Deep Learning

David Lekei

7700601

COMP3190 Research Project

April 5th, 2021

Abstract

The human brain is quite possibly the most complex machine to ever exist. It is capable of processing incredible amounts of data in amazingly short times. Computer scientists have often attempted to model algorithms after the human brain to take advantage of that processing power and speed. This has resulted in what is known as artificial Neural Networks.

Introduction

Neural Networks are made up of nodes, referred to as neurons. These neurons are constructed into layers, with one layer for input, one layer for output, and all other layers in between are referred to as hidden. To simulate the brain, all neurons in one layer are connected to all the neurons in the next layer. The neurons communicate by sending signals to the next layer.

In order to further model the human brain, many layers can be added between the input and output layers. The result of this is referred to as Deep Learning. Classic machine learning uses, at most, one hidden layer between the input and output layers. Deep learning utilizes two or more hidden layers, thus “deep” refers to the idea that the amount of hidden layers is ‘deep’. Increasing the number of layers increases the amount of processing done on the input, allowing the network to provide more specific and intelligent output.

Overview of the Mathematics Behind Deep Learning

Neural Networks consisting of two or more hidden layers create a Deep Learning model. To understand Deep Learning, we need to understand the math that powers Neural Networks. The first layer in the network is referred to as the Input layer. This layer consists of n neurons, where n is the number of data points provided. For example, a 28x28 pixel image would be converted into a two-dimensional array of pixel values, and then flattened into a one-dimensional array, consisting of 784 (28 times 28) values. Thus, for this data, the Input layer would consist of 784 neurons.

After the input layer, the Deep Neural Network contains two or more hidden layers. Each neuron in the previous layer is connected to each and every neuron in the current layer. Using our 784 input neuron example, let's say the second layer consisted of 20 neurons. This would result in 784x2=15680 connections between the two layers.

Each of these connections is weighted. As the network gets trained, these weights are adjusted. Connections that the network doesn't deem important are weighted lightly, and the connections that are highly valued are weighted heavily. These weight values are what powers the learning behind the network. Each neuron calculates the weighted sum of all of it's inputs.

To calculate a neurons value, z, is a simple linear algebra problem. For n inputs to a neuron, we have a vector of n values, V, and a vector of n weights, W. To calculate z, we do V x Z. This gives us:

$$z= w_1z_1 + w_2z_2 + ... + w_nz_n$$

This calculation is done for every neuron in the layer. The neuron will then output a value based on it's activation function. Activation functions will be discussed later, but basically these functions determine what value the neuron outputs, based on the value of the weighted sum, z.

This process is repeated throughout all the layers of the Deep Neural Network, until we reach the output layer. This layer is made up of m neurons, where m is the number of possible answers. For example, if the DNN is used for detecting handwritten digits, m would equal 9, since the possible outputs are the digits 0-9.

The output layer decides on what to output, typically based on a Sigmoid function. Sigmoid functions output a value between 0 and 1, where 1 represents 100% certainty. These types of functions are chosen so that we can represent the entire output vector as percentages, and choose which answer the model is most confident in (ie. The highest percentage).

The most important building block to make Deep Neural Networks possible is the concept of back-propagation. When training a model, we feed training data along with the correct label, or answer. This way, the model knows what the expected output is. In order to tweak the weights of the connections in the network, the neurons must know how much to change the weights by. The only way to do this is to know how far off from the correct answer the model was.

This is where back-propagation comes in. The model will use a loss function to calculate how far off the actual output was from the desired output. This loss function typically uses Gradient Descent to determine a minimum in the loss function. Once that value is determined, it gets sent back through the network, and is used by the neurons to adjust their connection weights accordingly. [Shrestha and Mahmood, 2019]

Finally, the last piece needed is the activation function. This function is what determines the value that a neuron outputs to the next layer. These functions can be linear or non-linear. The most commonly used function is the Rectified Linear Unit (ReLU) function, in which any negative inputs are mapped to 0, and all positive inputs are mapped to themselves (for example, -3 -> 0, 3 -> 3).

Training

A Deep Neural Network creates a model based on training data. There are two high-level types of training: supervised and unsupervised training. Supervised training is the type of training we've already discussed previously. Supervised training is when we provide the model with labelled data and the model will correct itself based those labels.

Unsupervised training is much trickier but can produce much more interesting results. As the name suggests, unsupervised training is when the model is fed unlabelled data and it makes it's own determinations of the data, without being told what it is.

