# Homework 4

## Introduction

We are using both a neural network and a logistic regression model to predict the digits 0-9 from images of handwritten digits. This could be useful for automatically reading zip codes on envelopes, for reading numbers on checks, or for anything that requires reading handwritten numbers. This model could be used to automate the process of reading handwritten numbers, which would save time and money for companies that need to read handwritten numbers.

## Methods

For the logistic regression model, we did not have to perform any pre-processing on the data, as we used the `mnist` dataset, which is already split into a training set and a test set. In addition, there is no need to drop any missing values due to it being a provided dataset. We then fit the model to the training set, and used the test set to evaluate the model's performance.

For the neural network, we perform some pre-processing on the data via reshaping data to be individual columns instead of image matrix. Basically we are taking data that's in a jumbled picture-like format and making it neat and organized by separating it into individual columns of information. We then rescale the data to be 0-1 instead of 0-255. We do this because it's easier for the computer to work with numbers between 0 and 1, and pixels are usually between 0 and 255. We then change the labels to be in the one-hot-encoded format using LabelBinarizer(), which is a fancy way of saying that we change the labels to be in a format that the computer can understand. After the pre-processing, we build the structure of the model with an input layer of 784 nodes, and an output layer of 10 nodes, in addition to using "softmax" as the activation function. The 10 nodes in the output layer correspond to the 10 possible digits that the model can predict. We also use 6 "inner" layers between the input and output layer, between 784 and 10 nodes. Think of it as having 784 entry points and 10 exit points for the computer to make predictions about digits (like 0 to 9). In the middle, we have 6 layers that help the computer understand things better, kind of like having multiple steps to solve a puzzle. Now, we set up the model's rules. We specify that the computer should use a technique called “categorical_crossentropy” to measure how well it’s doing in solving the puzzle. We also tell it to use a tool called “SGD” with a learning rate of 0.01 to get better at solving the puzzle. Lastly, we ask the computer to keep track of how accurate it is while solving by grabbing the “accuracy” metric. We then train the model by showing it lots of pictures (trainX) and tell it which numbers these pictures represent (trainYnn, like 0 to 9). This training process happens 100 times (or epochs) to make the model better at recognizing numbers. We also use different pictures (testX) to see how good the model has become at recognizing numbers (testYnn).

## Results

Both models did pretty well, but the Logistic Regression model had a slightly higher accuracy of 92.56%, while the neural network had an accuracy of about 92.28%. The neural network seemed to fit the training data a bit too closely, which might have caused a small drop in accuracy on the test data.I’m a bit surprised that the Logistic Regression did so well, but then again Logistic Regression models can perform when things are not too complicated, such as when there are only 10 possible digits (0-9) to predict. A neural network is more complicated and can work better for more complicated tasks, such as recognizing more complex images like pets.

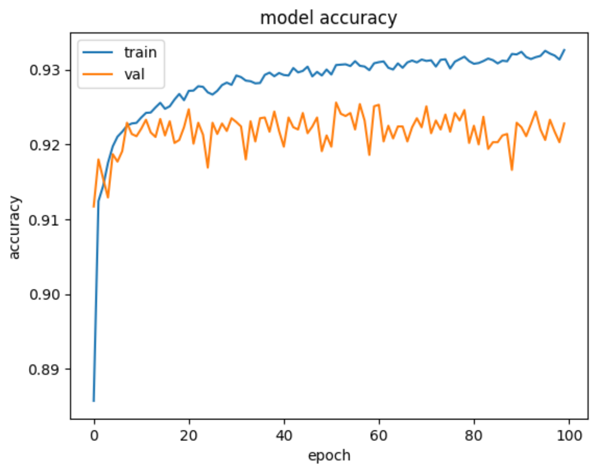

Figure 1: Neural Network Accuracy Over Time

As the number of epochs increases, the loss decreases and the accuracy increases for both the train and test set. This is because the model is getting better at recognizing numbers as it is trained more. The accuracy of the train set is higher than the accuracy of the test set, which is expected because the model is trained on the train set. The accuracy of the test set is still high, which means that the model is not overfitting. The loss of the train set is lower than the

