AI201 Programming Assignment 3
Implementing the Backpropagation Algorithm

Anthon Van M. Asares

Date of Submission: 2024-10-16

1. INTRODUCTION

Neural networks are a class of machine learning models inspired by the structure and function of the human brain. At their core, neural networks consist of layers of interconnected neurons, where each neuron processes and transforms input data. Training a neural network involves adjusting the parameters (weights and biases) of the network to minimize the difference between the predicted and actual outputs. The backpropagation algorithm is central to this training process, as it enables the network to learn from its errors by iteratively adjusting its weights in response to the calculated output error. This process is driven by the need to minimize a cost function, which quantifies how well the network's predictions match the true values.

The backpropagation algorithm works by calculating the gradients of the cost function with respect to each parameter by propagating the error backward through the network. Starting at the output layer, the error is propagated backward layer by layer, allowing each weight to be adjusted based on its contribution to the error. This iterative process, repeated over multiple training examples, enables the network to refine its parameters and improve its performance. Building this algorithm from scratch requires a deep understanding of the underlying mathematical principles, including the chain rule for derivatives and matrix operations. This paper aims to provide a detailed examination of the backpropagation algorithm, its mathematical foundations, and the step-by-step process of implementing it for training a neural network.

2. OBJECTIVES

The goal of this project is to construct and train an artificial neural network using the backpropagation algorithm, specifically a 2-layer Multilayer Perceptron (MLP) built from scratch. The network will be implemented using only numpy for vector computations, following the generalized delta rule. Training will be performed in mini-batches, with the code optimized for vectorized operations. During training, the performance of the model will be evaluated on a validation set every 5 epochs by measuring misclassifications and the sum of squared errors, and learning curves will be plotted. A confusion matrix will also be generated to visualize and calculate various performance metrics, including accuracy, precision, recall, F1 score, and the Matthews Correlation Coefficient. After training, the model will undergo a series of sanity checks to evaluate its performance on the validation set.

Given the highly imbalanced nature of the data, efforts will be made to address this issue by creating a balanced dataset for training, which will then be partitioned into training and validation sets. Following initial training, the model will be fine-tuned to further reduce errors. The performance of two network architectures will be compared: a tanh-tanh-logistic network (Network A) and a LeakyReLu-LeakyReLu-logistic network (Network B). Finally, the trained model will be used to predict labels for the data in the provided $test_set.csv$, with the trained weights loaded into the model for testing and prediction.

3. METHODOLOGY

3.1 Preprocessing

As stated earlier, the training data provided is highly imbalanced and is detrimental to training a neural network as can be observed in Figure 1.

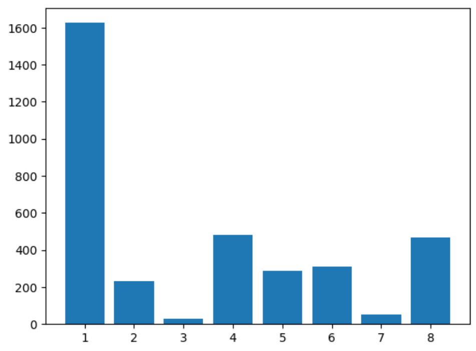


Figure 1: Labels in data.csv is skewed towards the dominant class

The solution for this is to use the Synthetic Minority Oversampling Technique (SMOTE) on the provided csv file. The package called imblearn was installed and used to synthetically create more data points to balance the dominant class. The distribution of classes is now balanced after applying SMOTE, as seen in Figure 2, which was then randomly partitioned into training_set.csv, training_labels.csv, validation_set.csv, and validation_labels.csv.

