

Comparing the Efficiency of a Neural Network

An EPQ By Sam Laister



June 30, 2020

Contents

[Small Introduction 2](#_Toc45042256)

[What will I be comparing? 2](#_Toc45042257)

[MNIST Database 2](#_Toc45042258)

[Contents 3](#_Toc45042259)

[EMNIST Database 3](#_Toc45042260)

[Exploring how Effective the Product is 3](#_Toc45042261)

[MNIST Database Testing 4](#_Toc45042262)

[Sample of a Random 50 4](#_Toc45042263)

[Testing each Number 10 Times 5](#_Toc45042264)

[EMNIST 6](#_Toc45042265)

[Sample of Random 100 6](#_Toc45042266)

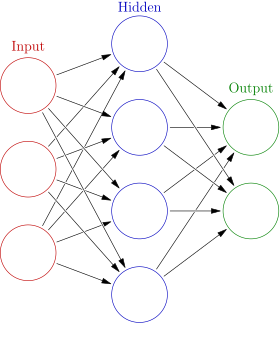
[Testing each Number 3 Times 7](#_Toc45042267)

[Conclusion 7](#_Toc45042268)

[Recourses Used (Credits) 9](#_Toc45042269)

Comparing the Efficiency of a Neural Network

# Small Introduction

Artificial Intelligence has always interested me. However, making such requires an extraordinary amount of time. So, for my EPQ, I decided it was the perfect opportunity to have a go and code one from scratch. As I am currently taking A level maths at college, I believed doing this would be in my comfort zone, to a degree I was right. However, this was a lot more challenging and time-consuming than I originally thought, and I didn’t think I’d make it to this point, with a finished product.

Figure

What is a Neural Network? Well it’s a lot of complex matrix maths all stacked on top of each other which gives the illusion that it’s intelligent. You will commonly see Neural Networks drawn like in Figure 1. Each column of circles is referred to as a ‘layer’ and the lines connecting them a ‘weight’. The weights are what give the illusion of it learning. By following a very complex algorithm, the weights are changed to produce a desired output from a set of inputs. So, for my neural network, each input will be a pixel, and the output will be a character. To use it correctly, you have to provide it with data to analyse, the output of said data is then compared to a desired output, and the weights are modified using back propagation. Eventually, after thousands of individual examples are fed through it, it will begin to understand testing data that hasn’t been seen by the network and guess a correct example.

# What will I be comparing?

By the end of this, I hope to test multiple things; how well my neural network is at image detection of digits, and letters. My initial idea was to include learning how to play a videogame, e.g. Mario. While this would be possible to do with the product I have created, I don’t understand enough about the emulation required to pull it off in time for the deadline. It also wouldn’t make sense in context of the test since detecting a handwritten 2 isn’t comparable with how long it could beat a simple level. But know that the maths and code required all boils down to exactly what I’ve created, so in the future it would be possible if I had more time.

## MNIST Database

So, for this test I have decided to compare detecting a number (0-9), a letter (a-z), and a simple drawing. In theory, these should all follow the same principles, just with different databases for it to learn from. The database for the numbers will be MNIST. From the website:

```The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on pre-processing and formatting. ```

Source: <http://yann.lecun.com/exdb/mnist/>

So, the general idea:

1. Feed the neural network each one of the 60,000 premade handwritten images
2. Feed the neural network a testing image that it hasn’t previously seen and chart the result.

I’m going to be collecting the data in excel and conclude what numbers it’s best and worst at guessing.

This will then be compared to the handwritten letters, ultimately concluding the neural network’s strengths and weaknesses without and recoding.

### Contents

The database contains 4 binary files:

* Training – Data
* Training – Labels
* Testing – Data
* Testing – Labels

In computing terms, parsing data means taking a document, and sorting it, logically, into a program. In the context of MNIST, it can be parsed using C# into an array or list object. The labels directly correspond to the data in said index, so parsing the data into an object that contains the data and label in one is very easy and simple to do. Since MNIST is a very popular database, there are plenty of methods out there publicly available for parsing the data. While researching, I found a user on Stack overflow use IEnumerables, and I will be using this method in my code:

<https://stackoverflow.com/questions/49407772/reading-mnist-database>

Note: This is the only section of code I am copying. The rest is written all by me

## EMNIST Database

The EMNIST Database is very similar to the MNSIT database, hence the name, except it’s for alphabet characters. Just like the MNIST Database, it contains a set of 60,000 drawings of letters. These are greyscale and of resolution 28x28. It then comes with 10,000 testing drawings which I will use to test and log the results.

From the website:

``` The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset ```

This is perfect since none of my code has to be rewritten to use this database.

