Homework 1 – Generative AI (CS/DS 552, Whitehill, Spring 2025)

Collaboration policy: You may complete this assignment with a partner, if you choose. In that case, both partners should sign up on Canvas in a **pre-made group** as a team, and only one of you should submit the assignment. You are permitted to use ChatGPT (or another AI tool) to help you with the visualization aspects of this assignment – e.g., how to call `matplotlib` and `networkx` to render the collages, graphs, etc. You may also use it on Problem 1b. *No other use of ChatGPT (or any other AI tool) is permitted.*

1. **Image Autoregression [20 pts]**

   In this exercise you will generate images by autoregressively sampling from a list of conditional probability tables (CPTs). This process will produce the pixels of a 5×5-pixel *binary* image $\mathbf{x} \in \{0,1\}^{25}$, where $\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_5 \\ & \ddots & \\ x_{21} & \dots & x_{25} \end{bmatrix}$. These CPTs have been pre-computed from an image dataset. To get started, first call `gunzip image_cpts.pkl.gz` (the expanded file is about 500MB) and then call `pickle.load(open("image_cpts.pkl", "rb"))` to load the CPTs. Each CPT i represents $p(x_i \mid x_1, \dots, x_{i-1})$, represented as a multidimensional numpy array with indices `[x1][x2]...[xi]`. (Python indices start at 0; hence, CPT $i = 1$ is stored at index 0 of the list in `image_cpts.pkl`.) Your tasks:

   (a) Use the CPTs and `np.random.rand` to sample 100 images from the joint distribution $p(\mathbf{x})$. To sample each image, first sample pixel $x_1 \sim p(x_1)$. Then, sample $x_2 \sim p(x_2 \mid x_1)$, i.e., the conditional distribution of pixel x_2 given the value of x_1 has already been drawn. Proceed for each x_i until you generate the entire image. After sampling 100 images in this way, display them in a 10×10 "collage" of images. If you sample correctly, the contents of these images should be recognizable objects. Include the collage in the PDF you submit. **[8 pts]**

   (b) Now, use PyTorch to train an LSTM RNN to fit the image dataset – this is a much more efficient representation of $p(\mathbf{x})$ than the brute-force CPTs from part (a). In particular, the input of the RNN at timestep i should be x_{i-1}, and the output should be $p(x_i \mid x_1, \dots, x_{i-1})$. (At timestep $i = 1$, you can just feed 0 as the input, which will serve as a "start symbol".) Autoregressively sample the pixels of an image from the trained RNN for $i = 1, \dots, 25$. Sample a total of 100 images and create a 10×10 collage of images (and include it in your PDF). How many total parameters are in your model, and how does this compare to the size of the CPTs? **Hints**: (1) To get started, first use part (a) to create the set of all the unique images (there are <30 of them), and create a `TensorDataset` out of them. (2) Since each $x_i \in \{0,1\}$, you can use `BCEWithLogitsLoss`. (3) Use `torch.sigmoid` and `torch.rand` to sample each x_i from the output of the RNN at timestep i. (4) In my own solution, I used a 2-layer LSTM with a hidden state dimension of 32. You may use ChatGPT for this task. **[10 pts]**

   (c) For the provided image dataset, why will the training loss (no matter how powerful the model or training algorithm) of the RNN never go to 0? **[2 pts]**

2. **Eigenfaces and Probabilistic Principal Component Analysis (p-PCA) [20 pts]**

   In this exercise you will apply p-PCA to a dataset of face images. This will enable you to generate new faces as well as alter existing face images by perturbing their latent feature vectors. Recall that p-PCA posits that (1) every observed example $\mathbf{x} \in \mathbb{R}^m$ is generated from a latent feature vector $\mathbf{z} \in \mathbb{R}^d$ (where $d < m$), and (2) latent features $\mathbf{z}$ can be sampled from a multivariate Gaussian distribution

with zero mean and identity covariance:

$$\begin{aligned} p(\mathbf{z}) &= \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ p(\mathbf{x} \mid \mathbf{z}) &= \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2\mathbf{I}) \\ p(\mathbf{z} \mid \mathbf{x}) &= \mathcal{N}(\mathbf{M}^{-1}\mathbf{W}^\top(\mathbf{x} - \mu), \sigma^2\mathbf{M}^{-1}) \end{aligned}$$

where $\mathbf{M} = \mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}$.

