CS 189 / 289 Introduction to Machine Learning

Fall 2023 Jennifer Listgarten, Jitendra Malik

HW7

Due 12/01/23 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 Graph Dynamics

This problem is borrowed from CS 182.

Generally, an unnormalized adjacency matrix between the nodes of an undirected graph is given by:

$$A_{ij} = \begin{cases} 1 & \text{there is an edge between node } i \text{ and node } j \\ 0 & \text{otherwise} \end{cases}$$

This will be a symmetric matrix for undirected graphs. For a directed graph, we have:

$$A_{ij} = \begin{cases} 1 & \text{there is an edge from node } i \text{ to node } j \\ 0 & \text{otherwise} \end{cases}$$

This need not be symmetric for a directed graph, and is in fact typically not a symmetric matrix when we are thinking about directed graphs.

Some graph neural network methods operate on the full adjacency matrix. Others, such as those discussed here `https://distill.pub/2021/gnn-intro/`, at each layer apply the same local operation to each node based on inputs from its neighbors.

This problem is designed to:

- show connections between these methods
- show that for a positive integer k, the matrix A^k, has an interesting interpretation. That is, the entry in row i and column j of A^k gives the number of walks of length k (i.e., a collection of k edges) leading from vertex i to vertex j.

To do this, let's consider a very simple deep linear network that is built on an underlying graph with n vertices. In the 0th layer, each node is represented by a one-hot vector of its own index, i.e., node i in the graph is represented by an n-dimensional vector that has a 1 in the ith index and 0s everywhere else.

The weights connecting node i in layer k to node j in layer $k+1$ of this neural network are simply 1 if vertices i and j are connected in the underlying graph and are 0 if those vertices are not connected in the underlying graph. We assume that there is no bias vector for any of the layers in this network. Since this is a linear network, all activation functions are identity and the forward pass proceeds as normal (recall homework 3).

1. Let A be the $n \times n$ size adjacency matrix for the underlying graph where entry $A_{ij} = 1$ if vertices i and j are connected in the graph and 0 otherwise. Given node j as input, write the output of this network at layer k in terms of the matrix A.

2. Here is some helpful notation: Let $V(i)$ be the set of vertices that are connected to vertex i in the graph. Let $L_k(i, j)$ be the number of distinct paths that go from vertex i to vertex j in the

graph, where the number of edges traversed in the path is exactly k. Recall that a path from i to j in a graph is a sequence of vertices that starts with i and ends with j, and for which every successive vertex in the sequence is connected by an edge in the graph. The length of the path is 1 less than the number of vertices in the corresponding sequence. **Show that the ith output of node j at layer k in the network is** $L_k(i, j)$, where by convention, there is exactly 1 path of length 0 that starts at each node and ends up at itself (in math notation, $L_0(i, i) = 1$ for all i).

hint: this is equivalent to showing that $L_k(i, j) = A^k_{ij}$.

hint: apply induction on k.

3. The structure of the neural network in this problem is compatible with a straightforward linear graph neural network since the operations done (just summing) are locally permutation-invariant at the level of each node and can be viewed as essentially doing the exact same thing at each vertex in the graph based on the inputs coming from their neighbors. This is called ”aggregation” in the language of graph neural nets. In the case of the computations in previous parts, **what is the update function that takes the aggregated inputs from neighbors and results in the output for this node?**

4. The simple GNN described in the previous part counts paths in the graph. If we were to replace the sum aggregation with the max aggregation, **what is the interpretation of the outputs of node j at layer k**?

2 Coding GNNs with PyTorch Geometric (PyG)

Download the notebook called `GNN Colab.ipynb` and upload it to Google Colab. 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 is 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 called `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 Expectation Maximization (EM) Algorithm: A closer look!

Note that this problem appears lengthy. Don't worry! Most of these parts are of a "tutorial" nature and there are 2 "demo" parts at the end. This leads to the lengthy appearance of the overall problem, even though it is not conceptually actually that long. Hopefully you enjoy working on this homework and learn several concepts in greater detail.