The EMNIST Database can be found here:

<https://www.nist.gov/itl/products-and-services/emnist-dataset>

# Exploring how Effective the Product is

To test how effective my neural network is, I will run similar tests for each of the aspects I am exploring and plot them in excel for easy comparison. I provide a Colour Code which should stay consistent throughout the tests, which can show at a glance how consistent the neural network is at said topic. I will also provide some notes I find interesting or notes I can easily compare within the topics.

## MNIST Database Testing

This section will cover the 0-9 digits using the MNIST database. I will conduct a random sample of 50 results in a row and how well it was at guessing correctly, then Each digit 10 times to work out a percentage of how well it did.

### Sample of a Random 50

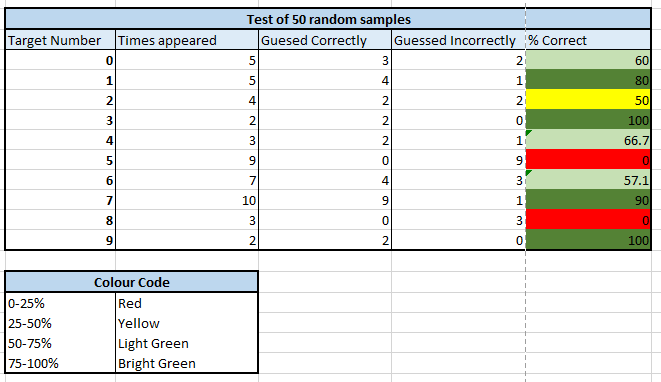


Figure 2 - Source: My own testing

Some interesting notes:

* For some reason, the neural network really struggles to guess 8 and 5 correctly, with getting 0 correct out of 50 attempts
* The neural network got 9 and 3 100% of times, however I wouldn’t say this is reliable since 9 and 3 only got randomly picked twice.
* Out of all the data, I’d say the neural network is reliably best at guessing 7. Since out of all 10 times it was picked, it correctly guessed 9. This surprised me since 7 is very similar to 1 and I expected the neural network to confuse the two.
* I expected the neural network to correctly guess 0 and 3 since they are very distinct numbers. However, I assume it thought 8 was 0 as well as 6 since they both contain circles which could be mistaken for a 0.

### Testing each Number 10 Times

A screenshot of a cell phone

Description automatically generated

Figure 3 - Source: My own testing

Things to note:

* As you can see, the neural network becomes pretty reliable when you test each number x amount of times. I assume after maybe 50 guesses on each number, the scores will plateau to be around 90% correct rate
* Once again, it struggles to guess the number 8
* This is where the neural network shines in my testing. It shows an extremely high rate of consistency and shows that with time, it could guess every digit every single time.

## EMNIST

This section will cover the effectiveness the Neural Network has on recognising handwritten letters. This is significantly more difficult for the AI since there are now 26 potential outcomes, and the letters are increasingly more complex while also being similar to each other when compared to simple the digits 0-9.

### Sample of Random 100

A picture containing screenshot, building

Description automatically generated

Things to note:

* The neural network is very clearly worse at guessing letters than numbers
* It failed to guess a,e,k,s,u,v and y every time they appeared. I assume this is because (minus the exception of k) they all look very similar to another letter.
* It is very good at guessing ‘m’, but not ‘w’. I assume it sees a ‘w’ and thinks it’s an ‘m’. This will depend on how many ‘w’s and ‘m’s it was fed when learning.
* Adding more hidden layers to the neural network gave it better correct / incorrect ratio.
* When guessing numbers, it got 0 nearly every time, and the same applies here to ‘o’. For some reason this type of neural network is very good at guessing round objects.

