Neural Networks

Oliver Temple

Exeter Mathematics School

olivertemple@exe-coll.ac.uk

Abstract

Now commonplace in todays world, neural networks are becoming increasingly popular. This project will explore the basic concepts behind neural networks, their history, how they are used in the real world and how the most simple neural networks work at a low level. The project will conclude by exploring how to code a neural network from scratch to recognize hand written digits from 0-9.

Neural networks are complex computational models that have many uses. Every one of the numerous different types of neural networks each has its own pros and cons which suits it for one or another of many possible tasks including computer vision, classification and prediction. The structure of neural networks can vary dramatically for different models, allowing for a wide range of applications. The fluidity of neural network structure allows them to adapt to a wide range of tasks.

Introduction

Neural networks are complex computational models used to perform a wide range of tasks. This project looks into what neural networks are, what they are used for and how to code a simple neural network to recognize hand written digits from 0 to 9, using the MNIST dataset to train the network.

Main Objectives

1. Explore the different types of neural network models.
2. Explore the uses of different types of neural network models.
3. Explore how a neural network works from a high level.
4. Explore how to use a neural network to classify handwritten digits from 0-9.
5. Code a neural network to recognize handwritten digits form 0-9.

Types of Neural Networks

There are many different types of neural networks, each with their own advantages and disadvantages that make them best suited for specific tasks. The most common types of neural networks are:

- Perceptron
- Feed Forward Neural Network
- Multi-layer Perceptron
- Convolutional Neural Network

Perceptron

The perceptron is the simplest component of a neural network, often referred to as a neuron. It accepts inputs ranging between 0 and 1 and uses the weight that each connection has to calculate the output using the vector dot product of the weights and the inputs:

$$\mathbf{y} = \sigma(\mathbf{W} \cdot \mathbf{x} + b) \tag{1}$$

Where **y** is the output, **W** is a vector of the weights, **x** is a vector of the inputs, b is the bias and σ is the activation function.

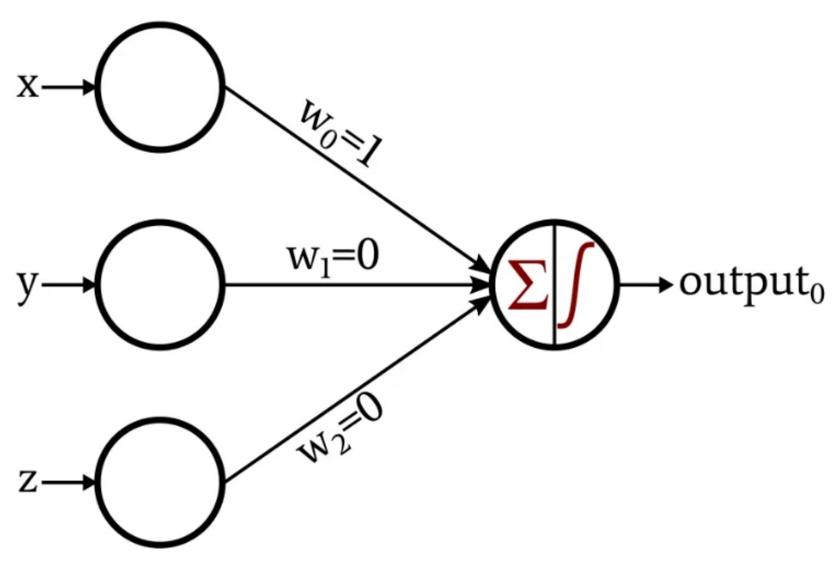

Figure 1: Perceptron

Feed Forward Neural Network

A feed forward neural network is constructed from a set of interconnected layers. Each neuron in a layer is connected to all of the neurons in the previous layer, and the output of each layer fed forward into the next layer and used as the input. The data only travels forwards through the network, making them useful only for linear classification. Each connection has a weight, and each neuron has a bias, which are used to calculate the output of the network. The output is calculated by:

$$\mathbf{y}_n = \sigma(\mathbf{W}_n \cdot \mathbf{y}_{n-1} + \mathbf{b}_n) \tag{2}$$

Where $\mathbf{y}_n$ is the output vector of the nth layer, $\mathbf{W}_n$ is the weight matrix of the nth layer, $\mathbf{y}_{n-1}$ is the output vector of the previous layer, $\mathbf{b}_n$ is the bias vector of the nth layer, and σ is the activation function.