The parameters of p-PCA are the matrix $\mathbf{W} \in \mathbb{R}^{m \times d}$ that projects the latent feature vector into the observation space, the mean $\mu \in \mathbb{R}^m$, and the variance $\sigma^2 \in \mathbb{R}$. The maximum likelihood estimators (MLEs) of these parameters are not trivial to derive (see this paper for details), but fortunately they are relatively easy to state. For a dataset containing n examples (where each example $\mathbf{x}^{(i)} \in \mathbb{R}^m$), whose mean is $\mu = \frac{1}{n}\sum_{i=1}^n \mathbf{x}^{(i)}$ and whose auto-covariance matrix $\mathbf{S} = \frac{1}{n}\sum_{i=1}^n \left(\mathbf{x}^{(i)} - \mu\right)\left(\mathbf{x}^{(i)} - \mu\right)^\top$, the MLEs are:

$$\begin{aligned} \sigma^2 &= \frac{1}{m-d}\sum_{j=d+1}^{m} \lambda_i \\ \mathbf{W} &= \mathbf{U}_q(\mathbf{\Lambda}_q - \sigma^2\mathbf{I})^{1/2} \end{aligned}$$

where $\mathbf{U}_q$ contains the first d principal eigenvectors of $\mathbf{S}$ (i.e., the eigenvectors corresponding to the d eigenvalues with largest magnitude), and $\mathbf{\Lambda}_q$ is a diagonal matrix containing the d largest-magnitude eigenvalues.

Your task is to apply p-PCA to the `eigenfaces.npy` file, which contains a set of 24×24-pixel face images $\mathbf{x} \in \mathbb{R}^{576}$. In particular:

(a) Train the p-PCA model by computing the MLE for this dataset using the equations above for a latent dimension d. **[7 pts]**

(b) For $d = 2$, make a scatter-plot containing the latent feature vectors $\mathbf{z}$ for the entire training dataset. It should look like a spherical Gaussian. Include this plot in your PDF. **[1 pt]**

(c) For $d \in \{16, 32, 64\}$, use the trained model to "reconstruct" 25 randomly selected images from the provided dataset: For each image $\mathbf{x}$, compute the expected latent vector $\hat{\mathbf{z}} = \mathbb{E}[\mathbf{z} \mid \mathbf{x}]$. Then, use this latent vector to compute the expected image $\mathbb{E}[\mathbf{x} \mid \hat{\mathbf{z}}]$. (**Hints**: (1) The expected value of a Gaussian distribution $\mathcal{N}(\mathbf{m}, \mathbf{C})$ is $\mathbf{m}$. (2) To compute $p(\mathbf{z} \mid \mathbf{x})$, do *not* call `np.linalg.inv`, which is numerically unstable. Instead, use `np.linalg.solve`.) Create a 5×5 collage of these images for each d. How does the reconstruction quality change with d? Include your answer and the collage in the PDF. **[4 pts]**

(d) Generate 100 new faces from scratch (just choose a value of d that gives good visual results) by sampling from $p(\mathbf{z})$ and then computing $\mathbb{E}[\mathbf{x} \mid \mathbf{z}]$. Create a 10×10 collage of these images and include it in your PDF. **[3 pts]**

(e) For $d = 16$, pick a random image $\mathbf{x}$ from the dataset. Compute the expected latent vector $\hat{\mathbf{z}} = \mathbb{E}[\mathbf{z} \mid \mathbf{x}]$. Then, perturb one of the latent dimensions (i.e., any dimension $i \in \{1, \ldots, d\}$) by a small amount (e.g., ± 0.05). For each perturbation $\tilde{\mathbf{z}}$ of $\hat{\mathbf{z}}$, use the trained p-PCA model to reconstruct the image, i.e., $\mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{z}}]$. Create a 5×10 collage of image reconstructions from the perturbed latent vectors (i.e., 5 different latent dimensions, and 10 different linearly-spaced perturbations each). Describe any patterns you see in how the reconstructed images change with larger perturbations for the various latent dimensions. Include your description and the collage in the PDF. **[5 pts]**

3. **Social Networks: MLE and Generation [25 pts]**

In this problem you will train a *graph generation model* and use it to sample new graphs that are similar to those in a provided dataset. In the file `classroom_graphs.pkl` you will find (synthetic)