### Testing each Number 3 Times

A screenshot of a cell phone

Description automatically generated

Things to note:

* The neural network is consistently very poor at guessing letters. This could be for many reasons, the complexity of the letters, the 26 outputs compared to the 10 of the digits.
* Reading letters is a weakness of the neural network. However, with more testing data and changing the topology of the network, I believe it could become much more consistent at guessing correctly.
* It fails to guess many of the letters at all, which wasn’t the case with the 0-9 digits.

# Conclusion

This project took months to finish, and I can say I’m happy with the results. It shows so much potential when guessing digits that I believe this can be adapted and altered to show the same level of consistency for the letters.

If I could go back and re-test all the data, I would:

* Add more hidden layers for the letters – I don’t think only having 16 hidden layers is enough for the level of complexity in these characters.
* Use a database other than EMNIST that separates upper and lowercase letters. I believe that the main reason the neural network failed to guess letters is because it was fed, for example, ‘R’ and ‘r’ and told it’s the same picture, when it clearly isn’t.
* I would also adapt the neural network to recognise ‘R’ and ‘r’ as separate outputs. Giving it 52 outputs rather than the current 26. This would increase its accuracy since all the data being fed looks very similar to each other for the same output.

Overall, I very much enjoyed this project. The book ‘How to create your own neural network’ (Linked in credits) was a very enjoyable read during summer.

All the source code for this project and the compiled exe so anyone can access and run the product are available at: <https://github.com/Soup666/NEA-Neural-Network>

# Recourses Used (Credits)

A massive thank you to these recourses, since I they helped immensely.

|  |  |
| --- | --- |
| What it is | Direct link |
| An archive of a livestream by the university professor who runs The Coding Train YouTube channel. | [The Coding Train’s – Live Stream #120, MNIST and Pendulums](https://www.youtube.com/watch?v=KhogNPC24eI&t=5557s) |
| A video demonstrating how a neural network can solve XOR and why a simple perceptron can not | [The Coding Train’s – Coding Challenge #92: XOR Problem](https://www.youtube.com/watch?v=188B6k_F9jU) |
| This is my main recourse used in developing the neural network. It’s an 8.5-hour long series where he goes through the maths, and code required for this project. And while I wish it was as simple as copying the code, I decided to challenge myself and code mine in a different language. The outcome is very similar however. | [The Coding Train’s – Neural Network – The Nature of Code Series](https://www.youtube.com/watch?v=XJ7HLz9VYz0&list=PLRqwX-V7Uu6aCibgK1PTWWu9by6XFdCfh) |
| Great help with understanding the theory behind it | [3Blue1Brown - But What is a Neural Network? | Deep learning, chapter 1](https://www.youtube.com/watch?v=aircAruvnKk) |
| ^ | [3Blue1Brown - Gradient descent, how neural networks learn | Deep learning, chapter 2](https://www.youtube.com/watch?v=IHZwWFHWa-w) |
| This is the first recourse I bought to ensure I stuck with the project. It’s a book I bough on my kindle which I gradually read through quarantine. It’s split into 3 sections; the theory, a worked example, and using it. The worked example was written in a programming language called Python, since mine is written in C# it didn’t help massively but gave me the general workflow for how it should work. The theory section is what inspired The Coding Train’s series, so it was crucial for me to read it | [Tariq Rashid - Make your own Neural Network](https://www.amazon.com/Make-Your-Own-Neural-Network-ebook/dp/B01EER4Z4G/ref=as_li_ss_tl?ie=UTF8&qid=1498492463&sr=8-1&keywords=make+your+own+neural+network&linkCode=sl1&tag=natureofcode-20&linkId=0d10fdc485d6452bb7fc2b62ab4ffd31) |
| The MNIST Database, used for the handwritten numbers | <http://yann.lecun.com/exdb/mnist/> |
| The EMNIST Database, used for the handwritten letters | <https://www.nist.gov/itl/products-and-services/emnist-dataset> |
| The code I used to parse the database is publicly available from here | <https://stackoverflow.com/questions/49407772/reading-mnist-database> |
| Image of a neural network | <https://en.wikipedia.org/wiki/Artificial_neural_network> |