# Neural Networks from Scratch

Hunor Laczko and Smriti Joshi

## I. INTRODUCTION

Deep learning has proved to be a useful tool capable of modelling complex functions present in real-life problems and providing accurate results. This laboratory report presents the analysis of the performance of basic supervised neural networks. Three types of neural networks are evaluated: Binary classification using single neuron, binary classification using single hidden layer and multiclass classification using single hidden layer. This report has the following structure: Tools used to train and evaluate the network are discussed in Section II, analysis of the data is presented in Section III, the methods and implementations are discussed in Sections IV and V and the results are summarized in section VI. Further, the report is concluded and potential future work is discussed in section VII.

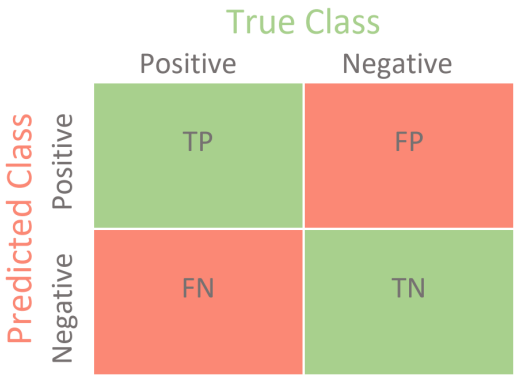


Fig. 1. Confusion Matrix ( TP: True positive, FP: False positive, FN: False Negative, TN: True Negative) Source: Towards Data Science

## II. PRELIMINARY

This section discusses the cost function and metrics used for training and evaluating the network.

### A. Cost function: Cross Entropy

Cross entropy is a popular choice for loss function, it measures the similarity between two vectors or distributions. In a typical multiclass classification application it is used to measure the similarity between the output of a softmax layer (the probability of object belonging to each class) and the one-hot-encoded ground truth label. Mathematically, it is represented by equation 1.

$$CE = -\sum_{i=1}^{C}(A_i \log B_i) \tag{1}$$

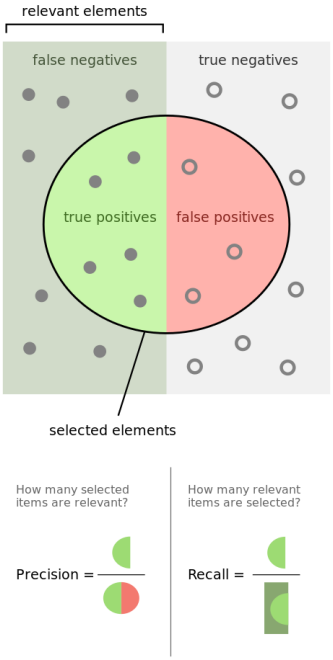


Fig. 2. Visualisation of metrics. Source: Wikipedia

For binary classification, a simplified version called the binary cross entropy is used. Mathematically it can be expressed as follows:

$$BCE = -\sum_{i=1}^{C=2}(A_i \log B_i) = -A_i \log B_i - (1-A_i)\log(1-B_i) \tag{2}$$

In equation 1 and 2, C is the total number of classes, *A* is the true label, *B* is the output probability.

### B. Evaluation metric: Confusion matrix

Confusion matrix is a table used to present the performance of binary classifiers in the case where true labels are known. This table is shown in Figure 1. False negative, also called type II error, refers to the case when the training example belongs to positive class but incorrectly classified as negative class. False positive, called as Type I error, refers to the case when training example is belongs to negative class but incorrectly labelled as positive class.

