Data length and the accuracy of defining a word in context using BERT.

To what extent does a sentence's wordcount affect BERT's accuracy to distinguish between a word's meanings in context?

Computer Science

Word Count: 3996

Table of Contents

Introduction:

Artificial Intelligence (AI) has become an increasingly popular term in recent years. The adaptability and effectiveness of the concept has allowed its implementation in practically every field: from traditional industries (autonomous driving) to creative fields (Autotune and CGI). Increasingly important to the everyday person, however, might be the use of AI in Natural Language Processing (NLP) – the study of "computational modeling of various aspects of language" (Joshi, 1991). A popular model used for NLP is Bidirectional Encoder Representations from Transformers (BERT), the main study of this essay. BERT has established itself as a leading state-of-the-art model in NLP (Devlin, Chang, Lee, & Toutanova, 2019), most notably seeing use in Google's search engine processing queries (Nayak, 2019). Not only is it noted for its increased speed and ability to understand language (Devlin, Chang, Lee, & Toutanova, 2019), but also its, theoretical, solution to the Vanishing Gradient Problem(VGP) (Vaswani, et al., 2017), a problem that has plagued NLP models because of its hindrance to longer input data.

To answer the research question: "**To what extent does a sentence's wordcount affect BERT's accuracy to distinguish between a word's meanings in context?**", this essay will focus on the field of NLP. The experiment's independent variable is the length of the sentence. Some uncontrolled variables include: the type of sentence (e.g. descriptive or narrative, etc), the type of word (e.g. noun or verb, etc), or the word's meaning. This essay will also only consider the English version of BERT Large (Devlin, Chang, Lee, & Toutanova, 2019) therefore it's important to note a different language might have different outcomes. Furthermore, in the

experiment, the lengths will be categorised into bands. This might have some minor effect on the results yet using bands instead of a discrete number of words means a more balanced number of sentences will be able to be collected for each band. This will minimize unintended errors as having roughly the same length sentence is more impactful than differentiating between, for example, 5 and 6 words in a sentence.

This research paper would allow us to ensure BERT has solved, or at least minimised, the Vanishing Gradient Problem for NLP (Vaswani, et al., 2017), therefore improving on the limitations of its predecessors. It is also worth testing whether BERT's accuracy is affected by sentence length when defining a word in context in order to further understand how BERT's accuracy could possibly vary in different environments, especially with all the information available for interpretation and analysis on the web such as social media (like Twitter) or other longer forms of media (like news outlets). As different contexts usually contain different lengthed sentences, answering the research question would allow us to consider any bias present in the results. Based on the analysis, this will be further explored in the conclusion.

In order to answer the research question and comprehend the reasoning behind it and the analysis, it is important to understand the theory leading to it. The Feedforward Artificial Neural Network (Mcculloch & Pitts, 1990), Recurrent Neural Network (Rumelhart, Hinton, & Williams, 1986) and Transformer Model (Vaswani, et al., 2017) will all be explored with regard to the issue they face: the Vanishing Gradient Problem and how Transformers and BERT supposedly solve the issue – the theoretical solution being the basis for the research question. BERT will

then be used in an experiment to test this theory and answer the research question, finally finishing with the analysis and conclusion.

Background Information:

Feedforward Artificial Neural Networks:

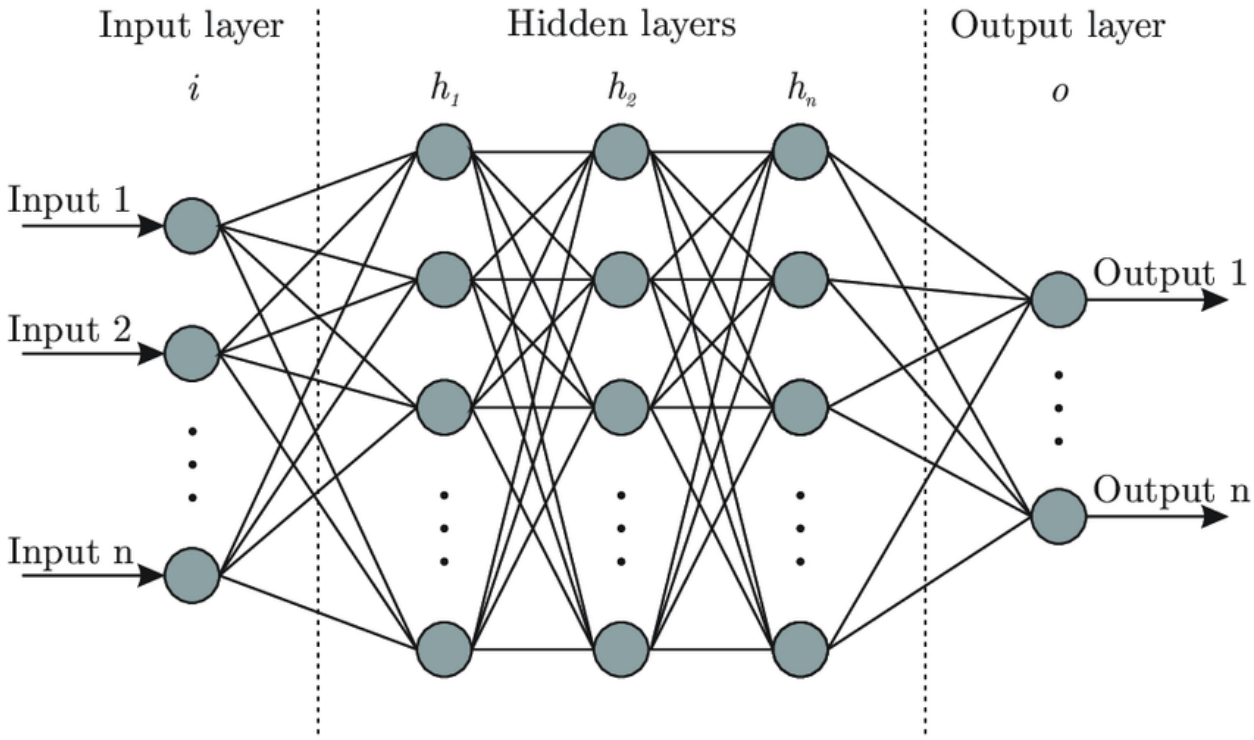

Figure 1. Artificial Neural Network Architecture (Bre, Juan, & Victor, 2017)

An Artificial Neural Network (ANN) is an architecture of connected neurons (nodes) inspired by the brain (Hwang & Ding, 1997). Feedforward ANNs (which will just be referred to as ANNs or basic ANNs) are the standard basis for artificial intelligence and more complex neural networks such as this essay's main focus: BERT. To gain a better understanding of what is being researched in this essay, it's important to have an intuition of ANNs, their inner-workings and henceforth the main inherent problem many ANNs face because of the structure of ANNs.