In this problem, we will work on different aspects of the EM algorithm to reinforce your understanding of this very important algorithm. This was touched upon back when we were discussing Gaussian Mixture models for clustering but this algorithm is used for so many applications that we don't think if would be fair to let you finish CS 189 without learning about it at least once.

For the first few parts, we work with the following one-dimensional mixture model:

$$Z \sim \begin{cases} 1 & \text{w.p. } \frac{1}{2} \\ 2 & \text{w.p. } \frac{1}{2} \end{cases}$$

$$X|Z = 1 \sim \mathcal{N}(\mu_1, \sigma_1^2), \quad \text{and}$$

$$X|Z = 2 \sim \mathcal{N}(\mu_2, \sigma_2^2).$$

Here, Z denotes the hidden label of the Gaussian distribution from which X is drawn. $p(Z = 1) = 0.5, p(Z = 2) = 0.5$ — the hidden label is a fair coin toss. X has two different Gaussian distributions based on what Z is. In other words, we have an equally weighted 2-mixture of Gaussians, where the variances and means for both elements of the mixture are unknown. (Note that the mixture weights are given to you. In general, even those would be unknown.)

For a given set of parameters, we represent the likelihood of (X, Z) by $p(X, Z; \boldsymbol{\theta})$ and its log-likelihood by $\ell(X, Z; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is used to denote the set of all unknown parameters $\boldsymbol{\theta} = \{\mu_1, \mu_2, \sigma_1, \sigma_2\}$.

Given a dataset presumed to consist of i.i.d. samples of only $(x_i, i = 1, \ldots, n)$ (and no labels z_i), our goal is to iteratively approximate the maximum-likelihood estimate of the parameters $\boldsymbol{\theta}$. In the first few parts, we walk you through one way of achieving this, namely the EM algorithm.

(a) **Write down the expression for the joint likelihood $p(X = x, Z = 1; \boldsymbol{\theta})$ and $p(X = x, Z = 2; \boldsymbol{\theta})$. What is the marginal likelihood $p(X = x; \boldsymbol{\theta})$ and the log-likelihood $\ell(X = x; \boldsymbol{\theta})$?**

(b) Now we are given an i.i.d. dataset where the label values Z are unobserved, i.e., we are given a set of n data points $\{x_i, i = 1, \ldots, n\}$. **Derive the expression for the log-likelihood $\ell(X_1 = x_1, \ldots, X_n = x_n; \boldsymbol{\theta})$ of a given dataset $\{x_i\}_{i=1}^n$.**

(c) Let q denote a possible distribution on the (hidden) labels $\{Z_i\}_{i=1}^n$ given by

$$q(Z_1 = z_1, \ldots, Z_n = z_n) = \prod_{i=1}^{n} q_i(Z_i = z_i). \tag{1}$$

Note that since $z_i \in \{1, 2\}$, q has n parameters, namely $\{q_i(Z_i = 1), i = 1, \ldots, n\}$. More generally if Z took values in $\{1, \ldots, K\}$, q would have $n(K - 1)$ parameters. To simplify notation, from now on, we use the notation $\ell(x;\boldsymbol{\theta}) := \ell(X = x;\boldsymbol{\theta})$ and $p(x,k;\boldsymbol{\theta}) := p(X = x, Z = k;\boldsymbol{\theta})$. **Show that for a given point x_i, we have**

$$\ell(x_i;\boldsymbol{\theta}) \geq \mathcal{F}_i(\boldsymbol{\theta};q_i) := \underbrace{\sum_{k=1}^{2} q_i(k)\log p(x_i,k;\boldsymbol{\theta})}_{\mathcal{L}(x_i;\boldsymbol{\theta},q_i)} + \underbrace{\sum_{k=1}^{2} q_i(k)\log\left(\frac{1}{q_i(k)}\right)}_{H(q_i)}, \tag{2}$$

where $H(q_i)$ denotes the Shannon-entropy of the distribution q_i. Thus **conclude that we obtain the following lower bound on the log-likelihood:**

$$\ell(\{x_i\}_{i=1}^{n};\boldsymbol{\theta}) \geq \mathcal{F}(\boldsymbol{\theta};q) := \sum_{i=1}^{n} \mathcal{F}_i(\boldsymbol{\theta};q_i). \tag{3}$$

