{sa}} = \frac{1}{k_{sa}}$
What Does Q-Learning learn

- Q-learning does not explicitly tell the agent what to do.
  - Given the Q-function, the agent can either exploit it or explore more.

Any effective strategy should

- Choose the predicted best action in the limit
- Try each action an unbounded number of times

We will look at two exploration strategies

- $\varepsilon$-greedy
- soft-max
$\varepsilon$-greedy

- Choose a random action with probability $\varepsilon$ and choose best action with probability $1 - \varepsilon$

\[
P(\text{random action}) = \varepsilon
\]
\[
P(\text{best action}) = 1 - \varepsilon
\]

- First GLIE condition (try every action an unbounded number of times) is satisfied via the $\varepsilon$ random selection

- What about second condition?
  
  - Select predicted best action in the limit.

- reduce $\varepsilon$ overtime!
Soft-Max

- Takes into account improvement in estimates of expected reward function $Q[s,a]$.
  - Choose action $a$ in state $s$ with a probability proportional to current estimate of $Q[s,a]$: $\frac{e^{Q[s,a]}}{\sum_a e^{Q[s,a]/\tau}}$.

- $\tau$ (tau) in the formula above influences how randomly actions should be chosen:
  - if $\tau$ is high, the exponentials approach 1, the fraction approaches $\frac{1}{\text{number of actions}}$, and each action has approximately the same probability of being chosen (exploration or exploitation?).
  - as $\tau \to 0$, the exponential with the highest $Q[s,a]$ dominates, and the current best action is always chosen (exploration or exploitation?).
**Soft-Max Example**

Assume only 3 actions:

\[ Q[s_i, a] \]

<table>
<thead>
<tr>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>2 $</td>
</tr>
<tr>
<td>a_2</td>
<td>3 $</td>
</tr>
<tr>
<td>a_3</td>
<td>1 $</td>
</tr>
</tbody>
</table>

**Probability of selecting an action**

\[
\frac{e^{Q(s,a)}}{\sum_a e^{Q(s,a)}}
\]

- **Action a_1**
  
  \[
  \frac{e^2}{e^1 + e^2 + e^3}
  \]

- **Action a_2**
  
  \[
  \frac{e^3}{e^1 + e^2 + e^3}
  \]

- **Action a_3**
  
  \[
  \frac{e^4}{e^2 + e^4 + e^6}
  \]

\[ \tau = 100 \]

\[ \frac{e^{Q(s,a)} / \tau}{\sum_a e^{Q(s,a)} / \tau} \]

\[ \tau = .5 \]

\[ \frac{e^{Q(s,a)} / \tau}{\sum_a e^{Q(s,a)} / \tau} \]
**Soft-Max**

- When in state \( s \), Takes into account improvement in estimates of expected reward function \( Q[s,a] \) for all the actions
  - Choose action \( a \) in state \( s \) with a probability proportional to current estimate of \( Q[s,a] \)

\[
\frac{e^{Q[s,a]}}{\sum_a e^{Q[s,a]}} \quad \text{vs} \quad \frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}
\]

- \( \tau \) (tau) in the formula above influences how randomly values should be chosen
  - if \( \tau \) is high, \( \gg Q[s,a] \)?

A. It will mainly exploit
B. It will mainly explore
C. It will do both with equal probability
Soft-Max

\[
\frac{e^{Q[s,a]/\tau}}{\sum_a e^{Q[s,a]/\tau}}
\]

- \( \tau \) (tau) in the formula above influences how randomly values should be chosen
  - if \( \tau \) is high, the exponentials approach 1, the fraction approaches \( 1/(\text{number of actions}) \), and each action has approximately the same probability of being chosen (exploration or exploitation?)
  - as \( \tau \to 0 \), the exponential with the highest \( Q[s,a] \) dominates, and the current best action is always chosen (exploration or exploitation?)
Lecture Overview

Finish Reinforcement learning

- Exploration vs. Exploitation
- On-policy Learning (SARSA)
- RL scalability
Learning before vs. during deployment

Our learning agent can:

A. act in the environment to learn how it works (before deployment)

B. Learn as you go (after deployment)

If there is time to learn before deployment, the agent should try to do its best to learn as much as possible about the environment

- even engage in locally suboptimal behaviors, because this will guarantee reaching an optimal policy in the long run

If learning while “at work”, suboptimal behaviors could be costly
Best way to present this

- If agent is not deployed it should do random all the time ($\epsilon=1$) and Q-learning
  - When Q values have converged then deploy

- If the agent is deployed it should apply one of the explore/exploit strategies (e.g., $\epsilon=.5$) and do sarsa
Consider, for instance, our sample grid game:

- the optimal policy is to go *up* in $S_0$
- But if the agent includes some exploration in its policy (e.g. selects 20% of its actions randomly), exploring in $S_2$ could be dangerous because it may cause hitting the -100 wall
- No big deal if the agent is not deployed yet, but not ideal otherwise

Q-learning would not detect this problem

- It does *off-policy learning*, i.e., it focuses on the optimal policy

*On-policy* learning addresses this problem
On-policy learning: SARSA

- On-policy learning learns the value of the policy being followed.
  - e.g., act greedily 80% of the time and act randomly 20% of the time
  - Better to be aware of the consequences of exploration has it happens, and avoid outcomes that are too costly while acting, rather than looking for the true optimal policy

- SARSA
  - So called because it uses \(<\text{state}, \text{action}, \text{reward}, \text{state}, \text{action}>\) experiences rather than the \(<\text{state}, \text{action}, \text{reward}, \text{state}>\) used by Q-learning
  - Instead of looking for the best action at every step, it evaluates the actions suggested by the current policy
  - Uses this info to revise it
On-policy learning: SARSA

Given an experience \(<s, a, r, s', a'>\), SARSA updates \(Q[s, a]\) as follows

\[
Q[s, a] \leftarrow Q[s, a] + \alpha ((r + \gamma Q[s', a']) - Q[s, a])
\]

What’s different from Q-learning?
On-policy learning: SARSA

- Given an experience <s, a, r, s’, a’>, SARSA updates Q[s,a] as follows

\[ Q[s, a] \leftarrow Q[s, a] + \alpha \left( (r + \gamma Q[s', a']) - Q[s, a] \right) \]

- While Q-learning was using

\[ Q[s, a] \leftarrow Q[s, a] + \alpha \left( (r + \gamma \max_{a'} Q[s', a']) - Q[s, a] \right) \]

- There is no more max operator in the equation, there is instead the Q-value of the action suggested by the current policy
\[ Q[s,a] \leftarrow Q[s,a] + \alpha (r + \gamma Q[s',a'] - Q[s,a]) \]

\[ Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k (r + 0.9Q[s_1, UpCareful] - Q[s_0, right]) \]

\[ Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k (r + 0.9Q[s_3, UpCareful] - Q[s_1, upCarfull]) \]

\[ Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k (r + 0.9Q[s_5, Left] - Q[s_3, upCarfull]) \]

\[ Q[s_3, upCarfull] \leftarrow 0 + 1(-1 + 0.9*0 - 0) = -1 \]

\[ Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k (r + 0.9Q[s_4, left] - Q[s_5, Left]) \]

\[ Q[s_5, Left] \leftarrow 0 + 1(0 + 0.9*0 - 0) = 0 \]

\[ Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k (r + 0.9Q[s_0, Right] - Q[s_4, Left]) \]

\[ Q[s_4, Left] \leftarrow 0 + 1(10 + 0.9*0 - 0) = 10 \]

Only immediate rewards are included in the update, as with Q-learning
\[ Q[s, a] \leftarrow Q[s, a] + \alpha (r + \gamma Q[s', a'] - Q[s, a]) \]

**k=2**

<table>
<thead>
<tr>
<th>Q[s,a]</th>
<th>s_0</th>
<th>s_1</th>
<th>s_2</th>
<th>s_3</th>
<th>s_4</th>
<th>s_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>upCareful</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Left</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Right</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Up</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ Q[s_0, right] \leftarrow Q[s_0, right] + \alpha_k (r + 0.9Q[s_1, UpCareful] - Q[s_0, right]); \]
\[ Q[s_0, right] \leftarrow -1 \]

\[ Q[s_1, upCarfull] \leftarrow Q[s_1, upCarfull] + \alpha_k (r + 0.9Q[s_3, UpCareful] - Q[s_1, upCarfull]); \]
\[ Q[s_1, upCarfull] \leftarrow 0 \]

\[ Q[s_3, upCarfull] \leftarrow Q[s_3, upCarfull] + \alpha_k (r + 0.9Q[s_5, Left] - Q[s_3, upCarfull]); \]
\[ Q[s_3, upCarfull] \leftarrow -1 + 1/2(-1 + 0.9*0 + 1) = -1 \]

\[ Q[s_5, Left] \leftarrow Q[s_5, Left] + \alpha_k (r + 0.9Q[s_4, left] - Q[s_5, Left]); \]
\[ Q[s_5, Left] \leftarrow 0 + 1/2(0 + 0.9*10 - 0) = 4.5 \]

\[ Q[s_4, Left] \leftarrow Q[s_4, Left] + \alpha_k (r + 0.9Q[s_0, Right] - Q[s_4, Left]); \]
\[ Q[s_4, Left] \leftarrow 10 + 1/2(10 + 0.9*0 - 10) = 10 \]

SARSA backs up the expected reward of the next action, rather than the max expected reward.
Comparing SARSA and Q-learning