As seen in Figure 1, ANNs are made up of the input layer (i), the output layer (o) and hidden layers (h_n) that perform all the calculations.

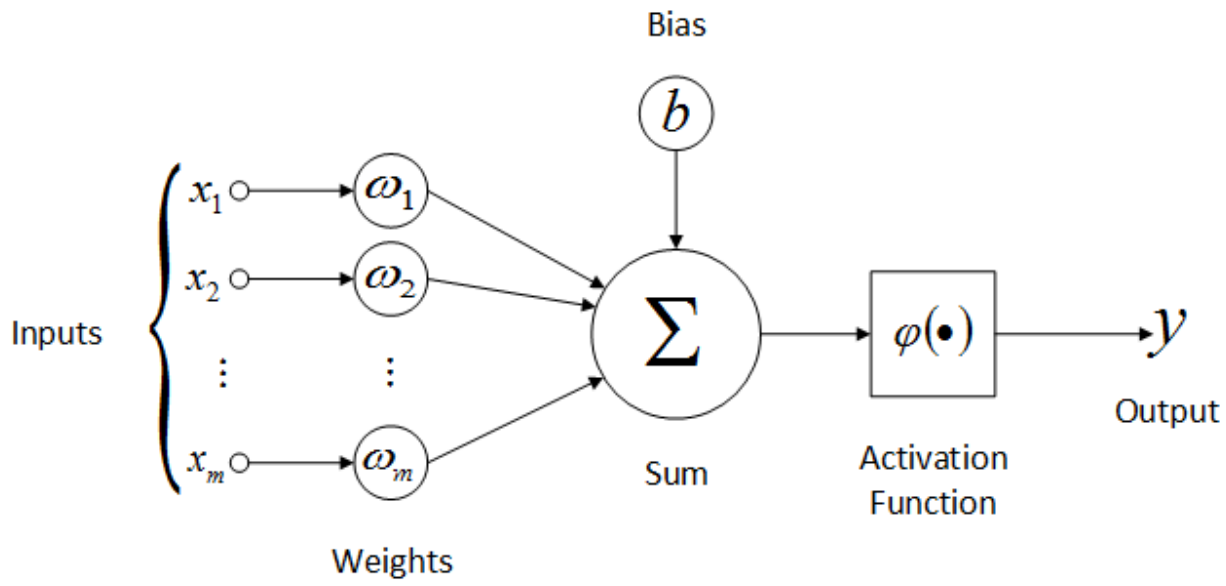

Figure 2. Structure of an artificial neuron (Jadidi, Menezes, Souza, & Lima, 2018)

$$Z = \sum_{i=1}^{n} x_i w_i + b$$

$$= (x_1 w_1) + (x_2 w_2) \; + \; \ldots \; + \; (x_n w_n) + b$$

Equation 1. The sum of Weights and Biases

At each hidden node, the input or previous result (x_i)is multiplied by its weight (w_i). Weights are defined as "numerical parameters which determine how strongly each of the neurons affects the other" (Ciaburro & Venkateswaran, 2017). Weights are first set randomly and later adjusted when training meaning as the ANN is trained, it tends to learn what 'path' to take along the ANN with different inputs.

A bias (b) is "an adjustable, numerical term added to a [node's] weighted sum of inputs and weights that can increase classification model accuracy." (Code Academy, n.d.). The bias helps make further minute details, increasing the accuracy.

The final sum (Z) is then passed through the activation function and the output is propagated forwards to the next nodes. The activation or threshold function is the "function that describes the output behaviour of a neuron" (Wilson, 1998): if the output reaches a certain level, it fires.

For ANNs to learn, they first need to be trained and, in most cases, this means supervised learning. At the highest level of abstraction, supervised training is performing a gradient descent: adjusting weights and biases in order to minimise the final loss function (3Blue1Brown, 2017). The most basic loss function is calculated by finding the difference between the real expected output and the ANN's predicted output (Baskaran, 2020). The loss function ensures the ANN is learning and repeats the training algorithm until it has used all of the training data. Backpropagation, "an algorithm for supervised learning of artificial neural networks using gradient descent [of the loss function]" (McGonagle, et al., n.d.), is typically used as the training algorithm.[1] The gradient of the loss function is calculated with respect to the weights (deeplizard, 2018) and biases; backpropagation then uses the gradient of the loss function and "repeatedly adjusts the weights of the connections in the network so as to minimize [the loss function]" (Rumelhart, Hinton, & Williams, Learning representations by back-propagating errors, 1986).

As previously stated however, ANNs are unviable for NLP; this is for the following two main reasons:

1. The number of inputs in an ANN is fixed so the ANN would only allow for a fixed number of words in a sentence.

[1] A reference of how gradient descent works can be seen in Appendix 1

2. ANNs cannot take into consideration the order of words in a sentence (this will be further explained later).

Hence a more specific model, based on the ANN, had to be created in order to solve this issue. The model, however, also had issues and Transformers, as well as BERT, were developed fix this problem which will later be discussed in this paper. The research question and experiment are grounded on proving BERT solves the issue. Hence, understanding the theory is crucial to analyse the data's significance.

Recurrent Artificial Neural Networks:

Based on the basic Feedforward ANN model architecture, Recurrent neural networks (RNNs) have been the de facto model in NLP for many years due to their ability to have "sequential memory" (Phi, Youtube, 2018). Being used to process sequences of words (sentences), RNNs were able to understand basic meaning and context. Figure 3, below, shows the difference between RNNs and ANNs, notably the loop in the hidden layers.

