MDPs: recap

Summary of MDPs

- **Markov decision processes** (MDPs) cope with uncertainty

- Solutions are **policies** rather than paths

- **Policy evaluation** computes policy value (expected utility)

- **Value iteration** computes optimal value (maximum expected utility) and optimal policy

- Main technique: write recurrences $\to$ algorithm

Summary of Reinforcement Learning

- Online setting: learn and take actions in the real world!

- Monte Carlo: estimate transitions, rewards, Q-values from data

- Bootstrapping: update towards target that depends on estimate rather than just raw data

- This concludes the technical part of reinforcement learning.
- The first part is to understand the setup: we are taking good actions in the world both to get rewards under our current model, but also to collect information about the world so we can learn a better model. This exposes the fundamental exploration/exploitation tradeoff, which is the hallmark of reinforcement learning.
- We looked at several algorithms: model-based value iteration, model-free Monte Carlo, SARSA, and Q-learning. There were two complementary ideas here: using Monte Carlo approximation (approximating an expectation with a sample) and bootstrapping (using the model predictions to update itself).

Covering the unknown

Epsilon-greedy: balance the exploration/exploitation tradeoff

Function approximation: can generalize to unseen states

Challenges in reinforcement learning

Binary classification (sentiment classification, SVMs):

- Stateless, full supervision

Reinforcement learning (flying helicopters, Q-learning):

- Stateful, partial supervision

💡 **Key idea: partial supervision**

Reward feedback, but not given the solution directly.

💡 **Key idea: state**

Rewards depend on previous actions $\Rightarrow$ can have delayed rewards.

States and information

	stateless	**state**
full supervision	supervised learning (binary classification)	supervised imitation learning (structured prediction)
partial supervision	multi-armed bandits	reinforcement learning

- If we compare simple supervised learning (e.g., binary classification) and reinforcement learning, we see that there are two main differences that make learning harder.
- First, reinforcement learning requires the modeling of state. State means that the rewards across time steps are related. This results in **delayed rewards**, where we take an action and don't see the ramifications of it until much later.
- Second, reinforcement learning requires dealing with partial supervision (rewards). This means that we have to actively explore to acquire the necessary supervision.
- There are two problems that move towards reinforcement learning, each on a different axis. Structured prediction introduces the notion of state, but the problem is made easier by the fact that we have full supervision, which means that for every situation, we know which action sequence is the correct one; there is no need for exploration; we just have to update our weights to favor that correct path.
- Multi-armed bandits require dealing with partial supervision, but do not have the complexities of state. One can think of a multi-armed bandit problem as an MDP with unknown random rewards and one state. Exploration is necessary, but there is no temporal depth to the problem.

Deep reinforcement learning

just use a neural network for $\hat{Q}_{\text{opt}}(s, a)$, π_{opt}, T, etc

Playing Atari [Google DeepMind, 2013]:

- last 4 frames (images) $\Rightarrow$ 3-layer NN $\Rightarrow$ keystroke

- ϵ-greedy, train over 10M frames with 1M replay memory

- Human-level performance on some games (breakout), less good on others (space invaders)

- Recently, there has been a surge of interest in reinforcement learning due to the success of neural networks. If one is performing reinforcement learning in a simulator, one can actually generate tons of data, which is suitable for rich functions such as neural networks.

- A recent success story is DeepMind, who successfully trained a neural network to represent the $\hat{Q}_{\text{opt}}$ function for playing Atari games. The impressive part was the lack of prior knowledge involved: the neural network simply took as input the raw image and outputted keystrokes.

Deep reinforcement learning

- Policy gradient: train a policy $\pi(a \mid s)$ (say, a neural network) to directly maximize expected reward

- Google DeepMind's AlphaGo (2016), AlphaZero (2017)

- One other major class of algorithms we will not cover in this class is **policy gradient**. Whereas Q-learning attempts to estimate the value of the optimal policy, policy gradient methods optimize the policy to maximize expected reward, which is what we care about. Recall that when we went from model-based methods (which estimated the transition and reward functions) to model-free methods (which estimated the Q function), we moved closer to the thing that we want. Policy gradient methods take this farther and just focus on the only object that really matters at the end of the day, which is the policy that an agent follows.
- Policy gradient methods have been quite successful. For example, it was one of the components of AlphaGo, Google DeepMind's program that beat the world champion at Go. One can also combine value-based methods with policy-based methods in actor-critic methods to get the best of both worlds.
- There is a lot more to say about deep reinforcement learning. If you wish to learn more, UC Berkeley's CS285 course website offers a nice introduction.
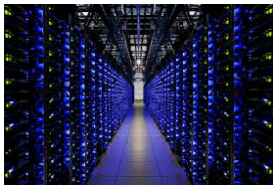
Applications
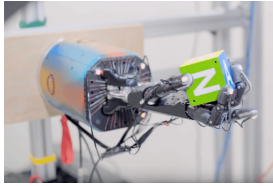
 Robotics Applications: learning dexterous manipulation, control helicopter to do maneuvers in the air
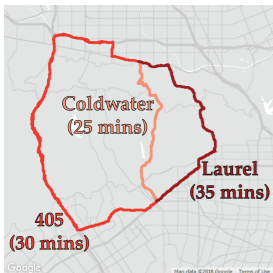
 Backgammon: TD-Gammon plays 1-2 million games against itself, human-level performance

 Games: DQN solving Atari Games, openAI Five playing Dota.

 Managing datacenters; actions: bring up and shut down machine to minimize time/cost

 Routing Autonomous Cars: bring down the total latency of vehicles on the road

- There are many other applications of RL, which range from robotics to game playing to other infrastructural tasks. One could say that RL is so general that anything can be cast as an RL problem.
- For a while, RL only worked for small toy problems or settings where there were a lot of prior knowledge / constraints. Deep RL — the use of powerful neural networks with increased compute — has vastly expanded the realm of problems which are solvable by RL.

Markov decision processes: against nature (e.g., Blackjack)
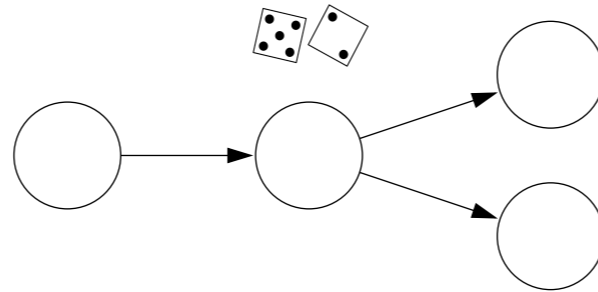
Next time...

Adversarial games: against opponent (e.g., chess)