Recognizing Handwritten Digits using Neural Network Algorithms

Michelle Kelman and Jihan Wang
The University of Texas at Dallas
800 W Campbell Rd
Richardson, TX 75080

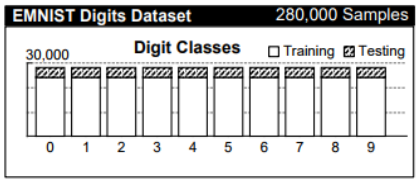
1 Introduction

Although in recent years digital document completion has become more common, there are still many types of documents that must be filled out by hand. This becomes especially troublesome when there are large quantities of documents to be processed such as addressed mail, bank checks, and handwritten forms. However, machine-led handwritten letter and digit recognition can be used to determine what is was written and digitize handwritten text.

Problem Character classification is a difficult task since humans have great variations in handwriting including size, orientation, and style. Therefore, computer vision and deep learning models (neural networks) must be used to learn patterns in handwritten letters and digits. Furthermore, different neural network activation functions classify patterns in data differently. Our goal is to determine which common activation function best learns these patterns and correctly classifies handwritten character data.

Need for Dimensionality Reduction Visualizing patterns in the data in each layer of our neural network algorithms aids our activation function analysis. However, the dimensions of our layers are large. In order to visualize the data, we use dimensionality reduction to represent the data in 2D while maintaining the original data information. Dimensionality reduction is also a useful tool for noise reduction and overfitting prevention.

2 Data



The Extended MNIST (EMNIST) dataset is a set of handwritten character digits derived from the National Institute of Standards and Technology (NIST) Special Database 19 and converted to 28x28 pixel image format. The EMNIST Digits dataset provides balanced handwritten digit data with a total of 10 classes (one per digit) and 280,000 samples: 240,000 samples in the train set and 40,000 samples in the test set (Cohen et al. 2017).

3 Algorithms

Our algorithm approach includes 4 different neural network models with 3 different activation functions, for a total of 12 considered algorithms. We also implemented 2 different dimensionality reduction methods.

Neural Network Models

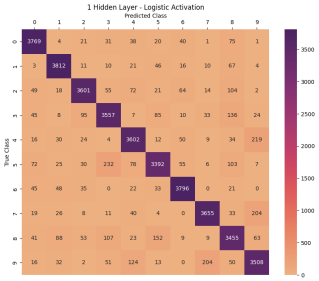
- **Our implementation of a 1 hidden layer neural network** with 28x28 input nodes, 300 hidden nodes, 10 output nodes, 1000 epochs, and a learning rate of 0.1. The number of hidden nodes was chosen based on the principle that the ideal number of nodes is approximately the average number of input and output nodes (Nielsen 2019) (Yousaf 2019) (Paudel 2020).
- **Our implementation of a 1 hidden layer neural network with Principle Component Analysis (PCA)**.
- **Scikit-Learn Neural Network MLPClassifier()** with the same parameters as above and the weight optimizer set to stochastic gradient descent. Although the same parameters were used, the final model is expected to perform better than our implementation due to additional optimization features.
- **Keras Convolutional Neural Network** with 3 2D convolution layers, 3 pooling layers, and 3 dense layers. This model was included in our analysis is to determine if activation functions affect different types of neural networks similarly (DataFlair 2020).

Activation Functions

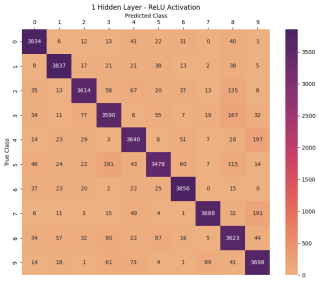
- Logistic (sigmoid) activation function
- ReLU (rectified linear unit) activation function
- Hyperbolic tangent (tanh) activation function

Dimensionality Reduction Methods

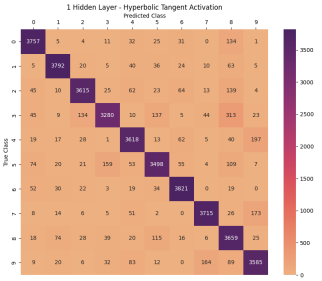
- **Our Principal Component Analysis (PCA) algorithm** with 784 total features linearly projected to 2 dimensions for visualization purposes and 549 dimensions (70% of the original dimensions) for testing the effect of dimension reduction (Maćkiewicz and Ratajczak 1993).
- **Our t-Distributed Stochastic Neighbor Embedding (t-SNE)** with 2 dimensions for visualization purposes and using the first 10,000 samples of the data set for testing due to RAM limitations (Maaten and Hinton 2008).



