Comparison of Neural Networks for Digit Recognition

To what extent does the number of layers and units per layer affect the performance of a sequential neural network when used in image processing.

A Computer Science Extended Essay

Personal Code: hnl938

June 2019

Word Count ~3950 Words

Table of Contents

Introduction

A.I. (artificial intelligence) is seen in everything from facial recognition to complex data analytics. A subset of A.I. is machine learning. Machine learning commonly referred to as ML, is where computers learn on their own without being explicitly programmed. This paper will be focusing on artificial neural networks (will be referred to as neural networks or artificial nets), a form of ML which attempts to replicate the design of the neurons seen in the human brain. It allows programmers to create software in a more abstract manner and program tasks that would take millions of lines of code within a few lines of code. While neural networks can be used to program many different tasks, this paper will be exploring the topic of neural networks in the case of text processing.

Text processing, otherwise referred to as OCR (Optical Character Recognition) comes in many forms and is seen in everything from programs that allow data entry for business documents to automatic number plate recognition. All forms of text processing take some sort of image input. (Fehr,2019)

What is an Artificial Neural Network?

Types of Machine Learning (See appendix for Machine Learning description and explanation)

Artificial neural networks are a subset of machine learning, specifically (in the case of this paper), it is supervised learning. Supervised learning, where the desired output is known as opposed to the second type which is unsupervised learning. This is when there is not really a fixed known output, but rather the algorithm is attempting to find patterns in datasets[2]. The final type of machine learning is reinforcement learning, where the algorithm is rewarded for getting closer to the desired output and punished for incorrect actions. It is analogous to training a pet. (Oladipupo, 2010)

Structure of an Artificial Neural Network

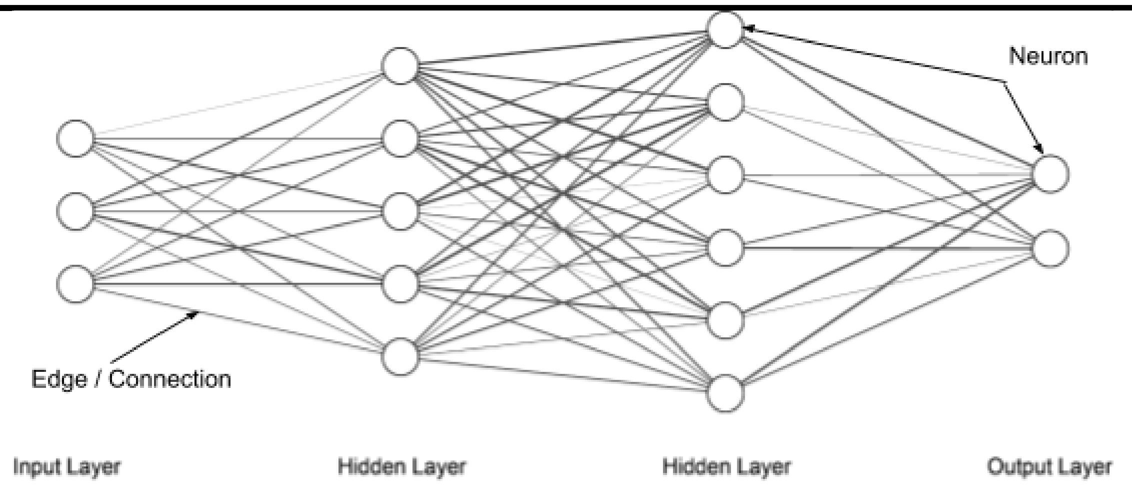


Figure 1.1 An example of an Artificial Neural Network

Source:http://alexlenail.me/NN-SVG/

To be more specific, an artificial neural network is comprised of many algorithms stuck together in the form of a neural network. This is commonly represented with a graph[3], as seen in *figure 1.1*. The network has an input layer which contains neurons (represented by the circles) for each numerical input. In the case of images, this means that there must be an input for every pixel in each channel of colour. Therefore, if an image has a resolution of 1920x1080 with 3 colour channels, it would result in a total of 6,220,800 neurons in the first layer. But, this can be reduced if the input is normalized, e.g. cropped to 28x28 pixels in grayscale (784 neurons). These input neurons then connect to the hidden layers through the edges of the graph. Throughout the neural network, each layer connects to the following layer such that every neuron in the first layer is connected to every neuron in the layer right after. These connections all have weights to them. For every neuron after the input layer, the value of that neuron is based on the weighted sum of each neuron in the previous layer. To clarify, the weight is a representation of the previous neuron's importance (among all the neurons connecting) to the receiving neuron.

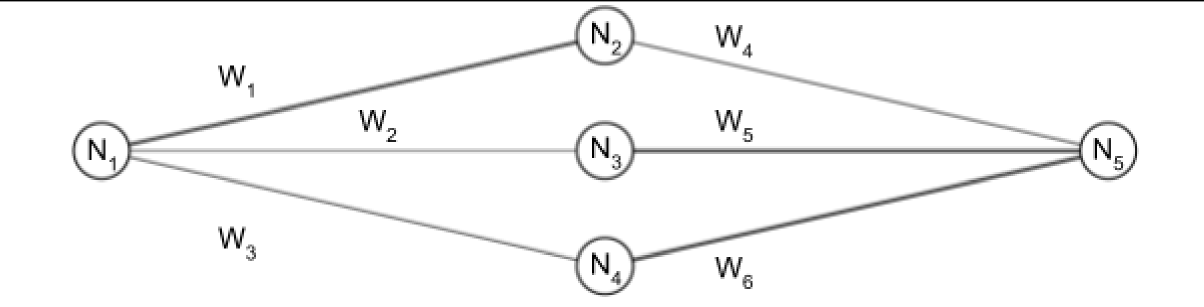


Figure 1.2 Example of weights in a Neural Network

Source:http://alexlenail.me/NN-SVG/

For example, in *figure 1.2* we can see the weights affecting each neuron. To get value at neuron E (before the activation function) the following equation would be used:

$$N_5 = W_4N_2 + W_5N_3 + W_6N_4$$

Where W_x is the weight and N_x are the outputs of the neurons.

In other words, since the weight of neuron C is greater than the weight of neuron B, we can say that it has more effect on the output and is more important to neuron E. This is applied throughout the network. However, at this point, it may seem like each neuron is simply adding numbers together and passing them on to the next neuron. This is not the case. One of the parts of artificial networks that resemble the neural networks seen in human brains are the neurons. More specifically, it is how the neurons work. In the case of artificial nets, the neurons have an activation function which tells the computer when they are activated or ‘fire’. This is attempting to simulate how neurons activate in the brains of living creatures and are commonly done via an activation function.