### C. Evaluation metric: F1-Score

This metric measures the performance of the network from the values of precision and recall in binary classification. This

is chosen over accuracy as evaluation metric due to high data imbalance for first two training networks. Precision is the number of correctly identified positive results divided by the number of all positive results, including those not identified correctly. Recall is the number of correctly identified positive results divided by the number of all samples that should have been identified as positive. This can be visualised in Figure 2. Finally, F-1 score is mathematically given by:

$$F1Score = 2\left(\frac{precision * recall}{precision + recall}\right) = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{3}$$

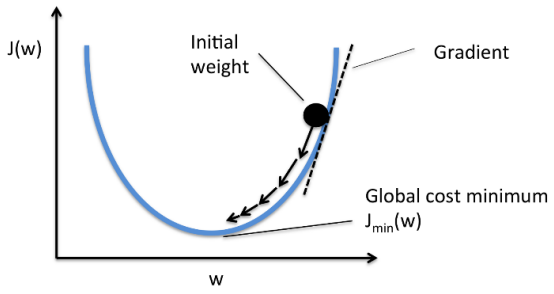


Fig. 3. Visualisation of metrics. Source: Wikipedia

### D. Evaluation metric: BACC Score

This score is useful in cases where there is high imbalance in the dataset between classes. It is known to be useful in deep learning for medical images. As later discussed in section III, we require such score for binary classification. Mathematically, it is given by:

$$BACC = \frac{\left(\frac{TP}{TP+FP} + \frac{TP}{TN+FN}\right)}{2} \tag{4}$$

### E. Optimizer: Gradient Descent

This is an optimization algorithm to calculate the local minimum of a differentiable function by taking steps along the negative of the gradient. The function used here was explained above II-A. This is an iterative method. The process is stopped when local minimum is reached. Figure 3. can be used to visualise this process. The size of the steps is determined by the learning rate as explained in Section V-E.1.

### F. Activation functions

Activation functions are popular to introduce non linearity in the network to fit complex distributions. In the networks, they are used to determine if the neuron should be fired and to what extent. Two kinds of activation functions are used in this assignment. They are explained in more detail below.

*1) Sigmoid:* Sigmoid activation, also known as logistic action, maps the input to the range 0 to 1. In the final layer, it outputs the probability which can be thresholded for binary classification. The function itself can be visualised in Figure 4. Mathematically, it is given by:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

This function is differentiable and monotonic. However, its derivative is not monotonic.

*2) Softmax:* Softmax function can be thought of as extension of logistic function to multiple dimensions. In this assignment, it is used as the last layer of multiclass-classification network to predict the probability distribution over all classes. Mathematically, this can be obtained with the following equation:

$$S(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{6}$$

where the summation is over probabilities for all inputs.

*3) Tanh:* This function is understood as rescaled sigmoid function and outputs values in range [-1,1]. This is believed to have an advantage over sigmoid as negative values are mapped to strongly negative and zero values are mapped to zero. The difference between the two function can be visualised in Figure 4. Mathematically, it is given by:

$$T(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{7}$$

This function is differentiable and monotonic. However, its derivative is not monotonic.

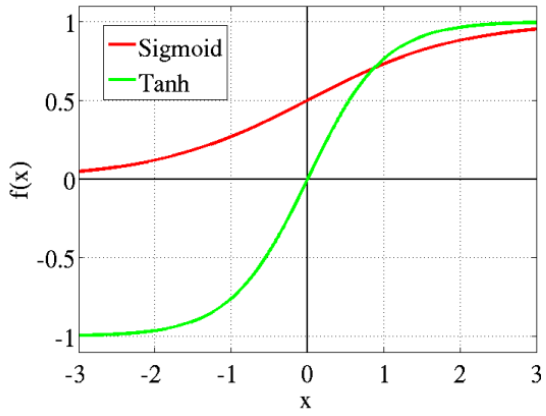


Fig. 4. Comparison of tanh and sigmoid function on a graph. Source: Towards Data Science

## III. DATASET

The dataset used for training and evaluating the two networks mentioned in section IV is MNIST dataset [1]. MNIST stands for Modified National Institute of Standards and Technology. This dataset contain 60,000 training and 10,000 testing images of size 28 x 28 square pixels containing handwritten numbers from 0 to 9. The classes for these

numbers are also 0 to 9 respectively. The training data is further split to create validation set of 10,000 images. The images and their labels can be visualised in Figure 5. The title of each image displays the ground truth label.

