CS 189 / 289 Introduction to Machine Learning
Fall 2024 Jennifer Listgarten, Saeed Saremi HW7

Due 12/06/24 11:59 pm PT

- Homework 7 consists of both written and coding questions.
- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted for the written questions.
- In all of the questions, **show your work**, not just the final answer.

Deliverables:

1. Submit a PDF of your homework to the Gradescope assignment entitled "HW 7 Write-Up". Submit your code to the Gradescope assignment titled "HW 7 Code". **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
   - In your write-up, please state with whom you worked on the homework. This should be on its own page and should be the first page that you submit.
   - In your write-up, please copy the following statement and sign your signature underneath. If you are using LaTeX, you can type your full name underneath instead. We want to make it *extra* clear so that no one inadvertently cheats.

     "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."
   - **Replicate all of your code in an appendix**. Begin code for each coding question on a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from the appendix to correct questions.

1 Denoising simple distributions

In class, we discussed that it is easier to sample from a distribution if we (1) add some (usually Gaussian) noise to the distribution, (2) sample from the noisy distribution, and then (3) denoise the noisy sample to a clean sample. In this problem, we will walk through the theory of denoising noisy samples.

(a) Suppose that we have some clean data $x \in \mathbb{R}$ to which we add some random noise $z \in \mathbb{R}$ to produce noisy data $y = x + z$. We'd like to determine the denoising function $\varphi(y) : \mathbb{R} \to \mathbb{R}$ that minimizes the following least squares objective

$$\mathcal{L}(\varphi) = \mathbb{E}\left[(x - \varphi(y))^2\right],$$

where the expectation is over both x and y. We have previously proved that the minimizer is

$$\varphi^*(y) = \mathbb{E}[x \mid y].$$

Show that we can rewrite the above conditional expectation as

$$\varphi^*(y) = \frac{\int x\, p(y \mid x) p(x)\, dx}{p(y)}.$$

(b) Suppose that z is Gaussian with mean 0 and variance σ^2. Prove that $\varphi^*(y) = y + \sigma^2 \nabla_y \log p(y)$. This is a version of Tweedie's formula. Do not assume any intermediate results provided in lecture.

Hint: Start by expanding $\nabla_y \log p(y)$.

(c) In the previous problem, we proved that the denoised value that minimizes the least squares objective is not just y, even though we're adding zero-mean noise. To help us better understand the $\sigma^2 \nabla_y \log p(y)$ correction term, let's assume that $x \sim \mathcal{N}(0, \sigma^2 = 2)$ and $z \sim \mathcal{N}(0, \sigma^2 = 1)$.

1. Compute $\varphi^*(y)$ in this specific case.
2. Explain why this formula for $\varphi^*(y)$ makes sense.

(d) So far, we have worked out the theory for estimating denoised samples from noisy samples. In the case when our clean data came from a Gaussian as well, we were able to analytically compute the denoising function. But when the clean data comes from a complex high-dimensional distribution, we might train a neural network to learn the denoising function.

However, we still need some way to sample from the noisy distribution in order to then denoise the noisy sample. Explain how you can use the denoising function $\varphi^*(y)$ to sample from $p(y)$.

2 Coding GNNs with PyTorch Geometric (PyG)

Fill out the Colab notebook ***here***. Read through the entire notebook and complete all sections marked either `Your code here` or `TODO`.

This notebook will introduce you to the PyTorch Geometric library, which builds upon standard PyTorch and provides an API for easily training and evaluating Graph Neural Networks. We will walk through how this library represents graphs as tensors and then train two GNNs for node-level and graph-level classification tasks on datasets from the Open Graph Benchmark.

At the end, you should have two CSV files, `ogbn-arxiv_node.csv` and `ogbg-molhiv_graph.csv`, which will contain your model's predictions on both tasks. Submit these CSV files along, along with the completed notebook to the Gradescope coding assignment. **Also include any code you write in the appendix and select it when submitting your writeup to the Gradescope written assignment.**

3 Markov Decision Processes and Value Computations

In this question, you'll be reasoning about maximizing reward when sequentially making decisions in a Markov Decision Process (MDP), as well as about the Bellman equation - the central equation to solving and understanding MDPs.