Activation Functions

An activation function is a mathematical model which dictates whether or not a neuron ‘fires’ based on the value received from the weighted sum as discussed above. In most cases, it also dictates how strong of a signal a neuron will send.

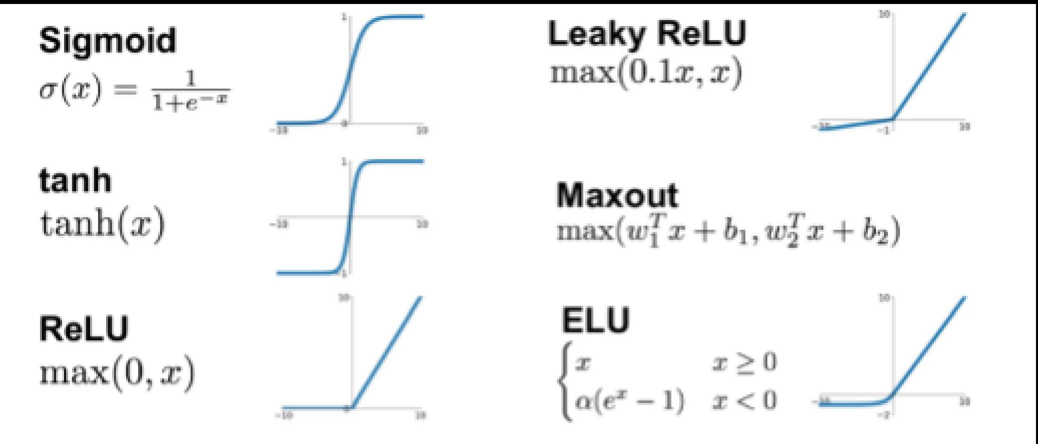


Figure 1.3 Examples of Activation Functions
Source: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

As can be seen from *figure 1.3,* there are many different types of activation functions which are all situational. ReLU and its variants (e.g. SReLU or LReLU) are one of the more commonly used functions because it works well in many situations. However, before the weighted sum is put through the activation function, there is a bias, a numerical value, added to each neuron in order to increase the ability for a neural network to solve tasks. This is added such that there is another variable which can be changed in order to get better results for the neural network.

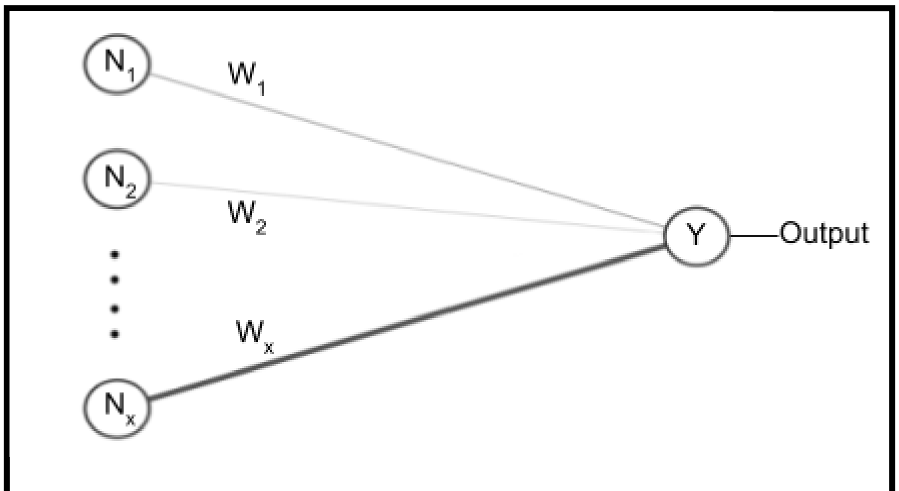


Figure 1.4 Example of the values comprising a Neuron

Source:http://alexlenail.me/NN-SVG/

To summarize, we can create an equation that will describe the output value of a neuron, seen in *figure 1.4*, as follows:

$$\mathit{Output\ of\ Neuron\ Y} \ = f(b + \sum_{i=1}^{x} W_i N_i)$$

Where b is the bias added, W_i are the weights for the corresponding neuron N_i and *f* is the activation function.

This output is then passed on to the neurons in the next layer and the process repeats until the output neuron is reached. In a neural network, it is very likely that there will be a combination of different activation functions in order to give an output which is more suited for the situation and increase the performance of the neural network. It is important to understand this is a slight simplification of the true neural networks faced when programming and some details have been excluded like hyperparameters, other types of networks, etc. Yet, it is still possible to build a neural net with knowledge only on the aforementioned ideas by utilizing certain libraries such as the ones seen in python like Keras.

How a Neural Network learns

Structure of Training Data

When creating a neural network with a programming language (referred to as a model by tensorflow) the main idea is to be able to give the model inputs and inform the computer of the corresponding correct outputs. There are different sizes of inputs given to a model which can each be controlled in order to affect the rate at which a model learns. At the smallest level, there is a *sample*, a single row of data containing both the input and the output. One level above this is a *batch*. The batch size is a hyperparameter[5] and affects the rate at which the model learns due to reasons explained later. A batch is simply the number of samples sent to the neural network before the model is updated. The hyperparameter *epoch* is one level higher than the batch size and is the number of times the model processes the entire training set (i.e. all samples in the given training data). (Brownlee,2018)

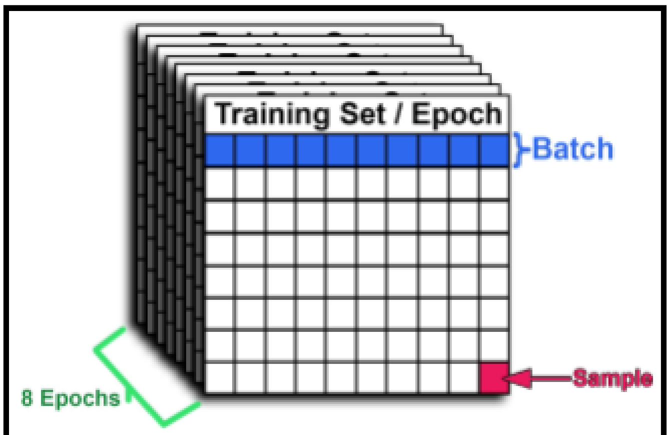


Figure 2.1 ***Visualization of data inputs to NN***

(Explanation in Appendix 4)

