Today

- Learn to play games
- Reinforcement Learning
Playing Games: Super Mario
Reinforcement Learning Resources

- RL tutorial – on course website
Learning algorithms differ in the information available to learner

- **Supervised**: correct outputs
- **Unsupervised**: no feedback, must construct measure of good output
- **Reinforcement learning**

More realistic learning scenario:

- Continuous stream of input information, and actions
- Effects of action depend on state of the world
- Obtain reward that depends on world state and actions
  - not correct response, just some feedback
World described by a discrete, finite set of states and actions

At every time step $t$, we are in a state $s_t$, and we:

▶ Take an action $a_t$ (possibly null action)
▶ Receive some reward $r_{t+1}$
▶ Move into a new state $s_{t+1}$

Decisions can be described by a policy

▶ a selection of which action to take, based on the current state

Aim is to maximize the total reward we receive over time

Sometimes a future reward is discounted by $\gamma^{k-1}$, where $k$ is the number of time-steps in the future when it is received
Make this concrete by considering specific example

Consider the game tic-tac-toe:

- **reward**: win/lose/tie the game (+1/−1/0) [only at final move in given game]
- **state**: positions of X’s and O’s on the board
- **policy**: mapping from states to actions
  - based on rules of game: choice of one open position
- **value function**: prediction of reward in future, based on current state

In tic-tac-toe, since state space is tractable, can use a table to represent value function
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)
Acting Under Uncertainty

- The world and the actor may not be deterministic, or our model of the world may be incomplete.
- We assume the **Markov property**: the future depends on the past only through the current state.
- We describe the **environment** by a distribution over rewards and state transitions:
  \[ P(s_{t+1} = s', r_{t+1} = r' | s_t = s, a_t = a) \]

- The **policy** can also be non-deterministic:
  \[ P(a_t = a | s_t = s) \]

- Policy is not a fixed sequence of actions, but instead a conditional plan.
Markov Decision Problem (MDP): tuple \((S, A, P, \gamma)\) where \(P\) is

\[
P(s_{t+1} = s', r_{t+1} = r' | s_t = s, a_t = a)
\]

Standard MDP problems:

1. **Planning**: given complete Markov decision problem as input, compute policy with optimal expected return
2. **Learning**: Only have access to experience in the MDP, learn a near-optimal strategy
Example of Standard MDP Problem

1. **Planning**: given complete Markov decision problem as input, compute policy with optimal expected return

2. **Learning**: Only have access to experience in the MDP, learn a near-optimal strategy

We will focus on learning, but discuss planning along the way
If we knew how the world works (embodied in $P$), then the policy should be deterministic

- just select optimal action in each state

But if we do not have complete knowledge of the world, taking what appears to be the optimal action may prevent us from finding better states/actions

Interesting trade-off:

- immediate reward (exploitation) vs. gaining knowledge that might enable higher future reward (exploration)
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
What to Learn

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

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

- 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

- 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
\( \gamma = 0.9 \)

\( r(s, a) \) (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:

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

- 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 $\hat{Q}(s, a) \leftarrow 0$
- 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
Assume the robot is in state $s_1$; some of its current estimates of $Q$ are as shown; executes rightward move

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

\[
\leftarrow r + 0.9 \max\{63, 81, 100\} \leftarrow 90
\]

Notice that if rewards are non-negative, then $\hat{Q}$ values only increase from 0, approach true $Q$. 

[Diagram showing initial state $s_1$ with rewards and next state $s_2$ with updated $\hat{Q}$ value]
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
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?
- We redefine $V, Q$ based on probabilistic estimates, expected values of them:

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

and

\[
Q(s, a) = E[r(s, a) + \gamma V^*(\delta(s, a))] = E[r(s, a) + \gamma \sum_{s'} p(s'|s, a) \max_{a'} Q(s', a')]
\]
Nondeterministic 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 \( s \)