Tutorial Agenda

- Refresh RL terminology through Tic Tac Toe
- Deterministic Q-Learning: what and how
- Q-learning Matlab demo: Gridworld
- Extensions: non-deterministic reward, next state
- More cool demos
Tic Tac Toe Redux
Tic Tac Toe Redux

\[ R = \begin{array}{|c|c|c|}
\hline
\text{Lose} & \text{Tie} & \text{Win} \\
\hline
-1 & 0 & +1 \\
\hline
\end{array} \]

\[ S_t = \begin{array}{|c|c|}
\hline
x & o \\
\hline
x & o \\
\hline
\end{array} \]

\( \pi : S \rightarrow A \)

\[ \pi \left( \begin{array}{|c|c|}
\hline
x & o \\
\hline
x & o \\
\hline
\end{array} \right) \rightarrow a \]

\[ \mathcal{V}^\pi : S \rightarrow R \]

\[ \mathcal{V}^\pi \left( \begin{array}{|c|c|}
\hline
x & o \\
\hline
x & o \\
\hline
\end{array} \right) \rightarrow r_{\text{future}} \]
Each board position (taking into account symmetry) has some probability

<table>
<thead>
<tr>
<th>State</th>
<th>Probability of a win (Computer plays “o”)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>
Each board position (taking into account symmetry) has some probability.

Simple learning process:

<table>
<thead>
<tr>
<th>State</th>
<th>Probability of a win (Computer plays “o”)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>
Each board position (taking into account symmetry) has some probability

- Simple learning process:
  - start with all values = 0.5

<table>
<thead>
<tr>
<th>State</th>
<th>Probability of a win (Computer plays “o”)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>
Each board position (taking into account symmetry) has some probability

Simple learning process:

- start with all values = 0.5
- policy: choose move with highest probability of winning given current legal moves from current state
RL & Tic-Tac-Toe

- Each board position (taking into account symmetry) has some probability

<table>
<thead>
<tr>
<th>State</th>
<th>Probability of a win (Computer plays “o”)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>x o</td>
<td>0.5</td>
</tr>
<tr>
<td>x</td>
<td>0.5</td>
</tr>
<tr>
<td>x x</td>
<td>1.0</td>
</tr>
<tr>
<td>x o o</td>
<td>0.0</td>
</tr>
<tr>
<td>x x x</td>
<td>0.0</td>
</tr>
<tr>
<td>o o x</td>
<td>0.5</td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>

Simple learning process:
- start with all values $= 0.5$
- **policy**: choose move with highest probability of winning given current legal moves from current state
- update entries in table based on outcome of each game
Each board position (taking into account symmetry) has some probability

Simple learning process:

- start with all values $= 0.5$
- policy: choose move with highest probability of winning given current legal moves from current state
- update entries in table based on outcome of each game
- After many games value function will represent true probability of winning from each state

<table>
<thead>
<tr>
<th>State</th>
<th>Probability of a win (Computer plays “o”)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 x</td>
<td>0.5</td>
</tr>
<tr>
<td>0 0 x</td>
<td>0.5</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1.0</td>
</tr>
<tr>
<td>x x 0</td>
<td>0.0</td>
</tr>
<tr>
<td>0 x x</td>
<td>0.0</td>
</tr>
<tr>
<td>0 0 x</td>
<td>0.5</td>
</tr>
<tr>
<td>etc</td>
<td></td>
</tr>
</tbody>
</table>
Each board position (taking into account symmetry) has some probability

Simple learning process:

- start with all values $= 0.5$
- policy: choose move with highest probability of winning given current legal moves from current state
- update entries in table based on outcome of each game
- After many games value function will represent true probability of winning from each state

Can try alternative policy: sometimes select moves randomly (exploration)
MDP Refresher

Familiar? Skip?
Goal: find policy $\pi$ that maximizes expected accumulated future rewards $V^\pi(s_t)$, obtained by following $\pi$ from state $s_t$:

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

Game show example:

I assume series of questions, increasingly difficult, but increasing payoff. I choice: accept accumulated earnings and quit; or continue and risk losing everything.

Notice that:

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$
Goal: find policy $\pi$ that maximizes expected accumulated future rewards $V^\pi(s_t)$, obtained by following $\pi$ from state $s_t$:

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Game show example:
Goal: find policy $\pi$ that maximizes expected accumulated future rewards $V^\pi(s_t)$, obtained by following $\pi$ from state $s_t$:

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Game show example:
- assume series of questions, increasingly difficult, but increasing payoff
Goal: find policy $\pi$ that maximizes expected accumulated future rewards $V^\pi(s_t)$, obtained by following $\pi$ from state $s_t$:

\[ V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \]
\[ = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \]

Game show example:
- assume series of questions, increasingly difficult, but increasing payoff
- choice: accept accumulated earnings and quit; or continue and risk losing everything