Data Propagation in a NN

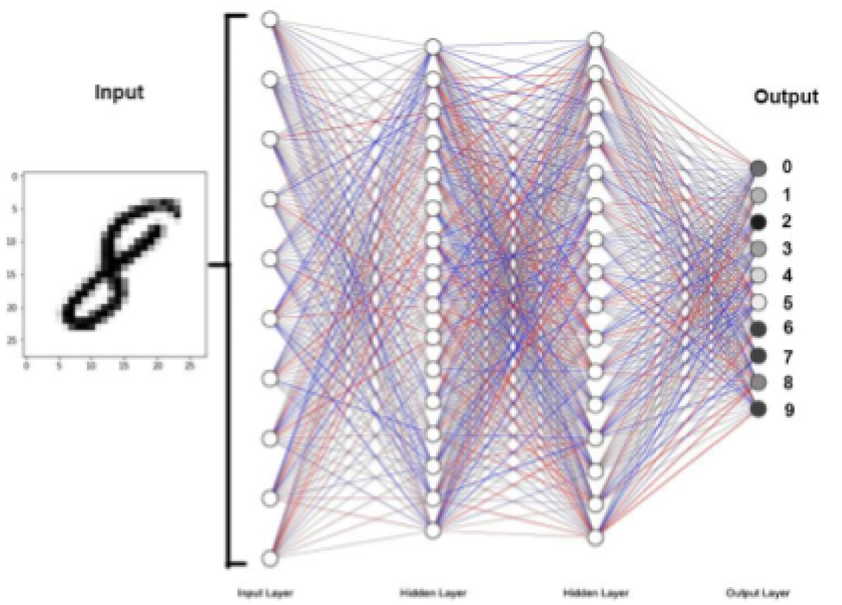


Figure 2.2 *Incoherent answer from randomly initialized NN*

Source:http://alexlenail.me/NN-SVG/

The process of learning begins during the initialization of the model. The computer randomly initializes the aforementioned weights and biases in the neural network such that any input at that moment would most likely not result in a coherent answer as seen in *figure 2.2*. A coherent answer would be one where neuron 8 is the most activated as compared to any other neuron, i.e. neuron 8 would have an output value of 0.8, while the other neurons would be around 0.1 or lower.

When the model gets the first sample sent through the output of the model is measured. This is then compared against the 'correct' value in a loss function. The loss function basically tells the neural network how to 'nudge' (small increments to the values) the biases and weights such that the same input would yield an output closer to the desired output *(Sanderson, 2017).* These changes are done layer by layer in the opposite direction of the neural network (i.e. backwards, from the output layer to input layer). For instance, the NN in *figure 2.2* starts off by wanting an output closer to 8, as seen in the image. Therefore, it wants 8 to be the most activated up neuron (the activation of the neuron is portrayed in *figure 2.2* by the darkness of the neuron. I.e. darker means more activation). This means that the weighted sum of the previous neurons connecting to the neuron representing 8 must be altered such that 8 is most activated. This results in 3 options. Either, the bias of neuron 8 itself can be increased, the weights of all the neurons connecting to neuron 8 can be increased, or the activation of the neurons in the previous layer can be affected to get the desired output. These will all increase the value inside the activation function and thus increase the chance/ value of the output of the activation function.

Neuron 8 =
$\Phi(w_1 n_1 + w_2 n_2 + \dots w_x n_x + b)$

Where:
n_z is the zth neuron

w_z is the weight for that neuron

Φ is the activation function

b is the bias

In order to increase activation of neuron 8:

1. Weights of previous neurons must **increase** proportional to n_z.
2. Bias must **increase.**
3. Previous neurons must **change** proportional to w_z.

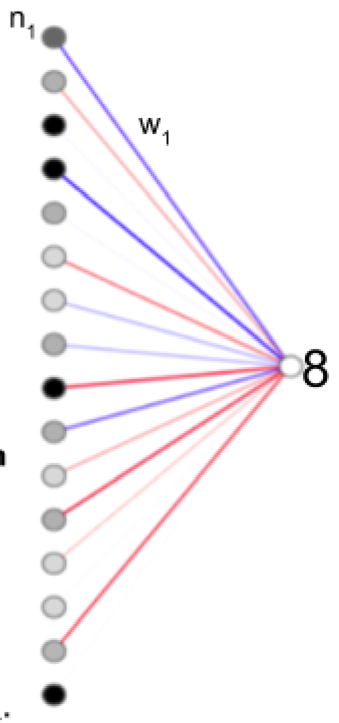
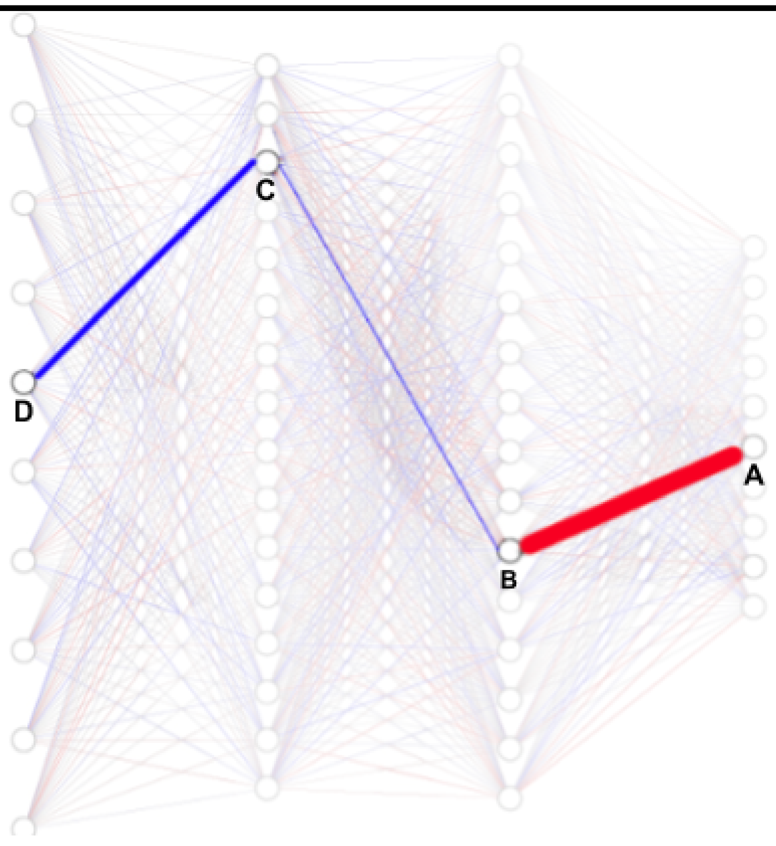


Figure 2.3 *How to increase activation of a neuron*

However, in order to increase the activation of 8 in an efficient manner, the weights of the neurons which are currently most activated (e.g. n_3 ,n_4 ,n_9 and n_{16}) should increase. This is because those neurons have the strongest influence on the activation of neuron 8. Therefore, we must increase the weights of the previous neurons *in proportion* to the influence those neurons have on neuron 8. In addition to this, the way the previous neurons should be affected is as follows: neurons with negative weights should decrease output value and neurons with positive weights should increase output value. Once again, this would increase the value inside the activation function of neuron 8. Similar to the weights, it would also be most efficient to alter these output values proportionally to the weights (rather than the neurons) because those are the neurons with most importance for the activation of neuron 8. This process is then repeated with every neuron in the network until it reaches the input layer (an example is seen in *figure 2.4)*. In actuality, this process takes a lot of time as one can imagine. The aforementioned hyperparameter describing the batch size can be changed such that this update only occurs once every so often, yet the loss function actually averages the losses such that all the samples have been taken into account.



