

IB Computer Science EE

Title: **Activation functions in convolutional neural networks.**

Research Question: **To what extent can certain activation functions produce more accurate predictions of handwritten digits in a convolutional neural network?**

Word Count: **3655**

Introduction	2
The neuroscience of image recognition	2
Structure of a CNN	3
1.The Convolutional layer	4
2.Activation Functions	6
3.The Pooling Layer	6
4.Classification Layer	7
5.Flattening	8
Structure of a multi-layered perceptron	8
1.How the MLP learns to recognize digits.	9
2.Backpropagation for MLP digit recognition	9
3.Gradient Descent for digit recognition	10
Summary of CNN functionality	10
Types of activation functions	11
1.No activation function	12
2.Hyperbolic tangent (Tanh)	13
3.Logistic activation function	14
4.Rectified linear activation function (RELU)	15
5.Leaky ReLU	16
Investigation	18
1.Methodology	18
2.Measured Parameters	18
3.Structure of program	18
4.Hypothesis	18
Conclusion	19
Evaluation and Limitations	21
Appendix	21
Bibliography	21

Introduction

Convolutional neural networks (CNN's) are computer programs that aim to replicate the brain's ability to recognize images and were created by D.H Hubel and T.N Wiesel. There are a variety of real life uses of CNN's ranging from medical image analysis, speech analysis, and recommendation systems. Within the CNN one of the many adjustable parameters is the activation function. The computer needs to recognize complex and intricate patterns, in order to recognize an image (*Cornelisse Daphne, 2018*). Activation functions are mathematical equations that are used for the CNN to learn these complicated patterns or in other words, non-linear relationships. While researching the variety of activation functions questions such as "What is the best activation function?" and "Do different images produce different accuracies?" crossed my mind. To answer these questions I decided to analyze different activation functions against the MNIST dataset of handwritten digits developed by Yann Lecun and came up with the research question "To what extent can certain activation functions produce more accurate predictions of handwritten digits in a convolutional neural network?". Accuracy will be defined through average accuracy (correct predictions out of 1000 tests), average loss (summations of errors) and training time, described in more detail later on.

The neuroscience of image recognition

To CNN is based on the human brain. It is therefore necessary that the basic neuroscience of how humans recognize images is understood, before understanding how computers can recognize images. While it is natural and easy for humans to recognize images, computers need to be trained on millions and millions of image before it is able to recognize a single image. This complex system evolved in human beings over 500 million years. Unlike human beings, computers can only see in terms of 1's and 0's (*4.10. Representing Images*). The figure below shows how a computer would recognize an image.

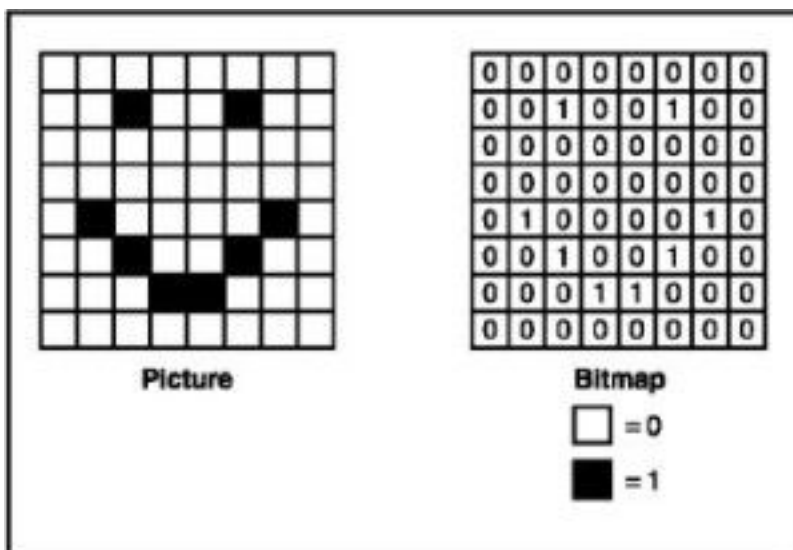


Figure 1.Example of an image to the pixel array (*mouldsm,2015*)

Biologically, when people recognize an image a neuron is fired in the brain. A section of the image (known as the receptive field) is what primarily triggers the neuron. The outermost layer of the brain (neocortex) stores information (and recognize images), in the grouping of neurons. These biological concepts were used first by a researcher named Fukushima in 1980 and later expanded on to CNN's by Yann Lecun in 1988 (*Cornelisse Daphne, 2018*). Now that the biological steps of image recognition are explained, the structure of a CNN can now be described.

Structure of a CNN

In his paper, Yann Lecun proposed the input images fed into the CNN to have three layers: height, width, and depth (shown below). The reason for the three layers is due to the nature of the image. The height and width of the picture require two layers, and the third layer is for the color of the image (some variation of the colors red, blue and green) (*Lightning Blade,2018*).

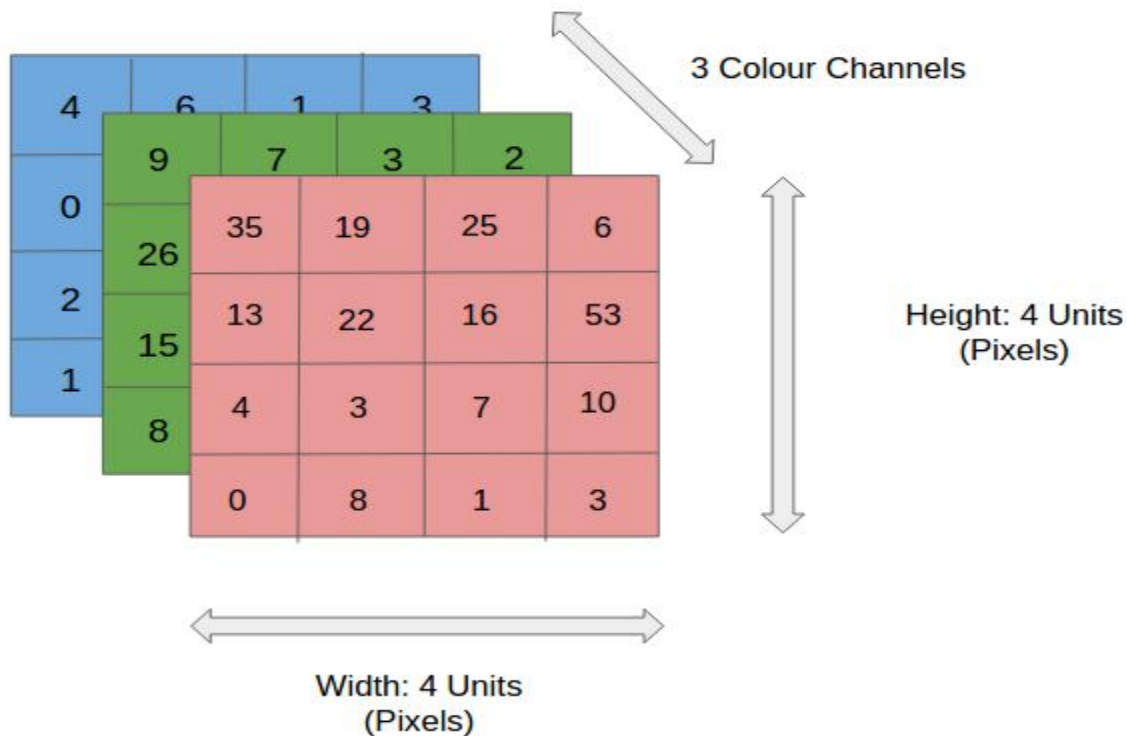


Figure 2. An example of a 4x4 image being fed into a CNN (Alexandr Honchar, 2017).

Convolutions and poolings are then repeated over and over on the image. This is so that the features of the image will be detected.

