CSC 311: Introduction to Machine Learning

Lecture 12 - Reinforcement Learning

Roger Grosse Rahul G. Krishnan Guodong Zhang

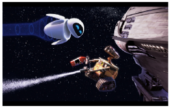
University of Toronto, Fall 2021

Reinforcement Learning Problem

- Recall: we categorized types of ML by how much information they provide about the desired behavior.
  - Supervised learning: labels of desired behavior
  - Unsupervised learning: no labels
  - Reinforcement learning: reward signal evaluating the outcome of past actions
- Bandit problems (Lecture 10) are a simple instance of RL where each decision is independent.
- More commonly, we focus on sequential decision making: an agent chooses a sequence of actions which each affect future possibilities available to the agent.

An agent

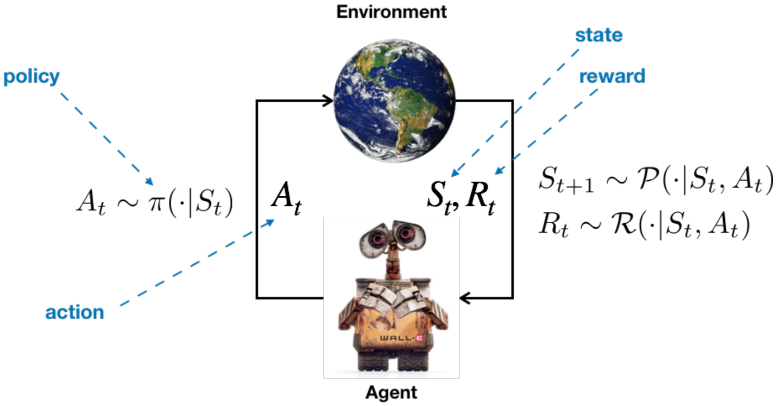
observes the world


takes an action and its states changes

with the goal of achieving long-term rewards.

Reinforcement Learning

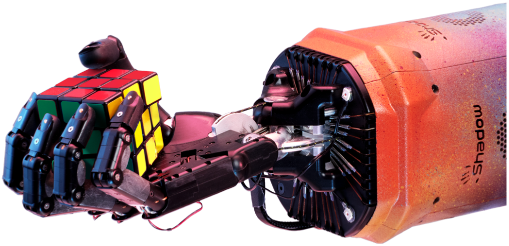
Most RL is done in a mathematical framework called a Markov Decision Process (MDP).


Environment
state
reward
policy
action
Agent
$A_t \sim \pi(\cdot|S_t)$
A_t
S_t, R_t
$S_{t+1} \sim \mathcal{P}(\cdot|S_t, A_t)$
$R_t \sim \mathcal{R}(\cdot|S_t, A_t)$

MDPs: States and Actions

- First let's see how to describe the dynamics of the environment.
- The state is a description of the environment in sufficient detail to determine its evolution.
  - Think of Newtonian physics.
    - What would be the state variables for a puck sliding on a frictionless table?
  - Markov assumption: the state at time $t+1$ depends directly on the state and action at time t, but not on past states and actions.
- To describe the dynamics, we need to specify the transition probabilities $\mathcal{P}(S_{t+1} \mid S_t, A_t)$.
- In this lecture, we assume the state is fully observable, a highly nontrivial assumption.

MDPs: States and Actions



- Suppose you're controlling a robot hand. What should be the set of states and actions?

- In general, the right granularity of states and actions depends on what you're trying to achieve.

MDPs: Policies

- The way the agent chooses the action in each step is called a policy.
- We'll consider two types:
  - Deterministic policy: $A_t = \pi(S_t)$ for some function $\pi : \mathcal{S} \to \mathcal{A}$
  - Stochastic policy: $A_t \sim \pi(\cdot \mid S_t)$ for some function $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. (Here, $\mathcal{P}(\mathcal{A})$ is the set of distributions over actions.)
- With stochastic policies, the distribution over rollouts, or trajectories, factorizes:

$$p(s_1, a_1, \ldots, s_T, a_T) = p(s_1)\, \pi(a_1 \mid s_1)\, \mathcal{P}(s_2 \mid s_1, a_1)\, \pi(a_2 \mid s_2) \cdots \mathcal{P}(s_T \mid s_{T-1}, a_{T-1})\, \pi(a_T \mid s_T)$$

- **Note:** the fact that policies need consider only the current state is a powerful consequence of the Markov assumption and full observability.
  - If the environment is partially observable, then the policy needs to depend on the history of observations.

MDPs: Rewards

- In each time step, the agent receives a reward from a distribution that depends on the current state and action

$$R_t \sim \mathcal{R}(\cdot \mid S_t, A_t)$$

- For simplicity, we'll assume rewards are deterministic, i.e.

$$R_t = r(S_t, A_t)$$

- What's an example where R_t should depend on A_t?
- The return determines how good was the outcome of an episode.
    - Undiscounted: $G = R_0 + R_1 + R_2 + \cdots$
    - Discounted: $G = R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots$
- The goal is to maximize the expected return, $\mathbb{E}[G]$.
- γ is a hyperparameter called the discount factor which determines how much we care about rewards now vs. rewards later.
    - What is the effect of large or small γ?