Multi-layer Perceptron

A multi-layer perceptron is much like a feed forward neural network, in the sense that it is made up of interconnected layers of neurons, however, the data can travel both forwards and backwards through the network allowing for the network to be trained. The process in which the data travels backwards through the network is called back propagation and involves the altering of the weights and biases of the network. The output of the network is calculated the same as a feed forward network (See equation 2).

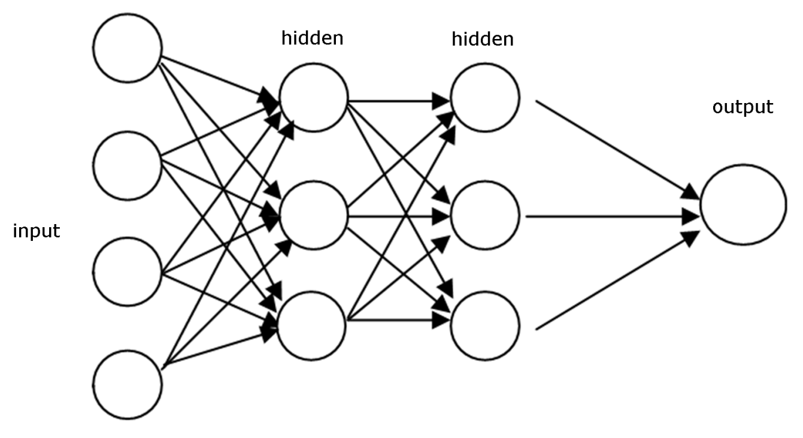

Figure 2: Feed Forward Neural Network

Convolutional Neural Network

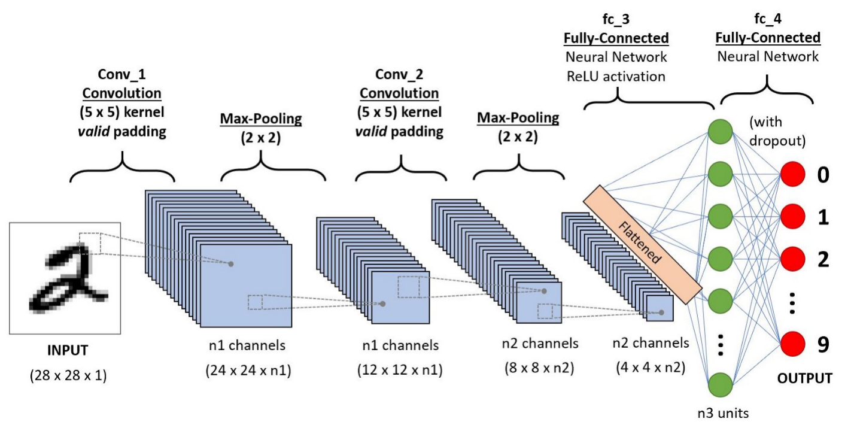

Figure 3: Convolutional Neural Network

A convolutional neural network is often used for image recognition and contains a three dimensional arrangement of neurons. Each neuron on the first layer only processes information from a small part of the field of view. We will not be using a convolutional neural network in this project, as a multi-layer perceptron will suffice.

Other Models

There are many other models beyond these that each have their own uses. The ones outlined here are some of the most common network models as well as one that is specialized for image recognition.

How Neural Networks Work

At its core, a neural network is a collection of interconnected neurons formed into layers. Each neuron has its own bias, and each connection has a weight, which are used to calculate the output of the network. For our model for classifying handwritten digits from 0-9, we will be using a multilayer perceptron.