(a) Logistic Activation Function



(b) ReLU Activation Function

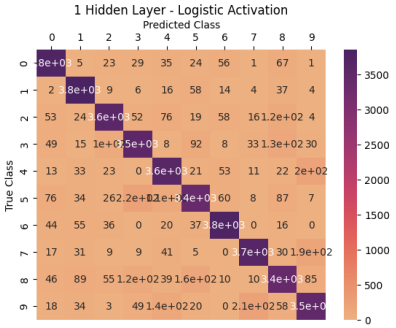


(c) Hyperbolic Tangent Function

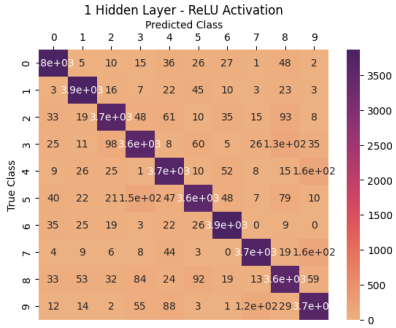
Figure 1: Confusion Matrices for 1 hidden layer neural network

Activation Function	1 Hidden Layer Neural Network						
	TPR	FPR	Precision	Recall	F1	AUC for ROC	Accuracy
Logistic	0.90368	0.01070	0.90379	0.90368	0.90360	0.89371	0.90368
ReLU	0.92145	0.00873	0.92227	0.92145	0.92154	0.90138	0.92145
Hyperbolic Tangent	0.90850	0.01017	0.91030	0.90850	0.90865	0.89140	0.90850

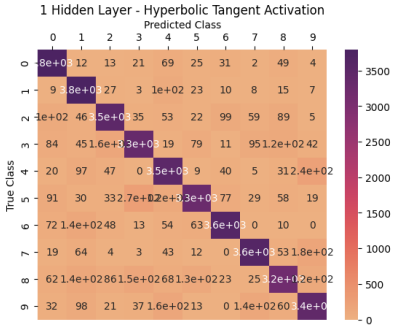
Table 1: Average Evaluation Metrics for 1 hidden layer neural network



(a) Logistic Activation Function



(b) ReLU Activation Function

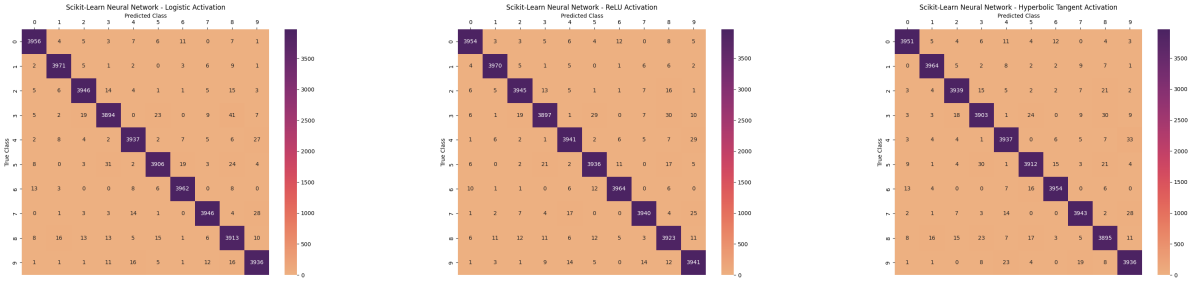


(c) Hyperbolic Tangent Function

Figure 2: Confusion Matrices for 1 hidden layer neural network with PCA processed input

Activation Function	1 Hidden Layer Neural Network(PCA processed)						
	TPR	FPR	Precision	Recall	F1	AUC for ROC	Accuracy
Logistic	0.90063	0.01104	0.90047	0.90163	0.90040	0.89049	0.90063
ReLU	0.92810	0.00799	0.92822	0.92810	0.92808	0.91317	0.92810
Hyperbolic Tangent	0.87685	0.01368	0.87754	0.87685	0.87643	0.87620	0.87685