MDPs: Rewards

- How might you define a reward function for an agent learning to play a video game?
  - Change in score (why not current score?)
  - Some measure of novelty (this is sufficient for most Atari games!)
- Consider two possible reward functions for the game of Go. How do you think the agent's play will differ depending on the choice?
  - **Option 1:** +1 for win, 0 for tie, -1 for loss
  - **Option 2:** Agent's territory minus opponent's territory (at end)
- Specifying a good reward function can be tricky.
  `https://www.youtube.com/watch?v=tlOIHko8ySg`

Markov Decision Processes

- Putting this together, a Markov Decision Process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$.
  - $\mathcal{S}$: State space. Discrete or continuous
  - $\mathcal{A}$: Action space. Here we consider finite action space, i.e., $\mathcal{A} = \{a_1, \dots, a_{|\mathcal{A}|}\}$.
  - $\mathcal{P}$: Transition probability
  - $\mathcal{R}$: Immediate reward distribution
  - γ: Discount factor $(0 \leq \gamma < 1)$
- Together these define the environment that the agent operates in, and the objectives it is supposed to achieve.

Finding a Policy

- Now that we've defined MDPs, let's see how to find a policy that achieves a high return.
- We can distinguish two situations:
  - Planning: given a fully specified MDP.
  - Learning: agent interacts with an environment with unknown dynamics.
    - I.e., the environment is a black box that takes in actions and outputs states and rewards.
- Which framework would be most appropriate for chess? Super Mario?

Value Functions

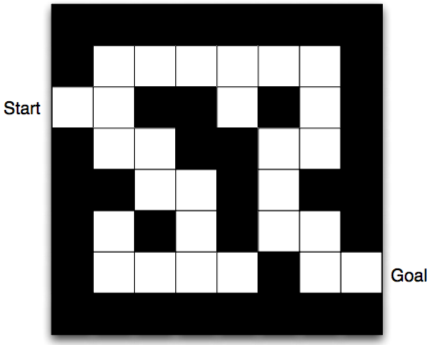
Value Function

- The value function V^π for a policy π measures the expected return if you start in state s and follow policy π.

$$V^\pi(s) \triangleq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s\right].$$

- This measures the desirability of state s.

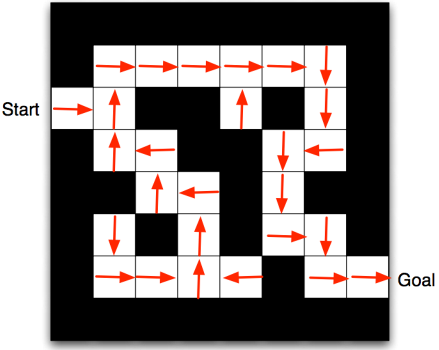
Value Function



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

[Slide credit: D. Silver]

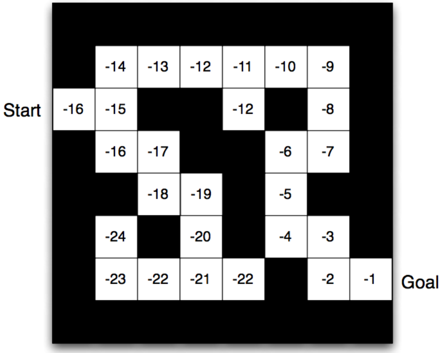
Value Function



- Arrows represent policy $\pi(s)$ for each state s

[Slide credit: D. Silver]

Value Function



- Numbers represent value $V^{\pi}(s)$ of each state s

[Slide credit: D. Silver]

Bellman equations

- The foundation of many RL algorithms is the fact that value functions satisfy a recursive relationship, called the Bellman equation:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid a, s)\, \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \right] \\ &= \sum_a \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid a, s)\, V^\pi(s') \right] \end{aligned}$$

- Viewing V^π as a vector (where entries correspond to states), define the Bellman backup operator T^π.

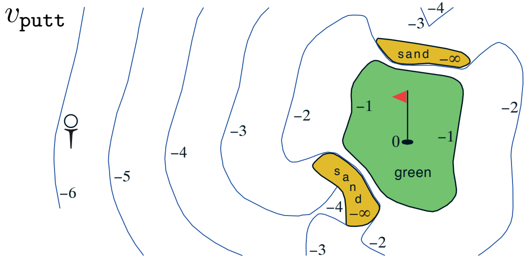
$$(T^\pi V)(s) \triangleq \sum_a \pi(a \mid s) \left[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid a, s)\, V(s') \right]$$

- The Bellman equation can be seen as a fixed point of the Bellman operator:

$$T^\pi V^\pi = V^\pi.$$

Value Function

A value function for golf:



— Sutton and Barto, *Reinforcement Learning: An Introduction*

State-Action Value Function

- A closely related but usefully different function is the state-action value function, or Q-function, Q^π for policy π, defined as:

$$Q^\pi(s,a) \triangleq \mathbb{E}_\pi\left[\sum_{k\geq 0} \gamma^k R_{t+k} \mid S_t = s, A_t = a\right].$$

- If you knew Q^π, how would you obtain V^π?

$$V^\pi(s) = \sum_a \pi(a \,|\, s)\, Q^\pi(s,a).$$

- If you knew V^π, how would you obtain Q^π?
  - Apply a Bellman-like equation:

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \,|\, a, s)\, V^\pi(s')$$

  - This requires knowing the dynamics, so in general it's not easy to recover Q^π from V^π.

State-Action Value Function

- Q^π satisfies a Bellman equation very similar to V^π (proof is analogous):

$$Q^\pi(s,a) = \underbrace{r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid a, s) \sum_{a'} \pi(a' \mid s') Q^\pi(s', a')}_{\triangleq (T^\pi Q^\pi)(s,a)}$$

Dynamic Programming and Value Iteration

Optimal State-Action Value Function

- Suppose you're in state s. You get to pick one action a, and then follow (fixed) policy π from then on. What do you pick?

$$\arg\max_a Q^\pi(s, a)$$

- If a deterministic policy π is optimal, then it must be the case that for any state s:

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

  otherwise you could improve the policy by changing $\pi(s)$. (see Sutton & Barto for a proper proof)

Optimal State-Action Value Function

- Bellman equation for optimal policy π^*:

$$\begin{aligned} Q^{\pi^*}(s,a) &= r(s,a) + \gamma \sum_{s'} \mathcal{P}(s', \mid s,a) Q^{\pi^\star}(s', \pi^\star(s')) \\ &= r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q^{\pi^\star}(s', a') \end{aligned}$$

- Now $Q^* = Q^{\pi^\star}$ is the optimal state-action value function, and we can rewrite the optimal Bellman equation without mentioning $\pi^\star$:

$$Q^*(s,a) = \underbrace{r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q^*(s',a')}_{\triangleq (T^* Q^*)(s,a)}$$

- Turns out this is *sufficient* to characterize the optimal policy. So we simply need to solve the fixed point equation $T^* Q^* = Q^*$, and then we can choose $\pi^*(s) = \arg\max_a Q^*(s,a)$.

Bellman Fixed Points

- **So far:** showed that some interesting problems could be reduced to finding fixed points of Bellman backup operators:
  - Evaluating a fixed policy π

$$T^{\pi} Q^{\pi} = Q^{\pi}$$

  - Finding the optimal policy

$$T^{*} Q^{*} = Q^{*}$$

- **Idea:** keep iterating the backup operator over and over again.

$$\begin{aligned} Q &\leftarrow T^{\pi} Q && \text{(policy evaluation)} \\ Q &\leftarrow T^{*} Q && \text{(finding the optimal policy)} \end{aligned}$$

  - We're treating Q^{π} or Q^{*} as a vector with $|\mathcal{S}| \cdot |\mathcal{A}|$ entries.
  - This type of algorithm is an instance of dynamic programming.

Bellman Fixed Points

- An operator f (mapping from vectors to vectors) is a contraction map if
$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq \alpha \|\mathbf{x}_1 - \mathbf{x}_2\|$$
for some scalar $0 \leq \alpha < 1$ and vector norm $\|\cdot\|$.
- Let $f^{(k)}$ denote f iterated k times. A simple induction shows
$$\|f^{(k)}(\mathbf{x}_1) - f^{(k)}(\mathbf{x}_2)\| \leq \alpha^k \|\mathbf{x}_1 - \mathbf{x}_2\|.$$
- Let $\mathbf{x}^*$ be a fixed point of f. Then for any $\mathbf{x}$,
$$\|f^{(k)}(\mathbf{x}) - \mathbf{x}^*\| \leq \alpha^k \|\mathbf{x} - \mathbf{x}_*\|.$$
- Hence, iterated application of f, starting from any $\mathbf{x}$, converges exponentially to a unique fixed point.

Finding the Optimal Value Function: Value Iteration

- Let's use dynamic programming to find Q^*.
- Value Iteration: Start from an initial function Q_1. For each $k = 1, 2, \ldots,$ apply

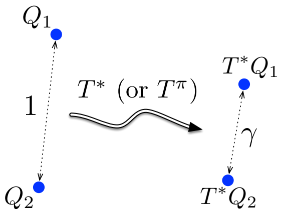
$$Q_{k+1} \leftarrow T^* Q_k$$

- Writing out the update in full,

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} Q_k(s',a')$$

- Observe: a fixed point of this update is exactly a solution of the optimal Bellman equation, which we saw characterizes the Q-function of an optimal policy.