Notice that the right hand side of the bound depends on q while the left hand side does not. *Hint: Jensen's inequality (for concave functions f, we know $f(\mathbb{E}[X]) \geq \mathbb{E}(f(X))$ since a line joining two points is below the function)*

Side note: hidden variables like Z are also called latent variables. In latent variable models, the paradigm of bounding the true log-likelihood by a lower bound, and optimizing it instead, is fairly common. Some models that use this idea include VAEs (variational autoencoders) and diffusion models.

(d) The EM algorithm can be considered a coordinate-ascent[1] algorithm on the lower bound $\mathcal{F}(\boldsymbol{\theta};q)$ derived in the previous part, where we ascend with respect to $\boldsymbol{\theta}$ and q in an alternating fashion. More precisely, one iteration of the EM algorithm is made up of 2-steps:

$$q^{t+1} = \arg\max_{q} \mathcal{F}(\boldsymbol{\theta}^t;q) \tag{E-step}$$

$$\boldsymbol{\theta}^{t+1} \in \arg\max_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{\theta};q^{t+1}). \tag{M-step}$$

Given an estimate $\boldsymbol{\theta}^t$, the previous part tells us that $\ell(\{x_i\}_{i=1}^{n};\boldsymbol{\theta}^t) \geq \mathcal{F}(\boldsymbol{\theta}^t;q)$. **Verify that equality holds in this bound if we plug in $q(Z_1 = z_1,\ldots,Z_n = z_n) = \prod_{i=1}^{n} p(Z = z_i|X = x_i;\boldsymbol{\theta}^t)$ and hence we can conclude that**

$$q^{t+1}(Z_1 = z_1,\ldots,Z_n = z_n) = \prod_{i=1}^{n} p(Z = z_i|X = x_i;\boldsymbol{\theta}^t). \tag{4}$$

is a valid maximizer for the problem $\max_q \mathcal{F}(\boldsymbol{\theta}^t;q)$ and hence a valid E-step update.

(e) Using equation (4) from above and the relation (1), we find that the E-step updates can be re-written as

$$q_i^{t+1}(Z_i = k) = p(Z = k|X = x_i;\boldsymbol{\theta}^t).$$

[1] A coordinate-ascent algorithm is just one that fixes some coordinates and maximizes the function with respect to the others as a way of taking iterative improvement steps. (By contrast, gradient-descent algorithms tend to change all the coordinates in each step, just by a little bit.)

Using this relation, show that the E-step updates for the 2-mixture case are given by

$$q_i^{t+1}(Z_i = 1) = \frac{\frac{1}{\sigma_1}\exp\left(-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}\right)}{\frac{1}{\sigma_1}\exp\left(-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}\right) + \frac{1}{\sigma_2}\exp\left(-\frac{(x_i-\mu_2)^2}{2\sigma_2^2}\right)}, \quad \textbf{and}$$

$$q_i^{t+1}(Z_i = 2) = \frac{\frac{1}{\sigma_2}\exp\left(-\frac{(x_i-\mu_2)^2}{2\sigma_2^2}\right)}{\frac{1}{\sigma_1}\exp\left(-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}\right) + \frac{1}{\sigma_2}\exp\left(-\frac{(x_i-\mu_2)^2}{2\sigma_2^2}\right)} = 1 - q_i^{t+1}(Z_i = 1).$$

Explain intuitively why these updates make sense.

