Introduction

For this assignment, we will perform binary classifications of facial expressions using a logistic regressor and a 1-layer neural network, as well as other classifiers from *scikit-learns*. The main goal is to successfully write the predict and train functions for both classifiers and compare these with values obtained from other classifiers. The dataset we are using is quite noisy and was obtained from the very 2013 Kaggle facial expression classification competition[1]. Although the original dataset contained 7 emotional classes (six of which are shown in Figure 1), we will only use two of them (happy and sad) for this assignment. To complete the assignment you will need to download the homework4.zip file from myCourses as it contains these instructions, the data files and code snippets required.

Figure 1: The top row shows three examples of correctly labeled faces from the Kaggle challenge; left to right - angry, disgust and fear. The bottom row shows three incorrectly labeled faces; left to right - happy, sad and surprise. Neutral face is not shown.

Requirements

You should perform this assignment in Python. It is due on **Wednesday December 5, 2019 by 11:59pm**. You are encouraged to ask questions and have discussions about the homework on myCourses, but please do not post your solutions or any closely related material. If there are parts of the assignment that are not clear to you, or if you come across an error or bug please don't hesitate to contact the TA or Instructor. Chances are that other students are also encountering similar issues.

You are allowed to collaborate with other students as far as discussing ideas and possible solutions, however you are required to code the solution yourself. Copying others' code and

[1] `https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-\ \challenge/data`

changing all the variable names is not permitted. You are not allowed to use solutions from similar assignments in courses from other institutions, or those found elsewhere on the web. If you access such solutions YOU MUST refer to them in your submission write-up. The data and starting code are provided in the zipped file **homework4.zip** which can be downloaded from myCourses.

Your solutions should be submitted via dropbox on myCourses. Your submitted zipped file for this assignment should be named **LastnameFirstname_hw4.zip**. Failure to follow this naming convention will result in delays in grading your work. Your zipped file should contain: (i) a PDF file named LastnameFirstname_hw4.pdf with your report; and (ii) the two files `LRModule.py` and `NNModule.py` used to generate the solutions. Please do **not** include the data files with your final submission as they are very large. For grading, we will be testing your code on a separate test dataset[2].

The Data Files

You are provided with two data files, where the first file `fer3and4train.csv` contains the training data with 12,066 data samples and the second file `fer3and4test.csv` contains the data that you will be testing your classifier on. Results should be reported on the test dataset which contains 2000 data samples. The files were created using `fer2013.csv` from Kaggle but have been shuffled and augmented to avoid the class imbalance problem.

All the files are stored as comma separated files with three columns each. The first column contains the label of the emotional expression, where 3=happy and 4=sad; the second column contains 2304 integer values (between 0 and 255) obtained by vectorizing 48×48 grayscale images of faces; and the last column states in which part of the development process the data should be used (i.e training or testing). Finally, the TA has another data set for final testing, which you will not have access to. This will be used when grading the goodness of your submitted classifiers.

The logistic regression classifier (Total 30 points)

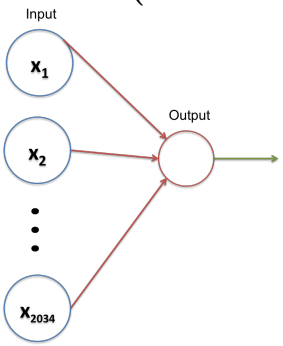

Figure 2: The logistic regression classifier

The file `LRModule.py` contains the skeleton code for your logistic regression module. This is accompanied by an auxiliary file called `helper.py` containing the names of some helper

[2]We will be using a different data split from what you have

functions needed for your module to run. In `LRModule.py`, different parts of the code have been marked for your implementation and they correspond to the following:

1. Load the training data samples $\mathbf{X}$ and their corresponding class labels Y using the helper function `getBinaryfer13Data`. Call the `train` function to learn the weights and bias of the unit. The following occur within the `train` function.

2. Initialize the weights W to small random numbers (variance - zero); also initialize the bias b to zero

3. Create a loop over the number of epochs specified. Within the loop, the following occur:

4. Call a `forward` function to calculate $P(Y|X)$ also known as pY. The `forward` function implements $\sigma(\mathbf{X}.\cdot\mathbf{W}+b)$. The argument of this equation can be implemented in *numpy* as $X.dot(W) + b$. Sigmoid is provided as a helper function.

5. Perform gradient descent using the equations below:

$$W = W - \eta\frac{\partial J}{\partial W} \Rightarrow W- = \eta \cdot \mathbf{X}^\top \cdot (pY - Y) \tag{1}$$

   Note: When doing matrix computation, the product the vectors $X^\top Y$ can be written as $np.dot(\mathbf{X}.\mathrm{T}, \mathbf{Y})$; Similarly,

$$b = b - \eta\frac{\partial J}{\partial b} \Rightarrow b- = \eta \cdot (pY - Y) \tag{2}$$

6. Apply the forward algorithm to predict the new labels for the validation data. Compute the sigmoid costs of predicting the validation data compared with the true labels, append resulting cost to a growing array.

