COMP 790-125: Goals for today

- Introduce Reinforcement Learning
- Deep Reinforcement Learning

Reading material:
- Reinforcement Learning: An Introduction, Sutton and Bardo
- Deep Reinforcement Learning tutorial
- Neural Net playing Atari Games
  http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html
- AlphaGo
  http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html
Reinforcement Learning

Reinforcement learning discovers how to act in different states of the world so as to maximize reward.

- Who acts? An agent, for example chess player, robot, stock trader.
- What is a world? Chess board, 3d model of the world around the robot, stock prices and current portfolio.
- What is the reward? Reward winning the game, robot is at the destination, portfolio is at target value.

Note: Rewards are immutable. Rewards should not incorporate strategy information.

Q: What would be the down side of allowing to change rewards?

Q: What would happen if we rewarded capturing chess pieces not just winning the game?
Elements of reinforcement Learning

Agent makes a sequence of actions \((A_1, A_2, \ldots)\) based on current state \((S_1, S_2, \ldots)\) and collects rewards \((R_1, R_2, \ldots)\).

Figure 3.1: The agent–environment interaction in reinforcement learning.
In supervised learning, we are provided with training set 
\{ (x_t, y_t) : t = 1, \ldots, T \} and asked to predict \( y^* \) given \( x^* \).

Reward is immediate and independent of other predictions.

In reinforcement learning, we are asked to repeatedly make actions that alter the environment.

Poor actions, early on, can affect later rewards.
Elements of reinforcement Learning

- **policy** A mapping from state to actions

  \[ \pi_t(a|s) \]

  probability that \( A_t = a \) if \( S_t = s \).

- **reward function** Immediate reward for a particular state

- **value function** Long run expected reward of a particular state
Reinforcement learning aims to maximize expected rewards

\[ G_t = R_{t+1} + R_{t+2} + \ldots + R_T \]

assuming that the task has finite horizon.

In more general setting, **continuing task**, \( G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \)

where \( 0 \leq \gamma \leq 1 \) is called **discount rate**.
We can think of dynamics of the world as a conditional distribution
\[
p(R_{t+1}, S_{t+1} \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t)\]
\[\text{complete history}\]

We can say that the dynamics of the world have the Markov property if
\[
p(R_{t+1}, S_{t+1} \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t) = p(R_{t+1}, S_{t+1} \mid S_t, A_t)\]
\[\text{complete history}\]

State and rewards are dependent on the previous state.
A Markov Decision Process (MDP) is defined by states, actions, and Markovian dynamics of the world

\[ p(s'|s, a) \]

called transition probabilities.

Reward realized by making a transition from state \( s \) to \( s' \) using action \( a \) is denoted by \( r(s, a, s') \)
An example of MDP

A recycling robot:

- searches for empty cans
- waits for someone to bring it an empty can
- recharges its batteries when they are low

Collecting cans by either waiting or searching is rewarded.

For the purposes of this task, the state is the level of battery in the robot.

If the robot’s batteries are depleted, someone will come and recharge them.

States $S = \{\text{low}, \text{high}\}$, actions $A = \{\text{wait}, \text{search}, \text{recharge}\}$
An example of an MDP

| $s$   | $s'$   | $a$    | $p(s'|s,a)$ | $r(s, a, s')$ |
|-------|--------|--------|-------------|---------------|
| high  | high   | search | $\alpha$    | $r_{\text{search}}$ |
| high  | low    | search | $1 - \alpha$| $r_{\text{search}}$ |
| low   | high   | search | $1 - \beta$ | $-3$          |
| low   | low    | search | $\beta$     | $r_{\text{search}}$ |
| high  | high   | wait   | $1$         | $r_{\text{wait}}$ |
| high  | low    | wait   | $0$         | $r_{\text{wait}}$ |
| low   | high   | wait   | $0$         | $r_{\text{wait}}$ |
| low   | low    | wait   | $1$         | $r_{\text{wait}}$ |
| low   | high   | recharge| $1$        | $0$            |
| low   | low    | recharge| $0$        | $0$            |

Table 3.1: Transition probabilities and expected rewards for the finite MDP of the recycling robot example. There is a row for each possible combination of current state, $s$, next state, $s'$, and action possible in the current state, $a \in A(s)$.

This MDP specifies **world’s dynamics**. There is no specification of how the agent behaves, just how the world responds to its actions. Robot’s behavior is encoded in policy $\pi(a|s)$. 
An example of an MDP – transition graph

Figure 3.3: Transition graph for the recycling robot example.
Value function

Value function estimates how good the current state is in terms of future rewards.

