# ECE1504: Assignment 1:
# Linear and Logistic Regression

Due date: 29 October 2018 (hard copy in Class)
Electronic submission to: ece1504.fall2018@gmail.com

## Objectives:

The purpose of this assignment is to investigate the classification performance of linear and logistic regression. In this assignment, you will gain some experience in training linear and logistic models using Stochastic Gradient Descent (SGD) and Adam optimization. All the implementations need to be done using Python and TensorFlow. You will use the built-in optimization functions provided by Tensorflow. You are encouraged to look up TensorFlow APIs for useful utility functions, at: https://www.tensorflow.org/api_docs/python/.

## General Note:

- Full points are given for complete solutions, including justifying the choices or assumptions you made to solve each question. A written report should be included in the final submission.
- Homework assignments are to be solved in groups of two. You are encouraged to discuss the assignment with other students, but you must solve it within your own group. Make sure to be closely involved in all aspects of the assignment. If you work in a group, please indicate the contribution percentage from each group member at the beginning of your report.

## Introduction

In this assignment, the performance of linear regression will be investigated. Linear models, compared to K-NN, make a heavier assumption about structure, namely that the output prediction of the model is a weighted linear combination of the inputs. This assumption yields predictions that are stabler (less vulnerable to overfitting) but possibly less accurate.

# 1 Linear Regression

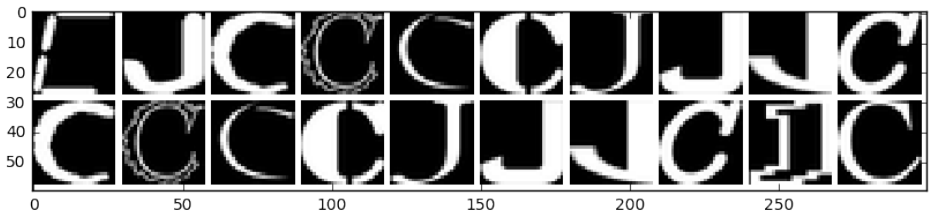
Linear regression can be used for classification in which the training targets are either 0 or 1. Once the model is trained, we can determine an input's class label by thresholding the model's prediction. We will consider the following Mean Squared Error (MSE) loss function $\mathcal{L}_\mathcal{D}$ and weight decay loss $\mathcal{L}_W$ for training a linear regression model, in which the goal is to find the best total loss,

$$\begin{aligned}\mathcal{L} =& \mathcal{L}_\mathcal{D} + \mathcal{L}_W \\ =& \sum_{n=1}^{N} \frac{1}{2N} \|W^T \mathbf{x}^{(n)} + b - y^{(n)}\|_2^2 + \frac{\lambda}{2}\|W\|_2^2\end{aligned}$$

You will train linear regression model for classification on the *two-class notMNIST* dataset (described in the next section) using the stochastic gradient descent (SGD) algorithm. You will be using the validation set to tune the hyper-parameter of the weight decay regularizer.

### Two-class notMNIST dataset

We use the following script to generate a smaller dataset that only contains the images from two letter classes: "C"(the positive class) and "J"(the negative class). This smaller subset of the data contains 3500 training images, 100 validation images and 145 test images.




```
with np.load("notMNIST.npz") as data :
    Data, Target = data ["images"], data["labels"]
    posClass = 2
    negClass = 9
    dataIndx = (Target==posClass) + (Target==negClass)
    Data = Data[dataIndx]/255.
    Target = Target[dataIndx].reshape(-1, 1)
    Target[Target==posClass] = 1
    Target[Target==negClass] = 0
    np.random.seed(521)
    randIndx = np.arange(len(Data))
    np.random.shuffle(randIndx)
    Data, Target = Data[randIndx], Target[randIndx]
    trainData, trainTarget = Data[:3500], Target[:3500]
    validData, validTarget = Data[3500:3600], Target[3500:3600]
    testData, testTarget = Data[3600:], Target[3600:]
```

