CS 6300: Artificial Intelligence

Midterm Review
Midterm Logistics

- In our classroom during normal class time
  - Wednesday from 3-4:20pm
- 1 sheet of notes (front and back)
- Simple calculator allowed but not needed (all math will be simple)
- Lots of extra-credit.
  - Choose your own adventure.
  - Focus on solving the easiest problems first and then move to the harder ones.
Topics you’ll need to know

- A* and consistent/admissible heuristics
- Alpha-Beta pruning for min-max search
- Expectimax search
- Probability
  - conditional prob, (cond.) independence, Bayes’ rule, chain rule
- MDPs
  - Value Iteration
  - Policy Iteration (iterative version, not the closed form solution)
  - Temporal difference learning
Topics you’ll need to know

- Q-Learning
- Linear value function approximation
- Policy Gradients
  - Be able to follow math for policy gradient derivation in slides.
  - I won’t ask you to rederive full policy gradient
- AlphaGo
  - Understand high-level pieces and how they connect
A search problem consists of:

- A state space
- A successor function (with actions, costs)
- A start state and a goal test

A solution is a sequence of actions (a plan) which transforms the start state to a goal state.
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        if STATE[child-node] is not in closed then fringe ← INSERT(child-node, fringe)
      end
    end
  end
A-star: Combining UCS and Greedy

- **Uniform-cost** orders by path cost, or *backward cost* $g(n)$
- **Greedy** orders by goal proximity, or *forward cost* $h(n)$

- **A* Search** orders by the sum: $f(n) = g(n) + h(n)$

Example: Teg Grenager
Admissible Heuristics

- A heuristic $h$ is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

- Examples:

- Coming up with admissible heuristics is most of what’s involved in using A* in practice.
Consistency of Heuristics

- **Main idea:** estimated heuristic costs ≤ actual costs
  - **Admissibility:** heuristic cost ≤ actual cost to goal
    
    \[ h(A) \leq \text{actual cost from } A \text{ to } G \]

  - **Consistency:** heuristic “arc” cost ≤ actual cost for each arc
    
    \[ h(A) - h(C) \leq \text{cost}(A \text{ to } C) \]

- **Consequences of consistency:**
  - The f value along a path never decreases
    
    \[ h(A) \leq \text{cost}(A \text{ to } C) + h(C) \]

  - A* graph search is optimal
Minimax Values

States Under Agent’s Control:

\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]

States Under Opponent’s Control:

\[ V(s') = \min_{s \in \text{successors}(s')} V(s) \]

Terminal States:

\[ V(s) = \text{known} \]
Minimax Implementation

**def max-value(state):**
initialize $v = -\infty$
for each successor of state:
    $v = \max(v, \text{min-value}(\text{successor}))$
return $v$

**def min-value(state):**
initialize $v = +\infty$
for each successor of state:
    $v = \min(v, \text{max-value}(\text{successor}))$
return $v$

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$
Minimax Implementation (Dispatch)

def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
Minimax Example

\[
\alpha = -\infty \\
\beta = +\infty
\]

\[
\alpha = 3 \\
\beta = +\infty
\]

\[
\alpha = 3 \\
\beta = +\infty
\]
**Alpha-Beta Implementation**

α: MAX’s best option on path to root  
β: MIN’s best option on path to root

At root you should initialize $\alpha = -\infty$ and $\beta = +\infty$

---

**def max-value(state, $\alpha$, $\beta$):**
- initialize $v = -\infty$
- for each successor of state:
  - $v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$
  - if $v \geq \beta$ return $v$
  - $\alpha = \max(\alpha, v)$
- return $v$

**def min-value(state, $\alpha$, $\beta$):**
- initialize $v = +\infty$
- for each successor of state:
  - $v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$
  - if $v \leq \alpha$ return $v$
  - $\beta = \min(\beta, v)$
- return $v$
**Alpha-Beta Quiz**

**α**: MAX’s best option on path to root  
**β**: MIN’s best option on path to root

**def max-value(state, α, β):**
- initialize $v = -\infty$
- for each successor of state:
  - $v = \max(v, value(successor, α, β))$
  - if $v ≥ β$ return $v$
  - $α = \max(α, v)$
- return $v$