Supervised training algorithms typically make use of regression and classification, such as random forest and classification trees. Unsupervised algorithms usually involve clustering and association. These algorithms include K-means and Hierarchical clustering.

At the end of the day, what truly makes the biggest difference in the Deep Neural Network is the amount of data it gets fed. Whether the training is supervised or unsupervised, if there is not enough training data being fed into the network, it will not be able to make any confident decisions. [Shrestha and Mahmood, 2019]

Deep Learning Models

In this section we will cover the many types of Deep Learning models that exist. Each model is suited for a specific purpose, but no one model is considered perfect.

Convolutional Neural Networks

These neural networks are designed in such a way that they are most effective when used on image data. As discussed above, most neural networks consist of neurons that are fully connected. That is, every neuron in one layer is connected to every neuron in the next layer. A convolutional neural network instead lowers the number of connections by only connecting neurons in a restricted region. The design of these networks was inspired by the visual cortex in our eyes being restricted to the visual field.

The architecture of convolutional neural networks consist of an input layer, many hidden layers, and an output layer. Within the hidden layers, often there are layers with specific functions, such as pooling layers and receptive fields. Pooling layers reduce the dimensions of the data, to further reduce the complexity of the network. The receptive field is the layer that limits the input from the previous layer to only a restricted area. [Saad et al, 2017]

These networks are especially effective on image analysis due to the fact that in many images there is a lot of data that is not necessary. For example, facial recognition only cares about the face, but rarely will an image contain just a persons face. There is a lot of noise around the face that the network simply doesn't care about. Using a convolutional neural network would give better results than a normal network. [Shrestha and Mahmood, 2019]

Long Short-Term Memory Networks

These types of networks are most useful where data is presented in a time series, such as speech recognition. In a traditional deep neural network, loss values are back-propagated through the network. However, over time, these loss values will tend to zero. Long short-term memory networks attempt to solve this problem by storing the loss value at each neuron, so when a value that is considered too low is back-propagated, it does not use it. [Hochreiter and Schmidhuber, 1997]

Recurrent Neural Network

These neural networks come in many different variants, but the main idea behind them is that the connections between nodes for a directed graph *[L.R. Medsker, L.C. Jain, 2001].* This graph can be either cyclic or acyclic. Cyclic graphs allow the network to use a neurons current or future outputs to influence and change itself. The figure below illustrates this. The graph on the left shows a Recurrent Neural Network where a neurons output is being fed back into itself. Compared to a standard feedforward network on the right, where all output is directed forward. These networks are typically used in natural language processing due to their ability to process input of any length.

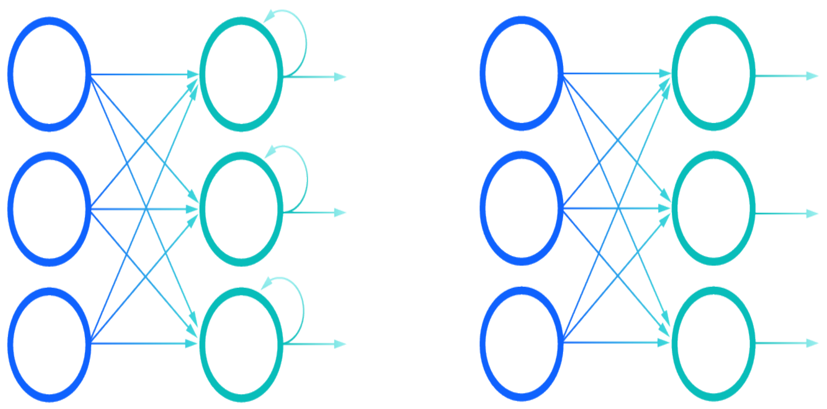


Figure 1: Left: Recurrent Neural Network with cyclic output. Right: Standard feed-forward neural network. [https://www.ibm.com/cloud/learn/recurrent-neural-networks]

Generative Adversarial Networks

This type of neural network relies on two types of data: real data and generated data. Generated data is constructed by a generator, which is an algorithm that is designed to generate plausible data. That is, the data should at least somewhat resemble the shape, size, color, etc of the real data. The generated data and real data are then both fed into a discriminator, which learns to tell the difference between the real and generated data.

