CS 280
Programming Assignment 2
Implementing the Backpropagation Algorithm

Jessa Faye M. Rili
jessa.rili@eee.upd.edu.ph
CS280 ThuG-A

I. The Data Set

The training data set and test data set given for this classification task is contained in *data.csv/data_labels.csv* and *test_set.csv*, respectively. Each row is one data sample and is described by 354 features. The training data set contains 3486 samples while the test data set contains 701 samples.

A. Handling Imbalanced Datasets

The procedures, details, and figures in this section is obtained by running the scripts in *Handling_Imbalanced_Data.ipynb.*

The number of training data samples labelled with each unique training label is described numerically in Table 1 and graphically in Figure 1. The information suggests that the training data set is highly imbalanced, i.e. some labels have a significantly greater ($> \sim 10 \times$) frequency among the other labels in the data set. Using the training data set as it is will make the neural network (or any machine learning algorithm) become biased towards the classes with a significantly higher frequency.

One way to handle this by **Undersampling** the biased class/es, e.g. by randomly removing data samples labelled with the biased class/es. This is done such that the data sample count for the biased class/es would be closer in magnitude to the data samples labelled with the unbiased class/es. This would be a viable option if our original training data set was large (e.g. tens of thousands). Unfortunately, that is not the case here.

Table 1 Counts per Class in the Training Data Set

Training Label	Data Sample Count
1	1625
2	233
3	30
4	483
5	287
6	310
7	52
8	466

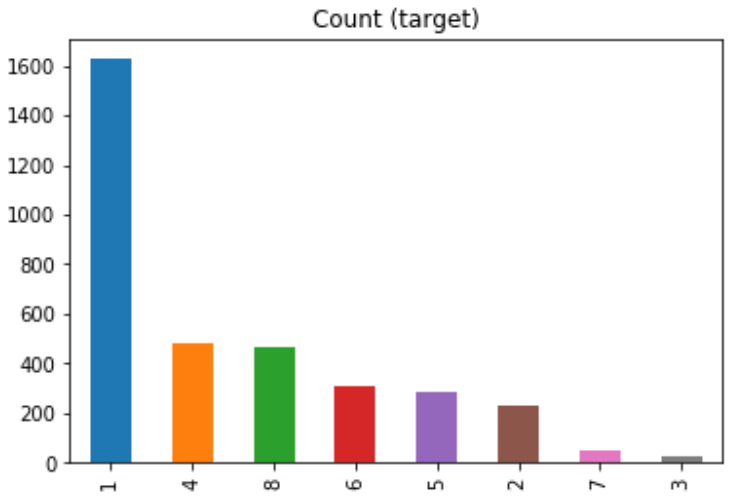

Figure 1 Counts per class (Before Balancing)

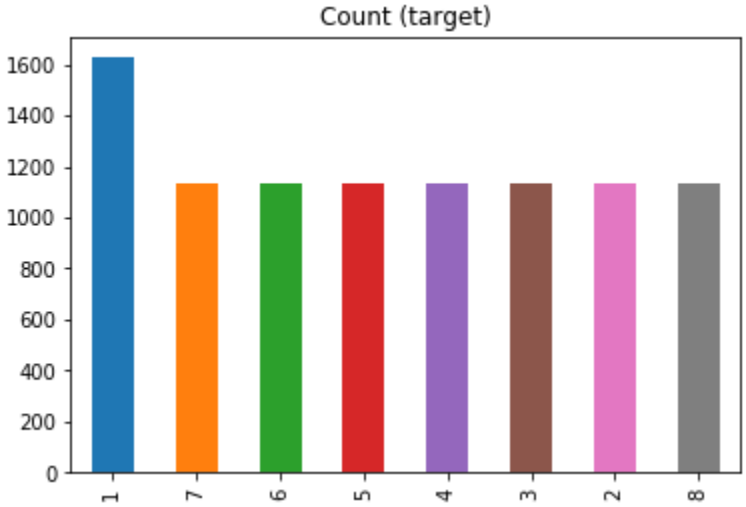

Figure 2 Counts per class (After Balancing)

Applying the same logic employed in the Undersampling method, **Oversampling** can instead be applied to the data samples labelled with the unbiased (i.e. the significantly less-occurring) classes. This can be done by randomly choosing existing data samples to be repeated in the data set such that the data sample counts for the unbiased classes will increase to a comparable amount with the biased class/es'.