1. The Convolutional layer

Convolutions take place in the convolutional layer of the CNN. The word convolution in mathematics means combining two functions to create a third one. In the CNN, convolutions are used to produce a **feature map**, by using a **filter**. The filter is a set window or square that slides across the image (shown in figure 3). A mathematical operation is performed and then the output of that operation sent to the feature map, the filter then moves to the right this is known as a **stride**. Once the filter reaches the end of the first row, it moves down to the second row and repeats the process. The feature map is the computerized version of the biological receptive field. The stride will produce a feature map that has a smaller size, so the concept of **padding** (where a pixel value of zero is surrounded by a padding of zeros) is used to maintain the same size. Through the use of padding, the performance of the CNN is increased as the filter fits the input, not wasting any pixels. (Hien, Dang Ha, 2017)

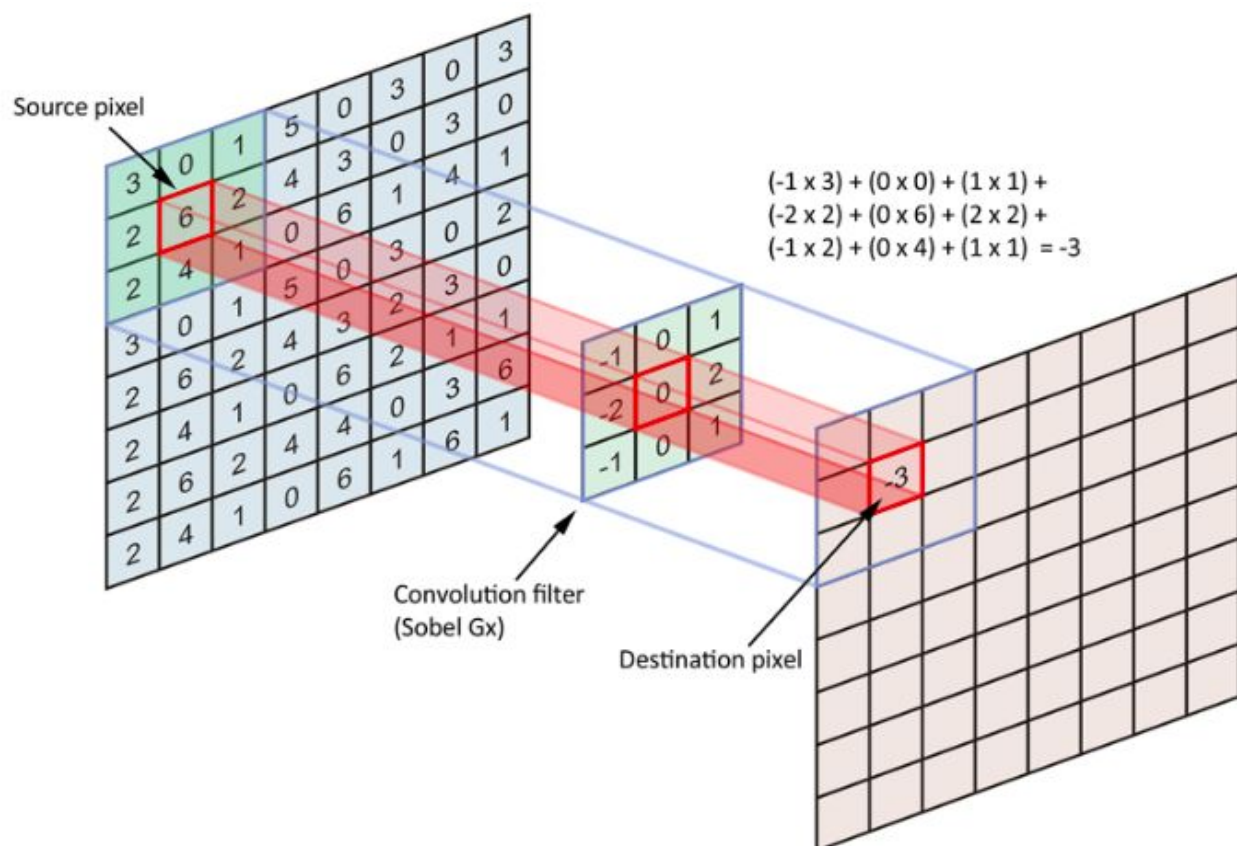


Figure 3. The destination pixel is on the feature map, the source pixel is on the input image (*Daphne Cornelisse, 2018*)

When a feature map of a different filter is used (for instance in figure 3 the filter used is 3x3), this produces a different feature map. This process repeats, producing several feature maps. The feature maps are stacked on top of each other (another mathematical operation is used to “add” all the layers together), and this stacked map what is outputted by the convolutional layer of the CNN.

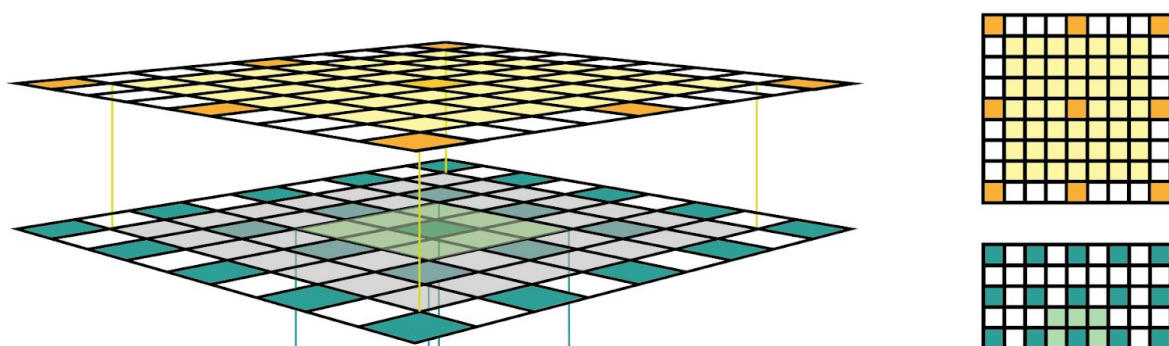


Figure 4. The lines represent mathematical operations from one map to another. (*Dang ha The Helian, 2017*).

2. Activation Functions

Activation functions are mathematical equations that are used to introduce non-linearity into the CNN. What is outputted by the convolutional layer is the stacked feature map. However, the mathematical operations that were used to calculate each map (such as the addition of pixels) are linear. This is a huge problem as computer image recognition cannot be modeled by a simple linear relationship. It is not as straightforward as recognizing the addition of pixels in each image. Image recognition is a more complex process that requires a more complex pattern to be found. Therefore by using activation functions the stacked feature map is represented as a complex pattern as opposed to a linear sum (*Sagar Sharma, 2017*). This complex pattern is modeled differently based on the activation function used. Some examples of activation functions are tanh, RELU and sigmoid but these will be explored in more depth, later on in the investigation portion of the EE.

3.The Pooling Layer

The purpose of the pooling layer is to reduce the size of the stacked feature map. This layer is almost always used as with a reduced stacked feature map, there are fewer parameters, computations, training time as well as a reduced risk of overfitting. Overfitting is a term used in statistics that refers to a data model being biased or heavily affected by a small amount of data. The most common way of using pooling is through the max pooling method (*Ardent Dertat, 2017*). This is when only the highest pixel value (within a certain window) of the feature map is taken (shown in figure.5 below). This way, the accuracy of the pixel value is kept and the new feature map is condensed.