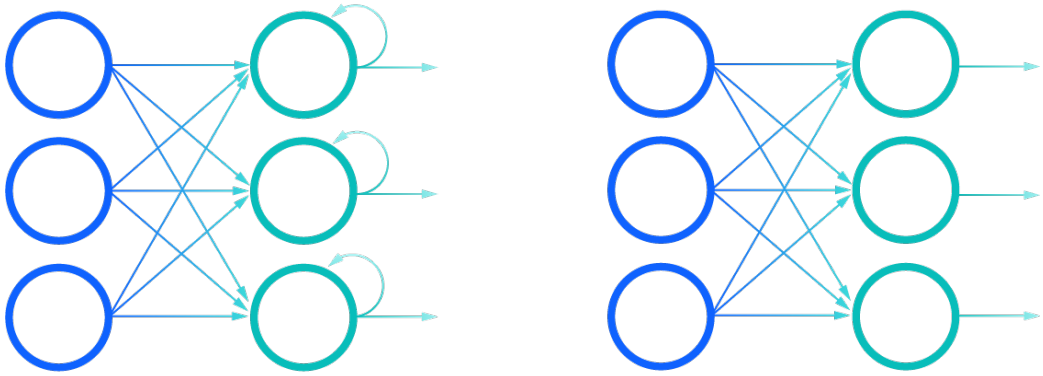

Figure 3: Comparison of Recurrent Neural Networks (on the left) and Feedforward Neural Networks (on the right) (IBM Cloud Education, 2020)

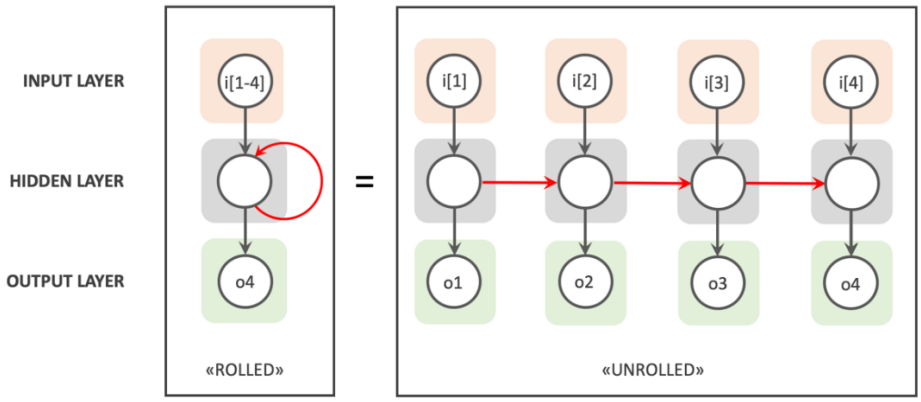

Figure 4. *An illustration of the Recurrent Neural Network Architecture. (West, 2020)*

As seen in Figure 4, RNNs can be thought of as many copies of the same basic Feedforward ANN joined together (West, 2020). Similarly to the basic Feedforward ANN structure, RNNs

also have input, output, and hidden layers with almost the same node structure, however, the hidden layer specifically 'talks' to itself: passing different data as input to itself the next time round. This allows for sequential data to be processed by keeping previous inputs as working memory (Phi, Youtube, 2018); for sentences, this means every word in the sentence is a new input. With every word as an input, it finally outputs a result (the output varying based on what the model is trained to do). Figure 5. shows a diagram of a simplified RNN node.

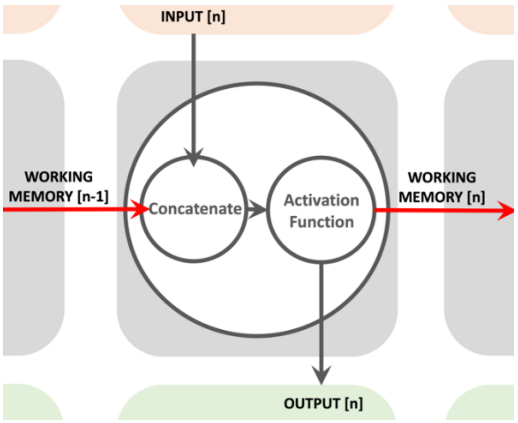

Figure 5. Internal operations in a Hidden Node of an RNN *(West, 2020)*

The input and working memory from the previous layer are concatenated to allow for 'memory' of previous data. Although not shown, this input goes through the same functions as an ANN node would.

To compare the classic feedforward ANN's 'bag-of words' (not in order) structure with RNN's sequential structure, an example is given by the textbook "Neural Networks and Deep Learning: A Textbook" (Aggarwal, 2018), considering these two sentences:

"The cat chased the mouse.

The mouse chased the cat."

"The two sentences are clearly very different […] [h]owever, the bag-of-words representation would deem them identical." This is because, with bag-of words, the order does not matter. RNNs on the other hand are able to differentiate the difference between specific words and the surrounding context including the order of words through the position or timestamps in a sentence.

Although RNNs do produce relatively good results, they face a problem known as the vanishing gradient problem (VGP). This problem is faced during training and is exacerbated by longer sequences due to the nature of how the gradient (the same gradient as in ANNs) is calculated: backpropagation. Simply put, because of backpropagation, the earlier a weight appears in the network, the more weights it will depend on. Figure 6 shows this in an ANN.

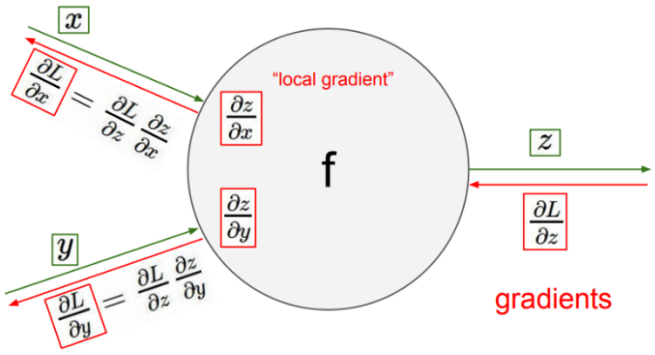

Figure 6. Gradients in an ANN (Li, Johnson, & Yeung, 2017)

As the dependencies of the gradient increase, the following change to the gradient will exponentially get smaller (just like how the product of for example 0.5 and 0.3 produces an even smaller number) (deeplizard, 2018). When the backpropagation adjusts the weights and biases based on the gradient and the learning rate, the change in the earlier weights will be miniscule,

therefore not allowing the earlier section of the network to 'learn'. The problem is known as the VGP because the gradient 'vanishes' to a miniscule and negligible amount. This is very problematic for ANNs, and especially RNNs designed for NLP as this has serious implications on the model's understanding of the sentence context. Take for example, the sentence:

Will you go to the beach with your best friends?

We as humans, can see that this sentence is in future tense, so we would respond in future tense too:

Yes, I will.

However, with the VGP, an RNN model might have lost the context of the first word "**Will**" and therefore might find it difficult to understand that the sentence is in future tense. Although this might be an exaggeration, the analogy is still useful to understand the concept.