1. **Tuning the learning rate**:
   Write a Tensorflow script that implements linear regression and the stochastic gradient descent algorithm with mini-batch size $B = 500$. Train the linear regression model on the *two-class notMNIST* dataset by using SGD to optimize the total loss $\mathcal{L}$. Set the weight decay coefficient $\lambda = 0$ and use 20000 iterations (one iteration is an SGD update through the entire pass of one mini-batch). Plot the training loss function vs number of epochs for learning rate $\eta = \{0.005, 0.001, 0.0001\}$ on the same plot (epoch means passing over the whole training dataset once, i.e., one epoch = 7 mini-batches for $B = 500$). Choose the best learning rate found and discuss the effect of the learning rate value on training convergence. **Note: Each epoch consists of one pass of all the training examples.**

2. **Effect of the mini-batch size**:
   Run the SGD algorithm with $B = \{500, 1500, 3500\}$ with the learning rate chosen from point 1, $\lambda = 0$, and 20000 iterations. Report the final training MSE for each mini-batch value. What is the best mini-batch size in terms of training time? Comment on your observation.

3. **Generalization**:
   Run SGD with mini-batch size $B = 500$, learning rate $\eta = 0.005$, and 20000 iterations to choose the best weight decay coefficient that gives the best classification accuracy on the validation set from $\lambda = \{0., 0.001, 0.1, 1\}$. Report the validation set accuracy for different values of $\lambda$. Report the best $\lambda$ based on the validation accuracy, what is the test accuracy for this $\lambda$? Comment on the effect of weight-decay regularization on the model performance. Also, comment on why we need to tune the hyper-parameter $\lambda$ using a validation set instead of the training set.

4. **Comparing SGD with normal equation**:
   For zero weight decay, write a TensorFlow script to find the optimal linear regression weights on the *two-class notMNIST* dataset using the "normal equation" of the least squares formula, i.e., closed form expression. Compare in terms of final training MSE, accuracy and computation time between SGD and normal equation. Also, discuss when SGD is more practical than using normal equation.

# 2 Logistic Regression

## 2.1 Binary cross-entropy loss

The MSE loss function works well for typical regression tasks in which the model outputs are real values. Despite its simplicity, MSE can be overly sensitive to mislabelled training examples and to outliers. A more suitable loss function for the classification task is the cross-entropy loss, which compares the log-odds of the data belonging to either of the classes. Because we only care about the probability of a data point belonging to one class, the real-valued linear prediction is first fed into a sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ that "squashes" the real-valued output to fall between zero

and one. The model output is therefore $\hat{y}(\mathbf{x}) = \sigma(W^T\mathbf{x} + b)$. The cross-entropy loss is defined as:

$$\begin{aligned}\mathcal{L} =& \mathcal{L}_{\mathcal{D}} + \mathcal{L}_W \\ =& \sum_{n=1}^{N} \frac{1}{N}\Big[ - y^{(n)} \log \hat{y}(\mathbf{x}^{(n)}) - (1 - y^{(n)}) \log(1 - \hat{y}(\mathbf{x}^{(n)}))\Big] + \frac{\lambda}{2}\|W\|_2^2\end{aligned}$$

The sigmoid function is often called the logistic function and hence a linear model with the cross-entropy loss is named "logistic regression".

1. **Learning**:
   Write a TensorFlow script that implements logistic regression and the cross-entropy loss (and you want to use the *tf.nn.sigmoid_cross_entropy_with_logits* utility function to avoid numerical underflow). Set the weight-decay coefficient $\lambda = 0.01$ and use 5000 iterations. Train the logistic regression model using SGD and a mini-batch size $B = 500$ on the *two-class notMNIST* dataset. Plot the best training and validation curves for both cross-entropy loss and classification accuracy vs. the number of epochs after tuning the learning rate. Report the best test classification accuracy obtained from the logistic regression model.