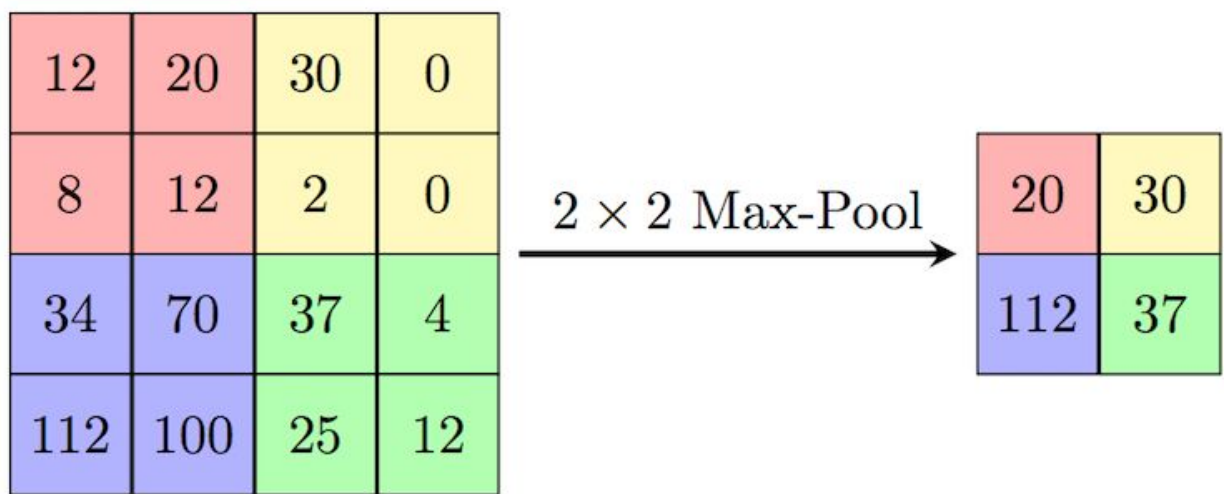


Figure 5. A two-dimensional representation of max pooling. The window is 2x2, showing the highest pixel value being taken and outputted in the new map. (*Max pool*)

4. Classification Layer

After both the convolutional and pooling layers of the CNN, the digits are now ready to be trained. The classification layer is where the CNN assigns a probability to the image being predicted by the CNN through the use of a multi-layered perceptron or MLP. For example, when training on handwritten digits by MNIST, the computer will determine the probability for each digit from 0-9 (shown in figure 6 below).





	<p><u>Prediction:</u> Digit 5 – 90% Digit 3 – 9% Digit 0 – 1%</p>		<p><u>Prediction:</u> Digit 5 – 57% Digit 3 – 38% Digit 8 – 5%</p>
	<p><u>Prediction:</u> Digit 3 – 50% Digit 5 – 49% Digit 0 – 1%</p>		<p><u>Prediction:</u> Digit 3 – 87% Digit 5 – 8% Digit 1 – 4% Digit 2 – 1%</p>

Figure 6. The CNN classification of digits (*kdnuggets,2016*)

5.Flattening

Classification is achieved through a fully connected layer also known as multi-layered perceptron (MLP). This layer will be described shortly, but it is important to note that it only accepts one-dimensional data. Recall that what is sent after the pooling layer is in three dimensions (length, width, colors). In order to

combat this problem, the three-dimensional stacked feature map is then **flattened** into a one-dimensional array. This is known as the flattening step. Now the one-dimensional array is ready to be put into an multi-layered perceptron so that digits can be classified.

Structure of a multi-layered perceptron

The structure of the multi-layered perceptron (MLP) is similar to figure 7 shown below. It is primarily composed of three parts the input, hidden and output layers (*Kang Nahua, 2017*). The input layers (shown in blue) is where the one-dimensional stacked feature map from the previous step is inputted. The hidden layers are shown in purple, note that each perceptron (or purple circle) is connected to all of the inputs (blue circles). In the picture, there is only one hidden layer but good neural networks (and CNN's) require more. Finally, the output layer is shown in orange.

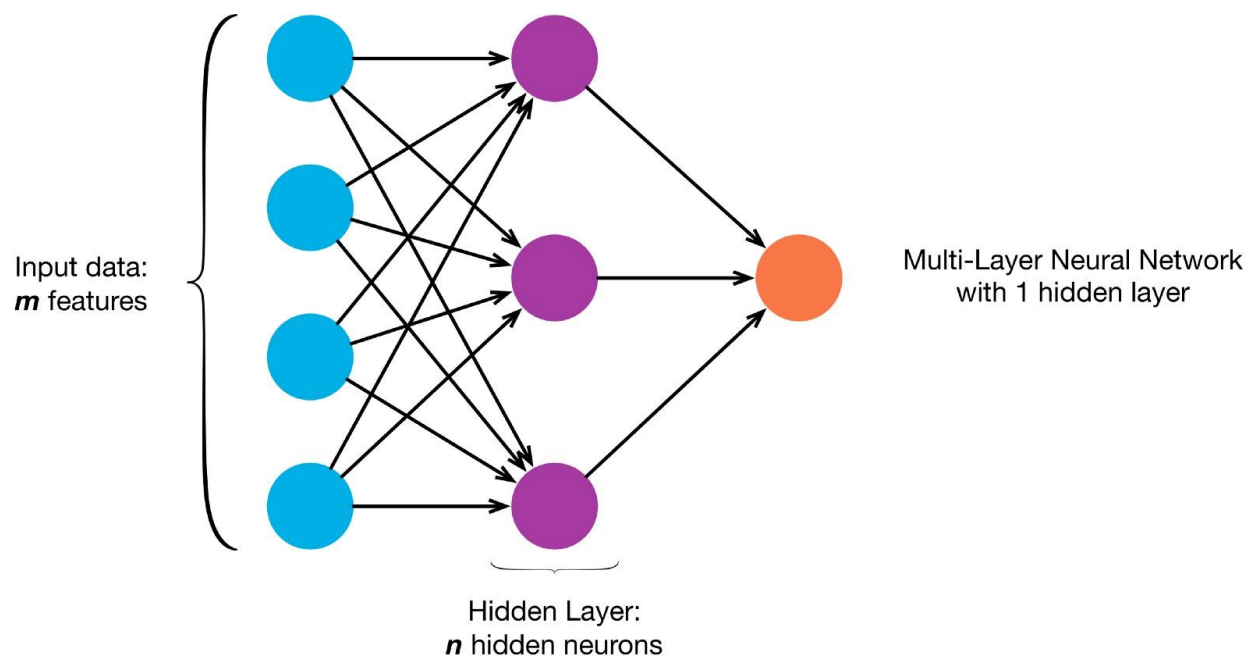


Figure 7 Structure of a MLP (*Nahua Kang, 2017*)

MLP's work by adjusting weights and biases with backpropagation until classification is sufficiently accurate. To understand this statement let us consider one perceptron (purple circle). Mathematically,

each perceptron in the hidden layer is $\sum x_i$ where x_i is each data inputted (or in other words the sum of each layer). However, instead of adding each input it is first multiplied by a weight w and the added by a bias b before being summed (Making each perceptron in the hidden layer equal to $\sum wx_i + b$). The bias is just a set number that measures how easy it is for a perceptron to reach a value of 1 or in biological terms how likely it is for a perceptron to fire. This will be useful later when training the MLP to recognize

digits. The concept of activation functions will now have to be introduced again, (because the data is not linear), making the equation $\sum \sigma(w x_i + b)$ for some activation function σ .

1.How the MLP learns to recognize digits.

All of the perceptrons sum up to the output layer where each handwritten digit is given a probability by the equation $\sum \sigma(w x_i + b)$. The highest probability is selected as the number the MLP chooses. Initially, the predicted numbers by the MLP will be inaccurate. This is where the backpropagation and gradient descent are introduced.

2.Backpropagation for MLP digit recognition

The MLP's predicted value is compared with the correct output so that an error function can be calculated. If we were to graph in 3 dimensions where x is the weight, y is the bias and z is the cost function. Every possible outcome of the cost function would be represented. This means that for certain weights and bias (x and y), at the local minimum of the graph, the cost function will be minimized, maximizing the classifier's accuracy. An example is shown below .