The VGP will also only worsen as sentences, and therefore sequences, grow. Some partial solutions have been suggested and implemented: such as using a specific "Long Short-Term Memory"(LSTM) (Hochreiter & Schmidhuber, 1997) or Gated Recurrent Unit(GRU) (Cho, et al., 2014) model instead of a general RNN, or using the RELU or Leaky RELU (Maas, Hannun, & Ng, 2013) activation functions rather than the sigmoid activation function; however, they have not completely eradicated the VGP, sometimes bringing in their own issues — like LSTM's increased computational and time costs.

Leading on from RNNs, Transformers were created to improve on RNNs' VGP. The next section will explore Transformers and how it supposedly solved or at least minimised the VGP, helping us understand the meaning of the research question as well as any implications an answer to the question might have.

Transformers:

In 2017, Google introduced transformers as a replacement for recurrent and convolutional neural networks: showing it was "superior in quality while being parallelizable and requiring significantly less time to train" (Vaswani, et al., 2017). Transformers paved the path for the next generation of NLP architectures, capturing long-term dependencies (Ravichandiran, 2021) and shortening the training time. Sustaining long-term dependencies (Ravichandiran, 2021) solved the VGP, in turn also allowing for longer sequences to be processed accurately. The transformer's success is mainly due to the attention mechanism that allows it to "[capture] the relationships between each word in a sequence with every other word" (Doshi, 2021).

Transformers are made up of two main sections: the encoder and decoder, however this essay will only discuss parts of the encoder in detail as BERT does not utilise the decoder, only creating BERT word-embeddings ("where words […] from the vocabulary are mapped to vectors of real numbers" (Word Embeddings, n.d.)), and because describing all of the encoder in detail would be too lengthy and not very useful for the scope of this essay.

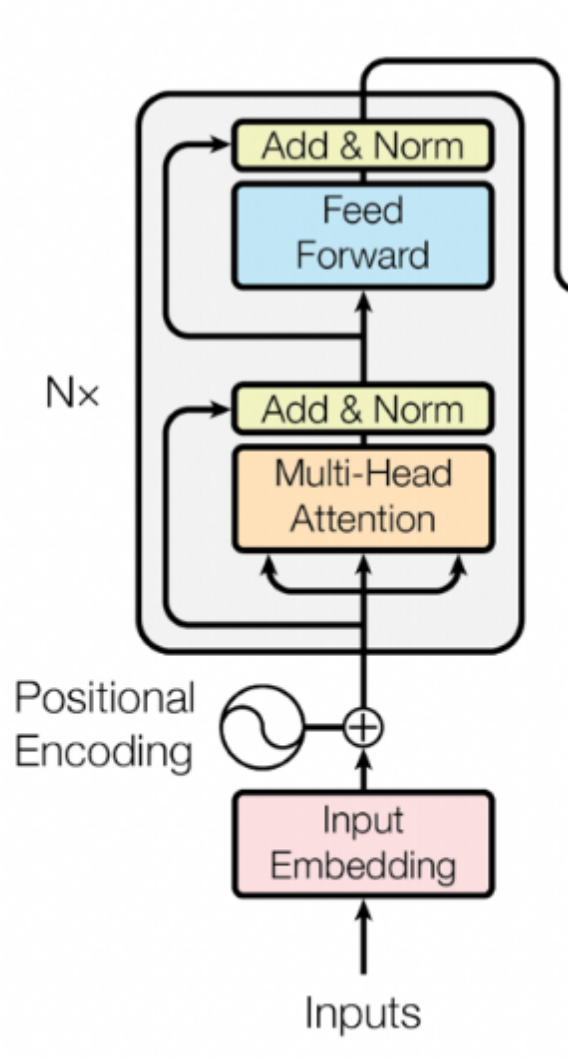

Figure 7: The encoder section of the transformer model (Vaswani, et al., 2017)

As seen above in Figure 7, the encoder is, in essence, composed of the inputs with a positional embedding added, then the encoder layer with a Multi-Headed Attention layer and a Feedforward layer. This produces a vector word-embedding of each word at the end.

In order to not use recurrence in transformers, positional encoding is used to indicate the position of the word in the sentence (Phi, 2020). This also ensures the order and context of the words is taken into consideration, therefore proving transformers still handle sequential data.

The encoder layer follows the input and position encoding. "The encoder layer's job is to map all input sequence into an abstract continuous representation that holds the learned information for that entire sequence" (Phi, 2020). The encoder layer is made up of the multi-headed attention layer and the Feedforward layer; the attention layer provides a method to associate each word in the sentence to every other word (Phi, 2020), calculating the importance of the relationship, producing a vector matrix similar to Figure 8.

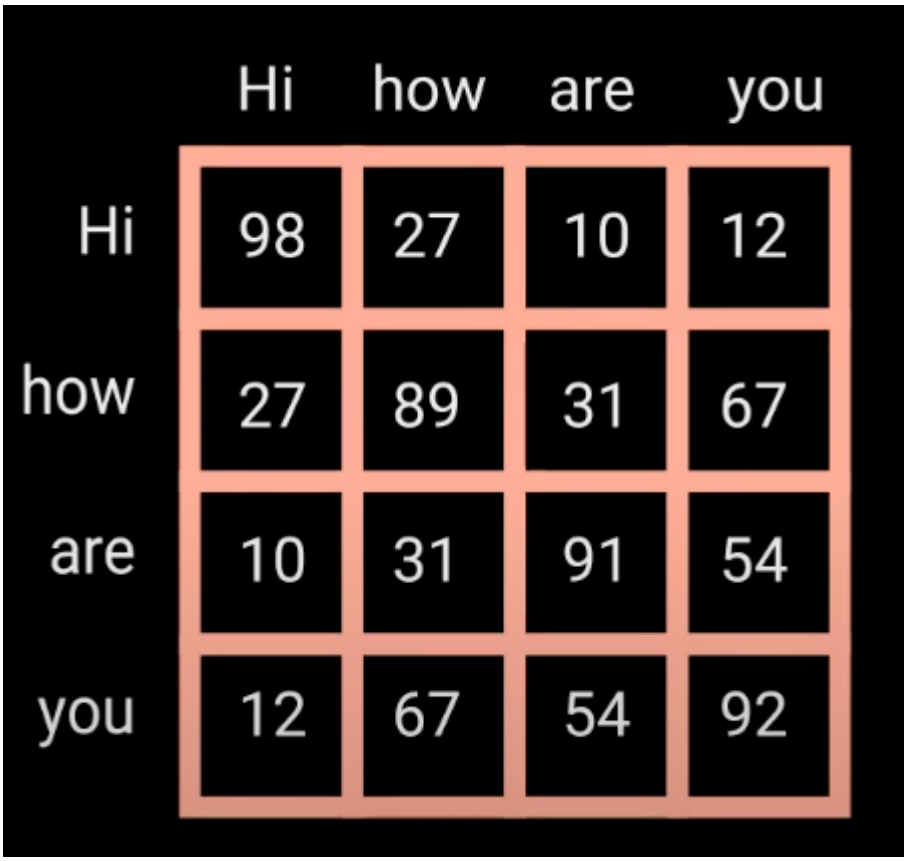

Figure 8. Attention scores from the dot product. (Phi, 2020)