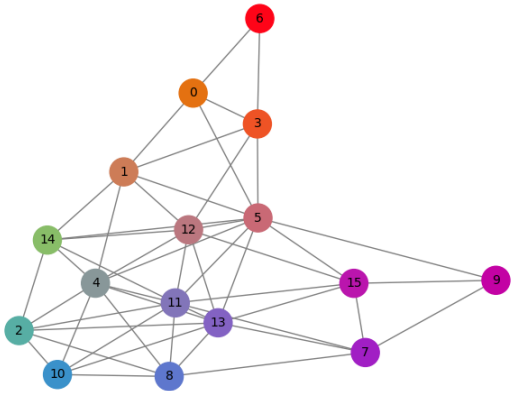
data of students in classrooms at an unnamed polytechnic institute in Worcester, Massachusetts. The dataset consists of 100 graphs, each of which contains both the *features* of each student as well as the *edges* between pairs of students that represents who is friends with whom. The process of sampling new graphs involves several steps: (1) Deciding on a generative model that produces each graph; (2) training the model by estimating its parameters from a dataset of graphs; and (3) sampling from the trained model to produce new graphs.

Graph generation model: For simplicity, we assume that each classroom always contains $n = 15$ students. (It would straightforward to make this more flexible, e.g., to use a geometric distribution to model n.) We further assume that each student i has a real-valued feature vector $\mathbf{x}^{(i)} \in [-1, +1]^m$. Let (binary) edge variable $e_{ij} = 1$ (i.e., students i and j are friends) if node i is connected to node j, and assume all edges are undirected, i.e., $e_{ij} = e_{ji}\ \forall i, j$. There are no self-connections and hence $e_{ii} = 0$. We model

$$p(e_{ij} = 1 \mid \mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma(a{\mathbf{x}^{(i)}}^\top \mathbf{x}^{(j)} + b)$$

where σ is the logistic sigmoid function, and a and b are learned parameters. Importantly, we assume that, given fixed parameters a and b, each edge e_{ij} is generated **independently** of all other edges.

To get intuition for this model, first suppose that $a = 1$ and $b = 0$. Then the probability that $e_{ij} = 1$ is large whenever the inner product between the two nodes' feature vectors is large. For the classroom social network example, the features might represent things like major, graduation year, hobbies, favorite animals, etc.; it is thus intuitive that similarity between the feature vectors influences the probability of friendship. In the graph shown below, each node's (3-dimensional) feature vector is represented by its color (RGB). Notice how the pairs of nodes with similar colors are more likely to be connected than those with very different colors.



Parameter a determines how much the similarity between two students' feature vectors impacts the probability of their friendship. Parameter b represents a "baseline" probability of friendship that is independent of the two persons' feature vectors.

Your tasks are the following:

(a) MLE derivation: Let $\mathbf{X} \in \mathbb{R}^{m \times n}$ contain the nodes' feature vectors, and let $\mathbf{E} = \begin{bmatrix} e_{11} & \dots & e_{1n} \\ & \ddots & \\ e_{n1} & \dots & e_{nn} \end{bmatrix} \in \{0, 1\}^{n \times n}$ represent the edges. Given that the probability of each edge is computed independently

of all other edges, we can factor the log-likelihood of the data given the model parameters as:

$$\begin{aligned} L(a,b) &\doteq \log p(\mathbf{E} \mid \mathbf{X}, a, b) \\ &= \sum_{i<j} \Big(e_{ij} \log p(e_{ij} = 1 \mid \mathbf{x}^{(i)}, \mathbf{x}^{(j)}) + (1 - e_{ij}) \log p(e_{ij} = 0 \mid \mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \Big) \\ &= \sum_{i<j} \Big(e_{ij} \log \sigma(a {\mathbf{x}^{(i)}}^\top \mathbf{x}^{(j)} + b) + (1 - e_{ij}) \log(1 - \sigma(a {\mathbf{x}^{(i)}}^\top \mathbf{x}^{(j)} + b)) \Big) \end{aligned}$$