Increase output of neuron *A* because it is desired output as calculated by loss function.

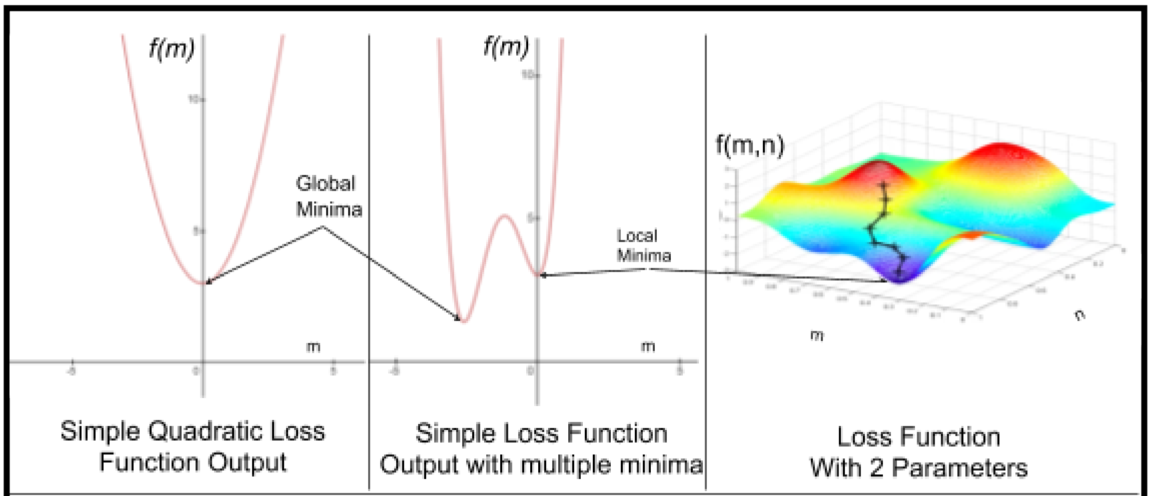
Pathing:

1. *B* has a heavy negative weight on neuron *A*, decrease neuron *B* by a large amount.
2. *C* has a small positive weight on neuron *B*, decrease neuron *C* by a small amount.
3. *D* has a medium positive weight on neuron *C*, decrease neuron D by medium amount.

Figure 2.4 *Example of Backpropagation on one neuron for one update to NN.*

Loss Function and Optimization

In simple terms, a loss function is a quantifiable value which informs the computer how incorrect the value produced by the neural network was. On top of this, it also shows the confidence of the neural network. Networks may get the right answer but simply be guessing, resulting in a high loss due to lack of confidence as opposed to high confidence resulting in low loss. If this was used in the real world, answers would end up being wrong more often due to incoherent answers similar to *figure 2.2*. During the training of the neural network, the loss function takes in the values of all the different variables that can be changed in the neural network and produces a value-based off the samples given that quantifies how incorrect the network is. This is simple to understand on a 2 or 3-dimensional level, where there are 1 or 2 parameters in the neural network which can be changed (e.g. a weight connecting 2 neurons). However, in real neural networks, there are a few thousands of these parameters which can be changed hence why the loss is used in a trial and error manner. For example, assume the loss function starts high for a certain set of parameters. The neural network will test another, slightly altered, version of the parameters and see whether the output of the loss function has decreased or increased. If the loss function has decreased, the computer understands that this is the direction in which the loss function is sloping downwards. On the other hand, if it increases, the computer understands that going in the opposite direction would result in a decrease of the loss function. This is stochastic gradient descent (SGD) and is commonly visualized like a ball rolling down a hill. In mathematical terms, this is analogous to finding the gradient and going in the opposite direction. Although this may result in a decrease in the loss function, it is not definite to get the lowest point. If the loss function produced a more complex graph with multiple minima, SGD would only find the local minima, not the global minima.



Where f( ) is output of loss function and m,n are parameters.

Figure 2.5 Examples of loss function outputs

Experimental Methodology

After understanding the inner workings of neural networks, it was decided that the main focus of this paper will be on the impact of the general structure of a feed-forward neural network on the performance of the network. More specifically the question generated was as follows:

To what extent does the number of layers and neurons per layer affect the performance of a sequential neural network when used in image processing.

An experimental methodology was chosen due to the lack of secondary research that investigates the specific question at hand. The methodology is also limited by technological aspects due to the availability of hardware and time. Before beginning the research, it was clear to define and clarify some terms described in the research question. Firstly, the term performance is very broad and can refer to many different measures when talking about neural network models. While accuracy is used more in the real world as it is a binary value which dictates whether or not the network’s prediction was correct, to keep things simple, performance will be based majoritively on loss because it is a quantified value that is designed for the evaluation of NNs. The term “image processing” will be simplified to text processing in images, specifically, the MNIST data set.

In order to try and eliminate biases, the experiment must be conducted in a manner such that every tested structure would contain the same hyperparameters, i.e. batch size, learning rate, etc. It was also vital to test a wide range of structures. The time taken should generally be the same but may vary slightly from structure to structure due to the change in the number of parameters in the network that can be tweaked (i.e. weights and biases).

Data Set

The MNIST data set is a premade set of images each with a label of the corresponding number (0 - 9) seen in the image (Deng,2012). As it is impossible to test every single neural network structure, a simple feed-forward neural network with regular neurons at each layer will be used. A total of 60,000 samples are provided for use. 10,000 of which will be used to evaluate the network on unseen data, leaving the rest 50,000 samples to be used for training. This was simply loaded in using Keras.

Model Structure

The input layer will contain neurons that amount to the number of pixels on each input image (28 by 28 pixels in one colour channel, i.e. 784 neurons). The output layer will contain 10 neurons, each with the *softmax* activation function. This is accomplished by using a discrete set of possible neuron configurations per layer (i.e. multiples of 20 up to 100 per layer). In addition, the depth of the neural network was required to conform to the fact that 'the optimal size of the hidden layer is usually between the size of the input and size of the output layers' (Heaton, 2017). In order to eliminate randomness, the network was initialized using the glorot uniform initializer and the seed '1234'. This allows the results to be repeatable and ensure that the randomness does not affect the results. All of these layers were made using keras dense layers. The batch size was set to 256 samples, which was found to be optimal between accuracy and time taken to train via simple trial and error on one model. The number of epochs per model was set to 3 due to the same reasoning as batch size. The loss function was set to *Sparse categorical cross entropy* as it is a variation of the most commonly used loss function (cross entropy) when testing with the MNIST data set (Hinton, Vinyals and Dean, 2015). In addition, *adam* was used as it is the most commonly utilized and best-suited optimizer for this task (Kingma and Ba, 2014).