These two components, the generator and the discriminator, are essentially pitted against each other. When the generator creates data that the discriminator deems implausible, it punishes the generator. In turn, this pushes the generator to creating data that will pass the discriminator's tests. Eventually, training reaches a point where the discriminator classifies fake data as real, meaning the generator has done it's job. [Goodfellow et al, 2014]

Drawbacks

Deep learning has resulted in numerous advances in technology in recent years, despite all the upsides, there are still some negative aspects that remain.

1) Amount of data required

In order for a deep neural network to output meaningful results, it must be trained on a large enough dataset. For some domains, there exists plenty of data. However, for more niche and narrow problem spaces, obtaining an adequate amount of data is not easy, or in some cases, possible. For example, when Google trained a neural network to determine the location of an image, they used a dataset containing 126 million images. That amount of data simply doesn't exist for many types of problems.

2) Computationally expensive

Assuming a network was provided with an adequate amount of data, processing that data is still expensive. The algorithms used have been improved to be more efficient, but at the end of the day, if you're processing millions upon millions of data points, it's still going to take computing time. Unfortunately, computing time means money, and there may not be enough funding to acquire that time.

3) Confidence and false positives

While many deep learning networks can reach nearly 100% accuracy, no model is perfect. Unfortunately, in the real world, there are many problem domains that simply cannot accept less than 100% accuracy. Tying into that, on the opposite side of the issue is false positives. For example, say you're driving a car which uses a deep learning network to predict oncoming crashes. It would be incredibly dangerous if the network failed to predict a crash and no safety measures were taken (airbags, etc). On the contrary, if the network predicted a crash that wasn't actually happening and set off the airbags randomly, it could be just as dangerous while driving.

Applications of Deep Learning

While many of the applications have been discussed previously, I would like to give a brief overview of some of the most popular and beneficial applications of deep learning.

Fraud detection – many credit card companies have switched to using deep learning models to detect credit card fraud. This allows the companies to save their customers money, as well as their employees time by not having to deal with fraudulent charges.

Video game cheat detection – One of the biggest video game developers, Valve, has recently created a deep learning model they call VACnet to detect players that are cheating in the video game Counter-Strike: Global Offensive.

Voice recognition – There exist many different home assistant technologies, such as Siri, Amazon's Alexa, Google, etc. All of these are powered by a deep learning network that is used to detect distinct voices, and recognize what's being said.

Technical Implementation/Prototype

To learn more about Deep Learning Neural Networks, I set out to implement one of my own. Inspired by the aforementioned VACnet to detect cheaters in a video game, my goal was to detect when the cursor is being controlled by a human or by a program. In many games, this is referred to as botting and has potentially massive negative impacts on the game and its community.