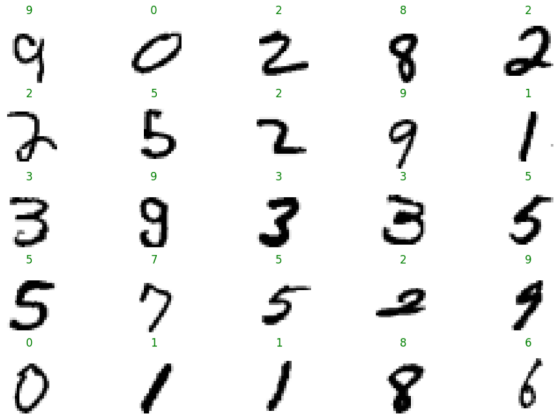

Fig. 5. Subset of dataset used for training for multiclass classification network

For each class, there are roughly equal number of instances in train, validation and test set as seen in the bar graph presented in Figure 6.

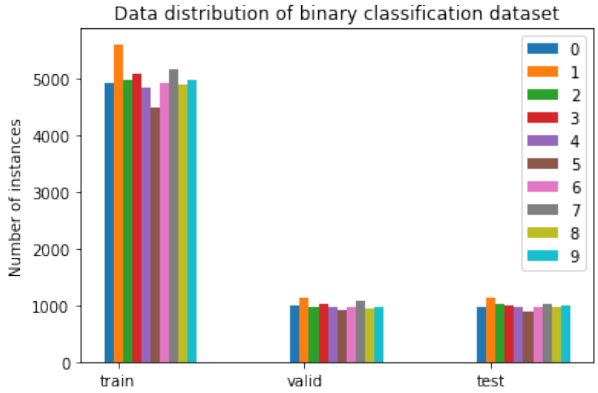


Fig. 6. Graph showing distribution of dataset for multiclass classification

To train the network, these labels are one-hot encoded in the multiclass case. For the binary classification problem, the labels are adjusted such that label for number 0 is '1' and for numbers 1-9 is '0'. Figure 7 displays a subset of this dataset for visualisation.

The title of each image displays the ground truth label. This way of creating labels leads to high imbalance in dataset. This imbalance can be visualised in Figure 8 which shows that there are almost 10 times more instances for '0' class than for '1' class all the three sets. This is why, metrics such as F-score and BACC are evaluated for the binary classification networks.

## IV. METHOD

This section discusses the methods for networks implemented in this assignment.

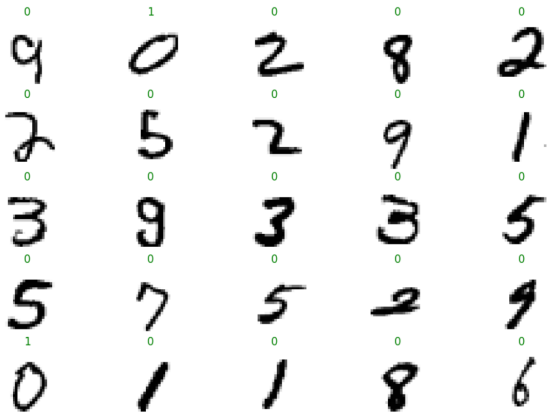

Fig. 7. Subset of dataset used for training for binary classification networks

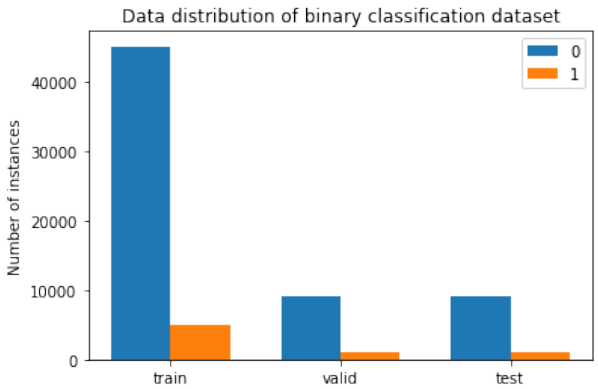


Fig. 8. Graph showing imbalanced dataset for binary classification

### A. Single Neuron

In this assignment, single neuron is used for binary classification of the created dataset explain in the previous section. Single neuron network can be visualised in Figure 9 where the input can be an image or a batch of images as discussed in V later. The output is the probability which is thresholded to give either true or false. Mathematically, single neuron operation looks as follows:

$$\hat{y} = f(\sum_i W_i x_i + b) \tag{8}$$

where $\hat{y}$ is the predicted output, f refers to the activation function, $W_i$ refers to the weights associated with the input pixels and b is the bias.