Dependent Variables

The variables being measured during the experiment are accuracy and loss.

Accuracy

Accuracy is a percentage which relays how many samples were predicted correctly. Unlike loss, this means that the value is either correct or incorrect. It calculates the number of correct predictions out of the total number of samples tested. This is useful in the real world as compared to loss.

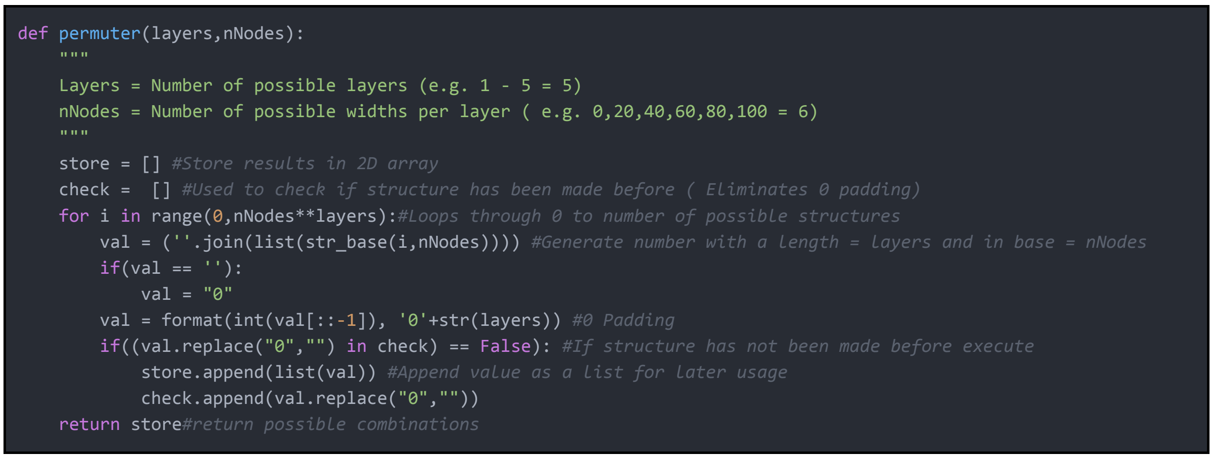
Loss

As mentioned above, loss is the quantitative value formed via a mathematical expression which dictates how incorrect the network was. This will be measured to see how close the network was to the right answers. When predictions get better, loss will decrease while accuracy may remain the same, this will show confidence in the answers. (Vallantin, 2018)

Programmatic generation of Network Structures

In order to create the possible structures of the neural network, a simple subprogram was created to generate a set of values that stores all possible combinations of neural network structures which follow the following rules: there can be 1-5 hidden layers, each with one of the following number of neurons: 0, 20,40,60,80,100.

The code is as follows:



```
def permuter(layers,nNodes):
    """
    Layers = Number of possible layers (e.g. 1 - 5 = 5)
    nNodes = Number of possible widths per layer ( e.g. 0,20,40,60,80,100 = 6)
    """
    store = [] #Store results in 2D array
    check =  [] #Used to check if structure has been made before ( Eliminates 0 padding)
    for i in range(0,nNodes**layers):#Loops through 0 to number of possible structures
        val = (''.join(list(str_base(i,nNodes)))) #Generate number with a length = layers and in base = nNodes
        if(val == ''):
            val = "0"
        val = format(int(val[::-1]), '0'+str(layers)) #0 Padding
        if((val.replace("0","") in check) == False): #If structure has not been made before execute
            store.append(list(val)) #Append value as a list for later usage
            check.append(val.replace("0",""))
    return store#return possible combinations
```

The Experimental Procedure

The entire neural network was designed in *Keras* and *TensorFlow* and starts off by generating the possible structures. The program automatically goes through each possible structure and does the following:

1. Clear any residual data from previous structures.
2. Generates a new fresh model with the same initialization as other structures.
3. Train and evaluate model and store data in a .txt file.
4. Repeat till all structures were completed.

This was run on a desktop with no interaction from keyboard or mouse till process completed.

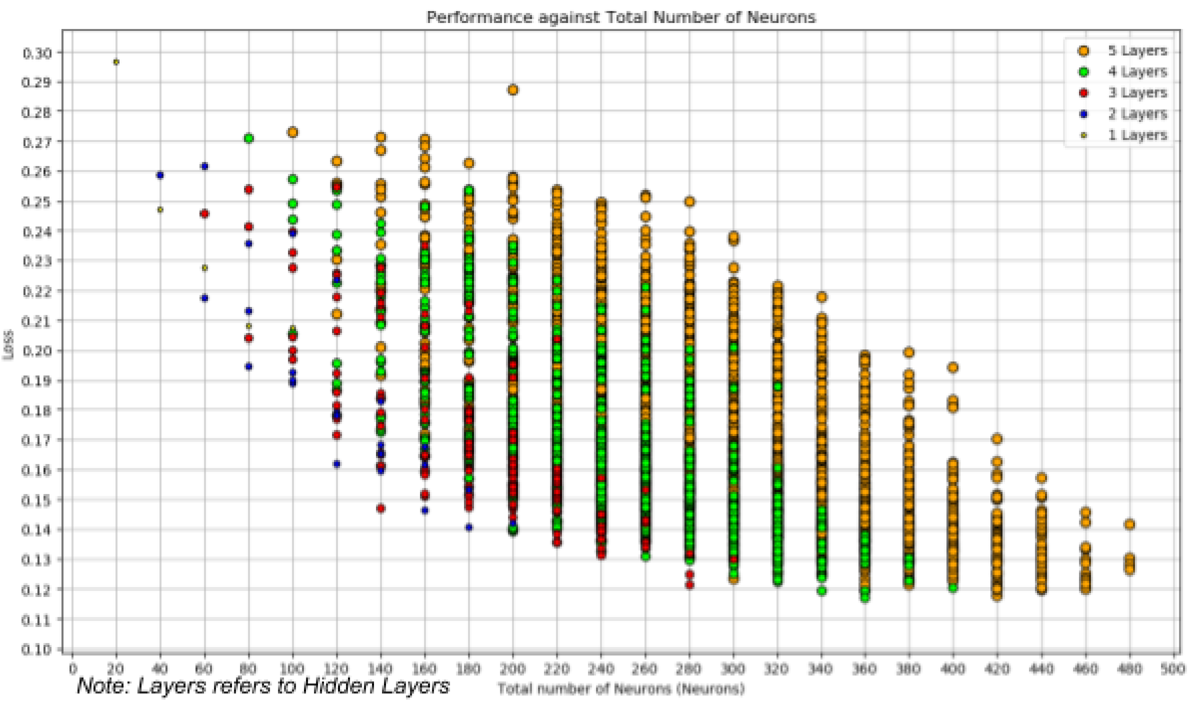
Experimental Results

The data below shows a few random entries in the raw results written to the text file. This is unparsed data. The structure column contains the number of neurons per layer divided by 20. Raw data is attached in the appendix.