**def min-value(state, α, β):**
- initialize $v = +\infty$
- for each successor of state:
  - $v = \min(v, value(successor, α, β))$
  - if $v ≤ α$ return $v$
  - $β = \min(β, v)$
- return $v$
Uncertain Search
Expectimax Pseudocode

```python
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v

v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10
Mixed Layer Types

- E.g. Backgammon
- Expectiminimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children
Probability
Probability Distributions

- Unobserved random variables have distributions
  
  \[
  P(T) \quad \quad P(W)
  \begin{array}{c|c}
  \text{hot} & 0.5 \\
  \text{cold} & 0.5 \\
  \end{array}
  \begin{array}{c|c}
  \text{sun} & 0.6 \\
  \text{rain} & 0.1 \\
  \text{fog} & 0.3 \\
  \text{meteor} & 0.0 \\
  \end{array}
  \]

- A distribution is a TABLE of probabilities of values

- A probability (lower case value) is a single number

  \[
P(W = \text{rain}) = 0.1
  \]

- Must have: \( \forall x \ P(X = x) \geq 0 \) and \( \sum_{x} P(X = x) = 1 \)

Shorthand notation:

\[
P(\text{hot}) = P(T = \text{hot}),
\]
\[
P(\text{cold}) = P(T = \text{cold}),
\]
\[
P(\text{rain}) = P(W = \text{rain}),
\]
... 

OK if all domain entries are unique
A joint distribution over a set of random variables: \( X_1, X_2, \ldots X_n \) specifies a real number for each assignment (or outcome):

\[
P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n)
\]

\[
P(x_1, x_2, \ldots x_n)
\]

- Must obey: \( P(x_1, x_2, \ldots x_n) \geq 0 \)

\[
\sum_{(x_1,x_2,\ldots x_n)} P(x_1, x_2, \ldots x_n) = 1
\]

Size of distribution if \( n \) variables with domain sizes \( d? \)

- For all but the smallest distributions, impractical to write out!

\[
\begin{array}{|c|c|c|}
\hline
T & W & P \\
\hline
\text{hot} & \text{sun} & 0.4 \\
\text{hot} & \text{rain} & 0.1 \\
\text{cold} & \text{sun} & 0.2 \\
\text{cold} & \text{rain} & 0.3 \\
\hline
\end{array}
\]
Quiz: Events

- $P(+x, +y) \, ?$

- $P(+x) \, ? = \sum_y P(X = +x, Y = y)$

- $P(-y \text{ OR } +x) \, ?$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+x$</td>
<td>$+y$</td>
<td>0.2</td>
</tr>
<tr>
<td>$+x$</td>
<td>$-y$</td>
<td>0.3</td>
</tr>
<tr>
<td>$-x$</td>
<td>$+y$</td>
<td>0.4</td>
</tr>
<tr>
<td>$-x$</td>
<td>$-y$</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Marginal Distributions

- Marginal distributions are sub-tables which eliminate variables
- Marginalization (summing out): Combine collapsed rows by adding

\[
P(T, W)
\]

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

\[
P(T) = \sum_s P(t, s)
\]

<table>
<thead>
<tr>
<th>T</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>0.5</td>
</tr>
<tr>
<td>cold</td>
<td>0.5</td>
</tr>
</tbody>
</table>

\[
P(W)
\]

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.6</td>
</tr>
<tr>
<td>rain</td>
<td>0.4</td>
</tr>
</tbody>
</table>

\[
P(X_1 = x_1) = \sum_{x_2} P(X_1 = x_1, X_2 = x_2)
\]
Quiz: Marginal Distributions

\[ P(X, Y) \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( Y )</th>
<th>( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>+x</td>
<td>+y</td>
<td>0.2</td>
</tr>
<tr>
<td>+x</td>
<td>-y</td>
<td>0.3</td>
</tr>
<tr>
<td>-x</td>
<td>+y</td>
<td>0.4</td>
</tr>
<tr>
<td>-x</td>
<td>-y</td>
<td>0.1</td>
</tr>
</tbody>
</table>

\[ P(x) = \sum_y P(x, y) \]

\[ P(y) = \sum_x P(x, y) \]

\[ P(X) \]

<table>
<thead>
<tr>
<th>( X )</th>
<th>( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>+x</td>
<td>\</td>
</tr>
<tr>
<td>-x</td>
<td>\</td>
</tr>
</tbody>
</table>

\[ P(Y) \]

<table>
<thead>
<tr>
<th>( Y )</th>
<th>( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>+y</td>
<td>\</td>
</tr>
<tr>
<td>-y</td>
<td>\</td>
</tr>
</tbody>
</table>
### Conditional Probabilities

- A simple relation between joint and conditional probabilities