(f) We now discuss the M-step. Using the definitions from equations (2) and (3), we have that

$$\mathcal{F}(\boldsymbol{\theta}; q^{t+1}) = \sum_{i=1}^{n}(\mathcal{L}(x_i; \boldsymbol{\theta}, q_i^{t+1}) + H(q_i)) = H(q^{t+1}) + \sum_{i=1}^{n}\mathcal{L}(\mathbf{x}_i; \boldsymbol{\theta}, q_i^{t+1}),$$

where we have used the fact that entropy in this case is given by $H(q^{t+1}) = \sum_{i=1}^{n} H(q_i^{t+1})$. Notice that although (as computed in previous part), q^{t+1} depends on $\boldsymbol{\theta}^t$, the M-step only involves maximizing $\mathcal{F}(\boldsymbol{\theta}; q^{t+1})$ with respect to just the parameter $\boldsymbol{\theta}$ while keeping the parameter q^{t+1} fixed. Now, noting that the entropy term $H(q^{t+1})$ does not depend on the parameter $\boldsymbol{\theta}$, we conclude that the M-step simplifies to solving for

$$\arg\max_{\boldsymbol{\theta}} \underbrace{\sum_{i=1}^{n}\mathcal{L}(\mathbf{x}_i; \boldsymbol{\theta}, q_i^{t+1})}_{=:\mathcal{L}(\boldsymbol{\theta}; q^{t+1})}.$$

For this and the next few parts, we use the simplified notation

$$q_i^{t+1} := q_i^{t+1}(Z_i = 1) \quad \text{and} \quad 1 - q_i^{t+1} := q^{t+1}(Z_i = 2)$$

and recall that $\boldsymbol{\theta} = (\mu_1, \mu_2, \sigma_1, \sigma_2)$. **Show that the expression for $\mathcal{L}(\boldsymbol{\theta}; q^{t+1})$ for the 2-mixture case is given by**

$$\begin{aligned}&\mathcal{L}((\mu_1, \mu_2, \sigma_1, \sigma_2); q^{t+1}) \\ &\quad = C - \sum_{i=1}^{n}\left[q_i^{t+1}\left(\frac{(x_i-\mu_1)^2}{2\sigma_1^2} + \log\sigma_1\right) + (1 - q_i^{t+1})\left(\frac{(x_i-\mu_2)^2}{2\sigma_2^2} + \log\sigma_2\right)\right],\end{aligned}$$

where C is a constant that does not depend on θ or q^{t+1}.

(g) Using the expression from the previous part, it is easy to show (make sure you know how to do this) that the gradients of $\mathcal{L}(\boldsymbol{\theta}; q^{t+1})$ with respect to $\mu_1, \mu_2, \sigma_1, \sigma_2$ are given by

$$\frac{\partial\mathcal{L}}{\partial\mu_1} = -\frac{\sum_{i=1}^{n} q_i^{t+1}(\mu_1 - x_i)}{\sigma_1^2}, \qquad \frac{\partial\mathcal{L}}{\partial\mu_2} = -\frac{\sum_{i=1}^{n}(1 - q_i^{t+1})(\mu_2 - x_i)}{\sigma_2^2},$$

$$\frac{\partial\mathcal{L}}{\partial\sigma_1} = \frac{\sum_{i=1}^{n} q_i^{t+1}(x_i - \mu_1)^2}{\sigma_1^3} - \frac{\sum_{i=1}^{n} q_i^{t+1}}{\sigma_1}, \quad \frac{\partial\mathcal{L}}{\partial\sigma_2} = \frac{\sum_{i=1}^{n}(1 - q_i^{t+1})(x_i - \mu_2)^2}{\sigma_2^3} - \frac{\sum_{i=1}^{n}(1 - q_i^{t+1})}{\sigma_2}.$$