Structure

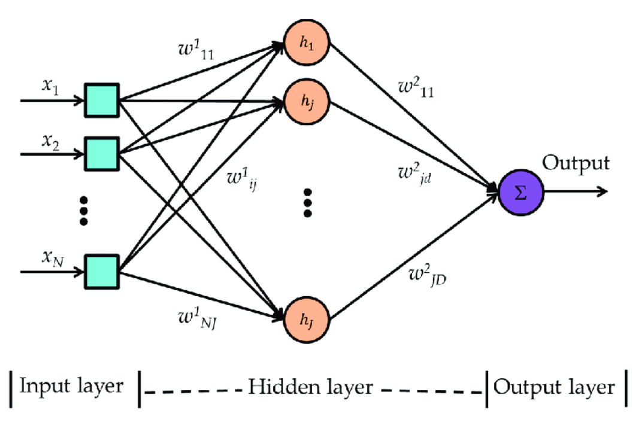

Figure 4: Multi-layer Perceptron Structure

The neuron with the highest output in the final layer will be the prediction of the network. The structure of the network is the first thing that must be defined. The input layer must consist of the same number of neurons as there are data points (e.g. one neuron per pixel in an image), and the output layer must consist of the same number of neurons as there are classification labels (the total number of outputs). There can be any number of hidden layers, which can have any number of neurons. The size and number of hidden layers should be varied to determine what sizes give the best results for the problem.

Forward Propagation

Forward propagation is when the data is fed forward through the network to get a prediction. To calculate the output, we take the weighted sum of the inputs, add the bias add pass it though the activation function (See equation 2).

Activation Functions

The activation function is applied to the output of each neuron before it is inputted into the next layer. The purpose is to prevent the input being too high or too low, and to add some non-linearity to the network. There are many different activations to choose from and we will be using the sigmoid function, due to the simplicity of the problem not requiring a complex activation function.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3}$$

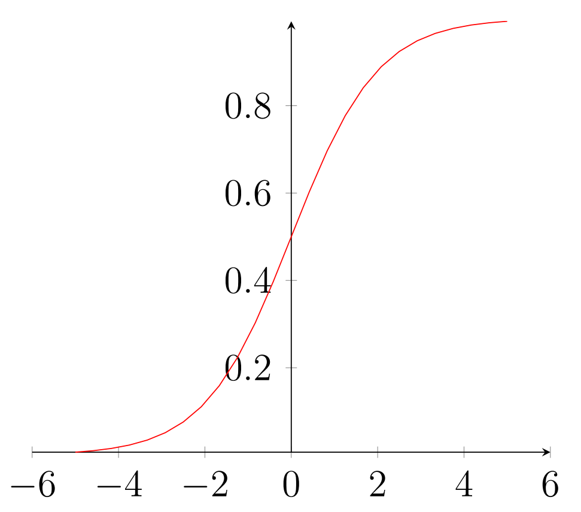

Figure 5: Sigmoid Function

Loss Functions

The loss function is used to calculate how good a network is. In order to calculate the loss we need the output of our network, and the expected output. Similar to activation functions, there are many loss functions to choose from, but we will be using the mean squared error (MSE) loss function due to the simplicity of the problem.

$$\texttt{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{4}$$

Where n is the number of samples, y_i is the desired output of the network, and $\hat{y}_i$ is the actual output of the network.

Back Propagation

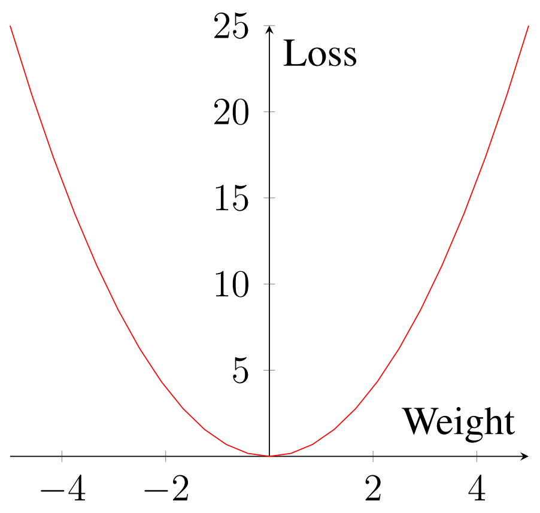

Figure 6: Example of Loss against Weight

Back propagation is the process in which the data moves backwards through the network, adjusting the weights and biases to minimize the loss function. To do so, the derivative of the loss function must first be calculated, as this will allow us to know whether to increase or decrease the weights and biases, and by how much.

For example, if a graph of loss against weight is plotted, as seen in Figure 6, the minimum of the function is calculated by using the derivative. However, as the loss function is usually in many more dimensions, the minimum cannot be calculated exactly and small steps towards the minimum are required.