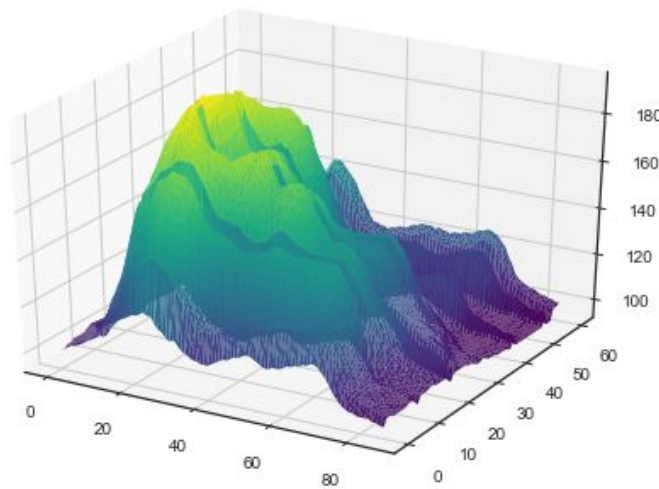


Figure 8. Example cost function graph
(Yan Holtz, 2017)

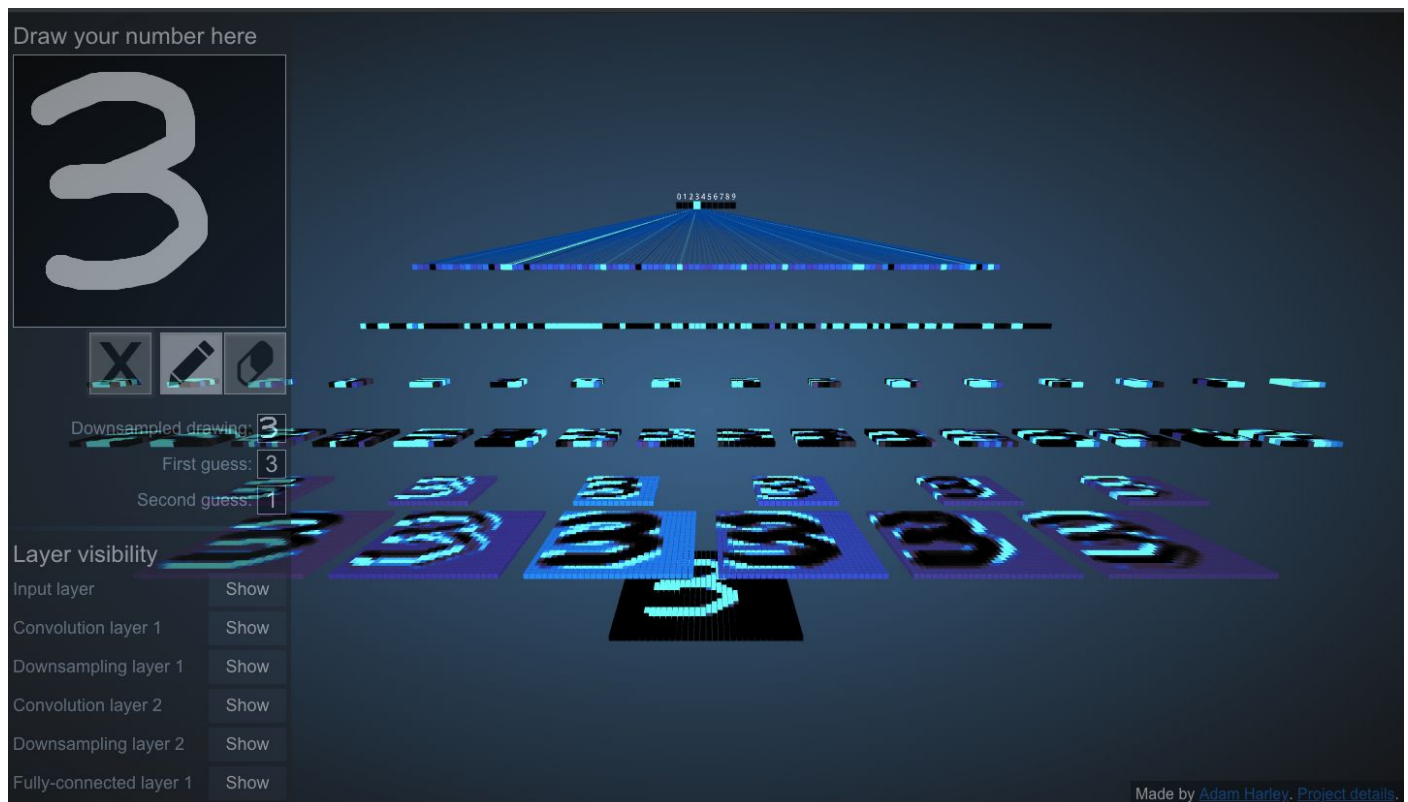
3.Gradient Descent for digit recognition

While the mathematics is too advanced and not necessary to this paper, it is important to recognize that Gradient Descent refers to an algorithm where the minimum of the cost function graph (which would give

the best weight and bias) are used. It uses the concepts of partial derivatives in order to reach the lowest cost value. Where each change in the x and y-direction (weight and bias) gets updated into the previous perceptron equation $\sum \sigma(w x_i + b)$. Ultimately, the adjustments of w and b values will yield the minimum cost, or in other words, the MLP (and therefore the entire CNN) will be trained to recognize handwritten digits accurately.

Summary of CNN functionality

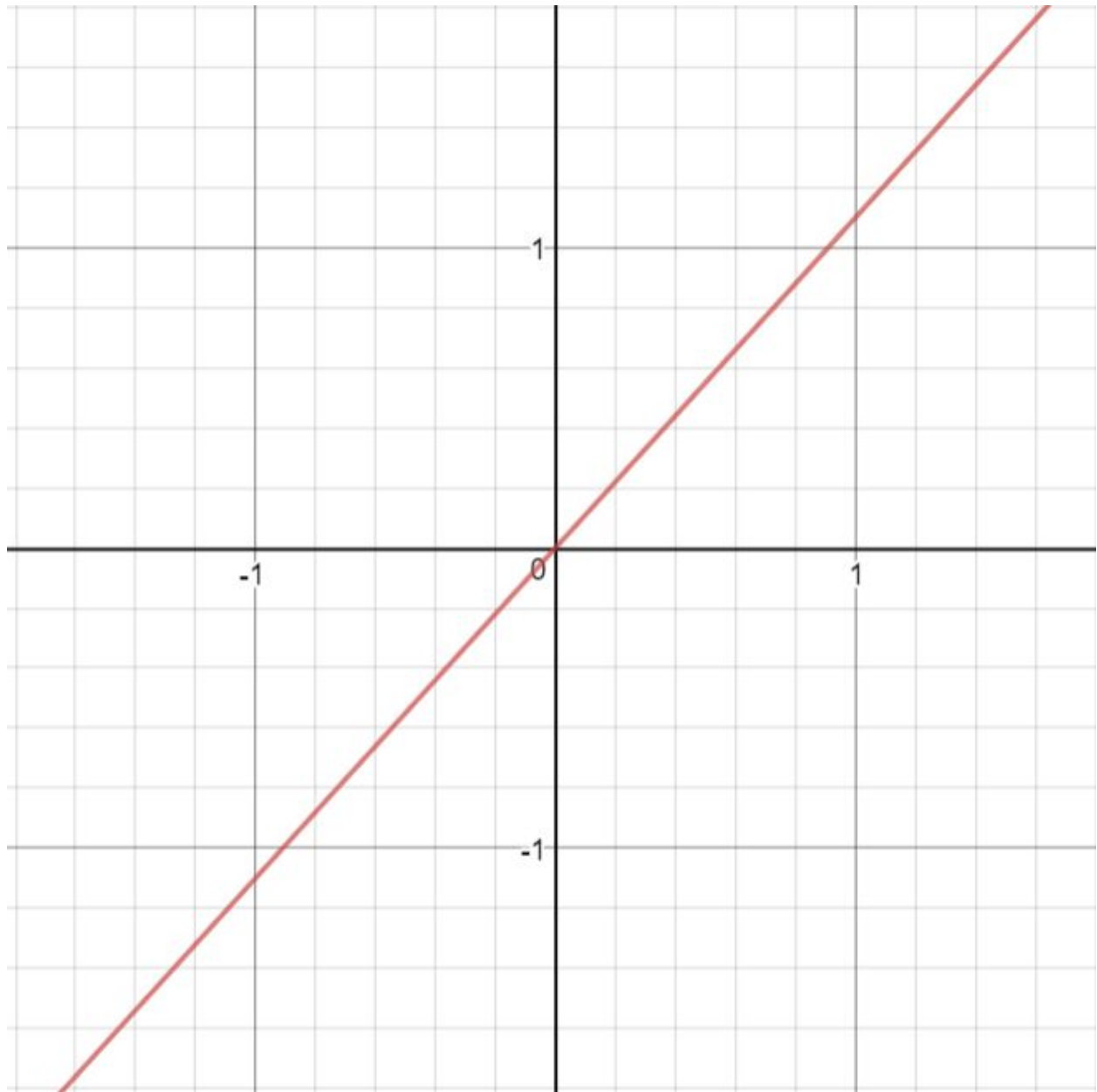
The entire process of a CNN can be visualized by the interactive visualization for neural networks project by Adam Harvey . It represents a CNN with two convolutional layers (2nd and 3rd layer), that produces 6 feature maps each. They are then put through two pooling layers (4th and 5th layer), the digit is now unrecognizable and reduced in size. Flattening (6th layer or first line) turns the entire set of data into a one-dimensional array. Finally, the training from the MLP in order to reach a prediction (top-most scale) is shown at the very top. As shown in the picture the numbers 3 is highlighted in blue (on the scale) and was the first guess of the CNN.