2. **Beyond plain SGD**:
   With the same weight-decay coefficient $\lambda = 0.01$, learning rate $\eta = 0.001$, and mini-batch size $B = 500$, train the logistic regression model with the Adam optimizer[1] (use the *tf.train.AdamOptimizer* function). Plot the best training cross entropy curve using Adam Optimizer along with the training cross entropy curve of SGD. Compare the two sets of learning plots and explain how the Adam optimizer may help learning from the notMNIST dataset. (**For the rest of the assignment, you should use the Adam optimizer in all numerical experiments.**)

3. **Comparison with linear regression**:
   For zero weight decay, compare the train, validation and test classification accuracy of the linear regression using "normal equation" to the optimal logistic regression learnt without weight decay.

   Also, for zero weight decay and learning rate of 0.001, plot the training cross entropy (or MSE) and accuracy curves for linear and logistic regression. For linear regression please use the Adam optimizer instead of normal equations in this part. What do you observe about the effect of the cross-entropy loss on the convergence ? To help explain your observation, you may wish to plot the cross-entropy loss vs squared-error loss as a function of the prediction $\hat{y}$ within the interval [0, 1] and a dummy target $y = 0$ in 1-D.

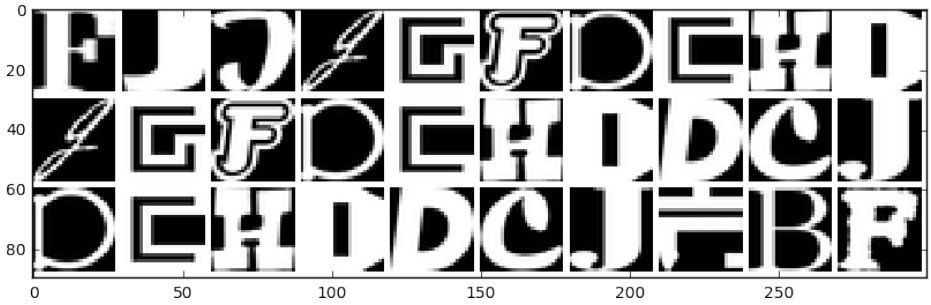
## 2.2 Multi-class classification

In this section, you will train multi-class logistic regression over two datasets; the *notMNIST* and the *FaceSrcub* as described hereafter.

[1] https://arxiv.org/abs/1412.6980

## notMNIST Dataset

The dataset that we will use in this assignment is a permuted version of notMNIST[2], which contains 28-by-28 images of 10 letters (A to J) in different fonts. This dataset has 18720 instances, which can be divided into different sets for training, validation and testing. The provided file is in **.npz** format which is for Python. You can load this file as follows.



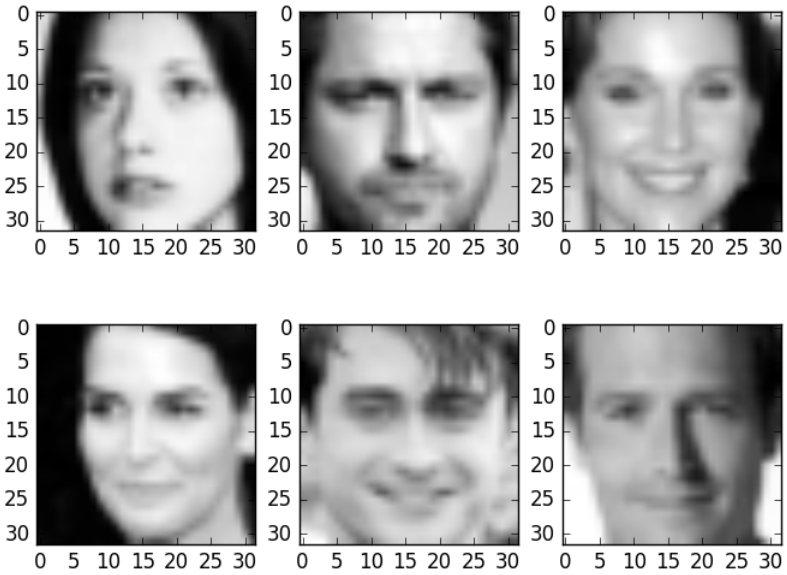

```
with np.load("notMNIST.npz") as data:
    Data, Target = data ["images"], data["labels"]
    np.random.seed(521)
    randIndx = np.arange(len(Data))
    np.random.shuffle(randIndx)
    Data = Data[randIndx]/255.
    Target = Target[randIndx]
    trainData, trainTarget = Data[:15000], Target[:15000]
    validData, validTarget = Data[15000:16000], Target[15000:16000]
    testData, testTarget = Data[16000:], Target[16000:]
```