Typically, the M-step updates are computed using the stationary points for $F(\boldsymbol{\theta}; q^{t+1})$. Using the expressions from previous parts and setting these gradients to zero, **conclude that the M-step updates are given by**

$$\mu_1^{t+1} = \frac{\sum_{i=1}^n q_i^{t+1} x_i}{\sum_{i=1}^n q_i^{t+1}}, \qquad \mu_2^{t+1} = \frac{\sum_{i=1}^n (1 - q_i^{t+1}) x_i}{\sum_{i=1}^n (1 - q_i^{t+1})},$$

$$(\sigma_1^2)^{(t+1)} = \frac{\sum_{i=1}^n q_i^{t+1} (x_i - \mu_1^{t+1})^2}{\sum_{i=1}^n q_i^{t+1}}, \qquad (\sigma_2^2)^{(t+1)} = \frac{\sum_{i=1}^n (1 - q_i^{t+1})(x_i - \mu_2^{t+1})^2}{\sum_{i=1}^n (1 - q_i^{t+1})}$$

Explain intuitively why these updates make sense.

(h) For the next few parts, we further simplify the scalar mixture model. We work with the following simpler one-dimensional mixture model that has only a single unknown parameter:

$$Z \sim \begin{cases} 1 & \text{w.p. } \frac{1}{2} \\ 2 & \text{w.p. } \frac{1}{2} \end{cases}$$

$$X|Z = 1 \sim \mathcal{N}(\mu, 1), \quad \text{and}$$

$$X|Z = 2 \sim \mathcal{N}(-\mu, 1),$$

where Z denotes the label of the Gaussian from which X is drawn. Given a set of observations only for X (i.e., the labels are unobserved), our goal is to infer the maximum-likelihood parameter μ. Using computations similar to part (a), we can conclude that the likelihood function in this simpler set-up is given by

$$p(X = x; \mu) = \frac{1}{2} \frac{e^{-\frac{1}{2}(x-\mu)^2}}{\sqrt{2\pi}} + \frac{1}{2} \frac{e^{-\frac{1}{2}(x+\mu)^2}}{\sqrt{2\pi}}.$$

For a given dataset of i.i.d. samples $\{x_i, i = 1, \ldots, n\}$, **what is the log-likelihood $\ell(\{x_i\}_{i=1}^n; \mu)$ as a function of μ?**

(i) We now discuss EM updates for the set up introduced in the previous part. Let μ_t denote the estimate for μ at time t. First, we derive the E-step updates. Using part (d) (equation (4)) and part (e), **show that the E-step updates simplify to**

$$q_i^{t+1}(Z_i = 1) = \frac{\exp\Big(-(x_i - \mu_t)^2/2\Big)}{\exp(-(x_i - \mu_t)^2/2) + \exp(-(x_i + \mu_t)^2/2)}.$$

Notice that these updates can also be derived by plugging in $\mu_1 = \mu$ and $\mu_2 = -\mu$ in the updates given in part (e).

(j) Next, we derive the M-step update. Note that we can NOT simply plug in $\mu_1 = \mu$ in the updates obtained in part (h), because the parameters are shared between the two mixtures. However, we can still make use of some of our previous computations for this simpler case. Plugging in $\mu_1 = \mu$ and $\mu_2 = -\mu$, $\sigma_1 = \sigma_2 = 1$ in part (f), **show that the objective for the M-step is given by**

$$\mathcal{L}(\mu; q^{t+1}) = C - \sum_{i=1}^n \left(q_i^{t+1} \frac{(x_i - \mu)^2}{2} + (1 - q_i^{t+1}) \frac{(x_i + \mu)^2}{2} \right).$$

where C is a constant independent of μ. Compute the expression for the gradient $\frac{d}{d\mu}(\mathcal{L}(\mu; q^{t+1}))$. And by setting the gradient to zero, conclude that the M-step update at time $t+1$ is given by