Notice that:

\[ V^\pi(s_t) = r_t + \gamma V^\pi(s_{t+1}) \]
What to Learn

- We might try to learn the function $V$ (which we write as $V^*$)
  
  $$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- Here $\delta(s, a)$ gives the next state, if we perform action $a$ in current state $s$
What to Learn

- We might try to learn the function $V$ (which we write as $V^*$)

$$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- Here $\delta(s, a)$ gives the next state, if we perform action $a$ in current state $s$

- We could then do a lookahead search to choose best action from any state $s$:

$$\pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$
What to Learn

- We might try to learn the function $V$ (which we write as $V^*$)
  \[ V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))] \]

- Here $\delta(s, a)$ gives the next state, if we perform action $a$ in current state $s$

- We could then do a lookahead search to choose best action from any state $s$:
  \[ \pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))] \]

- But there’s a problem:
What to Learn

- We might try to learn the function $V$ (which we write as $V^*$)

$$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- Here $\delta(s, a)$ gives the next state, if we perform action $a$ in current state $s$
- We could then do a lookahead search to choose best action from any state $s$:

$$\pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- But there’s a problem:
  - This works well if we know $\delta()$ and $r()$
What to Learn

- We might try to learn the function $V$ (which we write as $V^*$)

$$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- Here $\delta(s, a)$ gives the next state, if we perform action $a$ in current state $s$
- We could then do a lookahead search to choose best action from any state $s$:

$$\pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

- But there’s a problem:
  - This works well if we know $\delta()$ and $r()$
  - But when we don’t, we cannot choose actions this way
Q Learning

Deterministic rewards and actions
Define a new function very similar to $V^*$

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$
Q Learning

- Define a new function very similar to $V^*$

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

- If we learn $Q$, we can choose the optimal action even without knowing $\delta$!

$$\pi^*(s) = \arg\max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$
Q Learning

- Define a new function very similar to $V^*$

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

- If we learn $Q$, we can choose the optimal action even without knowing $\delta$!

$$\pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$= \arg \max_a Q(s, a)$$
Q Learning

- Define a new function very similar to $V^*$

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

- If we learn $Q$, we can choose the optimal action even without knowing $\delta$!

$$\pi^*(s) = \arg \max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$= \arg \max_a Q(s, a)$$

- $Q$ is then the evaluation function we will learn
\[ \gamma = 0.9 \]

\[ r(s, a) \text{ (immediate reward) values} \]

\[ Q(s, a) \text{ values} \]

\[ V^*(s) \text{ values} \]

\[ V^*(s_5) = 0 + \gamma 100 + \gamma^2 0 + \ldots = 90 \]

One optimal policy
\[ \gamma = 0.9 \]

\( r(s, a) \) (immediate reward) values

\( Q(s, a) \) values

\( V^*(s) \) values

\[ V^*(s_5) = 0 + \gamma 100 + \gamma^2 0 + \ldots = 90 \]

One optimal policy
Training Rule to Learn $Q$

- $Q$ and $V^*$ are closely related:

$$V^*(s) = \max_a Q(s, a)$$
Training Rule to Learn Q

- $Q$ and $V^*$ are closely related:
  \[ V^*(s) = \max_a Q(s, a) \]

- So we can write $Q$ recursively:
  \[ Q(s_t, a_t) = r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \]
Training Rule to Learn $Q$

- $Q$ and $V^*$ are closely related:

$$V^*(s) = \max_a Q(s, a)$$

- So we can write $Q$ recursively:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))$$

$$= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$
Training Rule to Learn $Q$

- $Q$ and $V^*$ are closely related:
  \[ V^*(s) = \max_a Q(s, a) \]

- So we can write $Q$ recursively:
  \[
  Q(s_t, a_t) = r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t)) \\
  = r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')
  \]

- Let $\hat{Q}$ denote the learner’s current approximation to $Q$
Training Rule to Learn $Q$

- $Q$ and $V^*$ are closely related:

$$V^*(s) = \max_a Q(s, a)$$

- So we can write $Q$ recursively:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))$$
$$= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$

- Let $\hat{Q}$ denote the learner’s current approximation to $Q$

- Consider training rule

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

where $s'$ is state resulting from applying action $a$ in state $s$
Q Learning for Deterministic World

- For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$
Q Learning for Deterministic World

- For each \( s, a \) initialize table entry \( \hat{Q}(s, a) \leftarrow 0 \)
- Start in some initial state \( s \)
Q Learning for Deterministic World

- For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state $s$
- Do forever:

  1. Select an action $a$ and execute it
  2. Receive immediate reward $r$
  3. Observe the new state $s_0$
  4. Update the table entry for $\hat{Q}(s, a)$ using $Q$ learning rule:
     \[
     \hat{Q}(s, a) \leftarrow r(s, a) + \max_{a_0} \hat{Q}(s_0, a_0)
     \]
  5. $s \leftarrow s_0$