The latter method, Oversampling, was used for this assignment. Looking at the data sample counts, it seems the training data set is biased towards Class 1 which has 1625 data samples. And thus, oversampling was applied to data samples labelled with the unbiased classes, i.e. Classes 2, 3, 4, 5, 6, 7 and 8 such that the said unbiased classes will have a data sample count of 80% of the biased class's. A visualized version is after performing oversampling is shown on Figure 2.

B. Partitioning the Training Data

After balancing the training data, the data then needs to be partitioned into training (i.e. data to be used during training) and validation (i.e. data excluded from training to be used to detect overfitting during training) sets. For this assignment, 80% of the original training set was randomly chosen to be part of the training data set while the remaining 20% is used as the validation data set.

II. Implementing the Artificial Neural Network Classifier

The information outlined in this section was obtained by running the scripts found in the first part of PA2_RILI.ipynb.

The task is to implement a four-layer artificial neural network, i.e. it contains 1 input layer, 2 hidden layers, and an output layer.

A. Choosing the number of hidden layer nodes

Choosing how many neurons to put in the ANN was done by trial-and-error, i.e., different values were tried while measuring the validation error. The idea is to choose the number of neurons that would yield the lowest validation error. It isn't as simple as it sounds, however, because validation errors were random for every training due to the random initializations and shuffling of the training data every epoch. To solve this, small values (e.g. 7, 5) and big values (e.g. 14, 10) was first tried and the validation error noted. Then try more values in-between to get an idea of the trend of the validation error. In the end, it boiled down to two values, (11, 9) and (10,11). To decide, each of the last two values were tried with the ANN nine (9) times and then the average validation error was computed. Finally, (11,9) was chosen since it yielded the smallest average validation error over the validation set. See

Table 2.

B. Altering the Learning Rate

While testing, it was observed that there was difficulty reducing the training error to the specified target stopping criterion, especially in the last few epochs when the training error is close to the stopping criterion. To solve this, a dynamically-changing learning rate was included during the training - the learning rate parameter would programmatically get smaller as the training error approached the target stopping criterion to adjust the granularity of updates to the weights and biases — therefore adjusting the granularity of the updates to the training error.

C. Plots of training and validation errors vs epoch number

For the chosen number of neurons for each hidden layer, the training error and validation error are plotted and shown in Figure 3.

Table 2: Different number of neurons in hidden layers

NUM_HIDDEN1_NEURONS	NUM_HIDDEN2_NEURONS	Final Validation Error
7	5	0.002389
14	10	0.002471365985
7	2	0.00605162675
5	5	0.002831948990
11	9	Average: 0.023650711 0.002113073 0.195887428 0.002113073 0.001905338 0.002222725 0.001881859 0.002166184 0.002329919 0.0022368
10	8	Average: 0.002412952 0.002357643 0.002454466 0.002131607 0.002740162 0.001612285 0.002817457 0.002385028 0.002482237 0.00273568
12	10	0.002294159842 0.002545774593
10	10	0.002307903756 0.001919131208
10	9	0.002301415621 0.002270380561
8	8	0.002736343907 0.002169085613
9	7	0.002897494506 0.002715720439