Consider the classic gridworld MDP, where an agent starts in cell (1, 1) and navigates around its environment:

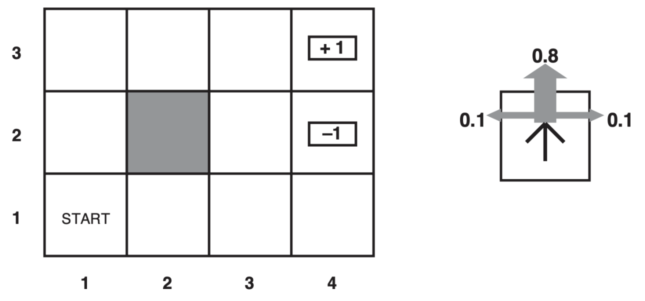

Figure 1: Gridworld MDP with stochastic transition probabilities.

In this world, the agent can take 4 actions in each cell: Up, Down, Left, or Right. The cells are indexed by (horizontal, vertical); that is, cell (4, 1) is in the bottom-right corner. The transition probabilities of the world work as follows: if the agent takes an action, it will move to the cell in the action's direction with probability 0.8, and it will slip to the action's relative right or left direction with probability 0.1 each. If the action (or slipping direction) is into a cell with no traversable tile (i.e., either a border or the wall in cell (2, 2)), that action keeps the agent at the cell it is currently at. For example, if the agent is in (3, 1), and it takes the action Up, it will land in cell (3, 2) with probability 0.8, in cell (2, 1) with probability 0.1, and (4, 1) with probability 0.1. If the agent is in cell (1, 3) and takes the action Right, it will land in cell (2, 3) with probability 0.8, in cell (1, 2) with probability 0.1, and (1, 3) with probability 0.1. When the agent reaches either of the defined reward states, at cells (4, 2) and (4, 3), the agent incurs the corresponding reward and the episode terminates.

Recall the Bellman equation for computing the *optimal value*, $V^*(s)$ of each state in an MDP, where we have a set of actions A, a set of states S, a reward value for each state $R(s)$, the transition dynamics of our world $P(s'|s,a)$, and a discount factor γ:

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s,a) V^*(s')$$

Lastly, we'll refer to policies as $\pi(s) = a$, where a policy π prescribes an action to take when in a given state.

(a) Consider an agent that starts in cell (1, 1) and takes the actions Up, Up in timesteps 1 and 2, respectively. Calculate which cells can be reached in each timestep from this action sequence and with what probabilities.

(b) Consider the reward function, $R(s)$, for all states that currently don't have a reward assigned to them (every cell except for (4, 2) and (4, 3).) Define what an optimal policy would be for an agent given the following reward values: (i.) $R(s) = 0$, (ii.) $R(s) = -2.0$, and (iii.) $R(s) = 1.0$. You may assume the discount factor to be a number arbitrarily close to 1, e.g. 0.9999. It may be helpful to draw out the gridworld and actions that should be taken at each state (remember that policies are defined over all states in an MDP!)

Note: You do not need to algorithmically compute the optimal policy. You must state the full policy for each of the three cases but only need to provide intuitive justifications.

(c) Sometimes MDPs are formulated with a reward function $R(s,a)$ that depends on the action taken, or with a reward function $R(s,a,s')$ that also depends on the outcome state. Write out the Bellman equations for the optimal value function for both the $R(s,a)$ and $R(s,a,s')$ formulations.

4 Jack's Car Rentals

This problem is inspired by an exercise from Sutton and Barto's Reinforcement Learning textbook.

Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting **the day after they are returned**. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved. We assume that the number of cars requested and returned at each day follow Poisson distributions. Suppose the rental requests and locations 1 and 2 follow Poisson distributions with rate parameters 3 and 4. Similarly, suppose the number of cars returned at locations 1 and 2 follow Poisson distributions with rate 3 and 2.