In Figure 8, the higher the value, the more 'importance' is given to each word, relating to the other word. This usually leads to words having a higher importance to themselves such as with "Hi" and "Hi"; Figure 9 also indicates this:

Figure 9. Diagram of Attention in a sentence (Vaswani, et al., 2017)

In this case, the darker the line, the more 'important' a word is (to that specific word). Take for example the word "Law" in the left column. It has three lines connecting it to "The", "Law", and "its"; this is likely because, in the sentence, these three words refer to the word "Law".

Finally, "a feedforward layer is applied to each position separately and identically" (Vaswani, et al., 2017) to further process the input. Because each word is processed separately, this work can

be parallelised, making full use of modern GPUs which are made with the idea of parallelisation in mind (Caulfield, 2009).

The main difference between RNNs and Transformers is that RNNs' inputs are passed one word at a time (sequentially); transformers, however, input words simultaneously (CodeEmporium, 2020). This also means the VGP is no longer problematic as there is no need for a 'deep' (many layers) Neural Network, instead a 'wide' (many nodes in a single layer) Neural Network can be used (Kapoor, 2022).

In order to implement the transformer architecture in a more accessible way, BERT was created. The next section will connect the structure of transformers to BERT and the VGP: the two main focuses of this essay. To summarise, the main similarities and differences between these three models can be seen in Table 1 below:

Model	Can handle sequential data?	Directionality	Is Parallelisable?	General use	Has the Vanishing Gradient Problem?
Feedforward Artificial Neural Network	No	One Directional	Yes	Regression and Classification	Yes
Recurrent Neural Network	Yes	One Directional	No*	NLP	Yes
Transformer	Yes	Omni-directional	Yes	NLP	No

Table 1: The 3 models and their differences. (selfmade)

** There have been some successes in the parallelization of RNNs, however, they are not very parallelizable by nature (CodeEmporium, 2020).*

BERT:

Bidirectional **E**ncoder **R**epresentations from **T**ransformers or BERT is an NLP model created by Google in 2018. Created with intention of "alleviat[ing] the… unidirectionality constraint" (Devlin, Chang, Lee, & Toutanova, 2019), which was the norm before this paper, BERT revolutionized the scene of NLP, being the "first deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus" (Devlin & Chang, 2018). BERT is based on and made up of the 2017 transformer model yet only uses the encoder section of transformers. In essence, it's an implementation of the transformer model in order to 'learn' and 'understand' language (CodeEmporium, 2020). Figure 10 shows BERT's structure:

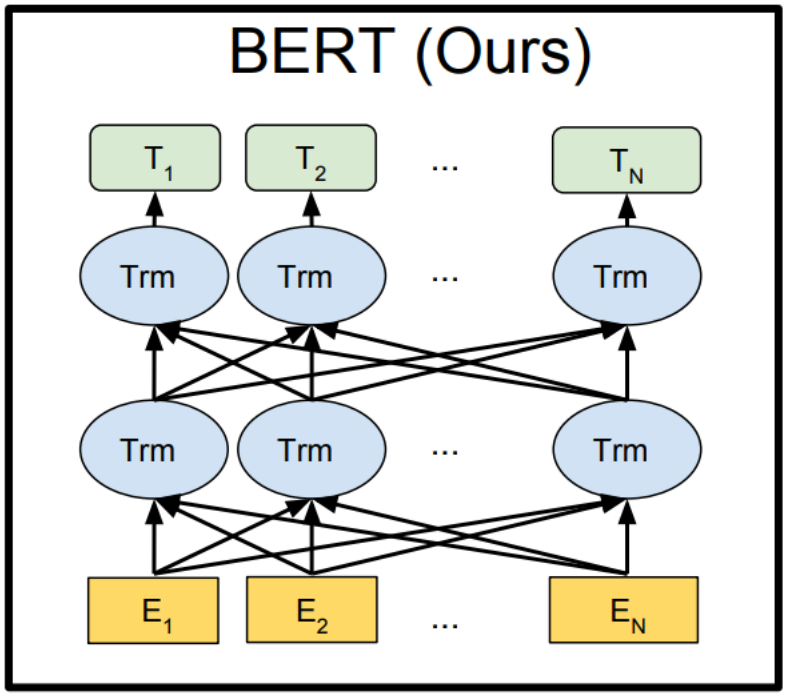

Figure 10. Structure of BERT (Devlin, Chang, Lee, & Toutanova, 2019). E_N stands for the input embedding. Trm stands for transformer. T_N stands for final contextualized representation.

This leads to the research question's worthiness. Transformers were created to solve the problem of "RNN[s'] performance start[ing] to decrease significantly… in the case of longer length

sentences" (Johri, Verma, & Paul, 2020) in other words, the VGP; thus BERT, made up of stacked transformers, should not face any challenges with longer sentences. This essay and its experiment will test this theory by answering "**To what extent does a sentence's wordcount affect BERT's accuracy to distinguish between a word's meanings in context?**".

Only using the transformer encoder means it can be initially pretrained to understand language properly and later finetuned to specific tasks, such as question answering, sentiment analysis, and text summarisation (Devlin, Chang, Lee, & Toutanova, 2019). Pretrained models have shown promise in NLP (Dai & Le, 2015) proving a simple understanding of language can be achieved. Pretraining meant BERT would be initially trained with a large dataset (Devlin, Chang, Lee, & Toutanova, 2019) and could later be used as a base to complete several NLP tasks after finetuning; this not only sped up transformer implementations, but also improved their quality.

However, although BERT is an implementation of the transformer model for NLP, therefore eliminating the VGP, it still has issues with longer sentences as "[l]onger sequences are disproportionately expensive because attention is quadratic to the sequence length" (Devlin, Chang, Lee, & Toutanova, 2019). This problem does not, however, directly affect the accuracy of the model, it only means longer pre-training or less computation time available for longer sentences. In theory, however, as the VGP does not affect BERT, long-range dependencies are kept and therefore longer sentences will not falter with accuracy. BERT has eliminated ANNs and RNNs' flaws and is theoretically apt for NLP. This, again, is the motivation for the research question, "**To what extent does a sentence's wordcount affect BERT's accuracy to distinguish between a word's meanings in context?**": **to prove whether BERT's accuracy is**

really not affected by the sentences' length and if it has solved, or at least minimised the VGP.

