Implementing an Artificial Neural Network to Recognize Handwritten Digits

INTRODUCTION

Computer vision is a rapidly developing field of computer science which aims to autonomously identify objects from digital images and videos. When humans look at symbols and objects, their brains effortlessly recognize them. Similar to the way humans use cone cells to recognize objects, computer vision systems view objects as pixelated images. One specific task within computer vision is to recognize and classify handwritten digits. In this project, a machine learning technique called an artificial neural network is implemented to recognize handwritten digits.

BACKGROUND RESEARCH

Simple logical rules cannot accurately classify handwritten digits. Rather than explicitly programming rules, machine learning techniques provide systems with the ability to learn on their own and make accurate preimplemented dictions on their own. The type of machine learning technique in this project is an artificial neural network. An artificial neural network is a model inspired by the structure of biological neurons in the brain [3]. Neural networks are a highly researched and developed field, and there are a variety of neural network structures.

A feed-forward, fully-connected artificial neural network consists of a number of artificial neurons arranged into layers. The first layer (the input layer) receives input from the real world, and the last layer is the calculated predictions. Each middle (hidden) layer processes the output of the previous layer. Each neuron is connected to each other neuron in neighboring layers by a weight. By adjusting the weights and biases of the network to minimize a cost function, the neural network learns to classify images.

PROBLEM

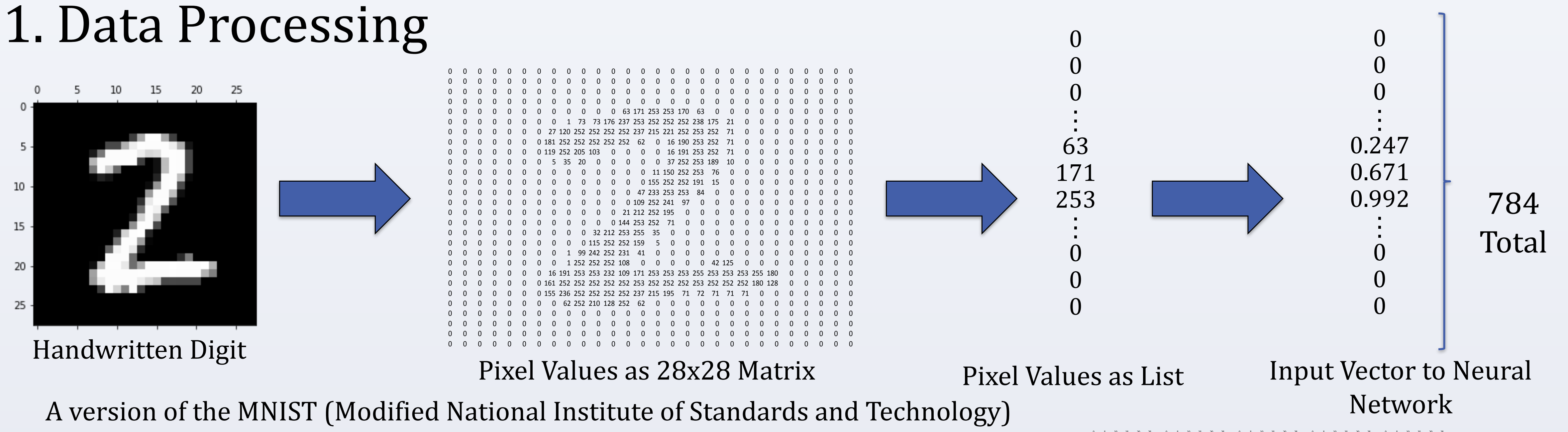
One specific type of computer image recognition involves recognizing human handwriting. It can be challenging for a computer algorithm to recognize handwritten digits because the specific pixel values of one digit may greatly differ across images of the same digit, while sometimes images of different digits may look very similar. In order to implement a machine learning algorithm for digit recognition, the algorithm must learn from numerous labeled training examples.

OBJECTIVE

The objective of this project is to implement an artificial neural network algorithm that can learn to accurately classify handwritten digits (zero through nine) from the MNIST dataset.