To simplify the problem slightly, we assume that there cannot be no more than 20 cars at each location (any additional cars are returned to the nationwide company and, thus, disappear completely from the problem), and a maximum of 5 cars can be moved from one location to the other in one night (once again, if the number of cars at a given location exceeds 20 after cars are moved overnight, the excess is removed from the system). We take the discount rate to be $\gamma = 0.9$ and formulate this as a finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net numbers of cars moved between the two locations overnight (a positive action indicates cars moving from location 1 to location 2 while a negative action indicates the reverse direction).

We define the policy $\pi(a \mid s)$ as the action Jack takes given the state s. In this problem, we will use Policy Iteration and Value Iteration to compute the optimal policy. **Add your code to the appendix of your writeup and also submit it to the Gradescope coding assignment.**

(a) Both algorithms depend on the Bellman equation: the Policy Iteration algorithm uses the Bellman expectation equation and the Value Iteration algorithm uses the Bellman optimality equation. In each case, we need to compute the sum below as a subroutine. Show that

$$Q(s,a) = \sum_{s',r} P(s',r \mid s,a)[r + \gamma V(s')] = \mathbb{E}[r \mid s,a] + \gamma\,\mathbb{E}[V(s') \mid s,a]$$

(b) Implement the Policy Iteration algorithm in the starter code, and use it to derive the optimal value function V^* and policy π^* that Jack should follow to maximize his returns. There are several helper functions that you will need to implement along the way.

Each location can have anywhere from 0 to 20 cars so the state space can be represented as $\mathcal{S} = [0,\dots,20] \times [0,\dots,20]$. As such, the policy $\pi : \mathcal{S} \to [-5,\dots,5]$ can be represented by a 21×21 2D matrix where π_{ij} is the action taken at state (i,j). The same goes for the value function $V : \mathcal{S} \to \mathbb{R}$. Plot the π and V matrices returned by the Policy Iteration algorithm after it converges. Describe what these plots represent. Do these align with your intuition?

(c) Implement the Value Iteration algorithm in the starter code, and use it to derive the optimal value function V^* and policy π^* that Jack should follow to maximize his returns.

Plot the π and V matrices returned by the Value Iteration algorithm. Does it return the same π and V as the Policy Iteration algorithm?

Now, suppose that one of Jack's employees at the first location rides a bus home each night and lives near the second location. They are happy to shuttle one car to the second location for free. Each additional car from location 1 to location 2 will still cost \$2, as do all the cars moved in the other direction. In addition, Jack has limited parking space at each location. If more than 10 cars are kept overnight at a location (after any moving of cars), then an additional cost of \$4 must be incurred to use a second parking lot (independent of how many cars are parked there) at said location.

These sorts of non-linearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimization methods other than dynamic programming.

(d) Repeat part (b) with the added non-linearities. Note that you only need to change how you compute the expected rewards, since the state transition probabilities remain the same.

How does the new policy compare against the previous policy? Qualitatively describe some of the differences in the new policy compared to the old policy, and what changes in the reward function might have encouraged them.

(e) Repeat part (c) with the added non-linearities. Does Value Iteration still return the same policy as Policy Iteration?

5 Kernels

For a function $k(x_i, x_j)$ to be a valid kernel, it suffices to show either of the following conditions is true:

1. k has an inner product representation: $\exists\ \Phi : \mathbb{R}^d \to \mathcal{H}$, where $\mathcal{H}$ is some (possibly infinite-dimensional) inner product space such that $\forall x_i, x_j \in \mathbb{R}^d,\ k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$.

2. For every sample $x_1, x_2, \ldots, x_n \in R^d$, the kernel matrix

$$\mathbf{K} = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & k(x_i, x_j) & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix}$$

   is positive semidefinite.

   Starting with part (c), you can use either condition (1) or (2) in your proofs.