The Experiment Methodology & Dataset:

The Dataset Used:

A premade dataset with sample sentences for words with multiple meanings could not be found, therefore I specifically created a dataset for this research. The sentences were primarily found using the website https://sentence.yourdictionary.com/, but other sources were also used as not enough data could be collected with only one source. I then classified the meaning of the words in the sentences by hand: adding, into a spreadsheet, the sentence and classifying the meaning of the word. To ensure the results were only affected by the sentence length, four words were defined as having two meanings: Address, Bark, Fall, Feet; and another four were defined as having three meanings: Bat, Date, Right, Tie. These words were tested as they have different uses in sentences which would allow for BERT's attention to work in distinct ways. The total number of sentences was 746.

Data Pre-processing:

To answer the research question, the data needed to be separated into length bands. I decided to separate them into bands of $0 < x \leq 8,\ 8 < x \leq 15,\ 15 < x \leq 20,\ 20 < x \leq 25,\ 25 < x$. The first band was chosen to be $0 < x \leq 8$ as there were very few sentences with $0 < x \leq 5$ and I found this to be a good-enough balance between the amount of data and still being representative of the band. To separate them, I wrote a piece of code (Appendix 2) and ran it several times before uploading it to GitHub as individual Gist files.

The Experiment:

To answer the research question, I modified a piece of code (Original code: https://colab.research.google.com/drive/1rbhuZYjMGezLJmpzc9p8T38gXqILTHt_) (Appendix 3: One example of the modified codes). Each code was then run with its respective data sample (each of the 8 words with each of the 5 bands) on Google Colab notebooks as it allowed for much faster processing, especially when a GPU hardware accelerator was used.

The minimum number of sentences for each data sample was 9 inclusive as the program had defined the number of neighbours in the K-NN classifier (K-Nearest Neighbour classifier) (Fix & Hodges, 1951) (Altman, 1992) model to be 8. The K-NN classifier was used to calculate the accuracy by finding the nearest, in this case, 8 'neighbours' or vector points for each point and then comparing the distances between each point and their 8 neighbours.

In order to better visualise the data on a graph, the program used a Principal Component Analysis (PCA) (Pearson, 1901) to reduce the dimensionality of the data to 2D. This helped better understand the data. More importantly, the program returned the accuracy and standard deviation which was manually saved into excel.

Evaluation of reliability and validity of method:

The number of sentences was not the same for each word due to human collection and classification of sentences being a limiting constraint. This might affect the results as small sample sizes can mean smaller accuracies.

Analysis:

Results:

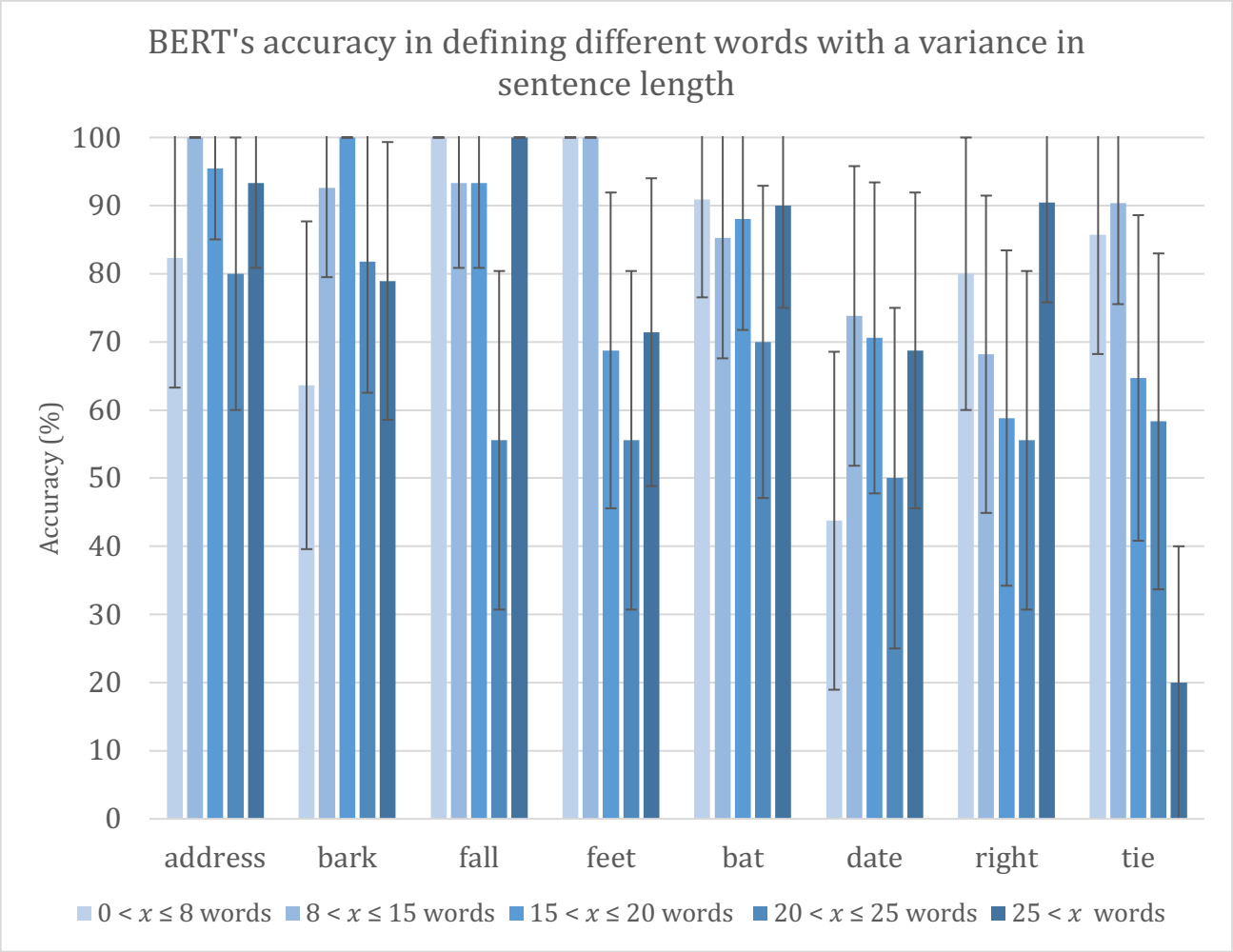

Figure 11 Bar Chart of Results

	Accuracy (%)					
Word	**0 < x ≤ 8 words**	**8 < x ≤ 15 words**	**15 < x ≤ 20 words**	**20 < x ≤ 25 words**	**25 < x words**	**Average**
address	82.353	100.00	95.455	80.000	93.333	**90.228**
bark	63.636	92.593	100.00	81.818	78.947	**83.399**
fall	100.00	93.333	93.333	55.556	100.00	**88.444**
feet	100.00	100.00	68.750	55.556	71.429	**79.147**
bat	90.909	85.294	88.000	70.000	90.000	**84.841**
date	43.750	73.810	70.588	50.000	68.750	**61.380**
right	80.000	68.182	58.824	55.556	90.476	**70.608**
tie	85.714	90.323	64.706	58.333	20.000	**63.815**
Average	**80.795**	**87.942**	**79.957**	**63.352**	**76.617**	