This estimate depends on policy \((\pi)\), probability of taking actions given states.
State-value function for policy \(\pi\) is defined as

\[
v_\pi(s) = E_\pi [G_t | S_t = s] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]
\]

where expectation is taken assuming that the agent follows policy\(^1\) \(\pi\).

Action-value function for policy \(\pi\) is defined as

\[
q_\pi(s, a) = E_\pi [G_t | S_t = s, A_t = a] = E_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]
\]

\(^1\)Policy \(\pi(a|s)\) specifies probability of selecting action \(a\) at state \(s\).
Value function

We can unpack the expectation to reveal a recursion

\[
\begin{align*}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \\
&= \mathbb{E}_\pi \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s \right] \\
&= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s' \right] \right] \\
&= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma v_\pi(s') \right], \quad (3.10)
\end{align*}
\]

This is known as **Bellman equation**.
Optimal value function

Value function depends on state and policy. Hence, we can seek policy that maximizes the value function. Maximizing over the policy yields optimal state-value function

\[ v_*(s) = \max_\pi v_\pi(s). \]

Similarly, optimal action-value function is given by

\[ q_*(s, a) = \max_\pi q_\pi(s, a) \]

and we can express it in terms of \( v_* \)

\[ q_*(s, a) = E[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \]

Note that the expectation is computed over \( S_{t+1} \) using \( p(s'|s, a) \)
Golf example

Driver is used for long distance shots. Putter for short. Value is number of strokes required to get the ball in the hole. Policy putt uses just putter.
Optimality equations

\begin{align}
  v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
  &= \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a, s') + \gamma v_*(s') \right] \\
  \end{align}

\begin{align}
  q_*(s, a) &= \mathbb{E}\left[ R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\
  &= \sum_{s'} p(s'|s, a) \left[ r(s, a, s') + \gamma \max_{a'} q_*(s', a') \right],
\end{align}

Once we obtain \( q^* \) we can easily make local decisions regarding optimal action

\[ a^{\text{next}} = \arg\max q_*(s, a). \]

By definition, this action provides optimal expected long-term return.
Optimality equations

\begin{align*}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma v_*(s') \right] \\
q_*(s, a) &= \mathbb{E} \left[ R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right] \\
&= \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma \max_{a'} q_*(s', a') \right],
\end{align*}

(4.1) (4.2)

Q: Does this formulation remind you of an algorithm we have already seen?
Dynamic programming

The formulation of the optimal value functions yields itself to application of dynamic programming.

We will first look at a simpler problem **policy evaluation**. For a given policy and a state, evaluate its value function

\[
v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\
= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma v_\pi(s') \right],
\]

(4.3) 

(4.4)
Recursive formulation

For \( k \to \infty \), \( v_{k+1} \) converges to \( v_\pi \), since \( v_\pi \) is a fix point of the equation above.

\[
v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]
= \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \left[ r(s, a, s') + \gamma v_k(s') \right], \tag{4.5}
\]
Iterative policy evaluation

Input $\pi$, the policy to be evaluated
Initialize an array $v(s) = 0$, for all $s \in S^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in S$:
  $\quad temp \leftarrow v(s)$
  $\quad v(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v(s')]$
  $\quad \Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $v \approx v_\pi$

Figure 4.1: Iterative policy evaluation.
Policy improvement

The idea: if \( q_\pi(s,a) > v_\pi(s) \) then action \( a \) is better than the action chosen by policy \( \pi \).

Hence, we can improve policy by adjusting it in cases where

\[
\max_a q_\pi(s,a) > v_\pi(s)
\]

We can define a new greedy policy

\[
\pi'(s) = \arg \max_a q_\pi(s,a) \\
= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\
= \arg \max_a \sum_{s'} p(s'|s,a) \left[ r(s,a,s') + \gamma v_\pi(s') \right],
\] (4.9)
Policy improvement

1. Initialization
   \( v(s) \in \mathbb{R} \) and \( \pi(s) \in \mathcal{A}(s) \) arbitrarily for all \( s \in S \)

2. Policy Evaluation
   Repeat
   \( \Delta \leftarrow 0 \)
   For each \( s \in S \):
   \( \text{temp} \leftarrow v(s) \)
   \( v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \left[ r(s, \pi(s), s') + \gamma v(s') \right] \)
   \( \Delta \leftarrow \max(\Delta, |\text{temp} - v(s)|) \)
   until \( \Delta < \theta \) (a small positive number)