(a) Show that the first condition implies the second one. That is, if for all x_i and x_j in $\mathbb{R}^d$ we have that $k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$, then the kernel matrix $\mathbf{K}$ is PSD.

(b) Let $\mathcal{X}$ be a finite set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, all of which are in $\mathbb{R}^d$. If the kernel matrix $\mathbf{K}$ generated from these vectors (as described in the second condition) is PSD, then there exists a feature map $\Phi_{\mathcal{X}}$: $\mathbb{R}^d \to \mathbb{R}^n$ such that $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi_{\mathcal{X}}(\mathbf{x}_i), \Phi_{\mathcal{X}}(\mathbf{x}_j) \rangle$ for all $\mathbf{x}_i$ and $\mathbf{x}_j$ in $\mathcal{X}$.

(c) Given a positive semidefinite matrix A, show that $k(x_i, x_j) = x_i^\top A x_j$ is a valid kernel.

(d) Suppose k_1 and k_2 are valid kernels with feature maps $\Phi_1\colon \mathbb{R}^d \to \mathbb{R}^p$ and $\Phi_2\colon \mathbb{R}^d \to \mathbb{R}^q$ respectively, for some finite positive integers p and q. Construct a feature map for the product of the two kernels in terms of Φ_1 and Φ_2, i.e. construct Φ_3 such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ we have

$$k(\mathbf{x}_1, \mathbf{x}_2) = k_1(\mathbf{x}_1, \mathbf{x}_2) k_2(\mathbf{x}_1, \mathbf{x}_2) = \langle \Phi_3(\mathbf{x}_1), \Phi_3(\mathbf{x}_2) \rangle.$$

Hint: It may feel more natural to define Φ_3 to output a matrix rather than a vector. If you do this, you should be able to make use of the Frobenius inner product: $\langle A, B \rangle = \mathrm{tr}(A^\top B)$.

6 Kernel Ridge Regression: Theory

(a) As we already know, the following procedure gives us the solution to ridge regression in feature space:

$$\operatorname*{argmin}_{\mathbf{w}} \|\mathbf{\Phi w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2 \tag{1}$$

Here $\mathbf{\Phi} \in \mathbb{R}^{n\times d}$ (where n is the number of datapoints) are the features extracted from our data $\mathbf{X} \in \mathbb{R}^{n\times k}$ by the function $\phi\colon \mathbb{R}^k \to \mathbb{R}^d$. Concretely, if $\mathbf{\Phi}_i$ denotes the i^{th} row of $\mathbf{\Phi}$, then $\mathbf{\Phi}_i = \phi(\mathbf{x}_i)$.

Recall that the solution to ridge regression is given by

$$\hat{\mathbf{w}} = (\mathbf{\Phi}^\top\mathbf{\Phi} + \lambda I_d)^{-1}\mathbf{\Phi}^\top y.$$

Show that we can rewrite $\hat{\mathbf{w}}$ as

$$\hat{\mathbf{w}} = \mathbf{\Phi}^\top(\mathbf{\Phi\Phi}^\top + \lambda I_n)^{-1}y.$$

(b) The prediction for a test point $\mathbf{x}$ is given by $\phi(\mathbf{x})^\top\hat{\mathbf{w}}$, where $\hat{\mathbf{w}}$ is the solution to (1). In this part you will show how $\phi(\mathbf{x})^\top\hat{\mathbf{w}}$ can be computed using only the kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top\phi(\mathbf{x}_j)$. Use the result from part (a) to show that

$$\phi(\mathbf{x})^\top\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i),$$

where $\boldsymbol{\alpha} = (\mathbf{K} + \lambda I)^{-1}\mathbf{y}$ is a vector in $\mathbb{R}^n$ with $\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$. In doing so, you will have proven that the prediction is a weighted linear combination of the kernels $k(\mathbf{x}, \cdot)$.

7 Honor Code

1. **List all collaborators. If you worked alone, then you must explicitly state so.**
2. **Declare and sign the following statement**:

   "I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

   Signature : ________________________________

   While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that the consequences of academic misconduct are *particularly severe*!