Loss	**Accuracy (%)**	**Structure**	**Time Taken (Per 10,000 Samples)**	**Number of Total Neurons**
0.422619	0.8973	0~0~0~0~0	0.532481	0
0.29664	0.9158	0~0~0~0~1	0.563729	20
0.247147	0.9307	0~0~0~0~2	0.650483	40
0.227438	0.9346	0~0~0~0~3	0.64998	60
0.207947	0.9409	0~0~0~0~4	0.420091	80
...3897 Rows not shown...				
0.15848	0.9508	5~5~5~5~1	0.50737	420
0.13165	0.9582	5~5~5~5~2	0.507218	440

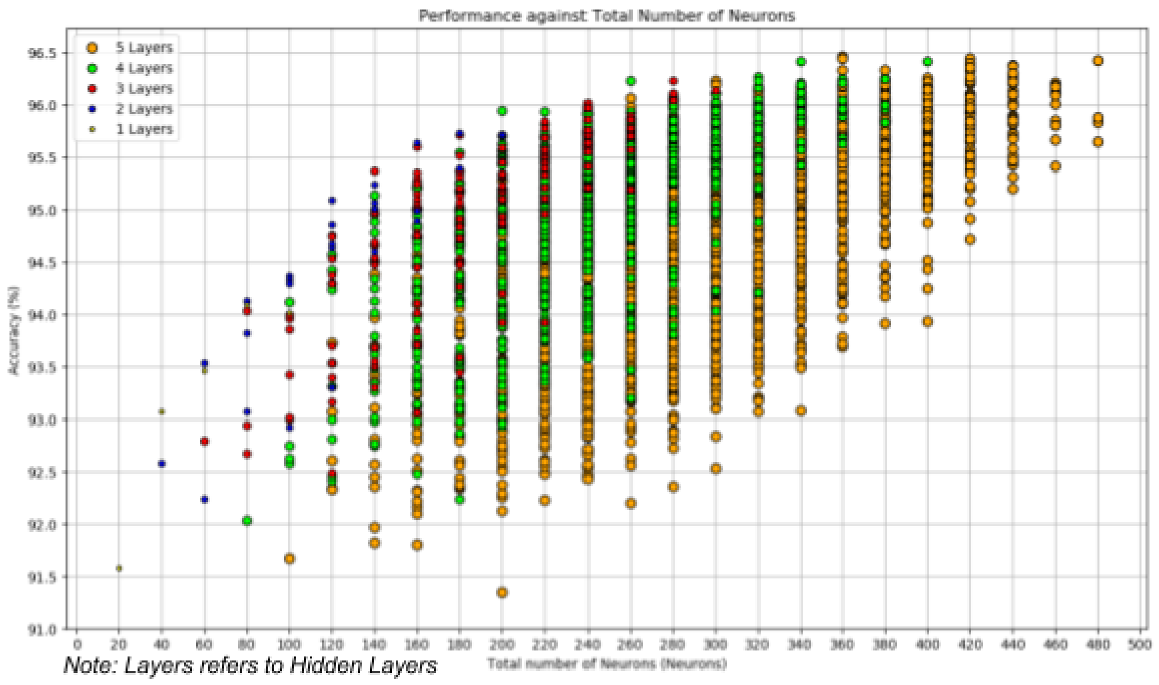
Graphical Representation Of Data

The data was then fed through a simple python script to use a library function, matplotlib, to create a graph with data on the total number of neurons against separate variables. Colour coding was used to indicate the number of hidden layers in the structure (total number of layers would have been # *hidden layers* + 2). To emphasize this, the size of each point is relative to the number of layers as well. ***Note: Not all variables included.***