Table 2: Average Evaluation Metrics for 1 hidden layer neural network with PCA processed input

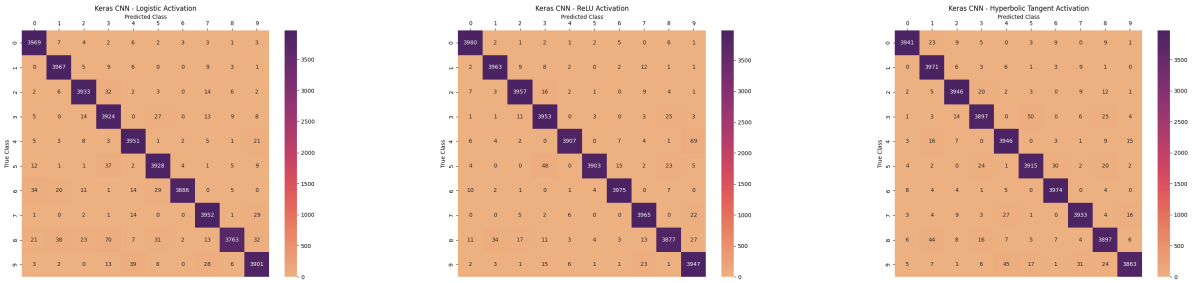


(a) Logistic Activation Function (b) ReLU Activation Function (c) Hyperbolic Tangent Function

Figure 3: Confusion Matrices for Scikit-Learn neural network

Activation Function	Scikit-Learn Neural Network						
	TPR	FPR	Precision	Recall	F1	AUC for ROC	Accuracy
Logistic	0.98417	0.00176	0.98419	0.98417	0.98418	0.99979	0.98418
ReLU	0.98528	0.00164	0.98529	0.98528	0.98528	0.99978	0.98528
Hyperbolic Tangent	0.98335	0.00185	0.98336	0.98335	0.98335	0.99978	0.98335

Table 3: Average Evaluation Metrics for Scikit-Learn neural network



(a) Logistic Activation Function (b) ReLU Activation Function (c) Hyperbolic Tangent Function

Figure 4: Confusion Matrices for Keras CNN

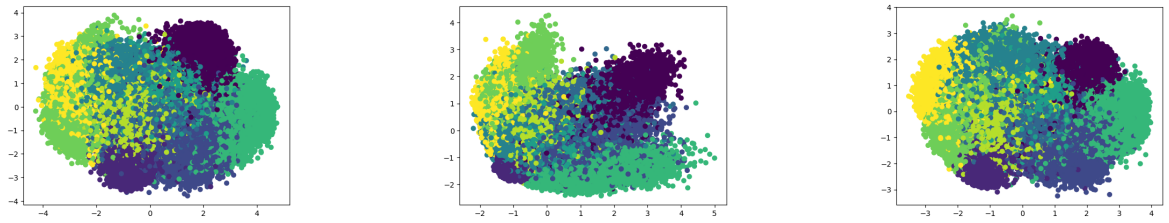
Activation Function	Keras Convolutional Neural Network						
	TPR	FPR	Precision	Recall	F1	AUC for ROC	Accuracy
Logistic	0.97935	0.00229	0.97950	0.97935	0.97932	0.98853	0.97935
ReLU	0.98568	0.00159	0.98575	0.98568	0.98568	0.99204	0.98567
Hyperbolic Tangent	0.98208	0.00199	0.98211	0.98208	0.98207	0.99004	0.98207