### B. Single Hidden Layer

*1) Binary Classification:* Single hidden layer in binary classification network replaces the neuron in the previous section by a layer of more than one neuron. This is usually called hidden layer. Mathematically this changes the computation as there are weights from input layer to hidden layer as well as from hidden layer to output layer. All these

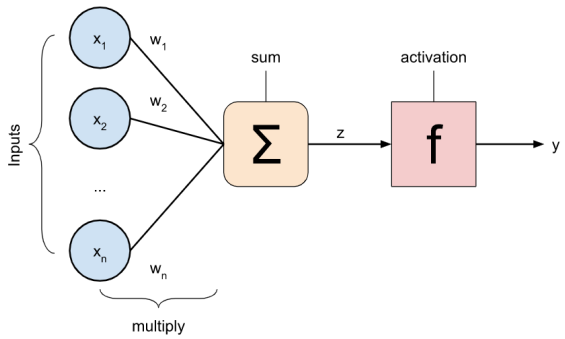


Fig. 9. Single Neuron Network. Source: Towards Data Science

parameters have to be updated at each iteration. This network can be visualised from Figure 10.

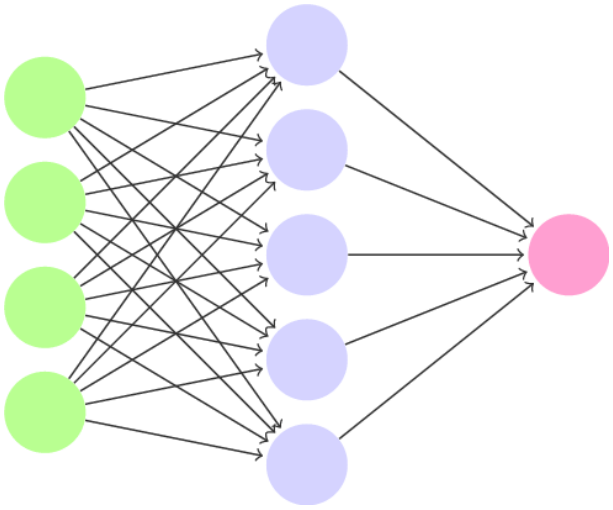

Fig. 10. Single Hidden Layer Network. [2]

### C. Multiclass Classification

Multiclass classification follows the similar routine as binary classification. The differences are as follows: The last layer of the network is replaced by softmax layer which outputs probability of each class. The loss function used to train this network is cross entropy discussed in section II-A. On MNIST dataset, the labels for each image are one-hot encoded.

## V. IMPLEMENTATION

These networks are written in Python with numpy library. The development platform used is: Windows using the PyCharm 2020.1.1.

### A. General structure

All implementations are encapsulated by their respective classes. They all have the same structure, only the implementation of the specific methods differ. First an instance of the class has to be created where the size of the input has to be given. In case of the networks, additional parameters need to be provided. On this instance the *train()* method can be called by giving the training data, validation data, a list of accuracy function which will be evaluated in each iteration, the number of iterations and the learning rate. The *train()* method will call the *forward()* method to perform the forward propagation step, calculate the loss with the *loss()* method, calculates the gradient with the *backprop()* method doing the backpropagation and updates the parameters in each iteration. In each iteration it also evaluates the current parameters by running the prediction and accuracy calculations on the validation data. The *train()* method returns the history of training, meaning a dictionary of the costs and accuracies at each iteration. The class provides an *evaluate()* method as well, which can evaluate the trained class on a given dataset (which is the test split of the dataset) with the given metrics. It also provides methods to save and load the learned parameters to be reused.

