# CSE4/574 Introduction to Machine Learning
# Programming Assignment 1
# Handwritten Digits Classification

Due Date: 11:59pm (EST) Thu Mar 13$^{th}$ 2025

## 1 Introduction

In this assignment, your task is to implement a Multilayer Neural Network and evaluate its performance in classifying handwritten digits. You will use the same network to analyze a more challenging face dataset and compare the performance of the neural network against a *deep* neural network and a convolutional neural network using the PyTorch library.

After completing this assignment, you are able to understand:

- How Neural Network works and use Feed Forward, Back Propagation to implement Neural Network?
- How to setup a Machine Learning experiment on real data?
- How *regularization* plays a role in the *bias-variance* tradeoff?
- How to use PyTorch library to deploy deep neural networks and understand how having multiple hidden layers can improve the performance of the neural network?
- How to use PyTorch library to deploy convolutional neural networks and understand the benefit of convolutional neural network compared to fully connected neural network?

To get started with the exercise, you will need to download the supporting files and unzip its contents to the directory you want to complete this assignment.

> **Warning:** In this project, you will have to handle many computing intensive tasks such as training a neural network. YOU MUST USE PYTHON 3 FOR IMPLEMENTATION. In addition, training such a big dataset will take a very long time, maybe many hours or even days to complete. Therefore, we suggest that you should start doing this project as soon as possible.

### 1.1 File included in this exercise

- *mnist_all.mat*: original dataset from MNIST. In this file, there are 10 matrices for testing set and 10 matrices for training set, which corresponding to 10 digits. You will have to split the training data into training and validation data.
- *face_all.pickle*: sample of face images from the CelebA data set. In this file there is one data matrix and one corresponding label vector. The preprocess routines in the script files will split the data into training and testing data.
- *nnScript.py*: Python script for this programming project. Contains function definitions -
  - *preprocess()*: performs some preprocess tasks, and output the preprocessed train, validation and test data with their corresponding labels. *You need to make changes to this function.*

  - *sigmoid()*: compute sigmoid function. The input can be a scalar value, a vector or a matrix. *You need to make changes to this function.*
  - *nnObjFunction()*: compute the error function of Neural Network. *You need to make changes to this function.*
  - *nnPredict()*: predicts the label of data given the parameters of Neural Network. *You need to make changes to this function.*
  - *initializeWeights()*: return the random weights for Neural Network given the number of unit in the input layer and output layer.

- *facennScript.py*: Python script for running your neural network implementation on the CelebA dataset. This function will call your implementations of the functions sigmoid(), nnObjFunc() and nnPredict() that you will have to copy from your nnScript.py. *You need to make changes to this function.*

- *deepnnScript.py*: Python script for calling the PyTorch library for running the deep neural network. *You need to make changes to this function.*

- *cnnScript.py*: Python script for calling the PyTorch library for a convolutional neural network. *You need to change this function.*

## 1.2 Datasets

Two datasets have been provided. Both consist of images.

### 1.2.1 MNIST Dataset

The MNIST dataset [1] consists of a training set of 60000 examples and testing set of 10000 examples. All digits have been size-normalized and centered in a fixed image of $28 \times 28$ size. In original dataset, each pixel in the image is represented by an integer between 0 and 255, where 0 is black, 255 is white and anything between represents different shade of gray.
You will need to split the training set of 60000 examples into two sets. First set of 50000 randomly sampled examples will be used for training the neural network. The remainder 10000 examples will be used as a validation set to estimate the hyper-parameters of the network (regularization constant $\lambda$, number of hidden units).

### 1.2.2 CelebFaces Attributes Dataset (CelebA)

CelebFaces Attributes Dataset (CelebA) [3] is a large-scale face attributes dataset with more than 200K celebrity images. CelebA has large diversities, large quantities, and rich annotations, including:

- 10,177 number of identities,

- 202,599 number of face images, and

- 5 landmark locations, 40 binary attributes annotations per image.

For this programming assignment, we will have provided a subset of the images. The subset will consist of data for 26407 face images, split into two classes. One class will be images in which the individual is wearing glasses and the other class will be images in which the individual is not wearing glasses. Each image is a $54 \times 44$ matrix, flattened into a vector of length 2376.