Table 4: Average Evaluation Metrics for Keras CNN

(a) PCA Visualization

(b) t-SNE Visualization

Figure 5: PCA and t-sne Processed Visualizations of Input Layer

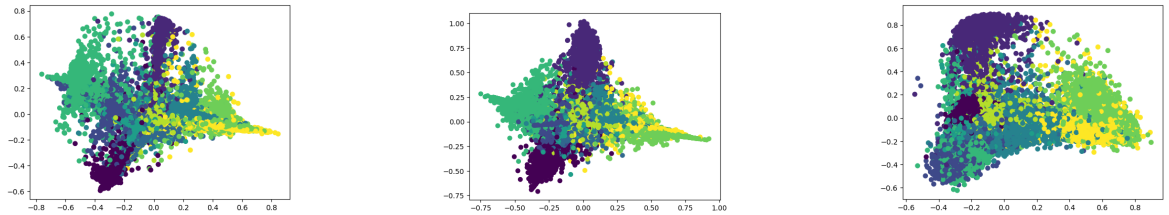


(a) Logistic Activation Function

(b) ReLU Activation Function

(c) Hyperbolic Tangent Function

Figure 6: PCA Processed Visualizations of Hidden Layer

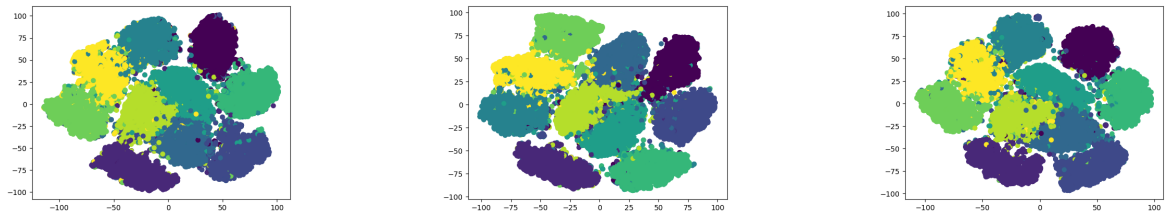


(a) Logistic Activation Function

(b) ReLU Activation Function

(c) Hyperbolic Tangent Function

Figure 7: PCA Processed Visualizations of Output Layer

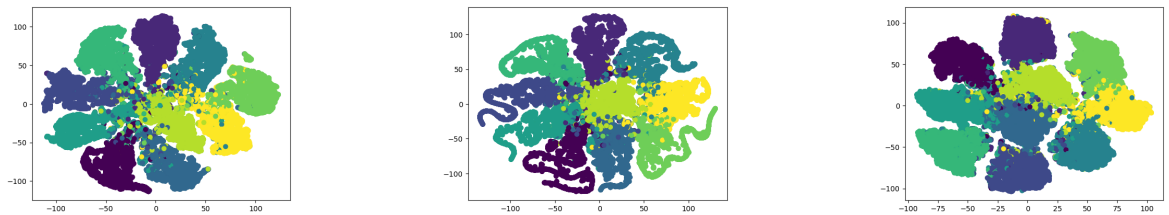


(a) Logistic Activation Function

(b) ReLU Activation Function

(c) Hyperbolic Tangent Function

Figure 8: t-sne Processed Visualizations of Hidden Layer



(a) Logistic Activation Function

(b) ReLU Activation Function

(c) Hyperbolic Tangent Function

Figure 9: t-sne Processed Visualizations of Output Layer

4 Results

To analyze the performance of each activation function, we evaluated confusion matrices (by class), accuracy, true positive rate, false positive rate, precision, recall, F1 score, and area under the ROC (AUC) for each algorithm.

For each model and activation function pair, we provided confusion matrices and evaluation metric values. For our implementations of PCA and t-SNE, we provided plots for visualizing the input, hidden, and output layers.

5 Discussion

Algorithm Analysis

Best Activation Function From analysis of the results, it is apparent that the ReLU activation function has the best overall scores for each evaluation metric (although there is less than 1% difference between scores in most models). This is expected since ReLU is the most commonly used activation function in deep learning: ReLU has been named computationally efficient because of its tendency to deactivate specific neurons.

Worst Activation Function Although it is easy to determine that ReLU is the best activation function, it is difficult to determine which activation function has the worst performance as it differs for each model. Determining the least efficient activation function requires more analysis.

Performance Comparison Overall, the ReLU activation function yields the best performance for each model (as represented by AUC for ROC). This is once again to due the efficiency of ReLU. However, it is interesting to note that for the Sci-Kit learn Neural Network, the ReLU activation function does not have the best AUC, with very small error.