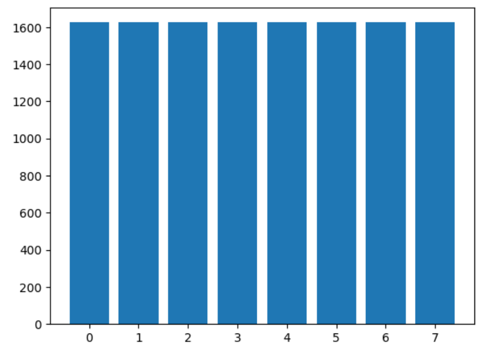


Figure 2: All labels are now balanced with 1625 datapoints each

3.2 Training

The four csvs were loaded as numpy arrays, with the labels underoing one hot encoding. Following the instructions sent, the backpropagation algorithm only implemented numpy as its sole computational module. The Layer class holds the weight and the bias terms, as well as what activation function is assigned to that layer, which can easily be slotted out for another. Three layers were instantiated to replicate two hidden layers and the output layer. The weights of the layers would be initialized using the Xavier initialization and the He initialization for logistic/tanh and leaky ReLU layers respectively. This would ensure that the initial weights of the model are optimal and would not encounter vanishing or exploding gradients [Kumar, 2017].

An epoch n of 100 was initially assigned to test the viability of the model. As stated above, training uses the concept of mini-batching, which segments the training set into different batches that contains a set amount of inputs–the batch size was set to 64 so that memory will not be a concern while still offering fast training time. As for selecting the number of neurons for the hidden layers, Heaton [2008] stated that there are many rules of thumb to determine the correct number of neurons used for the input layer, and these are:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

For the preliminary test, the number of neurons in both the first and second hidden layers is set to 8. The learning rate is configured at 0.1 to facilitate a quick evaluation of how low the validation error can reach. A summary of the parameters used in the initial run is provided in Table 1.

Batches will be generated from the entire training set per epoch, where each batch would then be used as an input to generate a prediction $y_j(n)$. This is known as the forward pass. Along with the desired response $d_j(n)$, the error signal $e_j(n)$ was then computed by $e_j(n) = d_j(n) - y_j(n)$.

Parameter	**Value**
Neurons for Hidden Layer 1	8
Neurons for Hidden Layer 2	8
Learning Rate (η)	0.1
Momentum (α)	0.9
Batch Size	64
Epochs	100

Table 1: The initial parameters for our neural network

Consequently, the cost function used for this paper, which is the sum of square errors (SSE), can now be computed by $\frac{1}{2}\sum_{j\epsilon C} e_j^2(n)$, or $\frac{1}{2}[e_j(n) \cdot e_j(n)]$ via vectorization.

The goal of minimizing the cost function is achieved by adjusting the weights, which is the expected result when performing the backpropagation algorithm using the generalized delta rule, and is depicted as follows

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_j(n) \tag{1}$$

where $\Delta w_{ji}(n)$ is the weight adjustment and is denoted by

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} \tag{2}$$

while $\delta_j(n)$ is the local gradient at node j and is defined as

$$\delta_j(n) = -\frac{\partial \varepsilon}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \tag{3}$$

The output of a neuron is defined as $y_j(n) = \varphi(v_j(n))$ where φ is the activation function. The assignment used three types of activation functions: the sigmoid function (logistic function), the hyperbolic tangent function (tanh), and Leaky ReLU, listed in the order they were described.

$$\varphi(v_j(n)) = \frac{1}{1 + exp(-av_j(n))}, a > 0 \qquad \text{(a = 2.0)}$$

$$\varphi(v_j(n)) = a \tanh(bv_j(n)) \qquad \text{(a = 1.716, b = 2/3)}$$

$$\varphi(v_j(n)) = \begin{cases} av_j(n) & \text{if } v_j(n) < 0 \\ v_j(n) & \text{if } v_j(n) > 0 \end{cases} \qquad \text{(a = 0.01)}$$

With these equations established, we can now proceed with the rest of the training process. An outline of the steps is provided in the pseudocode below:

Algorithm 1 Training Algorithm

```
for each $n$ in epochs do
  if n % 5 == 0 then
    Measure the number of misclassifications and compute the Sum of Square Errors in the validation set
  end if
  for each batch in batches do
    Forward pass to predict output
    Compute error
    Compute local gradients $\delta_j(n)$
    Compute the weight gradients $\Delta w_{ji}(n)$
    Update the biases using the local gradient
    Update the weights of the layers
    Compute batch Sum of Square Errors
  end for
  Compute epoch Sum of Square Errors
end for
```

As can be seen, minibatching entailed that we average the delta weights of the layers before adding them to weights, which happens per batch, and throughout all epochs. Tuning the hyperparameters were done done right after the initial validation.