Value Iteration



- **Claim:** The value iteration update is a contraction map:

$$\|T^*Q_1 - T^*Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$$

- $\|\cdot\|_\infty$ denotes the L^∞ norm, defined as:

$$\|\mathbf{x}\|_\infty = \max_i |x_i|$$

- If this claim is correct, then value iteration converges exponentially to the unique fixed point.
- The exponential decay factor is γ (the discount factor), which means longer term planning is harder.

Bellman Operator is a Contraction (optional)

$$
\begin{aligned}
|(T^*Q_1)(s,a) - (T^*Q_2)(s,a)| &= \left| \left[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) \max_{a'} Q_1(s',a') \right] - \right. \\
&\qquad \left. \left[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) \max_{a'} Q_2(s',a') \right] \right| \\
&= \gamma \left| \sum_{s'} \mathcal{P}(s' \mid s,a) \left[\max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \right] \right| \\
&\le \gamma \sum_{s'} \mathcal{P}(s' \mid s,a) \max_{a'} \left| Q_1(s',a') - Q_2(s',a') \right| \\
&\le \gamma \max_{s',a'} \left| Q_1(s',a') - Q_2(s',a') \right| \sum_{s'} \mathcal{P}(s' \mid s,a) \\
&= \gamma \max_{s',a'} \left| Q_1(s',a') - Q_2(s',a') \right| \\
&= \gamma \left\| Q_1 - Q_2 \right\|_\infty
\end{aligned}
$$

- This is true for *any* (s,a), so

$$\left\| T^*Q_1 - T^*Q_2 \right\|_\infty \le \gamma \left\| Q_1 - Q_2 \right\|_\infty ,$$

which is what we wanted to show.

Value Iteration Recap

- So far, we've focused on **planning**, where the dynamics are known.
- The optimal Q-function is characterized in terms of a Bellman fixed point update.
- Since the Bellman operator is a contraction map, we can just keep applying it repeatedly, and we'll converge to a unique fixed point.
- What are the limitations of value iteration?
  - assumes known dynamics
  - requires explicitly representing Q^* as a vector
    - $|\mathcal{S}|$ can be extremely large, or infinite
    - $|\mathcal{A}|$ can be infinite (e.g. continuous voltages in robotics)
- But value iteration is still a foundation for a lot of more practical RL algorithms.

Towards Learning

- Now let's focus on **reinforcement learning**, where the environment is unknown. How can we apply learning?
  1. Learn a model of the environment, and do planning in the model (i.e. model-based reinforcement learning)
     - You already know how to do this in principle, but it's very hard to get to work. Not covered in this course.
  2. Learn a value function (e.g. Q-learning, covered in this lecture)
  3. Learn a policy directly (e.g. policy gradient, not covered in this course)
- How can we deal with extremely large state spaces?
  - Function approximation: choose a parametric form for the policy and/or value function (e.g. linear in features, neural net, etc.)

Q-Learning

Monte Carlo Estimation

- Recall the optimal Bellman equation:

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{\mathcal{P}(s' \mid s,a)} \left[\max_{a'} Q^*(s',a') \right]$$

- **Problem:** we need to know the dynamics to evaluate the expectation
- Monte Carlo estimation of an expectation $\mu = \mathbb{E}[X]$: repeatedly sample X and update

$$\mu \leftarrow \mu + \alpha(X - \mu)$$

- **Idea:** Apply Monte Carlo estimation to the Bellman equation by sampling $S' \sim \mathcal{P}(\cdot \mid s,a)$ and updating:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Big[\underbrace{r(s,a) + \gamma \max_{a'} Q(S',a') - Q(s,a)}_{= \text{ Bellman error}} \Big]$$

- This is an example of temporal difference learning, i.e. updating our predictions to match our later predictions (once we have more information).

Monte Carlo Estimation

- **Problem:** Every iteration of value iteration requires updating Q for every state.
  - There could be lots of states
  - We only observe transitions for states that are visited
- **Idea:** Have the agent interact with the environment, and only update Q for the states that are actually visited.
- **Problem:** We might never visit certain states if they don't look promising, so we'll never learn about them.
- **Idea:** Have the agent sometimes take random actions so that it eventually visits every state.
  - ε-greedy policy: a policy which picks $\arg\max_a Q(s, a)$ with probability $1 - \varepsilon$ and a random action with probability ε. (Typical value: $\varepsilon = 0.05$)
- Combining all three ideas gives an algorithm called Q-learning.