ALGORITHM DESIGN

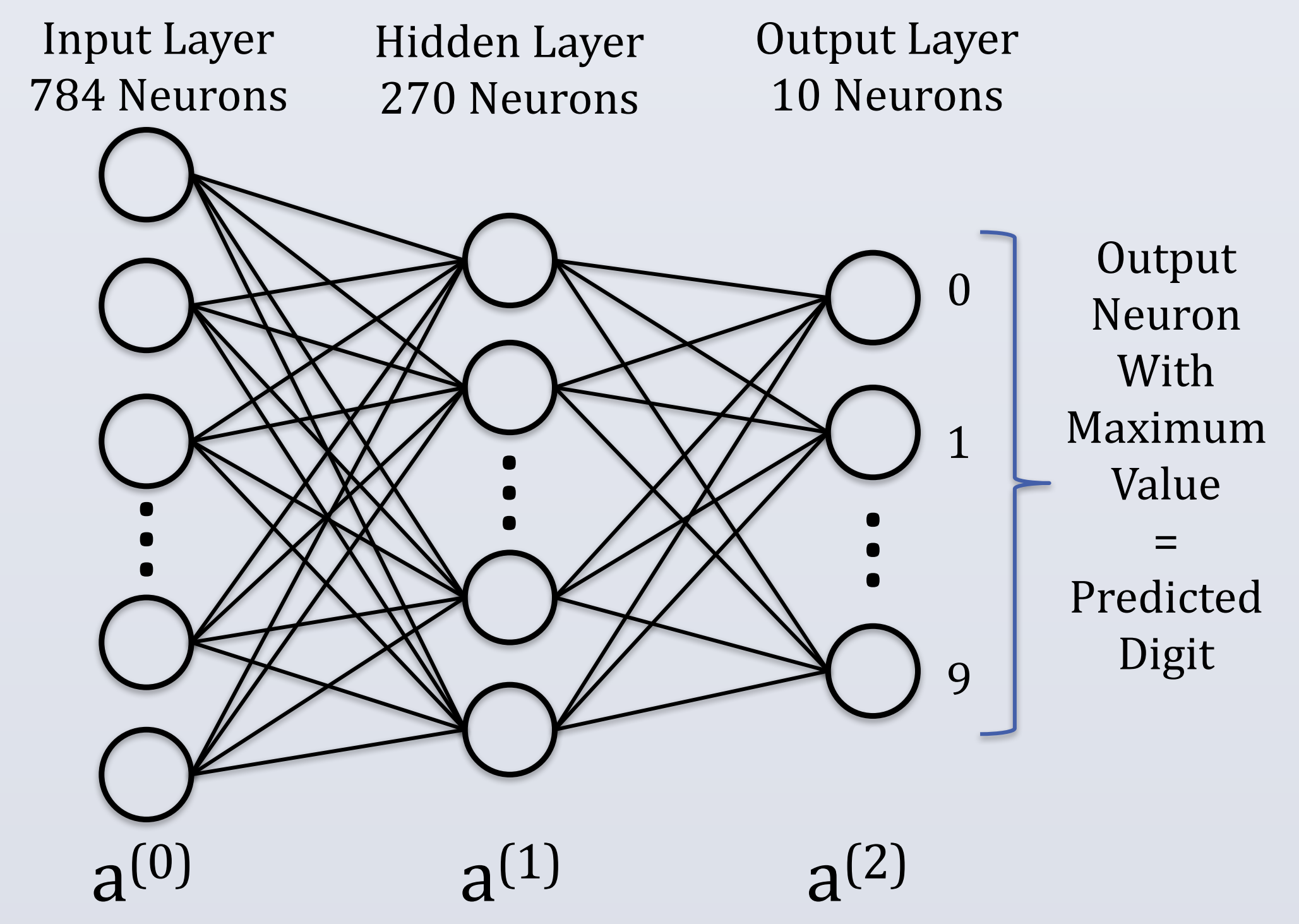
1. Data Processing



A version of the MNIST (Modified National Institute of Standards and Technology) dataset contains 42,000 labeled digits stored as a CSV file [2]. Each digit is 28 by 28 pixels, for a total of 784 pixels. Each row stores each digit's data. The first column contains a label, and the remaining 784 columns are the pixel values corresponding to the digit. Each pixel value ranges from 0 (black) to 255 (white).