3.3 Hyperparameter Tuning

Multiple parameters were adjusted to minimize the validation error. The first step in the tuning process was to vary the number of hidden neurons per layer. According to Heaton [2008], as long as the number of neurons lies between the size of the input and output layers, it is considered sufficient. However, Deng [2023] argued that maintaining a consistently large number of neurons per layer yields diminishing accuracy gains while significantly increasing time and computational complexity.

To address this trade-off, the second hidden layer in this experiment was assigned half the number of neurons as the first hidden layer. Consequently, the rule adopted for this study is to test configurations where the number of neurons ranges between 8 and 354, ensuring that:

- The first hidden layer has twice the neurons of the second hidden layer.
- The first hidden layer does not exceed the size of the input layer.
- All tested values are powers of 2.

Trial	Neurons in HL 1	Neurons in HL 2
1	16	8
2	32	16
3	64	32
4	128	64
5	256	128

Table 2: The number of neurons to be tested in each of the hidden layers

After determining the optimal neuron count per layer, the learning rate η was tuned. The learning rates to be tested are listed in Table 3. These values are considered sufficient, as excessively high learning rates may lead to divergence. Ensuring the stability of the model is crucial, and the cost function must demonstrate consistent convergence.

Trial	Learning Rates
1	0.01
2	0.025
3	0.005
4	0.075
5	0.1

Table 3: The learning rates to be tested

Finally, we retrain the neural network but up to 1000 epochs to try and minimize the cost function. The resultant weights were then saved to training_weights.csv.

3.4 Comparing With Another Network

A new LeakyReLU-LeakyReLU-logistic network, referred to as Network B, was trained for comparison with Network A. Both networks were used to predict a test set, with the results saved in separate CSV files. Learning curves were plotted separately for each network. Additionally, the confusion matrices for both networks were examined, and performance metric graphs were generated for further analysis.

4. EXPERIMENTAL RESULTS

Table 1 presents the results obtained after training the neural network using default parameter values. It is important to note that these initial results may not necessarily indicate learning, as they could still reflect random outcomes rather than meaningful patterns given the high amount of misclassifications obtained. The subsections below display the results when some of the hyperparameters were optimized.

Parameters	Values
Misclassifications	19
Validation Error	1.93×10^{-4}
Training Time	4.0 s

Table 4: Initial results of the neural network

4.1 Hyperparameter Tuning Results

The first hyperparameter tested was the number of neurons in the hidden layers. The summary is outlined below in Table 5.

Based on the Table 5, the first hidden layer would have 64 neurons while the second hidden layer would have 32 neurons since it had the lowest validation error among all the other configurations. Another aspect to consider would be the time it took to finish training the neural network. There is a direct relationship between the number of neurons and the training time. Using this configuration, different learning rates were then tested for their validation errors and is displayed in Table 6.

From Table 6, the learning rate that yielded the lowest error is $\eta = 0.025$. Notably, there appears to be no clear trend in

Neurons	Validation Error	Training Time
16 x 8	1.62×10^{-4}	3.30 s
32 x 16	5.98×10^{-5}	4.93 s
64 x 32	4.38×10^{-5}	15.94 s
128 x 64	6.23×10^{-5}	26.73 s
256 x 128	6.55×10^{-4}	52.24 s

Table 5: Results from testing different number of neurons per hidden layers. Given a x b, a would be the first hidden layer while b would be the second hidden layer.