  - In fact, this is taken as the *definition* of a conditional probability

\[
P(a|b) = \frac{P(a, b)}{P(b)}
\]

### Example

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- \(P(W = s|T = c) = \frac{P(W = s, T = c)}{P(T = c)} = \frac{0.2}{0.5} = 0.4\)
- \(= P(W = s, T = c) + P(W = r, T = c) = 0.2 + 0.3 = 0.5\)
### Quiz: Conditional Probabilities

#### $P(X, Y)$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$Y$</td>
<td>$P$</td>
</tr>
<tr>
<td>$+x$</td>
<td>$+y$</td>
<td>0.2</td>
</tr>
<tr>
<td>$+x$</td>
<td>$-y$</td>
<td>0.3</td>
</tr>
<tr>
<td>$-x$</td>
<td>$+y$</td>
<td>0.4</td>
</tr>
<tr>
<td>$-x$</td>
<td>$-y$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- $P(+x | +y)$ ?
- $P(-x | +y)$ ?
- $P(-y | +x)$ ?
The Product Rule

- Sometimes have conditional distributions but want the joint

\[ P(y)P(x|y) = P(x, y) \]

\[ \iff \]

\[ P(x|y) = \frac{P(x, y)}{P(y)} \]
The Product Rule

\[ P(y)P(x|y) = P(x, y) \]

**Example:**

| \( P(W) \) | \( P(D|W) \) | \( P(D, W) \) |
|----------------|----------------|----------------|
| R | P | | D | W | P |
| sun | 0.8 | | wet | sun | 0.1 |
| rain | 0.2 | | dry | sun | 0.9 |
|    |     | | wet | rain | 0.7 |
|    |     | | dry | rain | 0.3 |

\[
\begin{array}{ccc}
\text{D} & \text{W} & \text{P} \\
\text{wet} & \text{sun} & 0.1 \\
\text{dry} & \text{sun} & 0.9 \\
\text{wet} & \text{rain} & 0.7 \\
\text{dry} & \text{rain} & 0.3 \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{D} & \text{W} & \text{P} \\
\text{wet} & \text{sun} & \\
\text{dry} & \text{sun} & \\
\text{wet} & \text{rain} & \\
\text{dry} & \text{rain} & \\
\end{array}
\]
More generally, can always write any joint distribution as an incremental product of conditional distributions

\[
P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)
\]

\[
P(x_1, x_2, \ldots x_n) = \prod_i P(x_i|x_1 \ldots x_{i-1})
\]

- You can pick any order.
- Why is the Chain Rule always true?
Bayes’ Rule

- Two ways to factor a joint distribution over two variables:

\[ P(x, y) = P(x|y)P(y) = P(y|x)P(x) \]

- Dividing, we get:

\[ P(x|y) = \frac{P(y|x)P(x)}{P(y)} \]

- Why is this at all helpful?
  - Lets us build one conditional from its reverse
  - Often one conditional is tricky but the other one is simple
  - Foundation of many systems (e.g. ASR, MT, IRL)

- In the running for most important AI equation!
Independence

- Two variables are independent in a joint distribution if:

\[ P(X, Y) = P(X)P(Y) \]
\[ \forall x, y P(x, y) = P(x)P(y) \]

- Says the joint distribution factors into a product of two simple ones
- Usually variables aren’t independent!

- Can use independence as a modeling assumption
  - Independence can be a simplifying assumption
  - Empirical joint distributions: at best “close” to independent
  - What could we assume for \{Weather, Traffic, Cavity\}? 

- Independence is like something from CSPs: what?
Example: Independence?

### $P_1(T, W)$

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.4</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.1</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.3</td>
</tr>
</tbody>
</table>

### $P(T)$

<table>
<thead>
<tr>
<th>T</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>0.5</td>
</tr>
<tr>
<td>cold</td>
<td>0.5</td>
</tr>
</tbody>
</table>

### $P_2(T, W) = P(T)P(W)$

<table>
<thead>
<tr>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sun</td>
<td>0.3</td>
</tr>
<tr>
<td>hot</td>
<td>rain</td>
<td>0.2</td>
</tr>
<tr>
<td>cold</td>
<td>sun</td>
<td>0.3</td>
</tr>
<tr>
<td>cold</td>
<td>rain</td>
<td>0.2</td>
</tr>
</tbody>
</table>

### $P(W)$

<table>
<thead>
<tr>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun</td>
<td>0.6</td>
</tr>
<tr>
<td>rain</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Conditional Independence