(Harley, Adam)

Types of activation functions

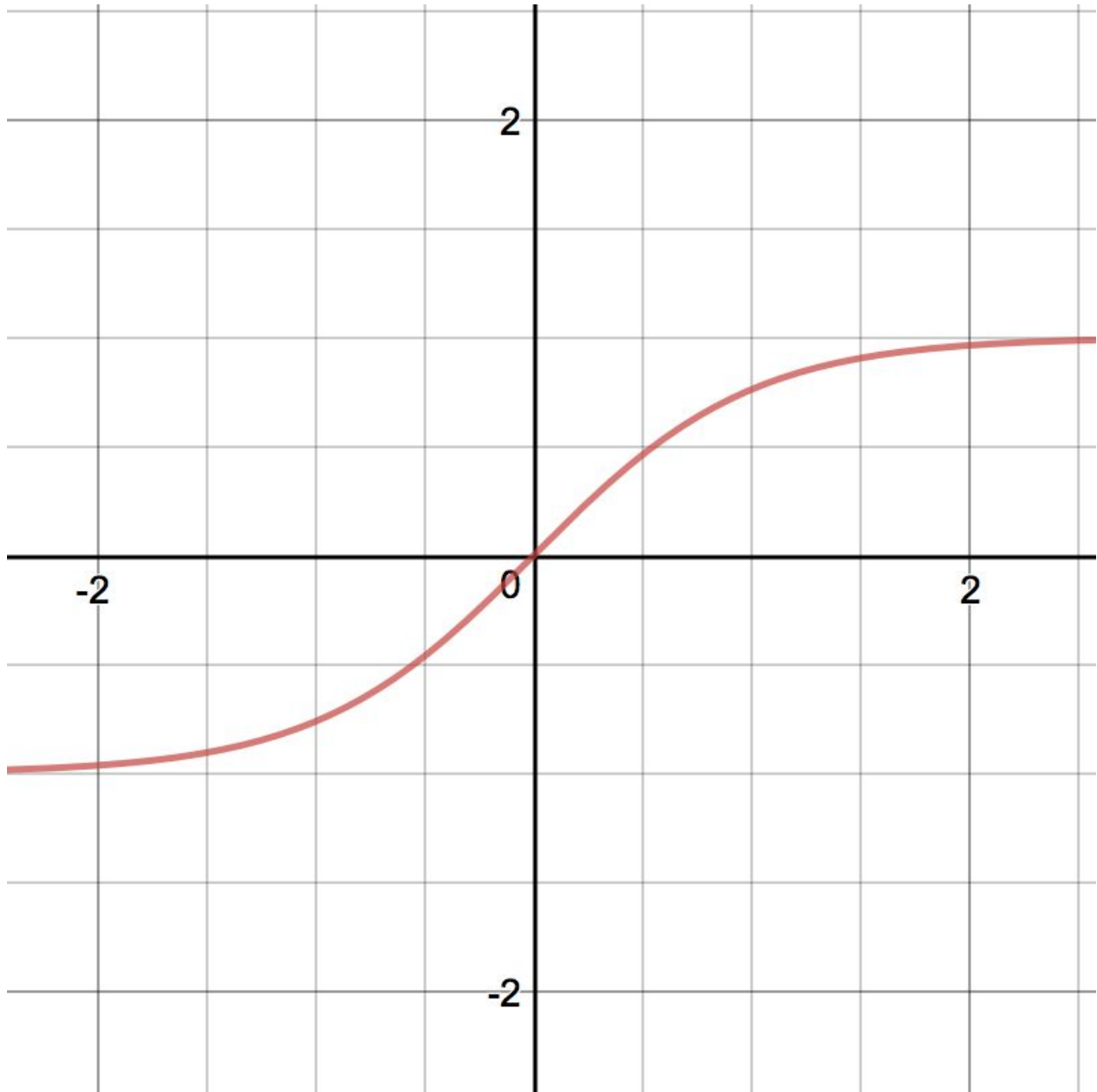
1.No activation function



(Brenden Fortuner, 2017)

If no activation function was used it would follow some variation of the linear equation $y=x$ (shown above). While it does include all real numbers in its range, the fact that it is a simple equation may hinder the classification of digits. This is because the relationship between pixels in an image and what digit it represents is not that straightforward. By using a linear relationship it suggests to the CNN that the more positive the number, the more likely it is to be a digit. This would make the structure of the MLP redundant as the same effect could be achieved using a single line of nodes as opposed to the MLP structure. (Sagar Sharma, 2017)

2. Hyperbolic tangent (Tanh):



(Brenden Fortuner, 2017)

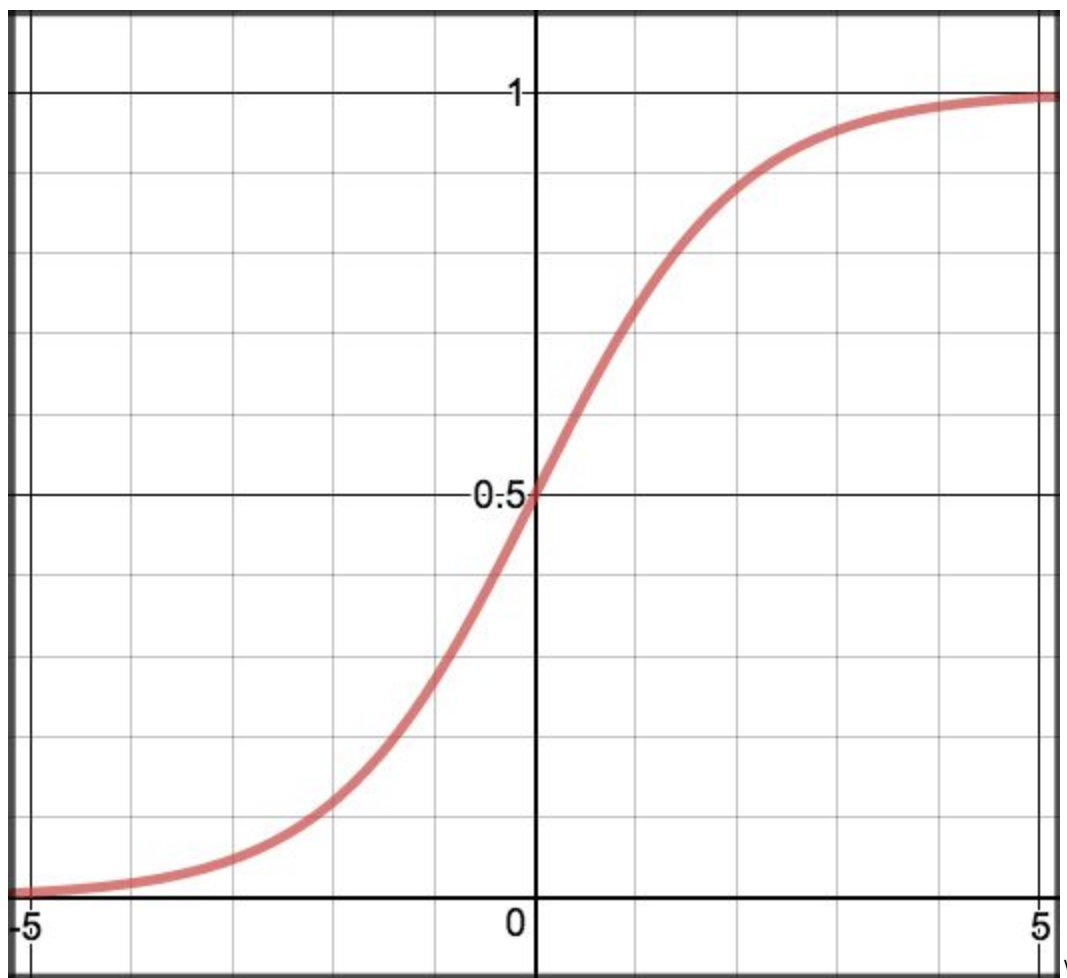
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