7. Keep note of the best error value on the validation data. This is required in the final report.

8. Construct a graph of your validation error (created from the above process) to show how the error changes over time. This is also required in the final report.

9. Lastly load a new dataset, your test data from the file `fer3and4test.csv`. Compute and display the accuracy (or classification rate) for predicting this new dataset from your trained network. This value should also be shown in your final report.

The neural network classifier (Total 40 points)

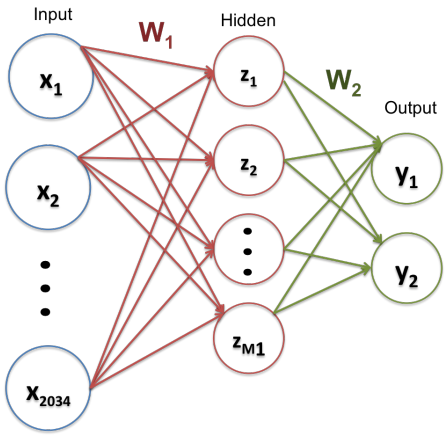

Figure 3: The single hidden layer neural network with two softmax output nodes

Similar to the previous exercise, the file `NNModule.py` contains the skeleton code for your neural network module; again there are some helper functions in `helper.py`. You are to implement the following:

1. Load the training data samples $\mathbf{X}$ and their corresponding class labels Y using the helper function `getBinaryfer13Data`. Call the `train` function to learn the weights and bias of the unit. The following occur within the `train` function.

2. Initialize the weights W_1, W_2 to small random numbers (variance - zero); also initialize the biases b_1, b_2 to zero. Set the number of hidden weights $M1$ here also. Note that the dimensions of these parameters here are different from those in the LRModule. W_1 is $D \times M1$, where $M1$ is the number of hidden nodes and the dimension of W_2 is $M1 \times k$, where k is the number of output classes.

3. Create a loop over the number of epochs specified. Within the loop, the following occur:

4. Call a `forward` function twice to calculate $P(Ytrain|X)$ also known as pY and $Ztrain$ (activations at hidden layer); and the other to calculate $P(Yvalid|Xvalid)$ and $Zvalid$ on the validation data. This implies that your `forward` function needs to return two values (i) pY, the output of the softmax classifier and (ii) the hidden activations Z, based on which activation function you choose (*tanh, sigmoid, or ReLU*).

5. Now we do back propagation by first performing gradient descent using equations (3) and (4) below;

$$W_2 = W_2 - \eta \frac{\partial J}{\partial W_2} \Rightarrow W_2 -= \eta \cdot \mathbf{Z}^\top \cdot (pY - Y) \tag{3}$$

$$b_2 = b_2 - \eta \frac{\partial J}{\partial b_2} \Rightarrow b_2 -= \eta \cdot (pY - Y) \tag{4}$$

then we propagate the errors we obtained from testing the newly updated W_2 and use this to update W_1 and b_1 via equations (5)-(7)

$$\frac{\partial J}{\partial Z} = (pY - Y) \cdot {W_2}^{\top} \cdot (1 - \mathbf{Z}^2) \tag{5}$$

$$W_1 = W_1 - \eta \cdot \mathbf{X} \cdot \frac{\partial J}{\partial Z} \tag{6}$$

$$b_1 = b_1 - \eta \cdot \frac{\partial J}{\partial Z} \tag{7}$$

Matrix multiplications in numpy will be sufficient to complete the processes.

6. Apply the forward algorithm to predict the new labels for the validation data and also compute the sigmoid costs of predicting the validation data compared with the true labels. Append the resulting cost to the growing array.

7. Keep note of the best error value on the validation data. This is required in the final report.

8. Construct the graphs of both your training and validation errors (created from the above process) to show how the errors change with time. This is also required in the final report.

9. Lastly load a new dataset, your test data from the file `fer3and4test.csv`. Compute and display the accuracy (or classification rate) for predicting this new dataset from your trained network. This value should also be shown in your final report.

Adding regularizers (Total 10 points)

After the initial training and testing, go back and add a regularizer to the cost function. Now report your new error rates and accuracies.

The Report (Total 20 points)

You should also turn in a well-written, neat and concise report which includes a description of the two classifiers along with the specific requirements made in the skeleton code. Extra credit (maximum of 10 points) would be given for applying other classifiers such as the SVM (an implementation exists on scikit-learns and also in Matlab) on the dataset.

You should turn in both your code and report discussing your solution and results to get full credit.

BONUS 10 points: Apply at least two other classifiers from say *scikit-learns* and report the accuracies from these. Discuss your findins briefly in your report. Two classifiers we covered in class include SVM, and Adaboost.