Intro to Machine Learning (CS771A, Autumn 2018)
Homework 4
Due Date: Nov 17, 2018 (11:59pm)

Instructions:

- Only electronic submissions will be accepted. Your main PDF writeup must be typeset in LaTeX (please also refer to the "Additional Instructions" below).
- Your submission will have two parts: The main PDF writeup (to be submitted via Gradescope `https://www.gradescope.com/`) and the code for the programming part (to be submitted via this Dropbox link: `https://tinyurl.com/cs771-a18-hw4-a`). Both parts must be submitted by the deadline. We will be accepting late submissions upto 72 hours after the deadline (with every 24 hours delay incurring a 10% late penalty). We won't be able to accept submissions after that.
- We have created your Gradescope account (you should have received the notification). Please use your IITK CC ID (not any other email ID) to login. Use the "Forgot Password" option to set your password.

Additional Instructions

- We have provided a LaTeX template file `hw4sol.tex` to help typeset your PDF writeup. There is also a style file `ml.sty` that contain shortcuts to many of the useful LaTeX commends for doing things such as boldfaced/calligraphic fonts for letters, various mathematical/greek symbols, etc., and others. Use of these shortcuts is recommended (but not necessary).
- Your answer to every question should begin on a new page. The provided template is designed to do this automatically. However, if it fails to do so, use the `\clearpage` option in LaTeX before starting the answer to a new question, to *enforce* this.
- While submitting your assignment on the Gradescope website, you will have to specify on which page(s) is question 1 answered, on which page(s) is question 2 answered etc. To do this properly, first ensure that the answer to each question starts on a different page.
- Be careful to flush all your floats (figures, tables) corresponding to question n before starting the answer to question $n+1$ otherwise, while grading, we might miss your important parts of your answers.
- Your solutions must appear in proper order in the PDF file i.e. solution to question n must be complete in the PDF file (including all plots, tables, proofs etc) before you present a solution to question $n+1$.
- For the programming part, all the code and README should be zipped together and submitted as a single file named `yourrollnumber.zip`. Please DO NOT submit the data provided.

Problem 1 (10 marks)

(Eigenchangers!) Suppose we wish to do PCA for an $N \times D$ matrix $\mathbf{X}$ and assume $D > N$. The traditional way to do PCA is to compute the eigenvectors of the covariance matrix $\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ (assuming centered data). Show that, if someone instead gives you an eigenvector $\boldsymbol{v} \in \mathbb{R}^N$ of the matrix $\frac{1}{N}\mathbf{X}\mathbf{X}^\top$, you can use it to get an eigenvector $\boldsymbol{u} \in \mathbb{R}^D$ of $\mathbf{S}$. What is the advantage of this way of obtaining the eigenvectors of $\mathbf{S}$?

Problem 2 (10 marks)

(A General Activation Function) Consider the following activation function: $h(x) = x\sigma(\beta x)$ where σ denotes the sigmoid function $\sigma(z) = \frac{1}{1+\exp(-z)}$. Show that, for appropriately chosen values of β, this activation function can approximate (1) the identity activation function, and (2) the ReLU activation function.

Problem 3 (15 marks)

(Mixtures meet Neural Nets!) Consider modeling some data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \{0, 1\}$, using a mixture of logistic regression models, where we model each binary label y_n by first picking one of the K logistic regression models, based on the value of a latent variable $z_n \sim$ multinoulli$(\pi_1, \ldots, \pi_K)$, and then generating y_n *conditioned* on z_n as $y_n \sim$ Bernoulli$[\sigma(\boldsymbol{w}_{z_n}^\top \boldsymbol{x}_n)]$.

Now consider the *marginal* probability of the label $y_n = 1$, given $\boldsymbol{x}_n$, i.e., $p(y_n = 1|\boldsymbol{x}_n)$, and show that this can be quantity can also be thought of as the output of a neural network. Clearly specify what is the input layer, hidden layer(s), activations, the output layer, and the connection weights of this neural network.

Problem 4 (30 marks)

(Probabilistix Formulation of Matrix Factorization with Side Information) Consider an $N \times M$ rating matrix $\mathbf{X}$, where the rows represents the N users and the columns represent the M items. We are also given some side information: for each user n, a feature vector $\boldsymbol{a}_n \in \mathbb{R}^{D_U}$, and for each item m, a feature vector $\boldsymbol{b}_m \in \mathbb{R}^{D_I}$. Let's model each entry of $\mathbf{X}$ as $p(X_{nm}|\boldsymbol{u}_n, \boldsymbol{v}_m, \theta_n, \phi_m) = \mathcal{N}(X_{nm}|\theta_n + \phi_m + \boldsymbol{u}_n^\top \boldsymbol{v}_m, \lambda_x^{-1})$.

In this model, $\boldsymbol{u}_n \in \mathbb{R}^K$ and $\boldsymbol{v}_m \in \mathbb{R}^K$ represent the user n and item m latent factors, respectively. In addition, for each user n, we have a user-specific bias $\theta_n \in \mathbb{R}$ (bias regardless of the item being rated), and for each item m, we have an item-specific bias $\phi_m \in \mathbb{R}$ ("popularity" of the item, regardless is who has rated this item).