Learning Rate	Validation Error	Training Time
0.01	8.45×10^{-5}	10.46 s
0.025	5.70×10^{-5}	10.36 s
0.05	1.27×10^{-4}	10.22 s
0.075	1.44×10^{-4}	10.27 s
0.1	1.40×10^{-4}	10.40 s

Table 6: Results from varying the learning rates

either the validation error or the training time as η is varied. Finally, the impact of increasing the number of epochs to 1000 is presented in Table 7.

Parameters	Values
Misclassifications	4
Validation Error	3.27×10^{-5}
Training Time	99.9 s

Table 7: Final result for Network A after updating the neuron counts for HL1 and HL2 to 64 and 32 respectively, as well as changing the learning rate to 0.025, and increasing the iterations to 1000

4.2 Comparison With Network B

The result of Network B after training it with the same hyperparameters of Network A is displayed in Table 8.

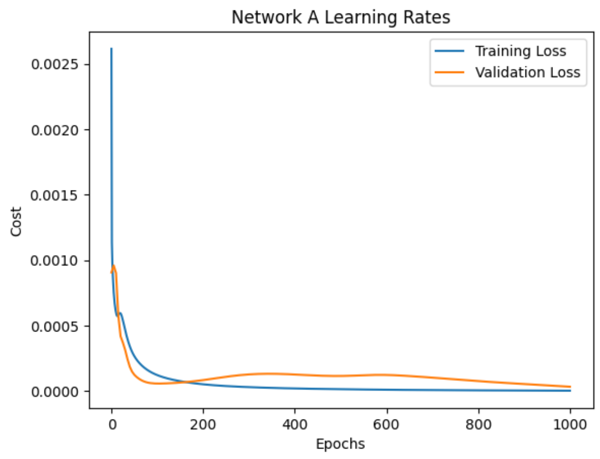
The plots for the training error and the validation error for both networks are displayed in Figure 3. At first glance, it is observed that both networks seem to converge as the iterations continue. However, it is worth noting that Network B converged faster than Network A as the latter was "spiking" during the earlier iterations. Moreover, Network B finished training somewhere around 2 minutes faster than Network A. Overall, the results suggest that Network B is better than Network A. The confusion matrices of both networks, as well as the bar graphs comparing their performance metrics, is displayed in figures 4, 5, 6.

5. ANALYSIS AND DISCUSSION OF RESULTS

The primary goal of this study was to develop an artificial neural network utilizing the backpropagation algorithm. Based on the results presented in Table 1, it can be concluded that the objective of this paper was achieved since the initial testing was satisfactory. The model displayed further promise when the hyperparameters were changed.

Parameters	Values
Misclassifications	3
Validation Error	1.58×10^{-6}
Training Time	82.4 s

Table 8: Final result for Network B after updating the neuron counts for HL1 and HL2 to 64 and 32 respectively, as well as changing the learning rate to 0.025, and increasing the iterations to 1000



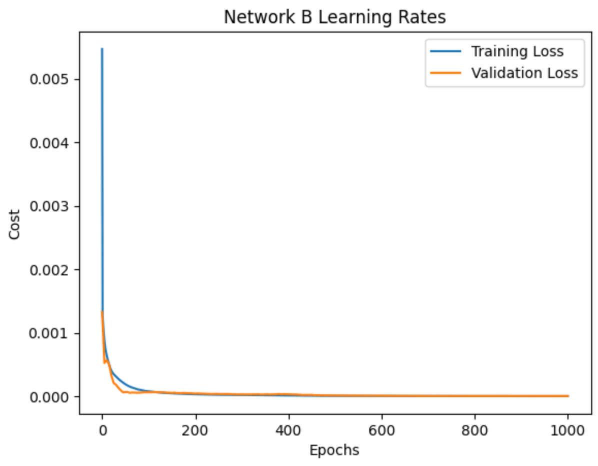


Figure 3: Training and Validation Errors for Networks A and B

Table 5 indicates that there is no direct correlation between the number of neurons in the layers and the validation error. It is important to note that while increasing the number of neurons can enhance the model's ability to learn complex patterns, it does not always result in a lower validation error. One possible reason for this is overfitting—adding more neurons increases the number of synaptic weights, which can hinder the model's ability to generalize well to new, unseen data. Additionally, a larger number of neurons introduces more local minima, potentially slowing convergence. Finally, there are increased time and computational costs associated with a more complex network.