# 2 Your tasks

- Implement **Neural Network** (forward pass and back propagation)

- Incorporate regularization on the weights ($\lambda$)

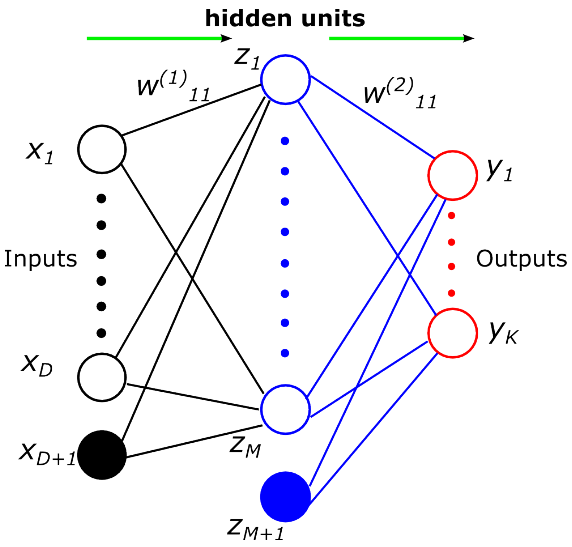

Figure 1: Neural network

- Use validation set to tune hyper-parameters for Neural Network (number of units in the hidden layer and $\lambda$).
- Run the deep neural network code we provided and compare the results with normal neural network.
- Run the convolutional neural network code and print out the results, for example the confusion matrix.
- Record a video to explain the experimental results.

# 3 Some practical tips in implementation

## 3.1 Feature selection

In the dataset, one can observe that there are many features which values are exactly the same for all data points in the training set. With those features, the classification models cannot gain any more information about the difference (or variation) between data points. Therefore, we can ignore those features in the pre-processing step. The task is to identify non-repetitive features.

Later on in this course, you will learn more sophisticated models to reduce the dimension of dataset (but not for this assignment).

*Note:* You will need to save the indices of the features that you use and submit them as part of the submission.

## 3.2 Neural Network

### 3.2.1 Neural Network Representation

Neural network can be graphically represented as in Figure 1.

As observed in the Figure 1, there are totally 3 layers in the neural network:

- The first layer comprises of $(d + 1)$ units, each represents a feature of image (there is one extra unit representing the bias).
- The second layer in neural network is called the hidden units. In this document, we denote $m + 1$ as the number of hidden units in hidden layer. There is an additional bias node at the hidden layer as well. Hidden units can be considered as the learned features extracted from the original data set. Since number of hidden units will represent the dimension of learned features in neural network, it's

our choice to choose an appropriate number of hidden units. Too many hidden units may lead to the slow training phase while too few hidden units may cause the the under-fitting problem.

- The third layer is also called the output layer. The value of $l^{th}$ unit in the output layer represents the probability of a certain hand-written image belongs to digit $l$. Since we have 10 possible digits, there are 10 units in the output layer. In this document, we denote $k$ as the number of output units in output layer.

The parameters in Neural Network model are the weights associated with the hidden layer units and the output layers units. In our standard Neural Network with 3 layers (input, hidden, output), in order to represent the model parameters, we use 2 matrices:

- $W^{(1)} \in \mathbb{R}^{m \times (d+1)}$ is the weight matrix of connections from input layer to hidden layer. Each row in this matrix corresponds to the weight vector at each hidden layer unit.
- $W^{(2)} \in \mathbb{R}^{k \times (m+1)}$ is the weight matrix of connections from hidden layer to output layer. Each row in this matrix corresponds to the weight vector at each output layer unit.

We also further assume that there are $n$ training samples when performing learning task of Neural Network.

In the next section, we will explain how to perform learning in Neural Network.

### 3.2.2 Feedforward Propagation

In Feedforward Propagation, given parameters of Neural Network and a feature vector $\mathbf{x}$, we want to compute the probability that this feature vector belongs to a particular digit.

Suppose that we have totally $m$ hidden units. Let $a_j$ for $1 \leq j \leq m$ be the linear combination of input data and let $z_j$ be the output from the hidden unit $j$ after applying an activation function (in this exercise, we use sigmoid as an activation function). For each hidden unit $j$ ($j = 1, 2, \cdots, m$), we can compute its value as follow:

$$a_j = \sum_{p=1}^{d+1} w_{jp}^{(1)} x_p \tag{1}$$

$$z_j = \sigma(a_j) = \frac{1}{1 + \exp(-a_j)} \tag{2}$$

where $w_{ji}^{(1)} = W^{(1)}[j][p]$ is the weight of connection from the $p^{th}$ input feature to unit $j$ in hidden layer. Note that we do not compute the output for the bias hidden node $(m+1)$; $z_{m+1}$ is directly set to 1.