Q-Learning with ε-Greedy Policy

- Parameters:
  - Learning rate α
  - Exploration parameter ε
- Initialize $Q(s, a)$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
- The agent starts at state S_0.
- For time step $t = 0, 1, ...,$
  - Choose A_t according to the ε-greedy policy, i.e.,

$$A_t \leftarrow \begin{cases} \operatorname{argmax}_{a \in \mathcal{A}} Q(S_t, a) & \text{with probability } 1 - \varepsilon \\ \text{Uniformly random action in } \mathcal{A} & \text{with probability } \varepsilon \end{cases}$$

  - Take action A_t in the environment.
  - The state changes from S_t to $S_{t+1} \sim \mathcal{P}(\cdot | S_t, A_t)$
  - Observe S_{t+1} and R_t (could be $r(S_t, A_t)$, or could be stochastic)
  - Update the action-value function at state-action (S_t, A_t):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

Exploration vs. Exploitation

- The ε-greedy is a simple mechanism for managing the exploration-exploitation tradeoff.

$$\pi_\varepsilon(S;Q) = \begin{cases} \operatorname{argmax}_{a\in\mathcal{A}} Q(S,a) & \text{with probability } 1-\varepsilon \\ \text{Uniformly random action in } \mathcal{A} & \text{with probability } \varepsilon \end{cases}$$

- The ε-greedy policy ensures that most of the time (probability $1-\varepsilon$) the agent exploits its incomplete knowledge of the world by chooses the best action (i.e., corresponding to the highest action-value), but occasionally (probability ε) it explores other actions.
- Without exploration, the agent may never find some good actions.
- The ε-greedy is one of the simplest, but widely used, methods for trading-off exploration and exploitation. Exploration-exploitation tradeoff is an important topic of research.

Examples of Exploration-Exploitation in the Real World

- Restaurant Selection
  - Exploitation: Go to your favourite restaurant
  - Exploration: Try a new restaurant
- Online Banner Advertisements
  - Exploitation: Show the most successful advert
  - Exploration: Show a different advert
- Oil Drilling
  - Exploitation: Drill at the best known location
  - Exploration: Drill at a new location
- Game Playing
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move

[Slide credit: D. Silver]

An Intuition on Why Q-Learning Works? (Optional)

- Consider a tuple (S, A, R, S'). The Q-learning update is

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S, A) \right].$$

- To understand this better, let us focus on its stochastic equilibrium, i.e., where the expected change in $Q(S, A)$ is zero. We have

$$\begin{aligned} &\mathbb{E} \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S, A) | S, A \right] = 0 \\ \Rightarrow &(T^* Q)(S, A) = Q(S, A) \end{aligned}$$

- So at the stochastic equilibrium, we have $(T^* Q)(S, A) = Q(S, A)$. Because the fixed-point of the Bellman optimality operator is unique (and is Q^*), Q is the same as the optimal action-value function Q^*.

Off-Policy Learning

- Q-learning update again:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S, A) \right].$$

- **Notice:** this update doesn't mention the policy anywhere. The only thing the policy is used for is to determine which states are visited.
- This means we can follow whatever policy we want (e.g. ε-greedy), and it still coverges to the optimal Q-function. Algorithms like this are known as off-policy algorithms, and this is an extremely useful property.
- Policy gradient (another popular RL algorithm, not covered in this course) is an on-policy algorithm. Encouraging exploration is much harder in that case.

Function Approximation

Function Approximation

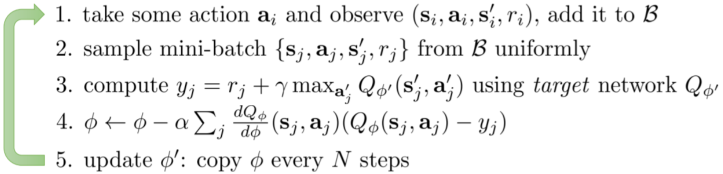
- So far, we've been assuming a tabular representation of Q: one entry for every state/action pair.
- This is impractical to store for all but the simplest problems, and doesn't share structure between related states.
- **Solution:** approximate Q using a parameterized function, e.g.
  - linear function approximation: $Q(\mathbf{s}, \mathbf{a}) = \mathbf{w}^\top \psi(\mathbf{s}, \mathbf{a})$
  - compute Q with a neural net
- Update Q using backprop:

$$t \leftarrow r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{t+1}, \mathbf{a})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(t - Q(\mathbf{s}, \mathbf{a})) \nabla_{\boldsymbol{\theta}} Q(\mathbf{s}_t, \mathbf{a}_t).$$

Function Approximation

- Approximating Q with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 ("deep Q-learning")
- They used a very small network by today's standards



```
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update $\phi'$: copy $\phi$ every $N$ steps
```

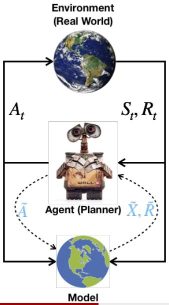
- Main technical innovation: store experience into a replay buffer, and perform Q-learning using stored experience
  - Gains sample efficiency by separating environment interaction from optimization — don't need new experience for every SGD update!