The training is not split into batches, at each iteration the whole dataset is processed. This can be done since the dataset is small and fits into the memory.

As metrics, accuracy was used and evaluated at each iteration of training. For binary classification, F1-score and BACC were used as additional metrics to account for the imbalance in the dataset.

### B. Single neuron

The implementation can be found in the *Neuron.py* file as the *Neuron* class. The single neuron uses *sigmoid* as its activation function. The loss applied is binary cross entropy.

### C. Single hidden layer binary

The implementation can be found in the *NeuralNetworkBinary.py* file as the *NeuralNetworkBinary* class. In this case there are a given number of units, meaning neurons, in a middle layer. For the hidden layer two activation functions were experimented with, *sigmoid* and *tanh*. Both were tested and the results are presented in Section VI. The output neuron has a sigmoid activation.

### D. Single hidden layer multiclass

The implementation can be found in the *NeuralNetworkMultiClass.py* file as the *NeuralNetworkMultiClass* class. Compared to the binary case, the only difference is the fact that the output is not a single number but an array of probabilities providing a probability for belonging to each class. This is done by using a softmax activation in place of the sigmoid activation in the output layer. Also the labels for the dataset are presented as one-hot-encoded vectors, meaning each label is a vector with as many elements as the number of classes and it contain a one in the place to which class it belongs.

### E. Parameters

All of the methods presented above have several tunable parameters. These can help achieve a better performance. Each will be be presented next section.

| Network | Metrics | | | | |
|---|---|---|---|---|---|
| | *Accuracy* | *Precision* | *Recall* | *F-Score* | *BACC* |
| *Network 1* | 0.9921 | 0.9948 | 0.9964 | 0.9956 | 0.9811 |
| *Network 2* | **0.9940** | **0.9964** | **0.9969** | **0.9969** | **0.9839** |

TABLE I

METRICS FOR BINARY CLASSIFICATION BY SINGLE NEURON (NETWORK 1) AND SINGLE HIDDEN LAYER (NETWORK 2)

*1) Learning Rate:* Learning rate is used to defined the size of steps taken during gradient descent. In the current application, learning rate is 1 for all the three networks. This parameter is chosen with experimentation with current value yielding best values for metrics.

*2) Iterations:* Number of iterations used in the code is '500' for all the three networks as suggested in the assignment description.

*3) Number of neurons in hidden layer:* The suggested number of neurons are usually between the number of inputs and number of outputs. Here, the input is image vector of size 784 and output is either one label or one-hot encoded vector of size 9. The number of neurons taken in the hidden layer are 64 as suggested in the assignment description. However, experimentation with 4/15 neurons was held which yields similar result.

## VI. RESULTS

All three methods were evaluated on the same splits of the data. The figures showing different costs and metrics are truncated from the front. This is because of the high learning rate, in the first few epoch the values change drastically, distorting the rest of the figures. SO in order to be able to view any changes during the training the results are shown starting from the $50^{th}$ iteration. For binary classification, the results of single neuron and single hidden layer can be compared in Table I.

### A. Single Neuron

Since the dataset is simple, and only a binary classification is needed, a single neuron already performs above 99% accuracy. It can be seen in Figure 11 that both training and validation loss are small and keep decreasing. They are almost the same, although at the very end the training loss decreases slightly more, but this is expected.

Looking at the evaluation metrics in Figure 12. it can be seen that both the accuracy and F1-score are above 99% meaning it performs very well. In this figure the difference between the training and validation values is more visible, but they are still close to each other and follow the same trend. The result on the test set are also similar as Table I shows.