If we get to absorbing state, restart to initial state, and run thru “Do forever” loop until reach absorbing state.
Q Learning for Deterministic World

- For each \( s, a \) initialize table entry \( \hat{Q}(s, a) \leftarrow 0 \)
- Start in some initial state \( s \)
- Do forever:
  - Select an action \( a \) and execute it
Q Learning for Deterministic World

- For each \( s, a \) initialize table entry \( \hat{Q}(s, a) \leftarrow 0 \)
- Start in some initial state \( s \)
- Do forever:
  - Select an action \( a \) and execute it
  - Receive immediate reward \( r \)
Q Learning for Deterministic World

- For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state $s$
- Do forever:
  - Select an action $a$ and execute it
  - Receive immediate reward $r$
  - Observe the new state $s'$
Q Learning for Deterministic World

- For each \( s, a \) initialize table entry \( \hat{Q}(s, a) \leftarrow 0 \)
- Start in some initial state \( s \)
- Do forever:
  - Select an action \( a \) and execute it
  - Receive immediate reward \( r \)
  - Observe the new state \( s' \)
  - Update the table entry for \( \hat{Q}(s, a) \) using \( Q \) learning rule:
    \[
    \hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')
    \]
Q Learning for Deterministic World

- For each $s, a$ initialize table entry $Q(s, a) \leftarrow 0$
- Start in some initial state $s$
- Do forever:
  - Select an action $a$ and execute it
  - Receive immediate reward $r$
  - Observe the new state $s'$
  - Update the table entry for $Q(s, a)$ using $Q$ learning rule:

$$Q(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q(s', a')$$

- $s \leftarrow s'$
Q Learning for Deterministic World

- For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$
- Start in some initial state $s$
- Do forever:
  - Select an action $a$ and execute it
  - Receive immediate reward $r$
  - Observe the new state $s'$
  - Update the table entry for $\hat{Q}(s, a)$ using $Q$ learning rule:

\[
\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')
\]

- $s \leftarrow s'$
- If we get to absorbing state, restart to initial state, and run thru ”Do forever” loop until reach absorbing state
Updating Estimated $Q$

- Assume the robot is in state $s_1$; some of its current estimates of $Q$ are as shown; executes rightward move

![Diagram showing the robot in states $s_1$ and $s_2$ with rewards and actions](image)

Important observation: at each time step (making an action $a$ in state $s$) only one entry of $\hat{Q}$ will change (the entry $\hat{Q}(s, a)$).
Updating Estimated $Q$

- Assume the robot is in state $s_1$; some of its current estimates of $Q$ are as shown; executes rightward move

$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
Updating Estimated $Q$

Assume the robot is in state $s_1$; some of its current estimates of $Q$ are as shown; executes rightward move

$$
\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')
$$

$$
\leftarrow r + 0.9 \max_{a} \{63, 81, 100\} \leftarrow 90
$$
Assume the robot is in state $s_1$; some of its current estimates of $Q$ are as shown; executes rightward move

$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow r + 0.9 \max_a \{63, 81, 100\} \leftarrow 90$$

Important observation: at each time step (making an action $a$ in state $s$) only one entry of $\hat{Q}$ will change (the entry $\hat{Q}(s, a)$)
Updating Estimated $Q$

- Assume the robot is in state $s_1$; some of its current estimates of $Q$ are as shown; executes rightward move

\[ \hat{Q}(s_1, a_{\text{right}}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a') \]
\[ \leftarrow r + 0.9 \max_a \{63, 81, 100\} \leftarrow 90 \]

- Important observation: at each time step (making an action $a$ in state $s$ only one entry of $\hat{Q}$ will change (the entry $\hat{Q}(s, a)$))

- Notice that if rewards are non-negative, then $\hat{Q}$ values only increase from 0, approach true $Q$
Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state.
Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action $a$ results in transition from state $s_i$ to $s_j$; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state

Each executed action $a$ results in transition from state $s_i$ to $s_j$; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule

Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back
Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state.

Each executed action $a$ results in transition from state $s_i$ to $s_j$; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule.

Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back:

1. All $\hat{Q}(s, a)$ start at 0.
Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action \( a \) results in transition from state \( s_i \) to \( s_j \); algorithm updates \( \hat{Q}(s_i, a) \) using the learning rule
- Intuition for simple grid world, reward only upon entering goal state \( \rightarrow Q \) estimates improve from goal state back
  1. All \( \hat{Q}(s, a) \) start at 0
  2. First episode – only update \( \hat{Q}(s, a) \) for transition leading to goal state
Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action $a$ results in transition from state $s_i$ to $s_j$; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
- Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back
  1. All $\hat{Q}(s, a)$ start at 0
  2. First episode – only update $\hat{Q}(s, a)$ for transition leading to goal state
  3. Next episode – if go thru this next-to-last transition, will update $\hat{Q}(s, a)$ another step back
Q Learning: Summary

- Training set consists of series of intervals (episodes): sequence of (state, action, reward) triples, end at absorbing state
- Each executed action $a$ results in transition from state $s_i$ to $s_j$; algorithm updates $\hat{Q}(s_i, a)$ using the learning rule
- Intuition for simple grid world, reward only upon entering goal state $\rightarrow Q$ estimates improve from goal state back
  1. All $\hat{Q}(s, a)$ start at 0
  2. First episode – only update $\hat{Q}(s, a)$ for transition leading to goal state
  3. Next episode – if go thru this next-to-last transition, will update $\hat{Q}(s, a)$ another step back
  4. Eventually propagate information from transitions with non-zero reward throughout state-action space
Gridworld Demo
Extensions

Non-deterministic reward and actions
• Have not specified how actions chosen (during learning)
Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s,a)$

\[ P(s'|s,a) = \exp(k \hat{Q}(s,a)) \]
Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$
- Good idea?
Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$
- Good idea?
- Can instead employ stochastic action selection (policy):

$$P(a_i|s) = \frac{\exp(k\hat{Q}(s, a_i))}{\sum_j \exp(k\hat{Q}(s, a_j))}$$
Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s, a)$
- Good idea?
- Can instead employ stochastic action selection (policy):

$$P(a_i | s) = \frac{\exp(k \hat{Q}(s, a_i))}{\sum_j \exp(k \hat{Q}(s, a_j))}$$

- Can vary $k$ during learning
Q Learning: Exploration/Exploitation

- Have not specified how actions chosen (during learning)
- Can choose actions to maximize $\hat{Q}(s,a)$
- Good idea?
- Can instead employ stochastic action selection (policy):

$$P(a_i|s) = \frac{\exp(k\hat{Q}(s,a_i))}{\sum_j \exp(k\hat{Q}(s,a_j))}$$

- Can vary $k$ during learning
  - more exploration early on, shift towards exploitation
Non-deterministic Case

What if reward and next state are non-deterministic?
Non-deterministic Case

- What if reward and next state are non-deterministic?
- We redefine $V, Q$ based on probabilistic estimates, expected values of them:

$$V^\pi(s) = E_\pi [r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots]$$

$$= E_\pi [\sum_{i=0}^{\infty} \gamma^i r_{t+i}]$$
Non-deterministic Case

What if reward and next state are non-deterministic?

We redefine $V, Q$ based on probabilistic estimates, expected values of them:

$$ V^\pi(s) = E_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots] $$

$$ = E_\pi[\sum_{i=0}^{\infty} \gamma^i r_{t+i}] $$

and

$$ Q(s, a) = E[r(s, a) + \gamma V^*(\delta(s, a))] $$

$$ = E[r(s, a) + \gamma \sum_{s'} p(s'|s, a) \max_{a'} Q(s', a')] $$
Non-deterministic Case: Learning $Q$

- Training rule does not converge (can keep changing $\hat{Q}$ even if initialized to true $Q$ values)

So modify training rule to change more slowly:

$$\hat{Q}(s, a)(1 + \nu) = \hat{Q}(s, a) + \nu[r + \max_{a_0} \hat{Q}(s_0, a_0)]$$

where $s_0$ is the state land in after $s, a$, and $a_0$ indexes the actions that can be taken in state $s_0$. $
u = 1 + \text{visits}$ is the number of times action $a$ is taken in state $s$. 

Zemel, Urtasun, Fidler (UofT)  
CSC 411: 19-Reinforcement Learning  
November 29, 2016  39 / 1
Non-deterministic Case: Learning $Q$

- Training rule does not converge (can keep changing $\hat{Q}$ even if initialized to true $Q$ values)
- So modify training rule to change more slowly

$$\hat{Q}(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n[r + \gamma \max_{a'}\hat{Q}_{n-1}(s', a')]$$

where $s'$ is the state land in after $s$, and $a'$ indexes the actions that can be taken in state $s'$

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

where visits is the number of times action $a$ is taken in state $s$
More Cool Demos
Other Examples:

Super Mario World
https://www.youtube.com/watch?v=L4KBBAwF_bE

Model-based RL: Pole Balancing
https://www.youtube.com/watch?v=XiigTGKZfks
Learn how to fly a Helicopter


• Formulate as an RL problem
  
  • State - Position, orientation, velocity, angular velocity
  
  • Actions - Front-back pitch, left-right pitch, tail rotor pitch, blade angle
  
  • Dynamics - Map actions to states. Difficult!
  
  • Rewards - Don’t crash, Do interesting things.

Slide credit: Nitish Srivastava