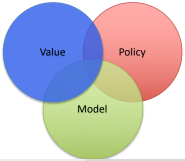
Atari

- Mnih et al., *Nature* 2015. Human-level control through deep reinforcement learning
- Network was given raw pixels as observations
- Same architecture shared between all games
- Assume fully observable environment, even though that's not the case
- After about a day of training on a particular game, often beat "human-level" performance (number of points within 5 minutes of play)
    - Did very well on reactive games, poorly on ones that require planning (e.g. Montezuma's Revenge)
- `https://www.youtube.com/watch?v=V1eYniJ0Rnk`
- `https://www.youtube.com/watch?v=4MlZncshy1Q`

Recap and Other Approaches

- All discussed approaches estimate the value function first. They are called value-based methods.
- There are methods that directly optimize the policy, i.e., policy search methods.
- Model-based RL methods estimate the true, but unknown, model of environment $\mathcal{P}$ by an estimate $\hat{\mathcal{P}}$, and use the estimate $\mathcal{P}$ in order to plan.
- There are hybrid methods.





Reinforcement Learning Resources

- Books:
  - Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, 2nd edition, 2018.
  - Csaba Szepesvari, Algorithms for Reinforcement Learning, 2010.
  - Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, 2010.
  - Dimitri P. Bertsekas and John N. Tsitsiklis, Neuro-Dynamic Programming, 1996.
- Courses:
  - Video lectures by David Silver
  - CIFAR and Vector Institute's Reinforcement Learning Summer School, 2018.
  - Deep Reinforcement Learning, CS 294-112 at UC Berkeley

Closing Thoughts

Overview

What this course focused on:

- Supervised learning: regression, classification
  - Choose model, loss function, optimizer
  - Parametric vs. nonparametric
  - Generative vs. discriminative
  - Iterative optimization vs. closed-form solutions
- Unsupervised learning: dimensionality reduction and clustering
- Reinforcement learning: value iteration

This lecture: what we left out, and teasers for other courses

CSC413 Teaser: Neural Nets

- This course covered some fundamental ideas, most of which are more than 10 years old.
- Big shift of the past decade: neural nets and deep learning
  - 2010: neural nets significantly improved speech recognition accuracy (after 20 years of stagnation)
  - 2012–2015: neural nets reduced error rates for object recognition by a factor of 6
  - 2016: a program called AlphaGo defeated the human Go champion
  - 2015–2018: neural nets learned to produce convincing high-resolution images
  - 2018–today: transformers demonstrate a sophisticated ability to generate natural language text and learn from few examples

CSC413 Teaser: Automatic Differentiation

- In this course, you derived update rules by hand
- Backprop is totally mechanical. Now we have automatic differentiation tools that compute gradients for you.
- In CSC413, you learn how an autodiff package can be implemented
    - Lets you do fancy things like differentiate through the whole training procedure to compute the gradient of validation loss with respect to the hyperparameters.
- With TensorFlow, PyTorch, etc., we can build much more complex neural net architectures that we could previously.

CSC413 Teaser: Beyond Scalar/Discrete Targets

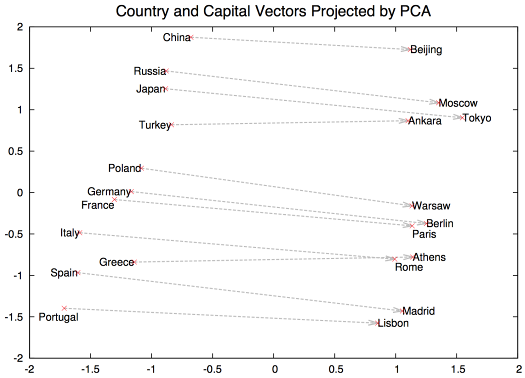
- This course focused on regression and classification, i.e. scalar-valued or discrete outputs
- That only covers a small fraction of use cases. Often, we want to output something more structured:
  - text (e.g. image captioning, machine translation)
  - dense labels of images (e.g. semantic segmentation)
  - graphs (e.g. molecule design)
- This used to be known as structured prediction, but now it's so routine we don't need a name for it.

CSC413 Teaser: Representation Learning

- We talked about neural nets as learning feature maps you can use for regression/classification
- More generally, want to learn a representation of the data such that mathematical operations on the representation are semantically meaningful
- Classic (decades-old) example: representing words as vectors
  - Measure semantic similarity using the dot product between word vectors (or dissimilarity using Euclidean distance)
  - Represent a web page with the average of its word vectors

CSC413 Teaser: Representation Learning

- Here's a linear projection of word representations for cities and capitals into 2 dimensions (part of a representation learned using word2vec)
- The mapping city $\rightarrow$ capital corresponds roughly to a single direction in the vector space:



Mikolov et al., 2018, "Efficient estimation of word representations in vector space"

CSC413 Teaser: Representation Learning

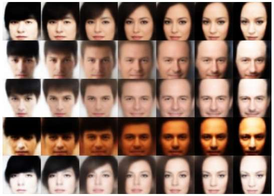
- In other words, vec(Paris) − vec(France) ≈ vec(London) − vec(England)
- This means we can analogies by doing arithmetic on word vectors:
  - e.g. "Paris is to France as London is to ______"
  - Find the word whose vector is closest to vec(France) − vec(Paris) + vec(London)
- Example analogies:

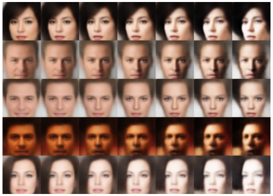
Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

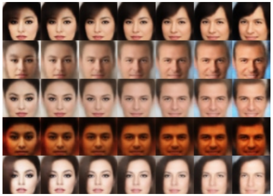
Mikolov et al., 2018, "Efficient estimation of word representations in vector space"

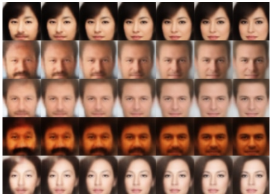
CSC413 Teaser: Representation Learning

One of the big goals is to learn *disentangled* representations, where individual dimensions tell you something meaningful


(a) Baldness (-6, 6)


(b) Face width (0, 6)

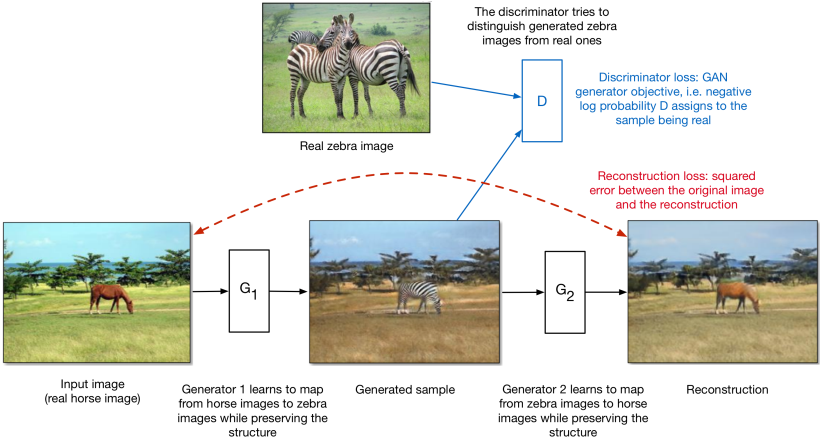

(c) Gender (-6, 6)


(d) Mustache (-6, 0)

Chen et al., 2018, “Isolating sources of disentanglement in variational autoencoders”

CSC413 Teaser: Image-to-Image Translation

Due to convenient autodiff frameworks, we can combine multiple neural nets together into fancy architectures. Here's the CycleGAN.

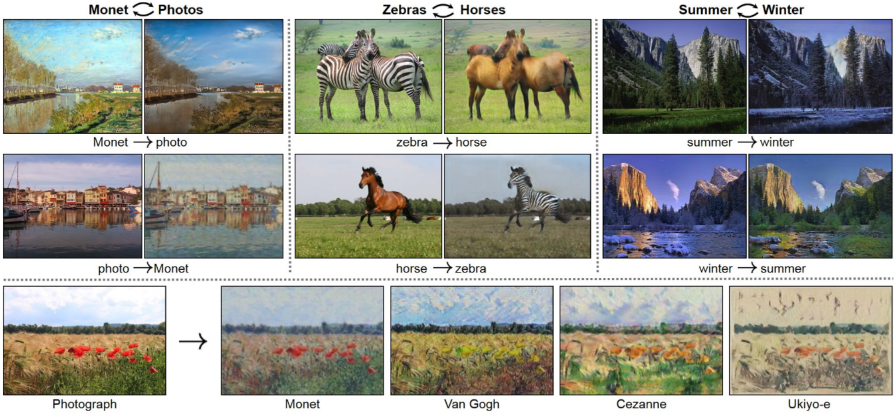

Real zebra image
The discriminator tries to distinguish generated zebra images from real ones
D
Discriminator loss: GAN generator objective, i.e. negative log probability D assigns to the sample being real
Reconstruction loss: squared error between the original image and the reconstruction
G1
G2
Input image (real horse image)
Generator 1 learns to map from horse images to zebra images while preserving the structure
Generated sample
Generator 2 learns to map from zebra images to horse images while preserving the structure
Reconstruction

Total loss = discriminator loss + reconstruction loss

Zhu et al., 2017, "Unpaired image-to-image translation using cycle-consistent adversarial networks"

CSC413 Teaser: Image-to-Image Translation

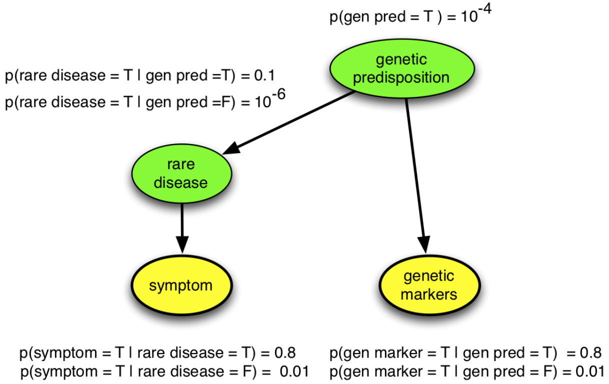
Style transfer problem: change the style of an image while preserving the content.