- Unconditional (absolute) independence very rare (why?)

- Conditional independence is our most basic and robust form of knowledge about uncertain environments.

- $X$ is conditionally independent of $Y$ given $Z$ ($X \perp Y \mid Z$) if and only if:

$$\forall x, y, z : P(x, y \mid z) = P(x \mid z)P(y \mid z)$$

or, equivalently, if and only if

$$\forall x, y, z : P(x \mid z, y) = P(x \mid z)$$
Probability Recap

- **Conditional probability**
  \[ P(x|y) = \frac{P(x, y)}{P(y)} \]

- **Product rule**
  \[ P(x, y) = P(x|y)P(y) \]

- **Chain rule**
  \[ P(X_1, X_2, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\ldots = \prod_{i=1}^{n} P(X_i|X_1, \ldots, X_{i-1}) \]

- **X, Y independent if and only if**: \( \forall x, y : P(x, y) = P(x)P(y) \)

- **X and Y are conditionally independent given Z if and only if**: \( X \perp Y|Z \)
  \[ \forall x, y, z : P(x, y|z) = P(x|z)P(y|z) \]
Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function $T(s, a, s')$
    - Probability that a from $s$ leads to $s'$, i.e., $P(s' | s, a)$
    - Also called the model or the dynamics
  - A reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - A start state
  - Maybe a terminal state

- MDPs are non-deterministic search problems
  - One way to solve them is with expectimax search
  - We’ll have a new tool soon
What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent.

- For Markov decision processes, "Markov" means action outcomes depend only on the current state.

\[
P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)
\]
\[
= P(S_{t+1} = s'|S_t = s_t, A_t = a_t)
\]

- This is just like search, where the successor function could only depend on the current state (not the history).

Andrey Markov (1856-1922)
Important Quantities

- The value (utility) of a state $s$:
  
  $V^*(s) = \text{expected utility starting in } s \text{ and acting optimally}$

- The value (utility) of a q-state $(s,a)$:
  
  $Q^*(s,a) = \text{expected utility starting out having taken action } a \text{ from state } s \text{ and (thereafter) acting optimally}$

- The optimal policy:
  
  $\pi^*(s) = \text{optimal action from state } s$
Bellman Equations

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed!

- Recursive definition of value:
  \[ V^*(s) = \max_a Q^*(s, a) \]
  \[ Q^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]
  \[ V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]
Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero.

- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$
V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]
$$

Bellman Update Equation

- Repeat until convergence.

- Complexity of each iteration: $O(S^2 A)$.

- Theorem: will converge to unique optimal values.
  - Basic idea: approximations get refined towards optimal values.
  - Policy may converge long before values do.
Policy Iteration

- Alternative approach for optimal values:
  - **Step 1: Policy evaluation:** calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy improvement:** update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It’s still optimal!
  - Can converge (much) faster under some conditions
Policy Iteration

- **Evaluation:** For fixed current policy \( \pi \), find values with policy evaluation:
  - Iterate until values converge:
    \[
    V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[ R(s, \pi_i(s), s') + \gamma V_{k}^{\pi_i}(s') \right]
    \]

- **Improvement:** For fixed values, get a better policy using policy extraction:
  - One-step look-ahead:
    \[
    \pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_{k}^{\pi_i}(s') \right]
    \]
Temporal Difference Learning

- **Big idea: learn from every experience!**
  - Update $V(s)$ each time we experience a transition $(s, a, s', r)$
  - Likely outcomes $s'$ will contribute updates more often