Similarly, Table 6 does not imply any relation between the

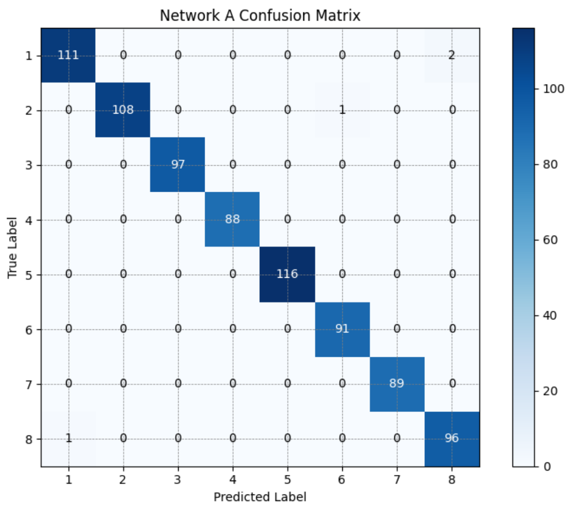


Figure 4: Confusion matrix for Network A.

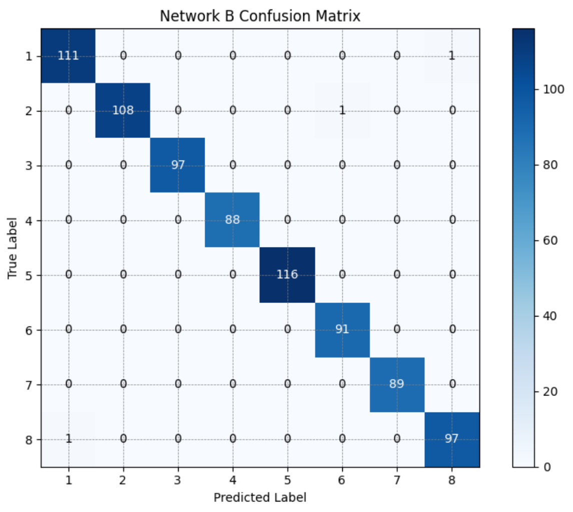


Figure 5: Confusion matrix for Network B.

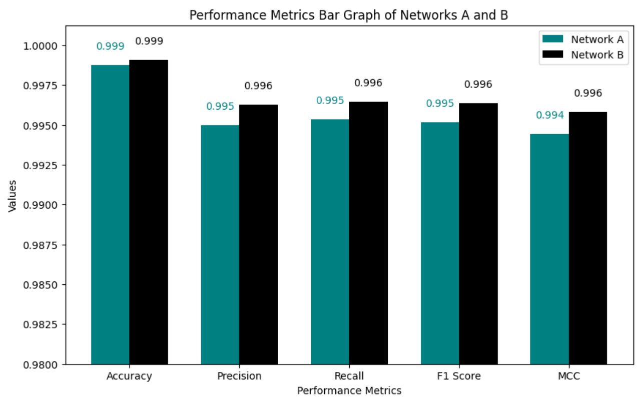


Figure 6: Performance metrics for Network A and Network B.

learning rate and the validation error. This was touched briefly in the earlier sections but a lower learning rate would converge slower compared to a larger one since the steps the model is taking towards the global minimum is small. In contrast, a higher learning rate might be overeager to arrive at the global minimum that it overshoots its target, misses the mark, and lands on another path leading to a local minimum, which it then tries to escape from to travel back to the global minimum. Furthermore, there is a change that it might get stuck in the local minimum, which in turn would cease any updates to the weights. Notably, there is no definitive literature on what constitutes an optimal learning rate. Much of the discussion surrounding the learning rate focuses on determining the best fit for a given model, typically through experimentation with multiple learning rate values. Just as learning rate has no relation with validation error, training time also has no relation with learning rate as there were no increase in training complexity.