Note: Layers refers to Hidden Layers

Graph 1.2 Performance against Total Number of Neurons

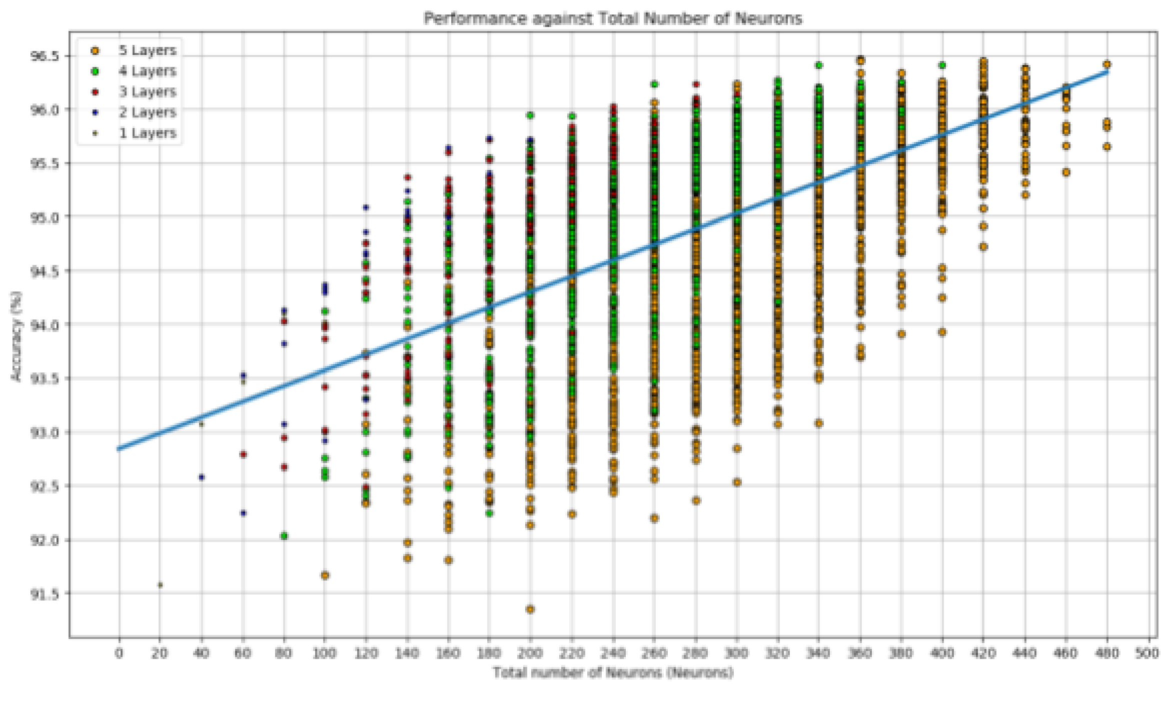


Note: Layers refers to Hidden Layers

Graph 1.1 Performance Against Total Number of Neurons

Analysing data

While a clear trend can be seen between the total number of neurons and the average performance (i.e. both loss and accuracy), it cannot be generalized due to the variable nature of neural networks. Given the fact that the seed was fixed, the loss function **may** have simply been optimized to be close to a local minimum. While in reality, had the network been randomized, the chance of reaching minima with a lower loss or potentially even the global minima would have been possible. In addition, the axis of the graphs has been zoomed in to emphasize the scale of the results. Upon closer inspection, it can be seen that the accuracy varies by fractions of a percentage when reaching higher neuron counts. The following line in *graph 1.3* was calculated using the numpy (a python library) function *polyfit*.



Graph 1.3 Performance against Total Number of Neurons (With Line of best fit)

This graph does not entail the full concept. This is because, while increasing the number of neurons does generally benefit the performance of a neural network, simply increasing the number of neurons to an arbitrary number like 1,000,000 will not guarantee an increase in performance. Nonetheless, having fewer than a certain amount of neurons per layer will prevent the neural network from being able to learn for this given problem due to the nature of neural networks. (chioka.in, 2016)

After processing the data, neural networks with the least loss values were taken and examined, these are the structures:

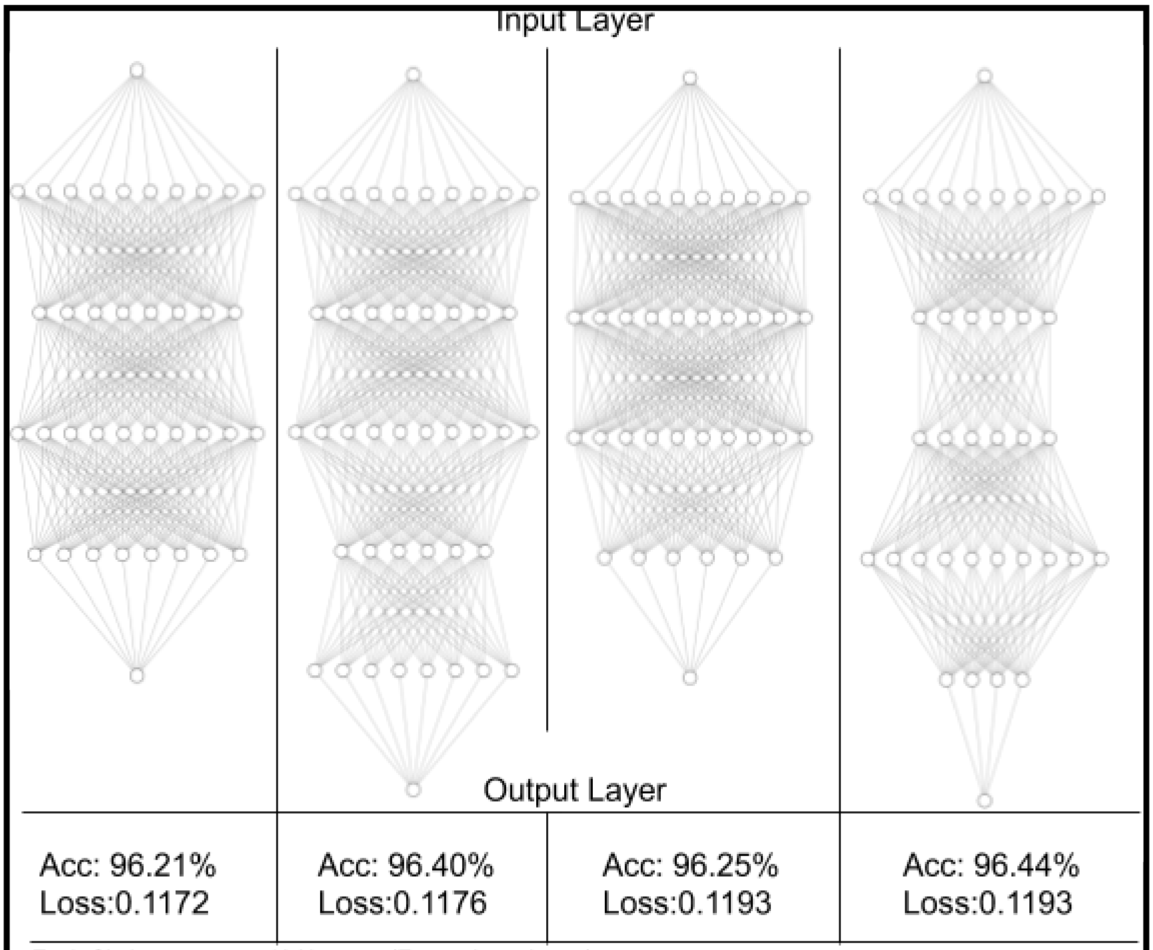


Figure 3.1 Networks with least *loss* in experiment

Evaluation

Through the data, it can be seen that a general trend is observed when the number total number of neurons increases. But, simply increasing the number of total neurons may not see a definite increase in the performance of a neural network due to the natural randomness of this feature.

The data gathered in this paper takes some arbitrary structures of neural networks. The increments in the experiment were set to cover a broader range of results at the same time as reducing the time taken to conduct the experiment and resources used. Had the batch size been set lower and the number of epochs set higher, the total loss was seen to decrease (along with an increase in accuracy) due to the more frequent updates and longer training time.

If the experiment were to be repeated, there should be further investigation on how the elements of the experiment can be more repeatable as even with the same hyperparameters, there was still some randomness observed(e.g. loss varied by 0.001 for the structure 1-4-3-2-1 when tested twice). Moreover, the time taken was not observed in this paper, a key factor that should be considered when modeling AI. In addition, research should be focused around convolutional neural networks (mentioned in the conclusion) or should widen the range of structures tested (i.e. more layers and more neurons).

Conclusion

While this experiment shows somewhat of a link between the number of neurons in a neural network and the performance of the neural network, in reality, a simple neural network is commonly not used when making an AI to classify text on images. Convolutional neural networks (CNN) are used. ‘CNNs are used for image classification and recognition because of its high accuracy.’ (Maladkar, 2018) Convolutions are a more complex form of neural networks which use filters to extrapolate structure, like curves and lines, from images. These are commonly much more accurate and have been used more often in recent years.