- **Temporal difference learning of values**
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Sample of $V(s)$:

$$sample = R(s, \pi(s), s') + \gamma V^\pi(s')$$

Update to $V(s)$:

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample$$

Same update:

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$$
Q-Learning

- Q-Learning: sample-based Q-value iteration

\[ Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right] \]

- 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:
    \[ sample = R(s, a, s') + \gamma \max_{a'} Q(s', a') \]
  - Incorporate the new estimate into a running average:
    \[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) \text{[sample]} \]
    \[ Q(s, a) \leftarrow Q(s, a) + \alpha (\text{sample} - Q(s, a)) \]
Using a feature representation, we can write a q function (or value function) for any state using a few weights:

\[
V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)
\]

\[
Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a)
\]

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!
Approximate Q-Learning

\[ Q(s,a) = w_1f_1(s,a) + w_2f_2(s,a) + \ldots + w_nf_n(s,a) \]

- Q-learning with linear Q-functions:
  
  \[ \text{transition} = (s,a,r,s') \]
  
  \[ \text{difference} = \left[ r + \gamma \max_{a'} Q(s',a') \right] - Q(s,a) \]
  
  \[ Q(s,a) \leftarrow Q(s,a) + \alpha \text{[difference]} \]
  
  \[ w_i \leftarrow w_i + \alpha \text{[difference]} f_i(s,a) \]

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state’s features

- Formal justification: online least squares
DQN

- Approximate Q-Learning at scale.
- Uses Neural Network for Q-value function approximation.
Two approaches to model-free RL

- **Learn Q-values**
  - Trains Q-values to be consistent. Not directly optimizing for performance.
  - Use an objective based on the Bellman Equation
    \[
    Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]
    \]

- **Learn Policy Directly**
  - Have a parameterized policy \( \pi_\theta \)
  - Update the parameters \( \theta \) to optimize performance of policy.
Policy Gradient RL

- Find a policy that maximizes expected utility (discounted cumulative rewards)

\[ \pi^* = \arg \max \pi \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R(s, \pi(s), s') \right] \]
Notation

- Trajectory (rollout, episode) $\tau = (s_0, a_0, s_1, a_1, \ldots)$
  - $s_0 \sim \rho_0(\cdot)$ (initial state distribution)
  - $s_{t+1} \sim P(\cdot | s_t, a_t)$ (transition probabilities)
- Rewards $r_t = R(s_t, a_t, s_{t+1})$
- Finite-horizon undiscounted return of a trajectory
  $$R(\tau) = \sum_{t=0}^{T} r_t$$
- Actions are sampled from a stochastic parameterized policy $\pi_{\theta}$
  $$a_t \sim \pi_{\theta}(\cdot | s_t)$$
Notation

- **Probability of a trajectory (rollout, episode)** $\tau = (s_0, a_0, s_1, a_1, \ldots)$

  $$P(\tau | \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t)$$

- **Expected Return of a policy** $J(\pi)$

  $$J(\pi) = \sum_\tau P(\tau | \pi) R(\tau) = E_{\tau \sim \pi} [R(\tau)]$$

- **Goal of RL: Solve the following optimization problem**

  $$\pi^* = \arg\max_\pi J(\pi)$$
The Policy Gradient

- We can now perform gradient ascent to improve our policy!

\[
\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta} J(\pi_{\theta}) \bigg|_{\theta_k}
\]

\[
\nabla_{\theta} J(\pi_{\theta}) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \ R(\tau) \right]
\]

Estimate with a sample mean over a set \( D \) of policy rollouts given current parameters:

\[
\approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} (\nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \ R(\tau))
\]
There will be one short answer question about AlphaGo.
Review high-level ideas from slides. Don’t worry about nitty-gritty details.