Assignment 4. Neural Network Classifiers with One and Two Hidden Layers

Due Date: **March 31, 11:30 pm**
Assessment: 9.5% of the total course mark.

Description:

In this assignment you are required to build neural network classifiers with one and two hidden layers for binary classification. You will use the banknote authentication data set (provided in the text file 'data_banknote_authentication.txt'). The data set consists of 1372 examples, each example representing a banknote. There are four predictor variables (i.e., features). The goal is to predict if a banknote is authentic (class 0) or a forgery (class 1).

Before starting building your classifiers you have to split the data into the test set, the validation set and the training set. You decide what splitting ratio to choose and mention it in the report. After you separate the test, validation and training data, you have to standardize the features, (i.e. center and scale them) in the training set and then apply the transformations to the features in the validation and test sets.

Use ReLU as the activation function at the hidden units and use sigmoid at the output unit. Train your networks using the average cross-entropy loss on the training set. Stochastic gradient descent (SGD) can be used in the optimization process. You can choose the learning rate 0.005 or another value.

You have to choose the number of units in the hidden layers (denoted by n_1 for hidden layer 1 and n_2 for hidden layer 2) using the validation set. Consider different pairs (n_1, n_2) and for each choice of (n_1, n_2), train the network and use early stopping to determine the weights. In other words, fix a number of epochs to run the SGD algorithm (for instance, 100 epochs - one epoch is one pass through all the training examples). After each epoch, compute the training cross-entropy loss and the validation cross-entropy loss. At the end, choose the weights that achieve the lowest validation cross-entropy loss.

Your goal is to find the best pair (n_1, n_2) with $n_1+n_2 \leq 10$, i.e., the pair with the smallest misclassification error. If you obtain more pairs achieving the smallest misclassification error, then choose the one with the smallest number of weights.

Similarly, you will have to implement a network with just one hidden layer and at most 10 hidden nodes ($n_1 \leq 10$) and you will have to find the best value of n_1.

Recall that SGD starts with a random assignment of values to the weights of the network. It then proceeds in iterations, which are organized in epochs. Each epoch represents one pass through all the training data. At the beginning of each epoch, shuffle the training examples, then iterate through all of them. At each iteration, you have to compute the current gradient using the forward pass followed by the backward pass. Next, use the gradient to update the weights and biases.

You have to write a report to present your results and their discussion. In your report you have to describe your strategy for selecting the pairs (n_1, n_2) (for the case of two

hidden layers) that you used in the validation process (in case you did not use exhaustive search) and to explain why you think that they were sufficient. Specify the number of hidden units of the best model and the weights of the model. Include a table with the training and validation misclassification error for each pair (n_1, n_2) that you tried. Plot the learning curves for the training and validation sets only for the best model. Indicate the misclassification error on the test set for the best model.

Do the same as above for the case with only one hidden layer.

Include in the report any other observations that you find valuable. Specify how you would try to improve the performance of the neural network classifier if the performance is not already optimal and if you had more time to work on the assignment.

Besides the report, you have to submit your numpy code. The code has to be modular. Write a function for each of the main tasks and for each task that is executed multiple times (e.g., computing the average error, computing the gradient of ReLU, forward pass, backward pass, network training, plotting, etc.). The code should include instructive comments. **Use vectorization whenever possible. Use vectorization when computing the average error!**

Code for initialization n:

You may use the following strategy for the initialization of weights and biases:

```
W1 = 0.1 * np.random.randn(D,n1)
W2 = 0.1 * np.random.randn(n1, n2)
W3 = 0.01 * np.random.randn(n2, 1)
b1 = np.zeros((n1))
b2 = np.zeros((n2))
b3 = 0
```

Submission Instructions:

- Submit the report in pdf format, the python file (with extension ".py") containing your code, and a short demo video. The video should be 1 min or less. In the video, you should scroll down your code, and **briefly explain what each part does**. Submit the files in the Assignments Box on Avenue.