This activation function ranges from -1 to 1. It follows the general equation

The function has the benefit of including negative numbers, this means that if a prediction by the MLP is inaccurate, it will strongly influence the updating of weights and biases when the partial derivatives are calculated with gradient descent. It is also a sigmoid, (shaped like the letter “s”), this allows for a range of numbers where the higher (or lower) a number is, the more likely (or less likely) it will be able to

influence the updating of weights and biases in the CNN, to reach a higher accuracy for classifying digits.(Sagar Sharma, 2017)

3.Logistic activation function

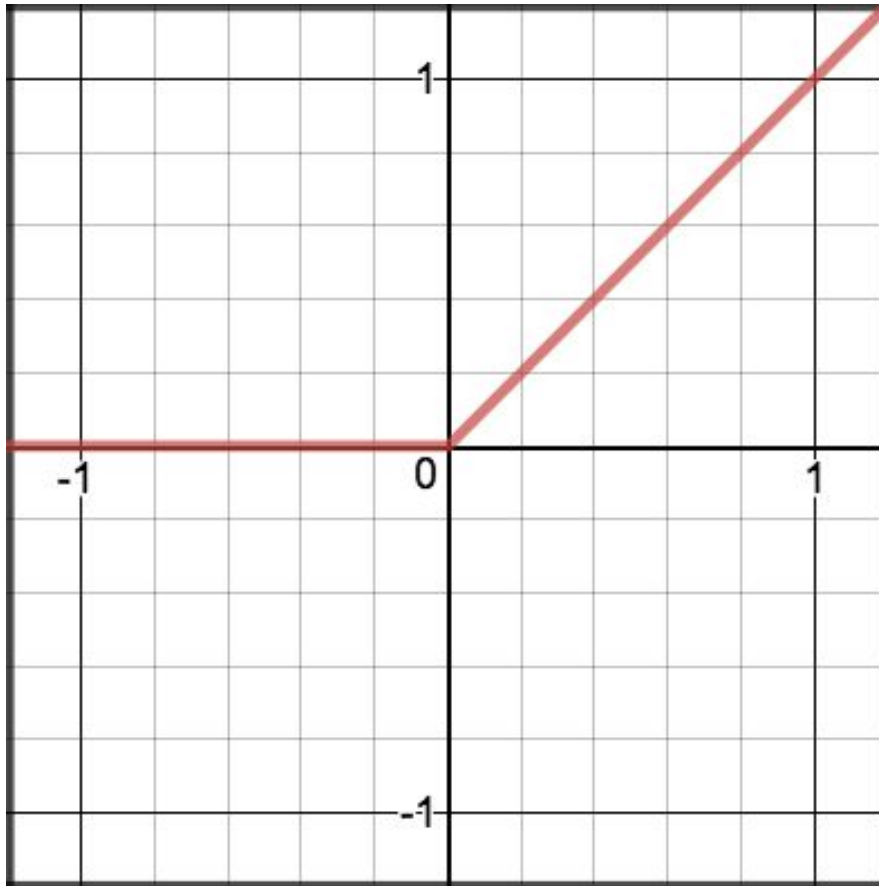


(Brenden Fortuner, 2017)

The equation for this activation function is $\frac{1}{1 + e^{-x}}$, it is similar to the tanh function however it does not produce negative numbers. This logistic activation function ranges from all positive numbers from zero to one.(Sagar Sharma, 2017) While there is still the benefit of being shaped like a sigmoid, (so that higher

and lower values have a proportional influence) the effect of only using positive numbers may reduce the influence of updating weights and biases. However, there is also the risk of getting “stuck” while training the CNN. This is because if the value (or in this case feature map) contains negative numbers, then the output of the function will be close to zero. Because this data is sent to a MLP that uses feed-forward processing, important data could be lost, when negative numbers are assumed to be 0 . This is known as the vanishing gradient problem.(Nielsen, Michael 1970).

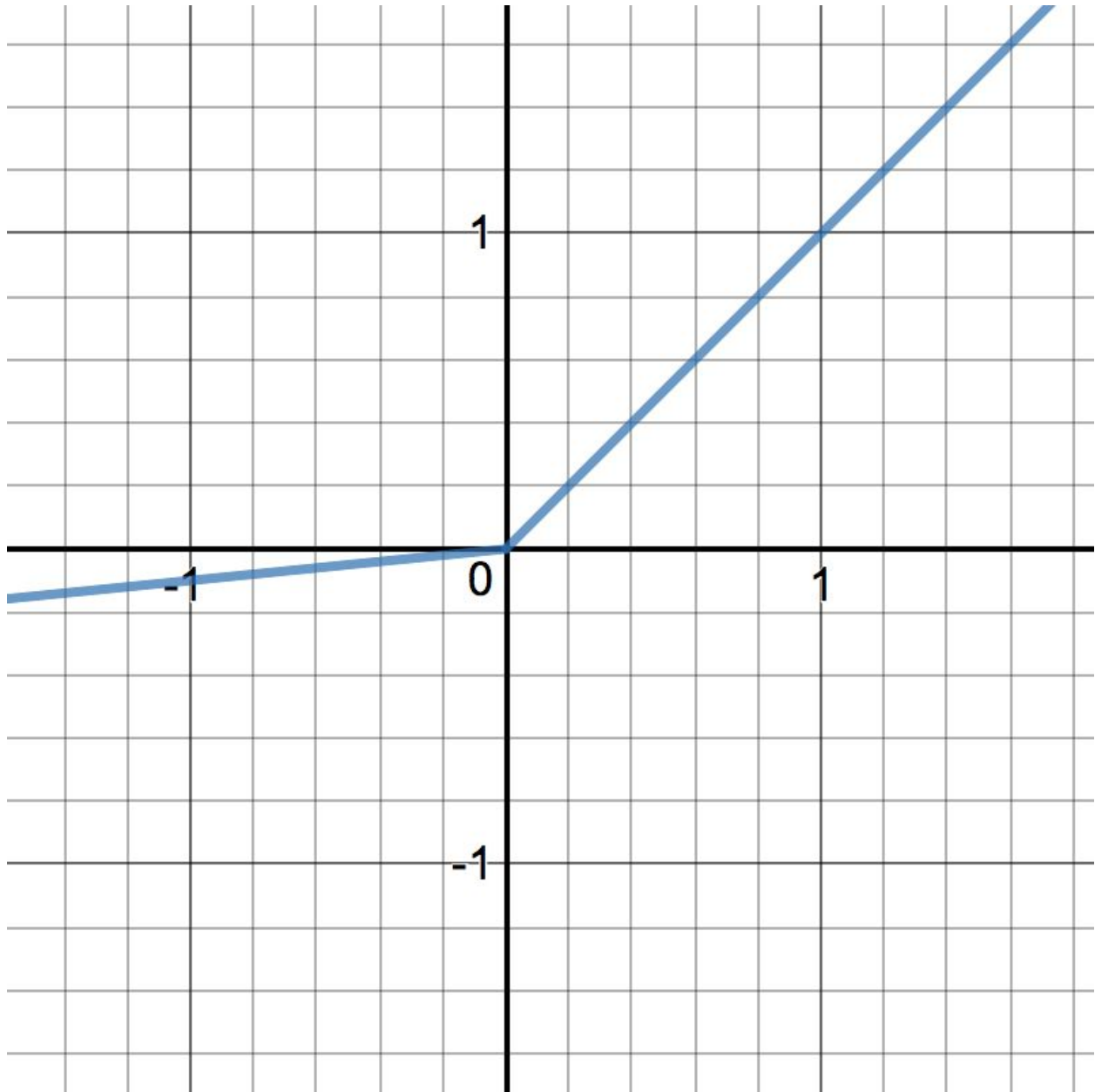
4.Rectified linear activation function (ReLU)