When we have the derivative of the loss function, we know whether to increase or decrease the weights and biases due to the sign of the gradient. The weights and biases can be changed proportionally to the gradient to prevent missing the minimum of the function.

Training

In order for a network to be used to classify data, it must first be trained. Before we can train the network, we must first initialize the weights and bias with small random values. Next, we must split our dataset into training data and testing data, so that once training has been completed we can test the model on unseen data to get an idea of how it would perform in the real world. Training a network is done through the following process:

1. Propagate all the values in the input layer thought the network (Forward Propagation).
2. Update the weights and biases of the network using the loss function (Back Propagation).
3. Repeat until the accuracy of the network is satisfactory.

Coding a neural network

Data

When training a neural network, one of the most important assets in the data that will be used to train the network. Any imperfections in the data, such as incorrect labels or biases will be reflected in the predictions of the network. For out model, we will be using the MNIST dataset, which is a large dataset of handwritten digits from 0-9.

Splitting the Data

When creating a neural network, it is important to split the dataset into a training set and a testing set. This will allow us to test the network on unseen data. For our model we will use 50000 training samples and 10000 testing samples.

Defining the Model

Before training the model, we have to define the structure. Our network will consist of layers of interconnected neurons, and when defining our model, we must decide how many layers we want and how many neurons should be in each layer. We also need to define our loss function and our activation function. We will be using the sigmoid function (equation 3) for our activation function, and the mean squared error (equation 4) loss function. We also need to define the derivative of our loss function for back propagation. The derivative of the sigmoid function is:

$$\sigma'(x) = x(1-x) \tag{5}$$

Layers

The input layer will consist of 784 neurons, as the size of the input images is 28x28. We will be using one hidden layer of 350 neurons. The output layer will consist of 10 neurons, as there are only 10 possible outputs.

Training the Model

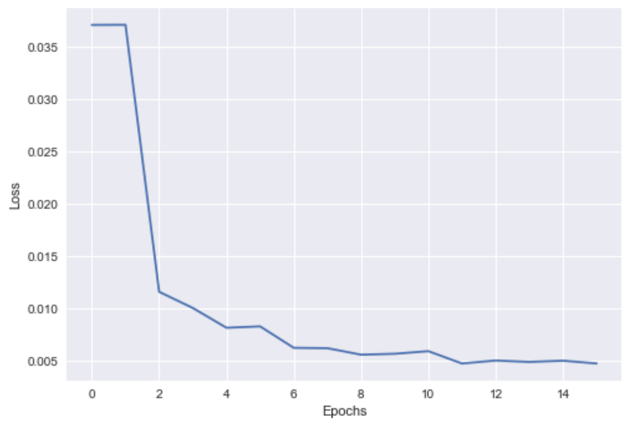

Figure 7: Loss of network over epochs

To train the model, the data is split up into equal sized batches. After a batch of data has been fed forward through the network, we will back propagate and adjust the weights and biases. Each batch is called a epoch. Training the network this way saves time, as back propagation can be computationally intensive.

Results

To calculate the accuracy of our trained model, we will feed every item in the testing dataset forward, keeping track of how many correct predictions there are. The accuracy of our model was 95.32%. To get an idea of what our neural network is predicting, we will feed 9 random testing samples into it, draw them, and compare them to what was predicted. The images shown in figure 8 were fed into the network, and the predictions were 8, 5, 8, 9, 1, 9, 7, 2, 8 and the actual values were 5, 5, 8, 9, 1, 9, 7, 2, 8. In this sample, we get an accuracy of 88%. This is lower, as only 9 samples were used.

Conclusion

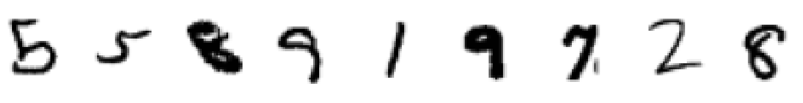

Figure 8: Sample Predictions

To conclude, neural networks are complex models that are used for different tasks. There are many different types of neural networks that each have their own merits and downfalls, that are best for different tasks. Although it is possible to code neural networks from scratch, it is much easier and quicker to use a library, as they are very well optimized and already have complex loss and activation functions implemented, as well as many different types of neural network.

Contact Details

Oliver Temple - olivertemple@exe-coll.ac.uk