# CS559-B HW2

Due: Oct. 29, 2020

**Problem 1 (30pt):** [Perceptron Algorithm]

In this problem, we learn the linear discriminant function for boolean NAND function. Suppose we have two dimensional $x = (x_1, x_2)$, $x_1$ and $x_2$ can be either 0 (false) or 1 (true). The boolean NAND function is defined as: $f(x_1, x_2) = x_1$ **NAND** $x_2$. Specifically, $f(0,0) =$ **true**, $f(1,0) =$ **true**, $f(0,1) =$ **true**, and $f(1,1) =$ **false** where **false** can be treated as *positive class* and **true** can be treated as *negative class*. You can think of this function as having 4 points on the 2D plane ($x_1$ being the horizontal axis and $x_2$ being the vertical axis): $P_1 = (0,1), P_2 = (1,1), P_3 = (1,0), P_4 = (0,0)$, $P_2$ in *positive class* and $P_1, P_3, P_4$ in *negative class*.

(1) [5pt] For boolean NAND function, is the negative class and positive class linearly separable?

(2) [25pt] Is it possible to apply the **perceptron algorithm** to obtain the linear decision boundary that correctly classify both the positive and negative classes? If so, write down the updation steps and the obtained linear decision boundary. (You may assume the initial decision boundary is $x_1 + x_2 - \frac{1}{2} = 0$, and sweep the 4 points in clockwise order, i.e., $(P_1, P_2, P_3, P_4, P_1, P_2, \dots)$, note that you **can not** write down the arbitrary linear boundary without updation steps. )

**Problem 2 (70pt):** [Principal Component Analysis, Dimension Reduction, Eigenface]

Face modelling serves as one of the most fundamental problems in modern artificial intelligence and computer vision, and can be useful in various applications like face recognition, identification etc. However, face images are usually of high-dimensional (e.g., a small $100 \times 100$ gray-scaled face image has dimension $100 \times 100 = 10,000$), therefore, find a suitable representation is utterly important. In this problem, we apply the linear model, principal component analysis (PCA), on face images to reduce the dimension and obtain eigenface representations.

**Dataset**: we use the dataset[†] which contains 177 face images. Each image contains $256 \times 256$ pixels and is gray-scaled (i.e., the value for each pixel is stored as unsigned integer between $[0, 255]$, typically, 0 is taken to be black and 255 is taken to be white). You need to split the dataset to be train/test set, e.g., you could use the first 157 images for training, and the rest 20 faces for testing.

(1) [30pt] Write the PCA codes to compute $K = 30$ eigenfaces and visualize the top 10 eigenfaces.

(2) [20pt] Use the learned $K$ eigenfaces from (1) to reconstruct testing images, and record the reconstruction error. The reconstruction error can be defined as $\frac{||\hat{Y} - Y||^2}{N}$, where $\hat{Y}$ is the reconstructed face using the learned eigenfaces, $Y$ is the testing faces and $N$ is the total number of testing data. Please show 5 testing images and their reconstructed ones.

(3) [20pt] Try different values of $K$, e.g., try $K = 10, 30, 50, 100, 150...$, and draw the curve to indicate the corresponding testing reconstruction errors. The x-axis of the curve can be different $K$ values, and the y-axis can be testing reconstruction error defined in (2).

[†] The dataset is coming from S.C. Zhu's previous Stats 231: Pattern Recognition and Machine Learning in UCLA.

Some useful hints:

(1) To construct the data matrix for PCA, you can reshape each image to a long vector. For example, the original 2D image of size $[h, w]$ becomes 1D vector of size $h \times w$. Each row of the data matrix represents one face image and data matrix has size $[N_{train}, D]$ where $N_{train}$ is the number of training samples and $D = h * w$ is the dimension of the reshaped image.

(2) To compute the PCA, you need to subtract the mean image from each training image. To get the mean image, just sum up all the training images and divide it by $N_{train}$.

(3) You could use **from PIL import Image**, **Image.open** to read the images, and use **matplotlib.pyplot** to show the images. Feel free to use other functions to read and process the images as well.

(4) For PCA, you could use *linalg* from *numpy* for eigendecomposition. Other eigenvalue decomposition/SVD functions can also be used. However, you **can not** directly call the **pca** functions in any languages for this problem.