1 Introduction

A large advancement in modern times has been the ability for computers to recognize images through the use of machine learning and neural networks. Linear Algebra is heavily used by these neural nets to analyze images and make best guesses as to the subject of the image. A large practical use of this is with OCR technology that can digitize hand-written text, an application that has been especially useful to organizations like the US Postal Service which uses this technology to parse the addresses on handwritten letters using computers instead of humans. In this paper I will be looking at how a neural network uses linear algebra in the image recognition of handwritten numeric digits.

2 Background

2.1 The Data

In order to teach a computer how to recognize handwritten images a database of handwritten numerals and the data on the actual numeral written are needed so that the neural network has a way to train. This data for numerical digits comes from the MNIST dataset. In the realm of machine learning, working with the MNIST database is the closest equivalent of a Ĥello Worldṕrogram in that it is considered an elementary example intended to introduce a person to working with machine learning.
The MNIST dataset consists of data points that have 2 components, the first being a 28x28 pixel image of a handwritten image, and the second being the label that gives the number that is written in the image. Processing these images in order to train a neural network with them is the first step in recognizing them.

3 Training the Neural Network

3.1 Processing the MNIST Data Points

The first step is creating a vector from the 28×28 pixel image where each entry in the vector is a number from 0 to 1 represnting the color of the pixel from black to white. This means flattening a 28×28 array into a 1×784 vector. Then the dataset of 55,000 of these data points can be represented by a 55000×784 array. We will label this array `mnist.train.images` and create another array of the labels with dimensions 55000×10 where each image would have a 1×10 one-hot vector representing its label. For example for a 9 the vector would be: $[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$. This array we will call `mnist.train.labels`.

3.2 Softmax Regression

The way that the neural network will identify images is by creating a probability distribution for each digit than an image could be. The network could identify a photo of a 3 with a 90% chance of being a 3 while having a 1% chance of it being a 7. In this case a softmax regression is used to generate this distribution.
The softmax regression creates a weighted sum of pixel intensities for an image being in a certain class where the weight is positive if there is evidence of the image being in the class and the weight is negative is there is evidence against the image being in the class. The formula for the evidence sum is $\sum_j W_{i,j} x_j + b_i$ where W is the weights and b is the bias applied to the function, and x is the input pixels. The output of this sum is then normalized so that the resulting probabilities for each classification sum to 1.

3.3 Creating the Loss Model

We need to create a method for the neural network to ́teachítself how to properly identify the numbers. This is done by creating a loss model that measures how the neural network performed in each iteration so that the bias used in the evidence sum can be modified to minimize the loss. The specific function used here is cross-entropy loss model. $H_{y'}(y) = -\sum_i y'_i \log y_i$. In this function y represents the probability distribution created by the neural network and y' is the true distribution (the one-hot vector).

4 Conclusion

With the neural network trained we find that with this process the computer has about a 92% success rate in correctly identifying the digit. With more advanced Machine Learning tools computers can achieve 99% accuracy using Convolutional Neural Networks.