(Brenden Fortuner,2017)

As shown in the graph ReLU or rectified linear activation functions assume all negative values to be zero. This could potentially have the same problem as logistic activation functions where negative outputs (by the feature map in the convolution step) are ignored (vanishing gradient). Furthermore, the relu graph is not a sigmoid. It is a piecewise equation where values below and equal to zero are zero and values greater than 0 increase linearly. This is represented in its equation $\max(0, x)$. One potential benefit is the range of this function (from 0 to infinity), by not limiting the upper values inputted into the function there could potentially be stronger influences with positive numbers when updating the weight and bias of the CNN, as before the upper limit was capped at 1.(Sagar Sharma, 2017)

5. Leaky ReLU

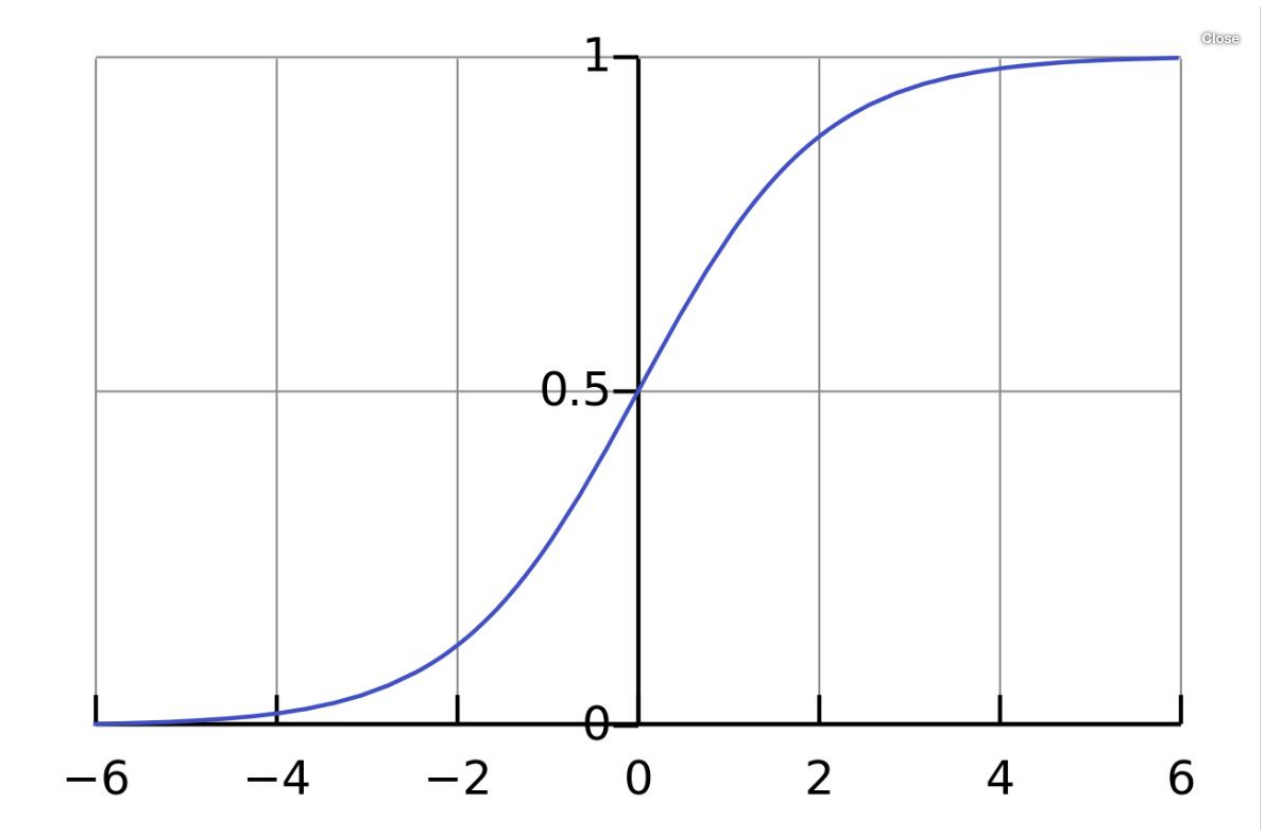


(Brenden Fortuner, 2017)

The Leaky ReLU activation function aims to get rid of the vanishing gradient problem by allowing negative numbers. Now, when a negative number is inputted (from the stacked feature map of the image) the values are not set to zero. This allows for a new range of outputs (from negative infinity to positive infinity or all real numbers). Whenever a negative value is inputted into the function it follows the equation $-0.01x$. Whenever a different coefficient is used (for example $-2x$ or $-3x$) it is known as randomized relu. All positive numbers still follow the output of x . For this EE, only the leaky relu (when

the coefficient is 0.01) will be explored as there are too many possibilities for coefficients if randomized ReLU was used.(*Sagar Sharma, 2017*)

6.Softmax activation function



(*Yashu Seth, 2018*)

The softmax function is what will be used for the MLP section of the CNN. This is because the MLP's purpose is to give probabilities to the predicted value of each digit. These probabilities range from 0 to 1 (the range of the function shown above) where a higher number corresponds to a higher likelihood, of the predicted number by the CNN. It is modelled by the equation The softmax activation function has been proven to be the best activation function to use, in the MLP.(*Sagar Sharma, 2017*)

Investigation

The activation functions will be analyzed by modifying the code provided by user treskai from Github. Activation functions are used in two places. The first when producing the feature maps and the second is in the MLP when training to recognize handwritten digits.

1.Methodology

For this investigation, the MLP's activation will be set as a softmax (for reasons described under the softmax section below) but the feature map activation function portion of the CNN will be altered and analyzed. The training will be done on 6000 images from the MNIST (Modified National Institute of Standards and Technology) dataset for handwritten digits and the testing will be done on 1000 MNIST images (for deeper analysis). These images The reason for using 1000 images is because it is a sufficient number of trials for an accurate result, and can be tested in a feasible amount of time.

2.Measured Parameters

To answer the question “To what extent can certain activation functions produce more accurate predictions of handwritten digits in a convolutional neural network?”, the program was altered to output average accuracy (correct predictions out of 1000), average loss (summations of errors in backpropagation step) and training time. These three parameters will be used to determine how effective each activation function is.

3.Structure of program

The overall structure of the CNN is similar to figure 9 and has two convolutional layers, one pooling layer (that uses max pooling with a window size of 2x2 and stride of one), one flattening layer and one MLP. This way, the question “To what extent can activation functions be used in a convolutional neural network” is explored.

4.Hypothesis

I expect all of the activation functions to have above 95% accuracy. This is because while different, all of them encompass some form of non-linearity. I also expect the training time for all programs to be around 2000 seconds, but the Leaky ReLu and tanh functions to have the highest training time due to weights being stuck by the vanishing gradient problem.

MNIST dataset testing results:

The data when tested against 1000 (handwritten) MNIST images (see Appendix A for sample).

	Average Accuracy (%)	Average loss	Training time (seconds)
Activation function			
None	9.8%	7.9381	2757
ReLU	99.2	0.0251	2272
Leaky ReLU	99.04	0.0292244202007274	2626.6
Logistic	11.35	2.3011261165618895	2596.8
Tanh	98.89	0.04653436026619177	2947.6

Conclusion

When analyzed against 1000 images the **ReLU** had the best results with an average accuracy of 99.2% (almost perfect), the lowest cost (0.0251) and the lowest training time (2272 seconds). This is likely because the ReLU function has no upper limit, making positive outputs have a stronger influence to the feature map. This performance seems to suggest that excluding negative numbers in the creation of the feature map does not hinder the accuracy when inputted into the MLP, when recognizing handwritten digits.