- For the little 6-states world

- Policy learned by Q-learning 80% greedy is to go up in $s_0$ to reach $s_4$ quickly and get the big +10 reward

<table>
<thead>
<tr>
<th>Iterations</th>
<th>$Q[s_0, \text{Up}]$</th>
<th>$Q[s_1, \text{Up}]$</th>
<th>$Q[s_2, \text{UpC}]$</th>
<th>$Q[s_3, \text{Up}]$</th>
<th>$Q[s_4, \text{Left}]$</th>
<th>$Q[s_5, \text{Left}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>400000000</td>
<td>19.1</td>
<td>17.5</td>
<td>22.7</td>
<td>20.4</td>
<td>26.8</td>
<td>23.7</td>
</tr>
</tbody>
</table>

- Verify running full demo, see http://www.cs.ubc.ca/~poole/aibook/demos/rl/tGame.html
Comparing SARSA and Q-learning

- Policy learned by SARSA 80% greedy is to go right in $s_0$
- Safer because avoid the chance of getting the -100 reward in $s_2$
- but non-optimal => lower Q-values

<table>
<thead>
<tr>
<th>Iterations</th>
<th>$Q[s_0,\text{Right}]$</th>
<th>$Q[s_1,\text{Up}]$</th>
<th>$Q[s_2,\text{UpC}]$</th>
<th>$Q[s_3,\text{Up}]$</th>
<th>$Q[s_4,\text{Left}]$</th>
<th>$Q[s_5,\text{Left}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>40000000</td>
<td>6.8</td>
<td>8.1</td>
<td>12.3</td>
<td>10.4</td>
<td>15.6</td>
<td>13.2</td>
</tr>
</tbody>
</table>

- Verify running full demo, see http://www.cs.ubc.ca/~poole/aibook/demos/rl/tGame.html
SARSA Algorithm

begin
  initialize $Q[S, A]$ arbitrarily
  observe current state $s$
  select action $a$ using a policy based on $Q$
repeatever:
  carry out an action $a$
  observe reward $r$ and state $s'$
  select action $a'$ using a policy based on $Q$
  $Q[s, a] \leftarrow Q[s, a] + \alpha (r + \gamma Q[s', a'] - Q[s, a])$
  $s \leftarrow s'$
  $a \leftarrow a'$
end-repeatever
end

This could be, for instance any $\varepsilon$-greedy strategy:
- Choose random $\varepsilon$ times, and max the rest
Another Example

- Gridworld with:
  - Deterministic actions *up*, *down*, *left*, *right*
  - Start from **S** and arrive at **G** (terminal state with reward > 0)
  - **Reward is -1 for all transitions**, except those into the region marked “Cliff”
    - Falling into the cliff causes the agent to be sent back to start: \( r = -100 \)
With an ε-greedy strategy (e.g., \( \varepsilon = 0.1 \))

A. SARSA will learn policy \( p_1 \) while Q-learning will learn \( p_2 \)

B. Q-learning will learn policy \( p_1 \) while SARSA will learn \( p_2 \)

C. They will both learn \( p_1 \)

D. They will both learn \( p_2 \)
Because of **negative reward for every step taken**, the optimal policy over the four standard actions is to take the shortest path along the cliff.

But if the agents adopt an ε-greedy action selection strategy with ε=0.1, walking along the cliff is dangerous:

- The optimal path that considers exploration is to go around as far as possible from the cliff.
Q-learning vs. SARSA

- Q-learning learns the optimal policy, but because it does so without taking exploration into account, it does not do so well while the agent is exploring
  - It occasionally falls into the cliff, so its reward per episode is not that great

- SARSA has better on-line performance (reward per episode), because it learns to stay away from the cliff while exploring
  - But note that if $\epsilon \to 0$, SARSA and Q-learning would asymptotically converge to the optimal policy
422 big picture: Where are we?

Deterministic

Logics
- First Order Logics

Ontologies
- Temporal rep.
  - Full Resolution
  - SAT

Stochastic

Belief Nets
- Approx. : Gibbs

Markov Chains and HMMs
- Forward, Viterbi….
- Approx. : Particle Filtering

Undirected Graphical Models
- Conditional Random Fields

Markov Decision Processes and Partially Observable MDP
- Value Iteration
- Approx. Inference

Reinforcement Learning

Hybrid: Det + Sto

Prob CFG
Prob Relational Models
Markov Logics

Applications of AI

CPSC 322, Lecture 34
Learning Goals for today’s class

You can:

• Describe and compare techniques to combine exploration with exploitation
• On-policy Learning (SARSA)
• Discuss trade-offs in RL scalability (not required)
TODO for Mon

- Read textbook 6.4.2
- Next research paper will be next Fri
- Practice Ex 11.B

- Assignment 1 due on Mon