The third layer in neural network is called the output layer where the learned features in hidden units are linearly combined and a sigmoid function is applied to produce the output. Since in this assignment, we want to classify a hand-written digit image to its corresponding class, we can use the one-vs-all binary classification in which each output unit $l$ ($l = 1, 2, \cdots, 10$) in neural network represents the probability of an image belongs to a particular digit. For this reason, the total number of output unit is $k = 10$. Concretely, for each output unit $l$ ($l = 1, 2, \cdots, 10$), we can compute its value as follow:

$$b_l = \sum_{j=1}^{m+1} w_{lj}^{(2)} z_j \tag{3}$$

$$o_l = \sigma(b_l) = \frac{1}{1 + \exp(-b_l)} \tag{4}$$

Now we have finished the **Feedforward pass**.

### 3.2.3 Error function and Backpropagation

The error function in this case is the negative log-likelihood error function which can be written as follow:

$$J(W^{(1)}, W^{(2)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{k}(y_{il}\ln o_{il} + (1 - y_{il})\ln(1 - o_{il})) \tag{5}$$

where $y_{il}$ indicates the $l^{th}$ target value in 1-of-K coding scheme of input data $i$ and $o_{il}$ is the output at $l^{th}$ output node for the $i^{th}$ data example (See (4)).

Because of the form of error function in equation (5), we can separate its error function in terms of error for each input data $\mathbf{x}_i$:

$$J(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{i=1}^{n} J_i(W^{(1)}, W^{(2)}) \tag{6}$$

where

$$J_i(W^{(1)}, W^{(2)}) = -\sum_{l=1}^{k}(y_{il}\ln o_{il} + (1 - y_{il})\ln(1 - o_{il})) \tag{7}$$

One way to learn the model parameters in neural networks is to initialize the weights to some random numbers and compute the output value (feed-forward), then compute the error in prediction, transmits this error backward and update the weights accordingly (error backpropagation).

The feed-forward step can be computed directly using formula (1), (2), (3) and (4).

On the other hand, the error backpropagation step requires computing the derivative of error function with respect to the weight.

Consider the derivative of error function with respect to the weight from the hidden unit $j$ to output unit $l$ where $j = 1, 2, \cdots, m + 1$ and $l = 1, \cdots, 10$:

$$\begin{aligned}\frac{\partial J_i}{\partial w_{lj}^{(2)}} &= \frac{\partial J_i}{\partial o_l}\frac{\partial o_l}{\partial b_l}\frac{\partial b_l}{\partial w_{lj}^{(2)}} && (8)\\ &= \delta_l z_j && (9)\end{aligned}$$

where

$$\delta_l = \frac{\partial J_i}{\partial o_l}\frac{\partial o_l}{\partial b_l} = -(\frac{y_l}{o_l} - \frac{1 - y_l}{1 - o_l})(1 - o_l)o_l = o_l - y_l$$

Note that we are dropping the subscript $i$ for simplicity. The error function (log loss) that we are using in (5) is different from the the squared loss error function that we have discussed in class. Note that the choice of the error function has "simplified" the expressions for the error!

On the other hand, the derivative of error function with respect to the weight from the $p^{th}$ input feature to hidden unit $j$ where $p = 1, 2, \cdots, d + 1$ and $j = 1, \cdots, m$ can be computed as follow:

$$\begin{aligned}\frac{\partial J_i}{\partial w_{jp}^{(1)}} &= \sum_{l=1}^{k}\frac{\partial J_i}{\partial o_l}\frac{\partial o_l}{\partial b_l}\frac{\partial b_l}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{jp}^{(1)}} && (10)\\ &= \sum_{l=1}^{k}\delta_l w_{lj}^{(2)}(1 - z_j)z_j x_p && (11)\\ &= (1 - z_j)z_j(\sum_{l=1}^{k}\delta_l w_{lj}^{(2)})x_p && (12)\end{aligned}$$

Note that we do not compute the gradient for the weights at the bias hidden node.