Digit Examples

2. Neural Network Structure



- Each neuron in the network is a function that outputs a value between 0 and 1 depending on the outputs of all the neurons in the previous layer.
- The first layer outputs the data of the digits after processing and the last layer's output indicates a confidence that the given image corresponds to each digit 0 through 9.
- There are a total of 280 biases and 214,380 weights in this network structure that can be adjusted.
- Each weight is initialized to a random number between -0.01 and 0.01.
- Each bias is initialized to 0.

3. Training

Forward Propagation:

Each of the digit input vectors is sent through the neural network. First, each neuron receives a weighted sum of the output from the previous layer. Then, each neuron activates depending on its input; the sigmoid function outputs a value between 0 and 1. These operations are performed with matrices.

$a^{(0)}$ = Input Vector to Neural Network

$z^{(1)} = a^{(0)} \bullet w^{(1)} + b^{(1)}$

$a^{(1)} = \mathrm{sigmoid}(z^{(1)})$

$z^{(2)} = a^{(1)} \bullet w^{(2)} + b^{(2)}$

$a^{(2)} = \mathrm{sigmoid}(z^{(2)})$

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \cdots \\ a_{783}^{(1)} \end{bmatrix} = \mathrm{sigmoid}\left(\begin{bmatrix} w_{(0,0)} & w_{(0,1)} & \cdots & w_{(0,783)} \\ w_{(1,0)} & w_{(0,1)} & \cdots & w_{(1,783)} \\ \cdots & \cdots & \cdots & \cdots \\ w_{(269,0)} & w_{(269,1)} & \cdots & w_{(29,783)} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \cdots \\ a_{783}^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \cdots \\ b_{783} \end{bmatrix} \right)$$

Back Propagation:

The goal of back propagation is to minimize the error function. The error function used is cross-entropy cost [4]:

$$\text{Cost} = -\frac{1}{M}\sum_{i=1}^{M}\left(y_i^T \cdot \log(a_i^{(2)}) + (1-y_i)^T \cdot \log(1-a_i^{(2)})\right)$$

1. Calculate derivatives of the loss with respect to each adjustable parameter in the model: w_2, b_2, w_1, and b_1 using chain rule:

Ex: $$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

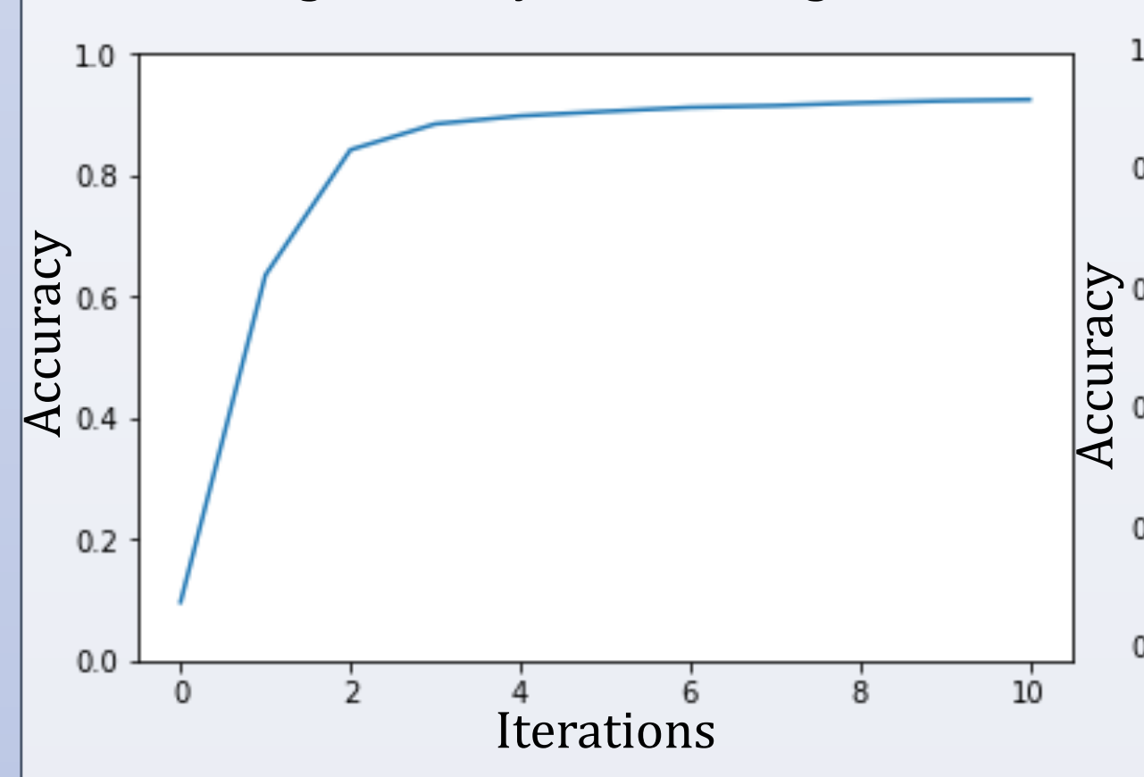
2. Update w_2, b_2, w_1, and b_1 in the direction that will minimize the error:

Ex: $$w_2 = w_2 - \alpha \cdot \frac{\partial L}{\partial w_2}$$

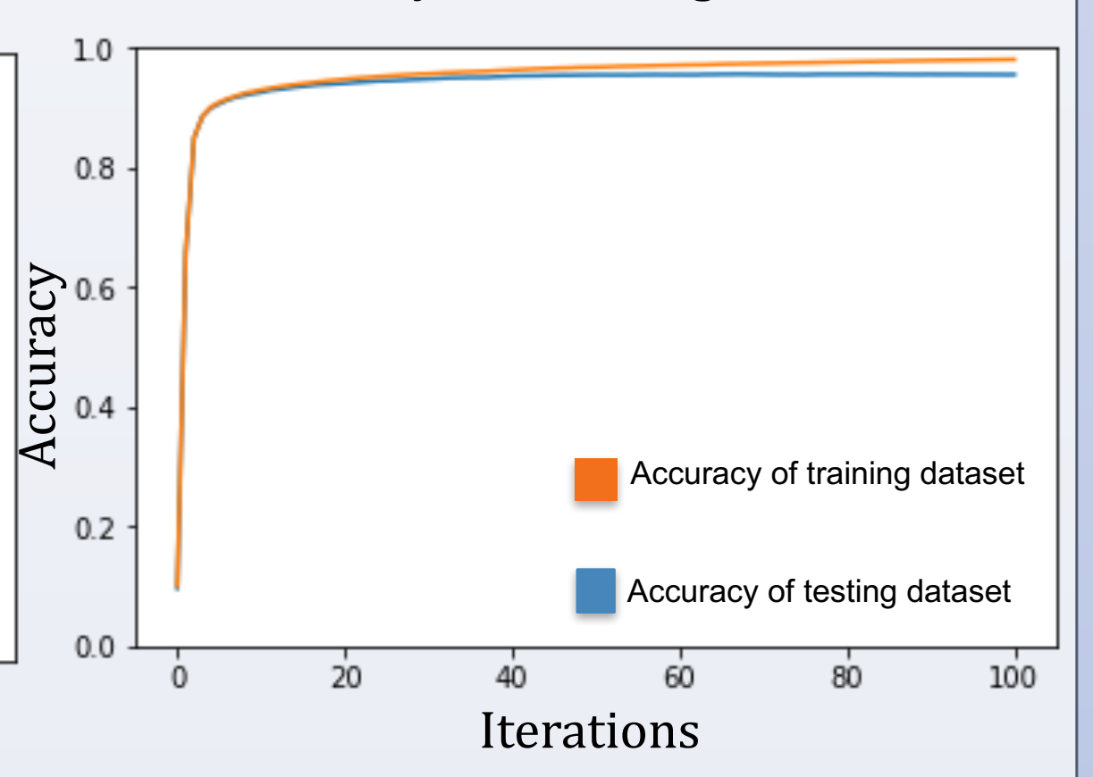
PERFORMANCE ANALYSIS

To evaluate the performance, the accuracy and cost were graphed as a function of training iterations.