## FaceScrub Dataset

You will use a tiny version of the FaceScrub dataset[3]. The FaceScrub dataset consists of 100,000+ images of 500+ celebrities. The tiny version consists only of six celebrities (three actors and three actress) with average of 150 image per subject. The images were downloaded and converted to grey using the available code in http://www.cs.toronto.edu/~guerzhoy/321/proj1/, then, the faces were cropped and resized to 32×32 images and stored as .npy file. You are provided two .npy files; the *data_facescrub.npy* file is an array of size 936×32×32 which contains 936 images of the six celebrities, and the *target_facescrub.npy* file that encodes the class of each image and of size 936×2. The first column encodes the name (ID) of the actor and the second column encodes the gender as follows:

[2] http://yaroslavvb.blogspot.ca/2011/09/notmnist-dataset.html
[3] http://vintage.winklerbros.net/facescrub.html

- The name (ID) of the actors: 'Lorraine Bracco', 'Gerard Butler', 'Peri Gilpin', 'Angie Harmon', 'Daniel Radcliffe', and 'Michael Vartan' are encoded as '0', '1', '2', '3', '4', and '5', respectively.
- The gender of the actors: 'Male' and 'Female' are encoded as '0' and '1', respectively.



You should use the following code to load the dataset.

```
def data_segmentation(data_path, target_path, task):
    # task = 0 >> select the name ID targets for face recognition task
    # task = 1 >> select the gender ID targets for gender recognition task
    data = np.load(data_path)/255
    data = np.reshape(data, [-1, 32*32])
    target = np.load(target_path)

    np.random.seed(45689)
    rnd_idx = np.arange(np.shape(data)[0])
    np.random.shuffle(rnd_idx)

    trBatch = int(0.8*len(rnd_idx))
    validBatch = int(0.1*len(rnd_idx))

    trainData, validData, testData = data[rnd_idx[1:trBatch],:], \
                                     data[rnd_idx[trBatch+1:trBatch + validBatch],:],\
                                     data[rnd_idx[trBatch + validBatch+1:-1],:]

    trainTarget, validTarget, testTarget = target[rnd_idx[1:trBatch], task], \
                                target[rnd_idx[trBatch+1:trBatch + validBatch], task],\
                                target[rnd_idx[trBatch + validBatch + 1:-1], task]
```

```
return trainData, validData, testData, trainTarget, validTarget, testTarget
```

1. Write a TensorFlow script that implements logistic regression using softmax output and with multi-class cross-entropy loss (you will want to use the *tf.nn.softmax_cross_entropy_with_logits* utility function, to avoid numerical underflow). Regularize the weights with $\lambda = 0.01$. Using a mini-batch size of $B = 500$ and a learning rate that you tune, train the logistic regression model on the full *notMNIST* dataset with 10 classes. For the best value of learning rate, plot the training and validation curves for both cross-entropy loss and classification accuracy vs. the number of epochs. Report the best test classification accuracy obtained from the logistic regression model. Is the classification performance better or worse than the binary-class problem in question 2.1? Explain intuitively the difference. Also, display one failure case where the classifier's output is not the same as the label (display the test image and the classifier's output ) and comment on your results.

2. Modify the multi-class logistic regression program in part 1 and train it over the *faceScrub* database for face recognition task. Use a mini-batch size of $B = 300$ and tune the weight decay and learning rate to get the best performance using the validation set. For the best value of learning rate and weight decay, plot the training and validation curves for both cross-entropy loss and classification accuracy vs. the number of epochs. Report the best test classification accuracy obtained from the logistic regression model. Also, display one failure case where the classifier's output is not the same as the label (display the test image and the classifier's output ) and comment on your results.