After finish computing the derivative of error function with respect to weight of each connection in neural network, we now can write the formula for the gradient of error function:

$$\nabla J(W^{(1)}, W^{(2)}) = \frac{1}{n}\sum_{i=1}^{n}\nabla J_i(W^{(1)}, W^{(2)}) \tag{13}$$

We again can use the gradient descent to update each weight (denoted in general as $w$) with the following rule:

$$w^{new} = w^{old} - \gamma \nabla J(w^{old}) \tag{14}$$

### 3.2.4 Regularization in Neural Network

In order to avoid overfitting problem (the learning model is best fit with the training data but give poor generalization when test with validation data), we can add a regularization term into our error function to control the magnitude of parameters in Neural Network. Therefore, our objective function can be rewritten as follow:

$$\widetilde{J}(W^{(1)}, W^{(2)}) = J(W^{(1)}, W^{(2)}) + \frac{\lambda}{2n}\left(\sum_{j=1}^{m}\sum_{p=1}^{d+1}(w_{jp}^{(1)})^2 + \sum_{l=1}^{k}\sum_{j=1}^{m+1}(w_{lj}^{(2)})^2\right) \tag{15}$$

where $\lambda$ is the regularization coefficient.

With this new objective function, the partial derivative of new objective function with respect to weight from hidden layer to output layer can be calculated as follow:

$$\frac{\partial \widetilde{J}}{\partial w_{lj}^{(2)}} = \frac{1}{n}\left(\sum_{i=1}^{n}\frac{\partial J_i}{\partial w_{lj}^{(2)}} + \lambda w_{lj}^{(2)}\right) \tag{16}$$

Similarly, the partial derivative of new objective function with respect to weight from input layer to hidden layer can be calculated as follow:

$$\frac{\partial \widetilde{J}}{\partial w_{jp}^{(1)}} = \frac{1}{n}\left(\sum_{i=1}^{n}\frac{\partial J_i}{\partial w_{jp}^{(1)}} + \lambda w_{jp}^{(1)}\right) \tag{17}$$

With this new formulas for computing objective function (15) and its partial derivative with respect to weights (16) (17) , we can again use gradient descent to find the minimum of objective function.

### 3.2.5 Python implementation of Neural Network

In the supporting files, we have provided the base code for you to complete. In particular, you have to complete the following functions in Python:

- *sigmoid*: compute sigmoid function. The input can be a scalar value, a vector or a matrix.
- *nnObjFunction*: compute the objective function of Neural Network *with regularization* and the gradient of objective function.
- *nnPredict*: predicts the label of data given the parameters of Neural Network.

Details of how to implement the required functions is explained in Python code.

**Optimization:** In general, the learning phase of Neural Network consists of 2 tasks. First task is to compute the value and gradient of error function given Neural Network parameters. Second task is to optimize the error function given the value and gradient of that error function. As explained earlier, we can use gradient descent to perform the optimization problem. In this assignment, you have to use the Python scipy function: **scipy.optimize.minimize** (using the option *method='CG'* for conjugate gradient descent), which performs the conjugate gradient descent algorithm to perform optimization task. In principle, conjugate gradient descent is similar to gradient descent but it chooses a more sophisticated learning rate $\gamma$ in each iteration so that it will converge faster than gradient descent. Details of how to use *minimize* are provided here: `http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.minimize.html`.

# 4 PyTorch Library

In this assignment you will only implement a single layer Neural Network. You will realize that implementing multiple layers can be a very cumbersome coding task. However, additional layers can provide a better modeling of the data set. The analysis of the challenging CelebA data set will show how adding more layers can improve the performance of the Neural Network. To experiment with Neural Networks with multiple layers, we will use PyTorch library (`https://pytorch.org/`). Please install PyTorch on personal machines. The code provided has been tested on PyTorch Version 1.12.1.