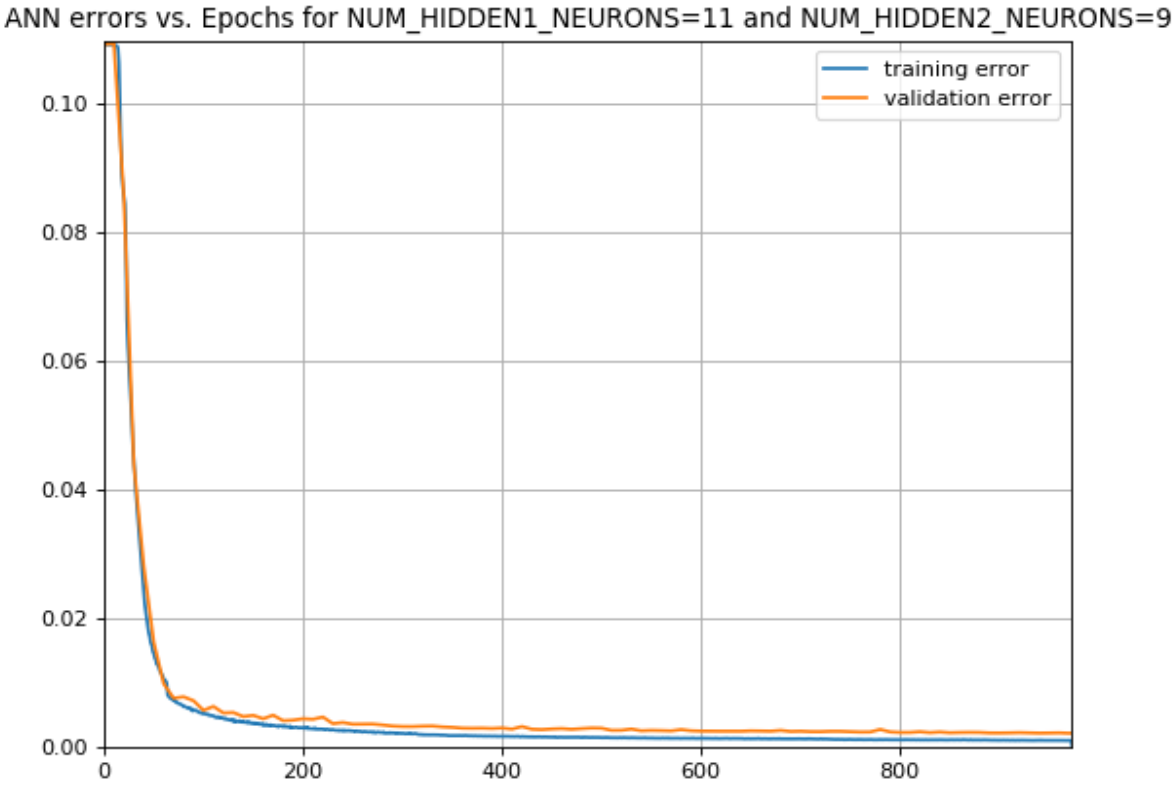

Figure 3 Error Plots

III. Implementing the Support Vector Machine Classifier

The information presented in this section was obtained by running the scripts in the later part of PA2_RILI.ipynb.

For implementing the SVM algorithm, the python library scikit-learn was used. An SVC object was instantiated and trained with the same training data as the one used with the ANN discussed previously.

To choose which kernel to use, the three kernel choices were trained and tested on the validation data set with the accuracies measured. The results are tallied in Table 3. The linear kernel is chosen because it yields the highest accuracy among the three.

Table 3 Choosing the Kernel for SVM

kernel	Accuracy on the validation data set
linear	0.9539267015706806
poly	0.16910994764397905
rbf	0.44293193717277485

IV. Comparison between ANN and SVM

Table 4 shows a simple comparison between the two algorithms. The ANN was able to score a higher accuracy on the validation set but it took a significantly longer time to train it than the SVM.

Table 4 ANN vs SVM

Algorithm	Accuracy on Validation Set	Running Time
ANN	0.987434554973822	8-15 minutes
SVM	0.9539267015706806	7 seconds

V. Conclusion

The Artificial Neural Network and its encompassing algorithms like forward-pass, back-propagation, etc., was successfully implemented in Python. There is, however, an observable gap between the training error and the validation error which suggests overfitting which may be caused by the way the imbalanced dataset was handled, among other things. A few things to try to reduce this gap is to (1) gather more real data to make the dataset balanced, (2) try a different error function (e.g. accuracy) to be used for the back propagation.

Also, the SVM was also implemented albeit using scikit-learn libraries using linear kernels which were proven to be most effective among the other kernels for this particular dataset. The SVM was able to score a high accuracy score over the validation set, but the ANN beat SVM's accuracy score. In terms of running/training time, however, SVM is significantly faster than ANN.