$$\mu_{t+1} = \frac{1}{n}\sum_{i=1}^{n}(2q_i^{t+1} - 1)x_i.$$

(k) Let us now consider a direct optimization method to estimate the MLE for μ: Doing a gradient ascent algorithm directly on the complete log-likelihood function $\ell(\{x_i\}_{i=1}^n; \mu)/n$ (scaling with n is natural here since the log-likelihood is a sum.). **Compute the gradient $\frac{d}{d\mu}(\ell(\{x_i\}_{i=1}^n; \mu)/n)$ and show that it is equal to**

$$\frac{d}{d\mu}\left(\frac{1}{n}\ell(\{x_i\}_{i=1}^n; \mu)\right) = \left[\frac{1}{n}\sum_{i=1}^{n}(2w_i(\mu) - 1)x_i\right] - \mu, \quad \textbf{where} \quad w_i(\mu) = \frac{e^{-\frac{(x_i-\mu)^2}{2}}}{e^{-\frac{(x_i-\mu)^2}{2})} + e^{-\frac{(x_i+\mu)^2}{2})}}.$$

Finally conclude that the gradient ascent scheme with step size α is given by

$$\begin{aligned}\mu_{t+1}^{\mathbf{GA}} &= \mu_t^{\mathbf{GA}} + \alpha\frac{d}{d\mu}\ell(\{x_i\}_{i=1}^n; \mu)\bigg|_{\mu=\mu_t^{\mathbf{GA}}} \\ &= (1-\alpha)\mu_t^{\mathbf{GA}} + \alpha\left[\frac{1}{n}\sum_{i=1}^{n}(2w_i(\mu_t^{\mathbf{GA}}) - 1)x_i\right].\end{aligned}$$

(l) **Comment on the similarity or dissimilarity between the EM and gradient ascent (GA) updates derived in the previous two parts.** Refer to the corresponding jupyter notebook. You can run the two algorithms for the simpler one-dimensional mixture with a single unknown parameter μ. Run corresponding part in the notebook first. The code first generates a dataset of size 100 for two cases $\mu_{\text{true}} = 0.5$ (i.e., when the two mixtures are close) and the case $\mu_{\text{true}} = 3$ (i.e., when the two mixtures are far). We also plot the labeled dataset to help you visualize (but note that labels are not available to estimate μ_{true} and hence we use EM and GA to obtain estimates). Starting at $\mu_0 = 0.1$, the code then computes EM updates and GA updates with step size 0.05 for the dataset. **Comment on the convergence plots (attach the convergence plots) for the two algorithms. Do the observations match well with the similarity/dissimilarity observed in the updates derived in the previous parts?**

(m) Suppose we decided to use the simplest algorithm to estimate the parameter μ: K Means! Because the parameter is shared, assuming $\mu > 0$, we can estimate $\hat{\mu} = \frac{1}{n_1+n_2}\left|n_1\hat{\mu}_1 - n_2\hat{\mu}_2\right|$, where n_1, n_2 denote the size of the two clusters at the time of update. $\hat{\mu}_1$ and $\hat{\mu}_2$ denote the cluster centroids determined by the K means. **Do you think this strategy will work well? Does your answer depend on whether μ is large or small?** To help you answer the question, we have given a numerical implementation for this part as well. **Run the code of corresponding part in jupyter notebook.** The code then plots a few data-points where we also plot the hidden labels to help you understand the dataset. Also code provides the final estimates of μ by EM, Gradient Ascent (GA) and K Means. **Use the plots and the final answers (report the final answers) to provide an intuitive argument for the questions asked above in this part. Do not spend time on being mathematically rigorous.**

Hopefully you are able to learn the following take away messages: For the simple one-dimensional mixture model, we have that

- EM works well: It converges to a good estimate of μ pretty quickly.
- Gradient ascent is a weighted version of EM: It converges to a good estimate of μ, but is slower than EM.
- K Means: Because of the hard thresholding, it converges to a biased estimate of μ if the two distributions overlap.