Your experiments should include the following:

- Evaluate the accuracy of single hidden layer Neural Network on CelebA data set (test data only), to distinguish between two classes - *wearing glasses* and *not wearing glasses*. Use *facennScript.py* to obtain these results.
- Evaluate the accuracy of deep Neural Network (try 3, 5, and 7 hidden layers) on CelebA data set (test data only). Use *deepnnScript.py* to obtain these results.
- Compare the performance of single vs. deep Neural Networks in terms of accuracy on test data and training time.
- Compare the performance of the deep Neural Networks vs. Convolutional Neural Networks in terms of accuracy on test data and training time. Use *cnnScript.py* to obtain these results (extra points)

# 5 Submission

You are required to submit a single file called *proj2.zip* using UBLearns.
File *proj2.zip* must contain a video presentation and a code folder: *demo.mp4* and *code.*

- For your video presentation *demo.mp4*, it should be no more than 15 minutes, and please introduce yourself and **both team members should participate in the presentation** and describe the parts they completed. If a member is absent in the video they may get a 0 for that project. The video can be a screen recording, where the team members can go through their code and the results they obtained. It is appreciable if the team members turn on the web camera when recording their presentation, but this is not mandatory.
- Folder *code* must contains the following updated files: *nnScript.py* and *params.pickle*[1]. File *params.pickle* contains the learned parameters of Neural Network. Concretely, file *params.pickle* must contain the following variables: list of selected features obtained after feature selection step (*selected_features*), optimal *n_hidden* (number of units in hidden layer), *w1* (matrix of weight $W^{(1)}$ as mentioned in section 3.2.1), *w2* (matrix of weight $W^{(2)}$ as mentioned in section 3.2.1), optimal $\lambda$ (regularization coeffient $\lambda$ as mentioned in section 3.2.4).[2]

**Video Presentation:** Your presentation should include the following explanation:

- Explanation of how to choose the hyper-parameters for Neural Network (number of hidden units, regularization term $\lambda$).

  We use regularization in Neural Network to avoid overfitting problem (more about this will be discussed in class). You are expected to change different value of $\lambda$ to see its effect in prediction accuracy in validation set. Your video presentation should include diagrams to explain the relation between $\lambda$ and performance of Neural Network. Moreover, by plotting the value of $\lambda$ with respect to the accuracy of Neural Network, you should explain in your presentation how to choose an appropriate hyper-parameter $\lambda$ to avoid both underfitting and overfitting problem. You can vary $\lambda$ from 0 (no regularization) to 60 in increments of 5 or 10.

[1] Check this to learn how to pickle objects in Python: `https://wiki.python.org/moin/UsingPickle`

[2] If you want to write more supporting functions to complete the required functions, you should include these supporting functions and a README file which explains your supporting functions.

- Compare the results of deep neural network and neural network with one hidden layer on the CelebA data set.

  You are expected to try different number hidden units to see its effect to the performance of Neural Network. Since training Neural Network is very slow, especially when the number hidden units in Neural Network is large. You should try with small hidden units and gradually increase the size and see how it effects the training time. Your video presentation should include some diagrams to explain relation between number of hidden units and training time. Recommended values: $4, 8, 12, 16, 20$.

# 6 Grading scheme

The TAs will deploy a testing script that will test the functionality of individual functions that you submit within the *nnScript.py* file. Full points will be awarded if the output of the function exactly matches the expected output. The second grading script will load the *params.pickle* file that you submit and then test a small testing data set. You get full points (10) if the accuracy using your model parameters is within ±5% of the accuracy reported by our code. Note that this data set **will not be** made available to you.

- Total 100 points + 20 extra points:
  - Successfully implement Neural Network: 50 points (*preprocess()* [8 points], *sigmoid()* [8 points], *nnObjFunction()* [26 points], *nnPredict()* [8 points]).
  - Video Presentation: 50 points
    - Explanation with supporting figures of how to choose the hyper-parameter for Neural Network: 26 points
    - Accuracy of classification method on the handwritten digits test data: 8 points
    - Accuracy of classification method on the CelebA data set: 8 points
    - Comparison of your neural network with a deep neural network on the CelebA data set in terms of accuracy and training time: 8 points
    - Present the results from convolutional neural network on the handwritten digits dataset in terms of accuracy and training time. 20 extra points

# References

[1] LeCun, Yann; Corinna Cortes, Christopher J.C. Burges. “MNIST handwritten digit database”.

[2] Bishop, Christopher M. “Pattern recognition and machine learning (information science and statistics)” (2007).

[3] Liu, Ziwei; Luo, Ping; Wang, Xiaogang; Tang, Xiaoou. “Deep Learning Face Attributes in the Wild”, Proceedings of International Conference on Computer Vision (ICCV) (2015).