Data: Two unrelated collections of images, one for each style

CSC412 Teaser: Probabilistic Graphical Models

- In this course, we just scratched the surface of probabilistic models.
- Probabilistic graphical models (PGMs) let you encode complex probabilistic relationships between lots of variables.



p(gen pred = T) = 10^-4
genetic predisposition
p(rare disease = T | gen pred =T) = 0.1
p(rare disease = T | gen pred =F) = 10^-6
rare disease
symptom
genetic markers
p(symptom = T | rare disease = T) = 0.8
p(symptom = T | rare disease = F) = 0.01
p(gen marker = T | gen pred = T) = 0.8
p(gen marker = T | gen pred = F) = 0.01

Ghahramani, 2015, "Probabilistic ML and artificial intelligence"

CSC412 Teaser: PGM Inference

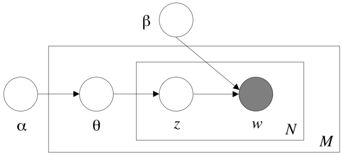
- We derived inference methods by inspection for some easy special cases (e.g. GDA, naïve Bayes)
- In CSC412, you'll learn much more general and powerful inference techniques that expand the range of models you can build
  - Exact inference using dynamic programming, for certain types of graph structures (e.g. chains)
  - Markov chain Monte Carlo
    - forms the basis of a powerful probabilistic modeling tool called Stan
  - Variational inference: try to approximate a complex, intractable, high-dimensional distribution using a tractable one
    - Try to minimze the KL divergence
    - Based on the same math from our EM lecture

CSC412 Teaser: Beyond Clustering

- We've seen unsupervised learning algorithms based on two ways of organizing your data
  - low-dimensional spaces (dimensionality reduction)
  - discrete categories (clustering)
- Other ways to organize/model data
  - hierarchies
  - dynamical systems
  - sets of attributes
  - topic models (each document is a mixture of topics)
- Motifs can be combined in all sorts of different ways

CSC412 Teaser: Beyond Clustering

Latent Dirichlet Allocation (LDA)



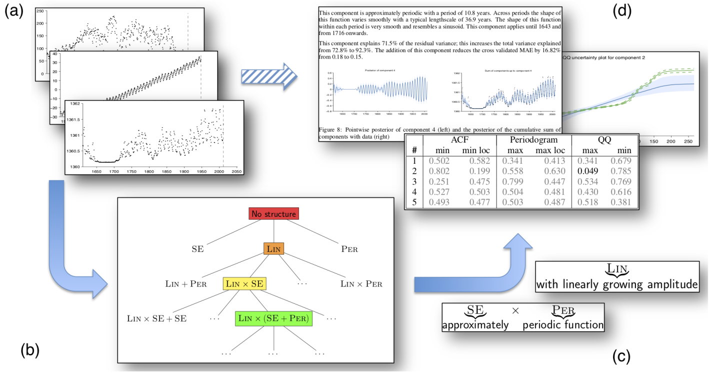
"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Blei et al., 2003, "Latent Dirichlet Allocation"

CSC412 Teaser: Automatic Statistician

Automatic search over Gaussian process kernel structures



$\underbrace{\text{LIN}}_{\text{with linearly growing amplitude}}$

$\underbrace{\text{SE}}_{\text{approximately}} \times \underbrace{\text{PER}}_{\text{periodic function}}$

This component is approximately periodic with a period of 10.8 years. Across periods the shape of this function varies smoothly with a typical lengthscale of 36.9 years. The shape of this function within each period is very smooth and resembles a sinusoid. This component applies until 1643 and from 1716 onwards.

This component explains 71.5% of the residual variance; this increases the total variance explained from 72.8% to 92.3%. The addition of this component reduces the cross validated MAE by 16.82% from 0.18 to 0.15.

Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

	ACF		Periodogram		QQ	
#	min	min loc	max	max loc	max	min
1	0.502	0.582	0.341	0.413	0.341	0.679
2	0.802	0.199	0.558	0.630	0.049	0.785
3	0.251	0.475	0.799	0.447	0.534	0.769
4	0.527	0.503	0.504	0.481	0.430	0.616
5	0.493	0.477	0.503	0.487	0.518	0.381

Duvenaud et al., 2013, “Structure discovery in nonparametric regression through compositional kernel search”

Image: Ghahramani, 2015, “Probabilistic ML and artificial intelligence”

Resources

Continuing with machine learning

- Courses
  - csc413/2516, “Neural Networks and Deep Learning”
  - csc412/2506, “Probabilistic Learning and Reasoning”
  - Various topics courses (varies from year to year)
- Videos from top ML conferences (NIPS/NeurIPS, ICML, ICLR, UAI)
  - Tutorials and keynote talks are aimed at people with your level of background (know the basics, but not experts in a subfield)
- Try to reproduce results from papers
  - If they’ve released code, you can use that as a guide if you get stuck
- Lots of excellent free resources avaiable online!