Table 2:Table of Results – Blue signifies an anomaly: 20% is much lower than any other accuracy.

Analysis of results:

The results show no noticeable relationship between the sentence's wordcount and accuracy, however, there does seem to be noise because it varies slightly. Although the highest average accuracy is in the $0 < x \leq 8$ words, and $8 < x \leq 15$ words bands, the difference to the other bands seems to be negligible. If an ANN or RNN was used instead of BERT, theoretically, the pattern would have been a decrease in accuracy as the number of words in a sentence increased because they face the VGP (Table 1).

	Accuracy (%)					
Word	**$0 < x \leq 8$ words**	**$8 < x \leq 15$ words**	**$15 < x \leq 20$ words**	**$20 < x \leq 25$ words**	**$25 < x$ words**	**Average**
address	82.353	100.00	95.455	80.000	93.333	**90.228**
bark	63.636	92.593	100.00	81.818	78.947	**83.399**
fall	100.00	93.333	93.333	55.556	100.00	**88.444**
feet	100.00	100.00	68.750	55.556	71.429	**79.147**
bat	90.909	85.294	88.000	70.000	90.000	**84.841**
date	43.750	73.810	70.588	50.000	68.750	**61.380**
right	80.000	68.182	58.824	55.556	90.476	**70.608**
tie	85.714	90.323	64.706	58.333	20.000	**63.815**
Average	**80.795**	**87.942**	**79.957**	**63.352**	**76.617**	

Table 3: Coloured Table of Results The tables have been shaded to better discern different patterns and important points to note. Green signifies a surprisingly high accuracy, yet a relatively low number of sentences. Orange signifies a relatively high accuracy, yet a relatively low number of sentences. Red signifies a low accuracy with a low number of sentences. Blue signifies an anomaly.

Reviewing the results suggests the cause of the variance in accuracy might be attributed to the number of sentences; comparing the accuracy results with the number of sentences, proves this hypothesis. This problem might occur due to the method used to calculate the accuracy – more specifically the K-NN classifier selecting the **8** nearest neighbours, as the worst performing groups were those at, and close to, 9 sentences. This pattern can also be seen **in the graphs below (figure 12 and 13)**, the number of sentences used as data seems to follow a similar pattern to the average accuracy.

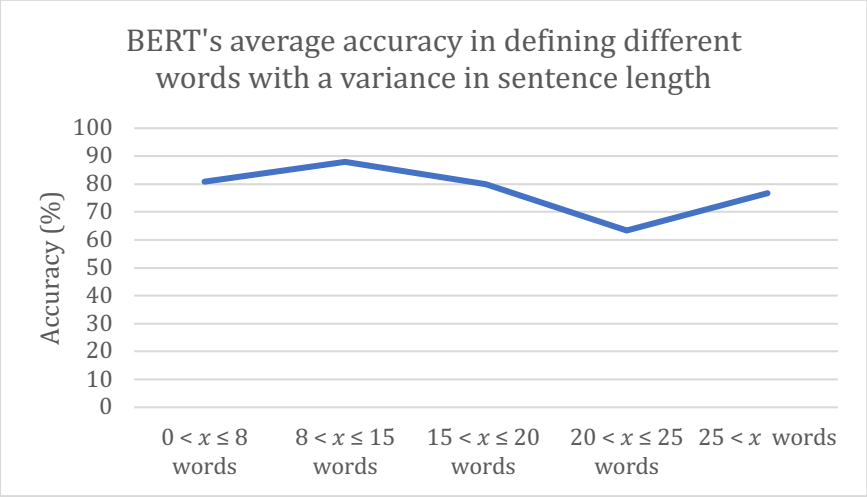

Figure 12: BERT's average accuracy in defining different words with a variance in sentence length

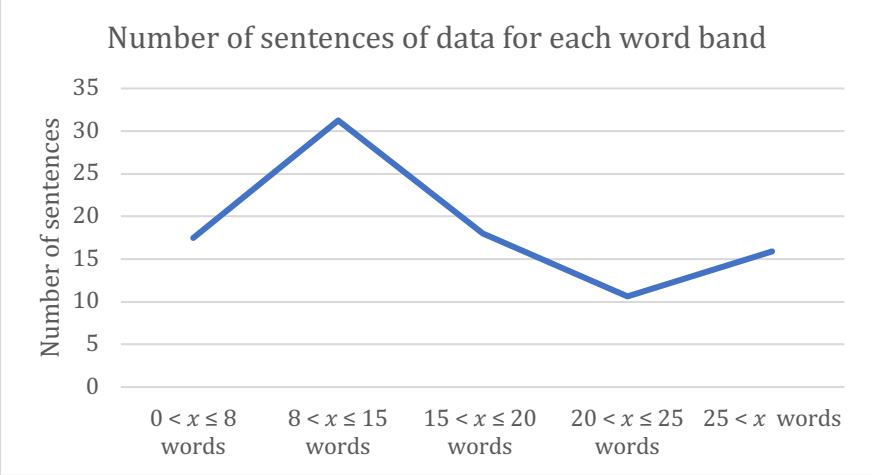

Figure 13: Number of sentences of data for each word band

Using the Pearson Correlation function on excel, with the following formula:

$$r = \frac{\sum(x - \overline{x})(y - \overline{y})}{\sqrt{\sum(x - \overline{x})^2 + \sum(y - \overline{y})^2}}$$

Equation 2: Pearson Correlation Formula (Microsoft, n.d.) (Microsoft, n.d.)

a correlation of $0.8786 \approx 0.88$ was found between the accuracy and the number of sentences used for each word band. This is a quite significant correlation indicating that perhaps the data sample size might have actually been responsible for the variance in BERT's accuracy when defining a word in context.