In comparing Network A to Network B, we look at Tables 7 and 8. Network B has lower misclassification, validation error, and even training time. This is corroborated by Chou et al. [2017], where they claim that the ReLU family of activation functions converge six times faster compared to a tanh function. Furthermore, it is far less computationally expensive than tanh which would explain its faster training time. To observe the performance of both these networks over all epochs, we look at Figure 3. Network B produced a very smooth curve for all plots relative to Network A, which had a very erratic behavior during the initial iterations, thus suggesting that a LeakyReLU activation function is superior to tanh. To emphasize this claim, we turn to figures 4, 5, and 6. The interpretation for each performance metrics is as follows:

- Accuracy: Network B has more correct classifications than Network A over all samples in the set
- Precision: Suppose we have an arbitrary class X. Network B is more likely to correctly identify that the correct label of the sample is X across all instances where the network predicts class X.
- Recall: Suppose we have an arbitrary class X. Network B is more likely to correctly identify that the correct label of the sample is X across all instances where the correct label of the samples is actually X.
- F1-score: Since both networks have similar precision and recalls, the F1-score is balanced as well. And since, Network B has greater precision and recall compared to Network A, then it naturally has a greater F1-score as well.
- MCC: Both networks display a very highly correlated observed and predicted labels. This is slightly in favor of Network B though since it misclassified one less sample by the end of 1000 epochs.

6. CONCLUSION

The objective of constructing a neural network from scratch using the backpropagation algorithm, capable of effectively classifying samples, has been successfully achieved. This experiment highlighted the critical importance of hyperparameter tuning in training a neural network.

One key observation was that increasing the number of neurons does not guarantee a lower validation error compared to configurations with fewer neurons. Furthermore, adding more neurons comes with trade-offs, as both time and computational complexity increase significantly. Adjusting the learning rate also proved crucial in reducing validation error. Unlike the number of neurons, increasing the learning rate to higher values does not always lead to better results. The general consensus is that finding the optimal learning rate is model-dependent and best achieved through trial and error.

Lastly, the use of the Leaky ReLU activation function proved to be significantly more effective than tanh in terms of faster convergence and reduced training time. This conclusion is supported by a comparison of the resulting confusion matrices and performance metrics from both networks.

6.1 Recommendations

The process of selecting the optimal number of neurons could have been approached more effectively. One potential method is pruning, where the network begins with a sufficiently large number of neurons in its layers. The model is trained, and neurons are then removed based on a predefined pruning criterion. The model is subsequently retrained, and this process is repeated until a satisfactory configuration is achieved [Thoma, 2017]. Another recommendation was to retrain the neural network for 2000 epochs or until the validation error converges since the validation error was still decreasing even when on the 1000th iteration. A better approach would be to implement early stopping whenever overfitting has begun. Another approach would be to introduce the idea of a "scheduler", which is a method that decreases η to avoid oscillations as the network gets trained in each iteration. [Patterson and Gibson, 2017].

References

Chun-Nan Chou, Chuen-Kai Shie, Fu-Chieh Chang, Jocelyn Chang, and Edward Chang. Representation learning on large and small data. 07 2017. doi: 10.48550/arXiv.1707.09873.

Tiancheng Deng. Effect of the number of hidden layer neurons on the accuracy of the back propagation neural network. *Highlights in Science, Engineering and Technology*, 74:462–468, 12 2023. doi: 10.54097/nbra6h45.

Jeff Heaton. *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, Inc., 2nd edition, 2008. ISBN 1604390085.

Siddharth Krishna Kumar. On weight initialization in deep neural networks. *CoRR*, abs/1704.08863, 2017. URL `http://arxiv.org/abs/1704.08863`.

Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc., 1st edition, 2017. ISBN 1491914254.

Martin Thoma. Analysis and optimization of convolutional neural network architectures. Masters's thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, June 2017. URL `https://martin-thoma.com/msthesis/`.