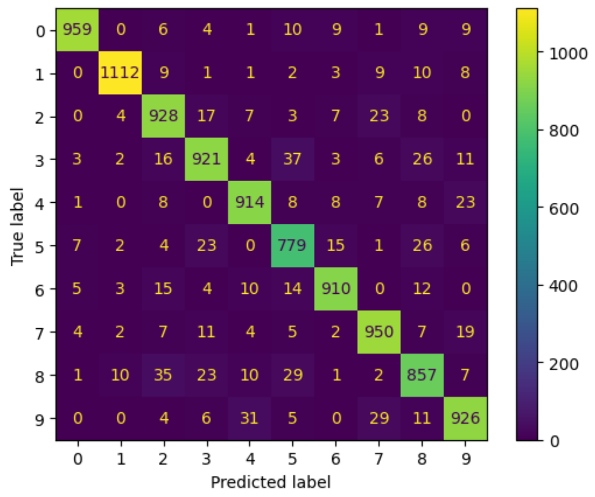

Figure 2: Confusion Matrix for Logistic Regression - X Axis is Predicted; Y Axis is the Actual Value

loss of the test set, which is expected because the model is trained on the train set. The loss of the test set is still low, which means that the model is not overfitting.

The job of a loss function is to measure how well a machine learning model is performing by comparing its predictions to the actual target values, kind of like a scoreboard. In this case, the model uses the "categorical_crossentropy" loss function. This loss function is appropriate for predicting digits (0-9) because it's designed for classification tasks where the output has multiple categories, which is the case when we're predicting one of ten possible digits. It helps the model penalize errors more when it makes predictions that are far from the correct digit, which is crucial for accurate digit recognition. In other words, this method is great for guessing from many options, just like picking a number from 0 to 9. If the model makes a big mistake, this method makes it pay more attention and try harder to get the number right. This helps the model become better at recognizing numbers accurately.

An activation function is like a decision-maker for each node in our neural network. In this case, we used an activation function called "softmax." Softmax is a good choice for predicting digits (0-9) because it helps the computer make a clear and confident choice among the ten possible numbers. It turns the computer's calculations into probabilities, making sure it picks one number with certainty, which is important for accurate digit prediction.

The last layer in our neural network has 10 nodes. We chose this number because there are 10 possible digits (0-9) that we want the model to predict. Each node corresponds to one of these digits, so having 10 nodes allows the model to make a separate prediction for each digit, making it suitable for classifying and recognizing the full range of digits.

Logistic regression likely will not perform as well on more complicated images like classifying pets as “Corgi” or “Not Corgi.” Logistic regression is a relatively simple algorithm that’s better suited for binary classification tasks, where there are only two possible outcomes, like “Yes” or “No.” In the case of classifying pets, there can be many different breeds and categories, in addition to the fact that a dog such as corgi is a much more complicated image vs the digits 0-9. More complex machine learning models, such as a neural network, are generally more effective for handling image classification tasksecause they can capture intricate patterns and features within the images, which logistic regression may struggle to do.

Deep Neural Networks, like super-smart computer systems, are really good at understanding and figuring out complicated things from a lot of information. They’re especially great with big and tricky problems, like recognizing pictures or understanding language. But, they need a lot of data and computer power. Simpler models are like basic tools that can work for simpler tasks, but these fancy networks are like having a super detective for the tough jobs. They’re not always necessary, but they can be really helpful for solving the most difficult problems.

## Discussion/Reflection

Through these analyses, I’ve learned that both neural networks and logistic regression can perform well in classification tasks. However, in this specific classication, Logistic Regression seems to perform a bit better as it is a relatively simple classification of 10 possible digits. If I were to perform this analysis again in the future, I would try to use a more complicated dataset that would require a neural network to perform well, such as classifying food on social media posts. I would also play around with different activation functions and loss functionsfor the neural network to see if it would perform better.