Neural Network Model Analysis Since we did not perform extensive parameter tuning, it is difficult to analyze the quality of our 4 models in comparison to each other. However, the Scikit-Learn neural network and Keras CNN models had the best performance of all the models and comparable performance due to each other due to their unique optimization features. There is a slight preference for the Scikit-Learn model, showing that CNNs might not be suitable for this problem, but more research is necessary.

Dimensionality Reduction Analysis

Best Dimensionality Reduction Method For visualization, t-SNE (non-linear) has better performance than PCA (linear) since some dimensions of the image data set can not be compressed linearly.

Performance Analysis for PCA PCA dimensionality reduction on the input data reduced running time with a slight performance compromise (less than 3.5%). Applying dimensionality reduction separately from training can lead to a performance gap between the train results and test results. Our solution projects the test set to the train set dimensions by taking the dot product of their eigenvectors, from processing the train set with PCA, to avoid performance loss from dimension mismatch.

6 Conclusion

Successes in our Implementation

- With little parameter tuning, we were able to increase our final accuracy to above 90% for most of our neural network implementations. This is a good starting point for future research to create an accurate handwritten character classification model.
- Our neural network implementation yielded the same results as the Scikit-Learn neural network and Keras CNN algorithms in terms of determining the best activation function for our problem and dataset.
- PCA and t-sne are powerful visualization tools for illustrating relations in the data. By testing their performance on each layer of our neural network algorithm, we prove that these visualization approaches are practical.
- In using PCA to reduce the dimensions of the input data, we significantly improved the running time of our models without significant damage to performance, showing the benefits of dimensionality reduction.

Issues with our Implementation

- There is a fairly accuracy gap between our neural network implementation and the Scikit-Learn algorithm with equivalent parameters. This was expected due to the Scikit-Learn algorithm's use of optimization algorithms, however it is important to consider for future work.
- The running times of our neural network and both of our dimensionality reduction implementations are significantly longer than the equivalent library versions.
- For t-SNE, we only tested our implementation on a subset of the dataset (10,000 samples) due to RAM limitations. In theory, we would have needed 179.45 GB of RAM to process all 240,000 train samples due to the use of the pairwise_distances library function.

Future Improvements

- For our neural network implementation, we will perform parameter tuning with cross validation to optimize model parameters. We will also try additional features like stochastic gradient descent, momentum factors, weight decay, and conjugate gradients to improve performance.
- We will implement linear algebra techniques to reduce the space required for the intermediate t-SNE matrix so that we can test on all samples.
- For both PCA and t-SNE, we will rebuild the program with CUDA and use the CuPy library to exploit the performance of the GPU and improve program speed.

Overall, our results show that the ReLU activation function is optimal for future deep learning research related to the handwritten classification problem. In addition, t-sne should be selected as the preferred visualization method for models using non-linear data. Although there are many improvements that could be made to equip this model for day-to-day use, determining the best activation function and visualization method will make development more efficient and give a reliable starting point for future work.

References

Cohen, Gregory et al. (2017). *EMNIST: an extension of MNIST to handwritten letters*. arXiv: 1702.05373 [cs.CV].

DataFlair (2020). *Handwritten Character Recognition with Neural Network*. https://data-flair.training/blogs/handwritten-character-recognition-neural-network/. [Online; accessed 06-May-2023].

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9, pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Maćkiewicz, Andrzej and Waldemar Ratajczak (1993). "Principal components analysis (PCA)". In: *Computers & Geosciences* 19.3, pp. 303–342. ISSN: 0098-3004. DOI: https://doi.org/10.1016/0098-3004(93)90090-R. URL: https://www.sciencedirect.com/science/article/pii/009830049390090R.

Nielsen, Michael (2019). *Using neural nets to recognize handwritten digits*. http://neuralnetworksanddeeplearning.com/chap1.html. [Online; accessed 06-May-2023].

Paudel, Ramesh (2020). *Building a Neural Network with a Single Hidden Layer using Numpy*. https://towardsdatascience.com/building-a-neural-network-with-a-single-hidden-layer-using-numpy-923be1180dbf. [Online; accessed 06-May-2023].

Yousaf, Sanwal (2019). *3 layer neural network from scratch*. https://www.kaggle.com/code/sanwal092/3-layer-neural-network-from-scratch/notebook. [Online; accessed 06-May-2023].