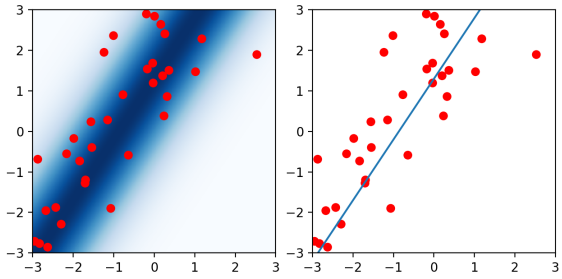
To find the MLE, you must differentiate L w.r.t. both a and b and then conduct gradient *ascent* (since you want to *maximize* the log-likelihood) using the provided data. In this exercise, **you may not use any auto-differentiation library**; rather, **you must derive the gradient terms yourself** – this is useful practice for understanding probabilistic machine learning models. Recall that $\sigma' = \sigma(1 - \sigma)$. Include your derivation of the gradient terms – make sure to simplify them as much as possible – in the PDF you submit. **[15 pts]**

(b) Gradient ascent: Using your MLE derivation, write Python/numpy code to initialize a and b randomly and then conduct gradient ascent to train the model on the provided data. Include in your PDF a screenshot of the last 10 ascent iterations and the associated data likelihood (it should hopefully be increasing). Show the final parameter estimates. Hint: to restrict the sum to $i < j$, you can call `np.tril_indices(n,k=-1)` and use its result to select elements of $\mathbf{E}$. **[5 pts]**

(c) Generation: After having learned good values for a and b from the training data, use your trained model to sample 5 randomly generated graphs (each with $n = 15$ students and $m = 3$ features) by first generating the students' features by sampling uniformly from $[-1, +1]^m$, computing the node probabilities based on these features, and then generating the edges between the nodes. Use the visualization code in `graph_viz.py` to render the color of each node by its features. Include pictures of the graphs in your PDF. **[5 pts]**

4. **Linear-Gaussian Model: MLE Derivation [15 pts]**

The goal of this exercise is to give you practice in (1) deriving maximum likelihood estimators and (2) working with Gaussian distributions, both of which are used heavily in generative AI (e.g., for variational auto-encoders and diffusions).



For supervised learning settings where we want to estimate some target value y based on a feature vector $\mathbf{x}$, probabilistic prediction models enable us to estimate not just the "most likely" or "expected" value of y (see figure above, right), but rather an entire *probability distribution* about which y values are more likely than others, given input $\mathbf{x}$ (see figure above, left). In particular, a linear-Gaussian model is a Gaussian distribution whose expected value (mean μ) is a linear function of the input features $\mathbf{x}$, and whose variance is σ^2:

$$p(y \mid \mathbf{x}) = \mathcal{N}(\mu = \mathbf{x}^\top \mathbf{w}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \mathbf{x}^\top \mathbf{w})^2}{2\sigma^2} \right)$$

Note that, in general, σ^2 can also be a function of $\mathbf{x}$ (heteroscedastic case). Moreover, *non*-linear Gaussian models are also completely possible, e.g., the mean (and possibly the variance) of the Gaussian distribution is output by a deep neural network. However, in this problem, we will assume that μ is linear in $\mathbf{x}$, and that σ^2 is the same for all $\mathbf{x}$ (homoscedastic case).

MLE: Suppose the training dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$. Let the parameters of the linear-Gaussian model be $\mathbf{w}$, which produces the Gaussian's mean $\mu = \mathbf{x}^\top \mathbf{w}$, and its variance σ^2. Prove that the MLE of $\mathbf{w}$ and σ^2 given $\mathcal{D}$ is:

$$\mathbf{w} = \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} {\mathbf{x}^{(i)}}^\top \right)^{-1} \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \right)$$

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} ({\mathbf{x}^{(i)}}^\top \mathbf{w} - y^{(i)})^2$$

Note that this solution – derived based on *maximizing* probability – is exactly the same as the optimal weights of a 2-layer linear neural network optimized to *minimize* mean squared error (MSE).

Hint: Start by defining the log-likelihood function $L(\mathbf{w}, \sigma^2) = \log \prod_{i=1}^n p(y^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{w}, \sigma^2)$ of the target values given the features & parameters. Then, follow the same strategy as the MLE derivation for a biased coin given in lecture. The log of the Gaussian likelihood simplifies beautifully. Submit your proof as part of the PDF.

Submission: Create a Zip file containing all your Python code (which can be in multiple files) and your PDF file, and then submit on Canvas. If you are working as part of a group, then only **one** member of your group should submit (but make sure you have already signed up in a pre-allocated team for the homework on Canvas).