3. Policy Improvement
   \( \text{policy-stable} \leftarrow \text{true} \)
   For each \( s \in S \):
   \( \text{temp} \leftarrow \pi(s) \)
   \( \pi(s) \leftarrow \arg \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a, s') + \gamma v(s') \right] \)
   If \( \text{temp} \neq \pi(s) \), then \( \text{policy-stable} \leftarrow \text{false} \)
   If \( \text{policy-stable} \), then stop and return \( v \) and \( \pi \); else go to 2
Value iteration

If the policy evaluation is fast, then policy improvement can be used to find optimal policy.

An alternative algorithm for finding optimal policy is called value iteration.

---

**Initialize array $v$ arbitrarily (e.g., $v(s) = 0$ for all $s \in S^+$)**

Repeat

$\Delta \leftarrow 0$

For each $s \in S$:

$\text{temp} \leftarrow v(s)$

$v(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v(s')]$

$\Delta \leftarrow \max(\Delta, |\text{temp} - v(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that

$$\pi(s) = \arg \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a, s') + \gamma v(s') \right]$$

---

*Figure 4.5: Value iteration.*
Beyond policy learning: Q-learning

Rather than learning the policy $\pi$ we can aim to learn an approximation of action-value function $q$.

The learning algorithm is defined by

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$
Q-learning

Initialize $Q(s, a), \forall s \in S, a \in A(s)$, arbitrarily, and $Q(terminal-state, \cdot) = 0$

Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$
    Take action $A$, observe $R, S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  until $S$ is terminal
Deep Q-learning$^3$

We can use a deep neural network as an approximation of the action-value function $q$.

A Q-network, parameterized by $\theta$, is trained by minimizing loss

$$L_i(\theta_i) = E_{s,a \sim \rho}[(y_i - Q(s, a; \theta_i))^2]$$

where

$$y_i = E_{s'}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s]$$

and $\rho$ is a behavior distribution$^2$

\[\text{with probability } 1 - \epsilon \text{ select best action according to } Q, \text{ otherwise perform random action.}\]

$^2$Data from: https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
Deep Q-learning

Algorithm 1 Deep Q-learning with Experience Replay

1. Initialize replay memory $\mathcal{D}$ to capacity $N$
2. Initialize action-value function $Q$ with random weights
3. for episode $= 1, M$ do
   4. Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
5. for $t = 1, T$ do
   6. With probability $\epsilon$ select a random action $a_t$
6. otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
7. Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
8. Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
9. Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
10. Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
11. Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
12. Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
### Atari games

<table>
<thead>
<tr>
<th>Method</th>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>354</td>
<td>1.2</td>
<td>0</td>
<td>-20.4</td>
<td>157</td>
<td>110</td>
<td>179</td>
</tr>
<tr>
<td>Contingency [4]</td>
<td>1743</td>
<td>6</td>
<td>159</td>
<td>-17</td>
<td>960</td>
<td>723</td>
<td>268</td>
</tr>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>Human</td>
<td>7456</td>
<td>31</td>
<td>368</td>
<td>-3</td>
<td>18900</td>
<td>28010</td>
<td>3690</td>
</tr>
<tr>
<td>HNeat Best [8]</td>
<td>3616</td>
<td>52</td>
<td>106</td>
<td>19</td>
<td>1800</td>
<td>920</td>
<td>1720</td>
</tr>
<tr>
<td>HNeat Pixel [8]</td>
<td>1332</td>
<td>4</td>
<td>91</td>
<td>-16</td>
<td>1325</td>
<td>800</td>
<td>1145</td>
</tr>
<tr>
<td>DQN Best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
<td>1075</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>B. Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S. Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>HNeat Best [8]</td>
<td>3616</td>
<td>52</td>
<td>106</td>
<td>19</td>
<td>1800</td>
<td>920</td>
<td>1720</td>
</tr>
<tr>
<td>HNeat Pixel [8]</td>
<td>1332</td>
<td>4</td>
<td>91</td>
<td>-16</td>
<td>1325</td>
<td>800</td>
<td>1145</td>
</tr>
<tr>
<td>DQN Best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
<td>1075</td>
</tr>
</tbody>
</table>

Table 1: The upper table compares average total reward for various learning methods by running an ε-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ε-greedy policy with $\epsilon = 0.05$. 
COMP 790-125: Goals for today

- Introduce Reinforcement Learning
- Deep Reinforcement Learning