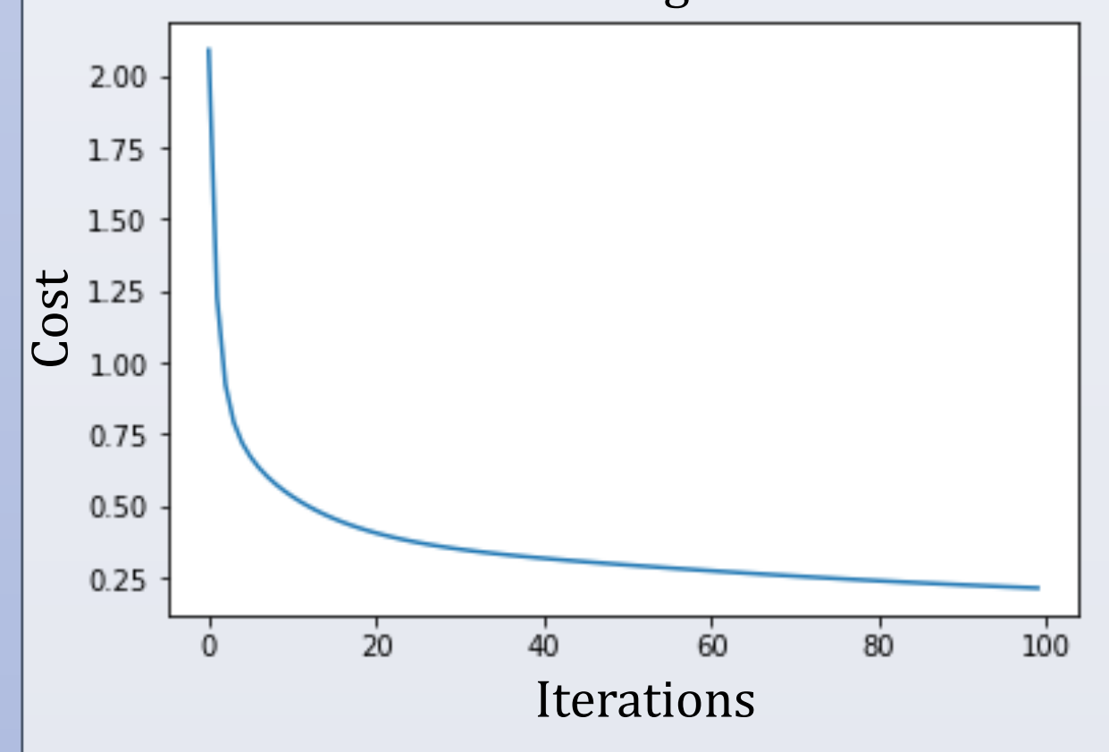
Testing Accuracy vs. Training Iterations



Accuracy vs. Training Iterations



Cost vs. Training Iterations



After 100 training iterations, the neural network with 270 hidden neurons correctly classified 98.1% of the training dataset and 97.4% of the test dataset.

Accuracy of testing dataset after 100 training iterations for various numbers of hidden neurons:

Hidden Neurons	1	10	30	90	270	810
Accuracy	11.3%	90.5%	95.6%	97.1%	97.4%	97.3%

Comparison to other published methods [1]:

Method	Test Accuracy
Mine: 2 layers, 270 hidden neurons	**97.4%**
Linear classifier (1-layer NN)	88.0%
K-nearest-neighbors, Euclidean (L2)	95.0%
3-layer NN, 300+100 hidden neurons	97.0%
Committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	99.8%

CONCLUSION

During this project, a neural network was implemented that could accurately classify digits from the MNIST dataset. This algorithm was programmed in Anaconda Python without any neural network libraries.

To improve the accuracy of this neural network, the number of hidden layers could be varied and more advanced techniques (regularization, deep learning) could be implemented. Applications of this technique include reading addresses on postage, processing images of checks, and autonomous vehicles.

REFERENCES

[1] Y. LeCun, C. Cortes, and C. Burges, "MNIST Handwritten Digit Database," [Online]. Available at: http://yann.lecun.com/exdb/mnist/. [Accessed 14 Jan. 2018].

[2] "Digit Recognizer," Kaggle, 2013. [Online]. Available at: https://www.kaggle.com/c/digit-recognizer. [Accessed 14 Jan. 2018].

[3] XeronStack, "Overview of Artifical Neural Networks and its Applications," Medium, July 16, 2017. [Online]. Available at: https://medium.com/@xenonstack/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7. [Accessed 14 Jan. 2018].

[4] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015