As it can be seen in Table II. there are slightly more false positives than false negatives. This can be expected, since the dataset is highly imbalanced having more positive instances. Some of these misclassified instances can be seen in Figure 13. As explained previously, all images with handwritten number 0 should have label '1' and '0' otherwise. It is seen

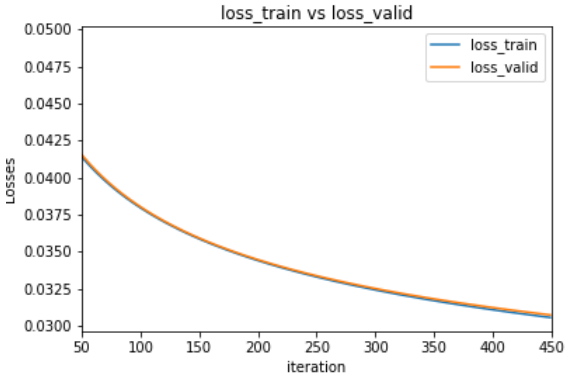


Fig. 11. Variation of costs across iterations in case of single neuron

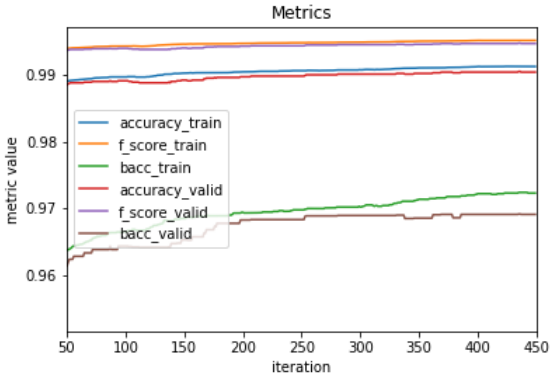


Fig. 12. Variation of different metrics across iterations in case of single neuron

that lot of numbers with label '0' have a generally curved structure resembling the number 0 itself.

| | Ground Truth | | |
|---|---|---|---|
| Predicted Class | | Positive | Negative |
| | Positive | 8973 | 47 |
| | Negative | 32 | 948 |

TABLE II

CONFUSION MATRIX FOR SINGLE NEURON

### B. Single Hidden Layer

*1) Binary case:* This method was tested with 64 units in the hidden layer. The training losses can be seen in Figure 14. As the figure shows there is some divergence between the training and validation loss, but this is expected, since the validation data has not been seen yet by the network. Further, the variation is from third decimal point indicating only a small difference.

The metric values shown in Figure 15. also show the expected results. All metrics keep increasing, both on training and validation data, meaning there is no overfitting. The values are only slightly higher than the single neuron case. Both training and validation values are close an follow the same trends.

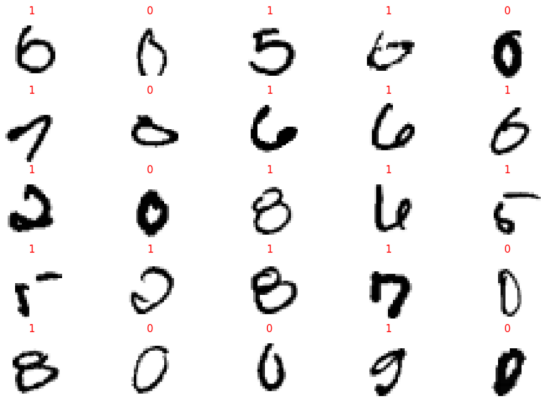

Fig. 13. Wrongly classified images for Single Neuron

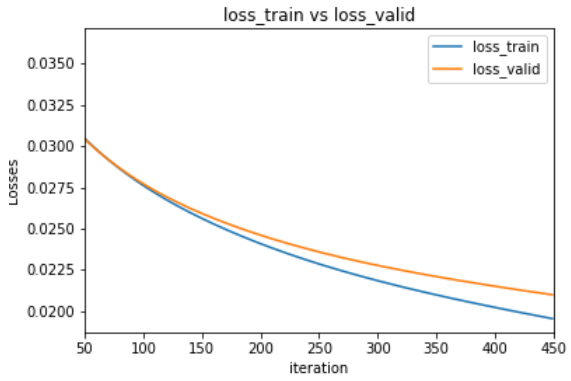


Fig. 14. Variation of costs across iterations in case of a hidden layer with binary output

The confusion matrix in Table III shows better results than the single neuron case, both false negative and false positive cases have decreased, although still slightly more false positive cases are present because of the class imbalance. Some of these wrongly classified instances can be seen in Figure 16.

| | Ground Truth | | |
|---|---|---|---|
| Predicted Class | | Positive | Negative |
| | Positive | 8988 | 32 |
| | Negative | 28 | 952 |

TABLE III
CONFUSION MATRIX FOR SINGLE HIDDEN LAYER NETWORK WITH BINARY OUTPUT

*2) Multiclass case:* Multiclass presents a more complex problem as there are 10 possible classes. The variation of the loss can be seen in Figure 17.