The **Leaky ReLU** also performed well with an accuracy of approximately 99% (while it is still less than the ReLU by 0.16%). Unexpectedly, by including negative numbers while there is a high accuracy, the ReLU performed better. The inclusion of negative values in the feature map (unlike ReLU where negative numbers are set to zero) did not appear to hinder accuracy. Furthermore, the high performance may have also been due to the fact that there was no risk of the vanishing gradient problem as there was no limit to the range of the Leaky ReLU. It is perhaps because of the inclusion of negative numbers that the leaky ReLU has a higher training time when compared against the ReLU. However, the difference between Leaky ReLU (99.04%) and ReLU (99.2%) could be considered negligible as they are both almost 100% accurate. The reason for their differences could be because of the limitations of the investigation described after the conclusion.

Another accurate activation function was the **hyperbolic tangent**, with an accuracy of 98.89% and a training time of 2947 seconds. This was surprising to me as I thought by limiting the output from negative

one to one, the vanishing gradient problem would cause inaccurate data when creating the feature maps to be used in the MLP. Additionally, while tanh activation functions have the risk of getting stuck at an accuracy (due to the functions cap at negative one and one) this did not appear to greatly hinder the accuracy, but may have contributed to the getting highest training time as well as a higher loss value when compared to the ReLU.

I was also surprised to see how poorly the **logistic activation** performed. It had an accuracy of around 11% and a high loss value (≈ 2.3). The tanh function seemed to suggest that capping the range does not impact accuracy, but when capped from zero to one the logistic function performed poorly. This may be because unlike the tanh function it does not accept negative values. Another possible reason for the low accuracy could be because the vanishing gradient problem influenced the data. While the tanh function did not seem to be affected by this risk, the ranges were capped to zero and one in both negative and positive values in the feature map with high magnitudes may have been rounded to one or zero losing accuracy when passed into the MLP greatly hindering the data. (*dustin stansbury, The Clever Machine*)

As expected when **no activation** functions were used on the feature map, there was a lowest accuracy (9.8%), highest loss (7.9381) and high training time (2757 seconds). This suggests that complex patterns such as digit recognition requires non-linear relationships in order to be effective. Interestingly, it does not have the highest training time, but is still the second highest. The high loss value suggests that when the feature map data was backpropagation to update weights and biases in the MLP there was consistent error, due to the simple linear relationship.

As shown by the reasoning above and data in MNIST testing, activation functions are useful to a large extent in a CNN. It is not to a great extent as, while most activation functions have almost perfect accuracy and little difference in training time and loss, some (such as logistic regression) are not suitable as they produce approximately the same accuracy as no activation function.

Evaluation and Limitations

One limitation of this investigation is that there was only one cycle and one average value found. While there were 1000 trials for each activation function, it is entirely likely that in another 1000 trial cycle the Leaky ReLU would have performed better than the ReLU. Next time it would be better to run 1000 trials 10 times and find the average of those average values (for accuracy percentage, loss and training time) to increase accuracy.

Furthermore, only the average was shown in the table. While this is feasible (because it would not be realistic to include $1000 * 5$ data values), there may have been some trends within the process of training and testing that could have been used for deeper analysis. While I tried to graph all data points, the coding process was too complex for me, but if done trends in training or testing can be shown for different activations.

Another limitation is that the MLP's activation function was set as softmax and only the feature

map activation function was changed. Also the stride step, pooling method and window size were all kept constant. While time efficient, This produces limited set of data as another combination would have produced different results. If given more time this limitation could have been fixed if every possible combination of activation functions and pooling parameters were done.

Finally, only five activation functions were used to answer the question “To what extent can certain activation functions produce more accurate predictions of handwritten digits in a convolutional neural network”. If done again, even more activation functions would be used, analyzed and explained to better answer this question.

Appendix A: Sample MNIST data



A sample of a portion of the MNIST data, used for testing and training (*Hariton Costin*).

Bibliography:

1. Cornelisse, Daphne. "An Intuitive Guide to Convolutional Neural Networks." FreeCodeCamp.org, FreeCodeCamp.org, 24 Apr. 2018, medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.
2. medium.com/@alexrachnog/neural-networks-for-algorithmic-trading-2-1-multivariate-time-series-ab016ce70f57.
3. Dertat, Arden. "Applied Deep Learning - Part 4: Convolutional Neural Networks." Towards Data Science, Towards Data Science, 8 Nov. 2017, towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.
4. Blade, Lightning. "Demystifying Convolutional Neural Networks – Lightning Blade – Medium." *Medium.com*, Medium, 2 Sept. 2018, medium.com/@eternalzerodayx/demystifying-convolutional-neural-networks-ca17bdc75559.
5. Hien, Dang Ha. "A Guide to Receptive Field Arithmetic for Convolutional Neural Networks." *Medium.com*, Medium, 5 Apr. 2017, medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807.
6. Kang, Nahua. "Multi-Layer Neural Networks with Sigmoid Function- Deep Learning for Rookies (2)." *Towards Data Science*, Towards Data Science, 27 June 2017, towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.
7. Harley, Adam. *3D Visualization of a Convolutional Neural Network*, scs.ryerson.ca/~aharley/vis/conv/.
8. Nielsen, Micheal, and Michael A. "Neural Networks and Deep Learning." *Neural Networks and Deep Learning*, Determination Press, 1 Jan. 1970, neuralnetworksanddeeplearning.com/chap5.html.
9. Honchar, Alexandr. "Medium." *Medium*, 6 June 2017, cdn-images-1.medium.com/max/800/0*2mE8rtuaYCMYyXo.png.
10. 1395550283894582, Sagar Sharma. "Activation Functions: Neural Networks – Towards Data Science." *Towards Data Science*, Towards Data Science, 6 Sept. 2017, towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.
11. *Handwritten MNIST*. *kdnuggets*, www.kdnuggets.com/2016/03/training-computer-recognize-handwriting.html.
12. "Jed-AI Blog." *JedAI - 7 Simple Steps To Visualize And Animate The Gradient Descent Algorithm*, jed-ai.github.io/py1_gd_animation/.
13. *Pixel image*. *mouldsm*, 21 Oct. 2015, mitchellkscscomputing.wordpress.com/2015/10/21/how-bitmap-images-are-represented-in-binary/.
14. 21/how-bitmap-images-are-represented-in-binary/.
15. Holtz, Yan. #371 *Surface plot*. *The python graph gallery*, Yan Holtz, python-graph-gallery.com/371-surface-plot/.
16. Nahua Kang. *Right: Perceptron with Hidden Layer*. *Medium*, 27 June 2017, towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.
17. *The filter slides over the input and performs its output on the new layer*. *free code camp*, Daphne Cornelisse, 24 Apr. 2018, medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.
18. Costin, Harrton. *Sample of the MNIST dataset of handwritten digits*. *Research Gate*, www.researchgate.net/figure/.
19. *Sigmoid Activation*. *yashuseth.blog*, YASHU SETH, 11 Feb. 2018, yashuseth.blog/2018/02/11/which-activation-function-to-use-in-neural-networks/.
20. *LeakyReLU. ML_cheatsheet*, Brenden Fortuner, ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html. Accessed 2017.

21. *ReLU. ML_cheatsheet*, Brenden Fortuner, ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html. Accessed 2017.
22. *Sigmoid. ML_cheatsheet*, Brenden Fortuner, ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html. Accessed 2017.
23. *Tanh. ML_cheatsheet*, Brenden Fortuner, ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html. Accessed 2017.
24. *Linear. ML_cheatsheet*, Brenden Fortuner, ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html. Accessed 2017.
25. *Max pool. Computer Science wiki*, 27 Feb. 2018, computersciencewiki.org/index.php/Max-pooling/_/_Pooling.