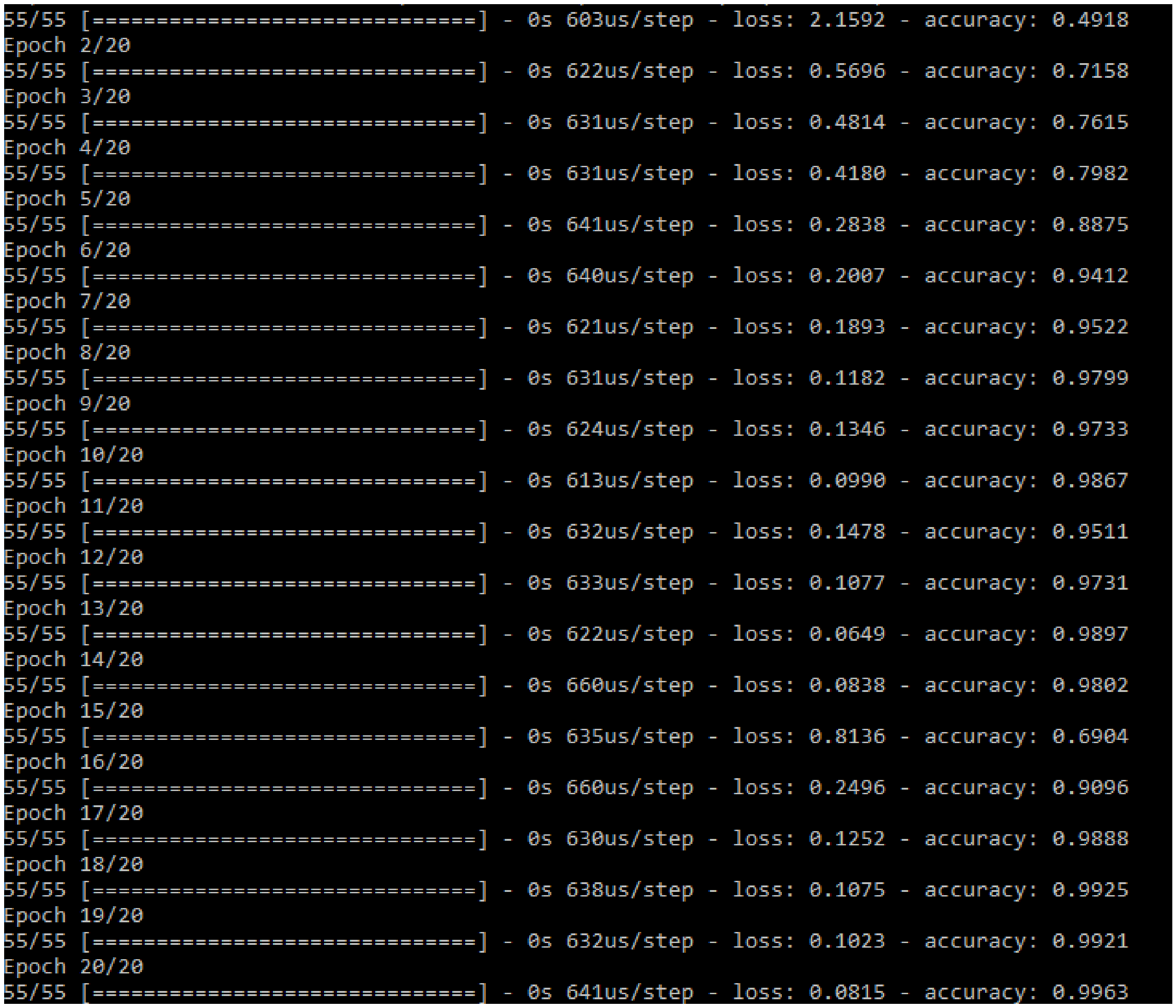
The first hurdle was collecting the data. I wrote a small C program that created a window with two buttons. When the left button was pressed, the program would start tracking the coordinates of the cursor, until the right button was pressed. This would allow me to collect data easily, so long as I could automate that process.

The next step was writing another small C program to control the mouse movement. I believed that moving the mouse in a complete straight line would make it a bit too easy for the network to detect the AI. On top of that, doing a complete straight line would almost make the use of a neural network pointless, as a human could just as easily detect if a Y coordinate never changed. Not to mention, most botting programmers would attempt to circumvent detection. Thus, I implemented a randomization factor to the mouse movements. As the cursor moves from left to right along the X-axis, it also gets moved up and down the Y-axis by 0-5 pixels. This value is randomized upon every single update of the cursor along it's path.

With the data collection out of the way, it was time to implement the deep learning neural network. To do this, I used the Python version of Tensorflow/Keras. Using python, I wrote a script that reads in the mouse data stored in a CSV file, stores it in an array, and then feeds it into the model.

The model used is a Sequential model. The model takes in an array of 57 Y-axis values of the mouse as input. For simplicity, the model only used two hidden layers. The first hidden layer was made up of 128 neurons, and used the reLU activation function. The second hidden layer used only 32 neurons, and also used the reLU activation function. The final output layer consisted of 2 neurons, one labelled “AI” and one labelled “HUMAN”. The output layer used the softmax activation function to map the final values between 0 and 1, which represents the percentage that the model believes the input was Human or AI. The output is in the form of a 1D array containing 2 elements. Array[0] is the percentage the model believes the input was human, and Array[1] is AI.