Figure 18. shows the accuracy across iterations during training. It shows that both the training and validation accuracies keep increasing so there is no overfitting. Initially, this network was implemented using sigmoid activation function on the hidden layer. For 500 iterations, it yielded an accuracy of 0.9308. However, as explained in Section II-F, tanh

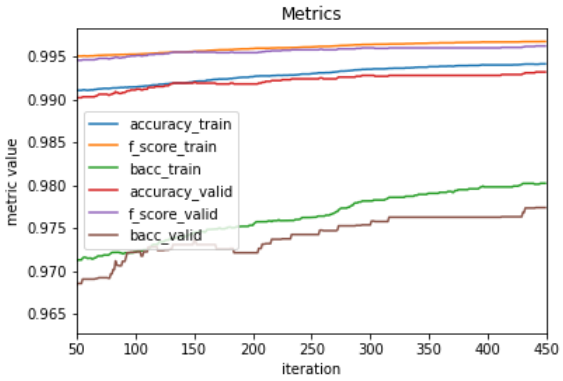


Fig. 15. Variation of different metrics across iterations in case of a hidden layer with binary output

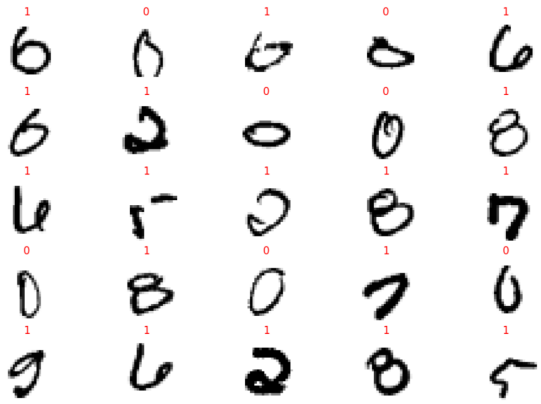

Fig. 16. Wrongly classified images for Single Hidden Layer Network with binary output

usually proves to be a better activation function for many networks. Therefore, a variant of this network with this activation function was experimented with. On the test set it achieves an accuracy of **0.9608** which is 3% higher than the sigmoid function. The accuracy is slightly lower than in binary case but it is expected as there are 10 classes instead of 2 making the problem more complex. Nonetheless, good performance is achieved for multiclass classification as well.

Inspecting the confusion matrix seen in Figure 19. it can be seen that there are only a few misclassifications as the accuracy also showed. the number of most of these misclassifications are in the single digits with only a few exceptions. These exceptions include for instance the pair of 2 and 7, or 4 and 9, which is most probably because these numbers slightly resemble each other so it is easier to confuse them. A few of these misclassified instances can be seen in Figure 20. It shows that these cases are usually hard to recognize, they are either more deformed or resemble another number.

## VII. CONCLUSIONS

In conclusion, all methods perform well in both binary and multiclass cases. For binary classification, the network

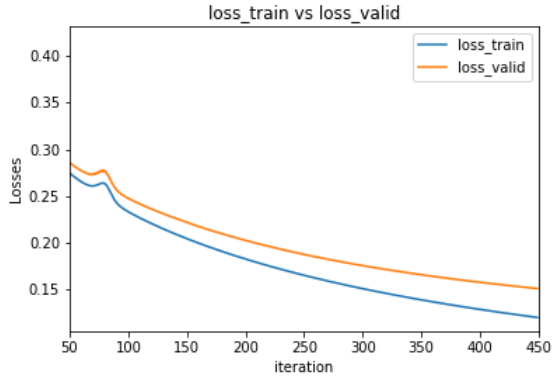


Fig. 17. Variation of costs across iterations in case of a hidden layer with a multiclass output