Assume Gaussian priors on the latent factors $\boldsymbol{u}_n \in \mathbb{R}^K$ and $\boldsymbol{v}_m \in \mathbb{R}^K$: $p(\boldsymbol{u}_n) = \mathcal{N}(\boldsymbol{u}_n|\mathbf{W}_u\boldsymbol{a}_n, \lambda_u^{-1}\mathbf{I}_K)$ and $p(\boldsymbol{v}_m) = \mathcal{N}(\boldsymbol{v}_m|\mathbf{W}_v\boldsymbol{b}_m, \lambda_v^{-1}\mathbf{I}_K)$. Note that the *mean* of the priors of the latent factors $\boldsymbol{u}_n$ and $\boldsymbol{v}_m$ depends on the given user and item features ($\boldsymbol{a}_n$ and $\boldsymbol{b}_m$, respectively) via a linear model with regression parameters matrices $\mathbf{W}_u \in \mathbb{R}^{K \times D_U}$ and $\mathbf{W}_v \in \mathbb{R}^{K \times D_I}$, respectively. You don't need to assume any priors on the bias parameters θ_n, ϕ_m, and the regression parameters $\mathbf{W}_u$, $\mathbf{W}_v$.

Assume $\Omega = \{(n, m)\}$ to denote the set of indices of the observed entries of $\mathbf{X}$, Ω_{r_n} to be the set of items rated by user n, and Ω_{c_m} to be the set of users who rated item m (similar notation that we saw in class).

Your goal is to estimate $\{\boldsymbol{u}_n, \theta_n\}_{n=1}^N$, $\{\boldsymbol{v}_n, \phi_m\}_{m=1}^M$, $\mathbf{W}_u$, and $\mathbf{W}_v$. Assume all other parameters to be known.

Write down the loss function for this model (this will be the negative of the MAP objective for the model) and use the ALT-OPT algorithm to derive the update equations for all the unknowns. The expressions must be in closed form (it is possible for this model).

Problem 5 (25+10 marks)

(Programming Problem 1) For this problem, your task is to implement a simplified version of the Probabilistic PCA (PPCA) model and play with it on a dataset of face images. The provided dataset contains a total of 165 face images of 15 individuals. Each image is 64x64 grayscale. So $\mathbf{X}$ is 165×4096 ($N = 165, D = 4096$).

The first simplication will be that we will assume $\sigma^2 = 0$ and the second simplification will be that we will use ALT-OPT instead of EM. The overall algorithm will basically be a simpler version of the algorithm given on slide 10 of lecture 18. Since you are using ALT-OPT instead of EM, you don't even have to compute the posterior or the expectation of the latent variables. It is really that simple! :-) You will effectively be doing PCA but using ALT-OPT instead of eigendecomposition! The ALT-OPT algorithm will alternate between estimating $\mathbf{Z}$ given $\mathbf{W}$, and $\mathbf{W}$ given $\mathbf{Z}$. If using MATLAB, this shouldn't need more than 8-10 lines or code (likewise for Python). You will run your ALT-OPT algorithm for a fixed number of iteration (100 iterations), though it might converge to a reasonable solution in fewer iterations.

Run your implementation with $K = 10, 20, 30, 40, 50, 100$ and using all the data (165 images). After the model has been learned, for each value of K, reconstruct any 5 of the images (of your choice!), using μ (this will be equal to the sample mean), $\mathbf{W}$, and the latent code of the corresponding image, and **visually inspect the reconstructed images** (note: to show the image, you will have to reshape $\hat{\boldsymbol{x}}_n$ as 64×64), comparing them with their original versions. Note that, the reconstruction of an image $\boldsymbol{x}_n$ will be $\hat{\boldsymbol{x}}_n \approx \mu + \mathbf{W}\boldsymbol{z}_n$.

Note: You should reconstruct the same 5 images for each value of K. The goal is to see how (or whether) increasing K improves the reconstruction (or not). What do you observe **visually** as K increases? Does the reconstruction get better or worse? Briefly explain the reason for what you observe.

For each K, also show 10 of the *basis* images (columns of the $\mathbf{W}$ matrix, each reshaped as 64×64) that you think look sort of interesting (for $K = 10$, it will mean showing all the columns of $\mathbf{W}$; for $K = 20, 30, 40, 50, 100$, you will need to select a subset of 10 columns from $\mathbf{W}$ and show those).

Deliverables: Submit your code (named `ppca_altopt.py` or `ppca_altopt.m`) and the images (for each K, the 5 original and reconstructed images, and the 10 basis images). Everything should be zipped together in a single file (following the naming conventions as in previous homework). Also include these images and your experimental observations in the main PDF writeup (you can scale the sizes of images to be small enough so that they don't take too much space in the PDF).

(Programming Problem 2) You are provided a subset of the MNIST data consisting of 10,000 images of digits 0-9. Each image is of size 28×28 (784 dimensional). The dataset also contains the digit labels.

Your goal is to run K-means clustering on *two-dimensional* embedding of this data. To get the two-dimensional embeddings, you will use two approaches: PCA and tSNE. You can use any implementation of PCA and tSNE, e.g., from sklearn if you are using Python (don't need to implement any of these) or any MATLAB implementation. If you want, you can use this very nice tSNE implementation (available in several langugages) from this webpage: `https://lvdmaaten.github.io/tsne/`. For PCA, you can even use your code from part 1 with the number of latent factors set to 2. You also don't need to implement K-means on your own (you already did it in HW3). You can use any existing implementation of K-means.

Run K-means with $K = 10$ for both cases (PCA based 2D embeddings and tSNE based 2D embeddings), using 10 different initializations and show the clustering results in form of the plots of the obtained clusterings with each cluster shown in a different color. Visually, which of the two 2D embedding methods (PCA/tSNE) seems to work better for the clustering task?

Deliverables: No deliverables except the clustering plots in the writeup PDF.