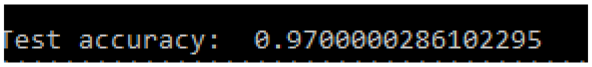
The model was trained using 1750 data points. Seen below is result of running 20 epochs on the training data.



```
55/55 [==============================] - 0s 603us/step - loss: 2.1592 - accuracy: 0.4918
Epoch 2/20
55/55 [==============================] - 0s 622us/step - loss: 0.5696 - accuracy: 0.7158
Epoch 3/20
55/55 [==============================] - 0s 631us/step - loss: 0.4814 - accuracy: 0.7615
Epoch 4/20
55/55 [==============================] - 0s 631us/step - loss: 0.4180 - accuracy: 0.7982
Epoch 5/20
55/55 [==============================] - 0s 641us/step - loss: 0.2838 - accuracy: 0.8875
Epoch 6/20
55/55 [==============================] - 0s 640us/step - loss: 0.2007 - accuracy: 0.9412
Epoch 7/20
55/55 [==============================] - 0s 621us/step - loss: 0.1893 - accuracy: 0.9522
Epoch 8/20
55/55 [==============================] - 0s 631us/step - loss: 0.1182 - accuracy: 0.9799
Epoch 9/20
55/55 [==============================] - 0s 624us/step - loss: 0.1346 - accuracy: 0.9733
Epoch 10/20
55/55 [==============================] - 0s 613us/step - loss: 0.0990 - accuracy: 0.9867
Epoch 11/20
55/55 [==============================] - 0s 632us/step - loss: 0.1478 - accuracy: 0.9511
Epoch 12/20
55/55 [==============================] - 0s 633us/step - loss: 0.1077 - accuracy: 0.9731
Epoch 13/20
55/55 [==============================] - 0s 622us/step - loss: 0.0649 - accuracy: 0.9897
Epoch 14/20
55/55 [==============================] - 0s 660us/step - loss: 0.0838 - accuracy: 0.9802
Epoch 15/20
55/55 [==============================] - 0s 635us/step - loss: 0.8136 - accuracy: 0.6904
Epoch 16/20
55/55 [==============================] - 0s 660us/step - loss: 0.2496 - accuracy: 0.9096
Epoch 17/20
55/55 [==============================] - 0s 630us/step - loss: 0.1252 - accuracy: 0.9888
Epoch 18/20
55/55 [==============================] - 0s 638us/step - loss: 0.1075 - accuracy: 0.9925
Epoch 19/20
55/55 [==============================] - 0s 632us/step - loss: 0.1023 - accuracy: 0.9921
Epoch 20/20
55/55 [==============================] - 0s 641us/step - loss: 0.0815 - accuracy: 0.9963
```

Figure 2: The model being trained on 1750 training inputs.

On the training data, the model reached an impressive 99.63% accuracy.

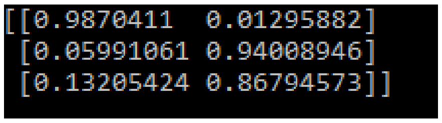


```
Test accuracy:  0.9700000286102295
```

Figure 3: The model's accuracy on the test data after being trained

Running the model on the testing data resulted in 97.00% accuracy.

Finally, it was time to actually use the model to make predictions about input it had never seen before. I used three data points, the first being from a human, and the second/third being from a program.



```
[[0.9870411  0.01295882]
 [0.05991061 0.94008946]
 [0.13205424 0.86794573]]
```

Figure 4: Percentages representing the likelihood of the input being Human or AI.

For all three data inputs, the model was able to correctly predict whether or not the mouse coordinates were generated by a human or not!

All of the code and data is available for download at https://github.com/DavidLekei/CheatDetection

Conclusion

Deep Learning is a vast and incredibly interesting field of Artificial Intelligence. As our technology improves and we gain more and more computing power, Deep learning networks can solve some amazing problems. There are many different models to design a deep neural network, all of which serve a purpose. No one model is perfect, and we may never reach perfection. Many of the problems facing deep learning neural networks may never get solved. However, over time, the amount of data we have will only increase, making deep learning networks stronger and more reliable, and one day they may change our world in ways we cannot even imagine.

Bibliography

[Saad et al, 2017] Albawi, Saad, Abed Mohammed, Tareq Saad, "*Understanding of a Convolutional Neural Network*", 10.1109/ICEngTechnol.2017.8308186, August 2017.

[Medsker and Jain, 2001] Medsker, L.R and Jain, L.C, "*Recurrent Neural Networks, Design and Applications*", 2001.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter, Jurgen Schmidhuber, "*Long Short-Term Memory*", Natural Computation 9(8):1735-1780, 1997.

[Shrestha and Mahmood, 2019] A. Shrestha and A. Mahmood, "*Review of Deep Learning Algorithms and Architectures*", IEEE Access, vol. 7, pp. 53040-53065, April 2019.

[Pouyanfar et al, 2019] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyenga, "*A Survey on Deep Learning: Algorithms, Techniques, and Applications*", ACM Comput. Surv. 51, 5, Article 92 January 2019.

[Goodfellow et al, 2014] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "Generative Adversarial Nets", arXiv:1406.2661, June 2014.