Overall,this paper shows that there is no fixed number of neurons that will generate the best result. It must be fine tuned depending on the problem and situation. Researchers should look into which structure and type of ML best fits their situation.

Works Cited

Raval, S. (2017). Which Activation Function Should I Use?. [online] YouTube. Available at: https://www.youtube.com/watch?v=-7scQpJT7uo [Accessed 10 Jun. 2019].

Moroney, L. and Allison, K. (2019). Machine Learning Zero to Hero (Google I/O'19). [online] YouTube. Available at: https://www.youtube.com/watch?v=VwVg9jCtqaU [Accessed 26 May 2019].

Brownlee, J. (2018). What is the Difference Between a Batch and an Epoch in a Neural Network?. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ [Accessed 7 Jun. 2019].

Stats StackExchange. (2019). How to choose the number of hidden layers and nodes in a feedforward neural network?. [online] Available at: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw [Accessed 6 Jun. 2019].

Stathakis, D. (2007). How many hidden layers and nodes?. [online] Dstath.users.uth.gr. Available at: http://dstath.users.uth.gr/papers/IJRS2009_Stathakis.pdf [Accessed 6 Jun. 2019].

Sanderson, G. (2017). What is backpropagation really doing? | Deep learning, chapter 3. [online] YouTube. Available at: https://www.youtube.com/watch?v=Ilg3gGewQ5U [Accessed 26 Jun. 2019].

Joshi, P. (2019). Understanding Xavier Initialization In Deep Neural Networks. [online] PERPETUAL ENIGMA. Available at:

https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/ [Accessed 7 Jul. 2019].

Brownlee, J. (2017). How to Get Reproducible Results with Keras. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/reproducible-results-neural-networks-keras/ [Accessed 8 Jun. 2019].

Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980 [Accessed 7 Jun. 2019].

Hinton, G., Vinyals, O. and Dean, J. (2015). Distilling the Knowledge in a Neural Network. [online] arXiv.org. Available at: https://arxiv.org/abs/1503.02531 [Accessed 17 Jul. 2019].

Heaton, J. (2017). The Number of Hidden Layers. [online] Heaton Research. Available at: https://www.heatonresearch.com/2017/06/01/hidden-layers.html [Accessed 9 Aug. 2019].

Simard, P., Steinkraus, D. and Platt, J. (2003). Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. [online] Cs.cmu.edu. Available at: https://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2016/pdfs/Simard.pdf [Accessed 20 Aug. 2019].

Dodge, S. and Karam, L. (2016). Understanding How Image Quality Affects Deep Neural Networks. [online] arXiv.org. Available at: https://arxiv.org/abs/1604.04004 [Accessed 16 Aug. 2019].

Maladkar K (2018). Overview Of Convolutional Neural Network In Image Classification [online] analyticsindiamag.com Available at: https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/

How Does the Number of Hidden Neurons Affect a Neural Network's Performance. (2016). Retrieved 31 August 2019, from http://www.chioka.in/how-does-the-number-of-hidden-neurons-affect-a-neural-networks-performance/

Oladipupo, T. (2010). Types of Machine Learning Algorithms. Retrieved 14 August 2019, from http://cdn.intechweb.org/pdfs/10694.pdf

Vallantin, W. (2018). Metrics to measure machine learning model performance. Retrieved 1 September 2019, from https://medium.com/@wilamelima/metrics-to-measure-machine-learning-model-performance-e8c963665476

Fehr, T. (2019). How We Sped Through 900 Pages of Cohen Documents in Under 10 Minutes. Retrieved 5 July 2019, from https://www.nytimes.com/2019/03/26/reader-center/times-documents-reporters-cohen.html?rref=collection%2Fsectioncollection%2Freader-center&action=click&contentCollection=reader-center&region=rank&module=package&version=highlights&contentPlacement=2&pgtype=sectionfront

All neural network model images were made using the following website:

http://alexlenail.me/NN-SVG/

Appendix

Raw Data generated can be found via this link:
https://pastebin.com/JXMHYNpn
Code written and used through experimental process:
https://pastebin.com/C6vN3RA2

Regular Programming VS Machine Learning

Before attempting to understand an artificial neural network, it is key to understand how regular programming works as opposed to machine learning. In regular programming, the programmer creates rules and gives in inputs. These inputs then have the rules applied to them by the computer and result in answers. For example, if the computer was given the rule “Multiply input by 3” and the input “2”, it would result in an output of “6”. On the other hand, machine learning makes it such that the programmer gives the computer multiple inputs and the answers to those inputs, then the computer tries to figure out the rules on its own. For example, the computer is given input “2”, it is also told that the output is “6”. It has to now try and figure out the rule(s). Which it calculates to be “Multiply input by 3”. However, this is a vast oversimplification. In reality, machine learning is used for problems where the rules would be too many to the point where programming rules would be overly inefficient.

Take the image processing case of object classification for the use of rock, paper and scissors, i.e. getting an image from the camera and figuring out which one of the three the hands is currently showing. In regular programming, the computer is told to look at each individual pixel’s[1] data and try applying some patterns to the image that are hardcoded (i.e. written by the programmer into the code), for example, look for 20 skin coloured pixels in a vertical line to find a finger. However, there are a few issues with this rule such as, how does a computer know what colour the skin is? What is defined as a vertical line (if the image is tilted, then the hand may not be pointing straight up, or what if the finger is slightly bent)? Coding for all these different situations like skin tone and how much the hand is tilted, the size of the hand requires many rules determined using trial and error. This is very inefficient as can be imagined. Machine learning, on the other hand, would be given a dataset full of images of hands and their corresponding shape (i.e. rock/paper/scissors) and it would try to create those rules on its own. The input would contain a variety of hand shapes, sizes, image qualities, skin tones, etc. The rules it generates would then be applied to new images it is given and then a prediction would be generated.

[1] An image is simply a series of numbers with a value in each colour channel stored in the form of a table. In the case of 3 colour channels, the stored information will be 3 different values each from 0-255 that indicate the brightness of the specific colour at that pixel in the form of red, green and blue. For example, a pixel at position 15 along the horizontal and 10 along the vertical could possibly be 204, 0, 255 to give the colour purple in that pixel.

[2] An example of this would be clustering, separating data into segments or ‘clusters’ based on properties of said data.

[3] A graph is from the mathematical field of graph theory.

[4] An example of this structure is seen in figure 2.1, where the hyperparameters are set as follows: the training data sent to the neural network will contain 8 epochs and each epoch will have a batch size of 10, resulting in a total of 8 batches per epoch. Typically the batch

sizes are powers of two (i.e. 32,64,128) in order to keep time taken for training down at the same time as keeping performance up.

[5] A parameter which has a value that is set before the learning process begins.