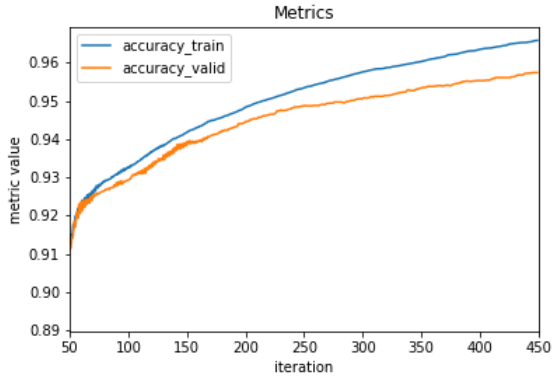


Fig. 18. Variation of accuracy across iterations in case of a hidden layer with a multiclass output

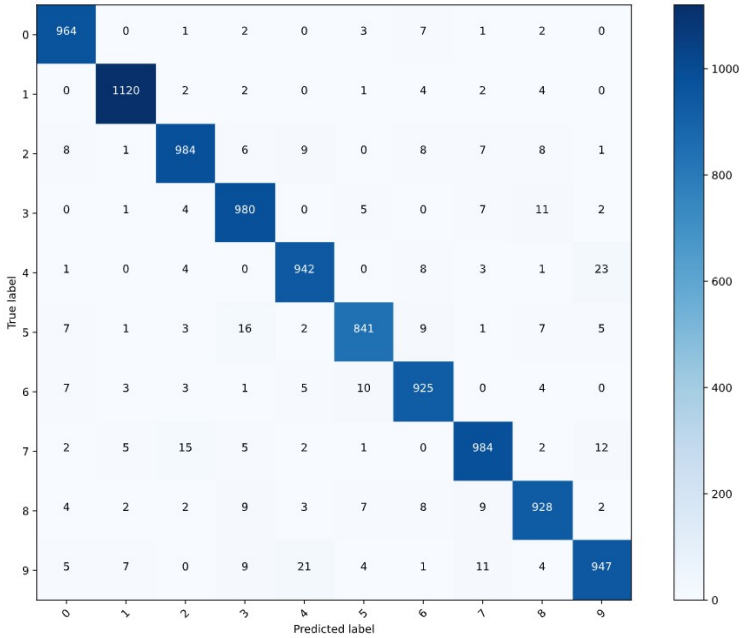


Fig. 19. Confusion matrix for single hidden layer network with multiclass output

with single hidden layer performs slightly better than single neuron. For multiclass classification, the network with tanh activation function achieves 3% more accuracy than the sigmoid activation function. The comparisons of validation loss and training loss also indicate no case of overfitting in either of the networks.

As future work, deeper and larger networks can be used to further increase the accuracy for multiclass classification problem. Moreover, other datasets offering more challenging scenarios can be trained and evaluated with simple and complex networks.

## REFERENCES

[1] LeCun, Y. & Cortes, C. (2010). MNIST handwritten digit database
[2] Source: Sánchez-Monedero, Javier. (2013). Challenges in ordinal classification: artificial neural networks and projection-based method

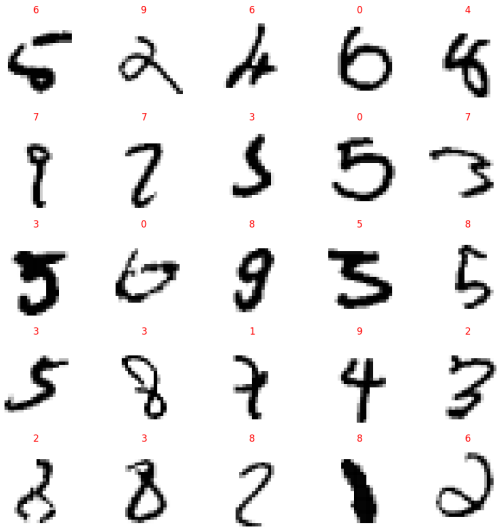

Fig. 20. Misclassified instances when using multiclass classification