In Table 3, the first anomaly (date, $0 < x \leq 8$ words) is strange as with 16 sentences, not a low number of sentences, BERT was only able to achieve an accuracy of 43.75%, the second lowest. One possible reason might be a difference in sentence complexity or type (e.g. descriptive or narrative, etc); the words themselves also lead to very different results; date for example seemed to have trouble – possibly (as mentioned in the introduction) due to very similar word meanings. Considering the three meanings for date were: 'Time i.e., Year, Month, Day', 'a dinner with a lover', and 'the person one goes on a date with'. The last two definitions are particularly similar and are usually used in the same contexts. For this reason, BERT's attention head (Vaswani, et al., 2017) possibly faced more similar words used in tandem with the word 'date' and had trouble distinguishing between the two definitions, leading to the lowest average accuracy of any word at 61.3796%.

The second anomaly (tie, $25 < x$ words) is strange due to its exceedingly low accuracy at 20% (with the average lowest being at 55.556). This might have been due to a problem with the sentences being too complex (as mentioned in the introduction) or the word's definition in the sentence being vague.

Conclusion:

To conclude, through the collection and thorough analysis of the results, it is quite clear that, with this range of words, there doesn't seem to be any correlation between the number of words in a sentence and BERT's accuracy to define a word in context. This suggests BERT does not have trouble with longer sentences and is not afflicted by the vanishing gradient problem, unlike ANNs and RNNs. Through the experiment, we can conclude that **a sentence's wordcount does not, to any extent, affect BERT's accuracy to distinguish between a word's meanings in context.** In theory, this is exactly what we would expect from BERT – contrasting the theoretical outcome of ANNs and RNNs. This means BERT is very adaptable and wouldn't have an issue with different lengthed text types such as social media. However, other variables previously mentioned might affect the accuracy and would likely be valuable to investigate.

Through my analysis, I realised it might have been beneficial to test more word bands with a bigger dataset or to verify BERT's variance in accuracy with a different test rather than defining a word in context; this would mean a wider range of data could be analysed. A further study should be conducted with a bigger and wider dataset to ensure the results found in this study are not limited in scope. Moreover, a further study should also be conducted comparing ANNs, RNNs, BERT, and other implementations of Transformers and their performance accuracy with respect to the number of words in a sentence. Additionally, as the type of word might have played an impact on the accuracy, further studies should be conducted with constant word types (i.e. only verbs) and with the word type being the independent variable.

Bibliography:

3Blue1Brown. (2017, November 3). Retrieved from Youtube: https://www.youtube.com/watch?v=Ilg3gGewQ5U

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning: A Textbook.*

Altman, N. S. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician, 46*(3), 175–185.

Baskaran, V. (2020, April 15). *What is Loss in Neural Nets? Is cost function and loss function are same ?* Retrieved from Medium: https://medium.com/@vinodhb95/what-is-loss-in-neural-nets-is-cost-function-and-loss-function-are-same-ef069a570e95

Bre, F., Juan, G. M., & Victor, F. D. (2017). Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. *ResearchGate*, 4.

Caulfield, B. (2009, December 16). *What's the Difference Between a CPU and a GPU?* Retrieved from Nvidia Blog: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.* arXiv.

Ciaburro, G., & Venkateswaran, B. (2017). *Neural Networks with R* . Packt.

Code Academy. (n.d.). *Perceptron*. Retrieved from Code Academy: https://www.codecademy.com/learn/machine-learning/modules/perceptron/cheatsheet

CodeEmporium. (2020, May 4). *BERT Neural Network - EXPLAINED!* Retrieved from Youtube: https://www.youtube.com/watch?v=xI0HHN5XKDo

CodeEmporium. (2020, January 13). *Transformer Neural Networks - EXPLAINED! (Attention is all you need)*. Retrieved from Youtube: https://www.youtube.com/watch?v=TQQlZhbC5ps

Crypto1. (2020, October 2). *How Does the Gradient Descent Algorithm Work in Machine Learning?* Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/

Dai, A. M., & Le, Q. V. (2015). Semi-supervised Sequence Learning. *In Advances in neural information processing systems*, 3079–3087.

deeplizard. (2018, March 24). Retrieved from Youtube: https://www.youtube.com/watch?v=qO_NLVjD6zE

Devlin, J., & Chang, M.-W. (2018, November 2). *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing*. Retrieved from Google AI Blog: https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for. *arXiv*.

Doshi, K. (2021, June 3). *Transformers Explained Visually — Not Just How, but Why They Work So Well*. Retrieved from Towards Data Science: https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3

Fix, E., & Hodges, J. L. (1951). *Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties.* Universal City: USAF School of Aviation Medicine, Randolph Field.

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000, June 22). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature, 405*, pp. 947–951.

Hendrycks, D., & Gimpel, K. (2016, June 27). Gaussain Error Linear Units (GELUs). *arXiv*.

Hochreiter, S., & Schmidhuber, J. (1997, November 15). Long short-term memory. *Neural Computation, 9*(8), pp. 1735–1780. Retrieved from https://www.bioinf.jku.at/publications/older/2604.pdf

Hwang, J. T., & Ding, A. A. (1997). Prediction Intervals for Artificial Neural Networks. *Journal of the American Statistical Association*, 748-757.

IBM Cloud Education. (2020, September 14). *Recurrent Neural Networks*. Retrieved from IBM: https://www.ibm.com/cloud/learn/recurrent-neural-networks#toc-what-are-r-btVB33l5

Jadidi, A., Menezes, R., Souza, N. d., & Lima, A. d. (2018). A Hybrid GA–MLPNN Model for One-Hour-Ahead Forecasting of the Global Horizontal Irradiance in Elizabeth City, North Carolina. *ResearchGate*, 8.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? *IEEE, 12th International Conference on Computer Vision*, 2146-2153.

Johri, P., Verma, J. K., & Paul, S. (2020). *Applications of Machine Learning.* Springer.

Joshi, A. J. (1991). Natural Language Processing. *Science, 253(5025)*, 1242–1249.

Kapoor, H. (2022). *Wide vs Deep vs Wide & Deep Neural Networks.* Retrieved from Kaggle: https://www.kaggle.com/hkapoor/wide-vs-deep-vs-wide-deep-neural-networks

Kobran, D., & Banys, D. (2019). *Activation Function.* Retrieved from AI WIKI: https://docs.paperspace.com/machine-learning/wiki/activation-function

Li, F.-F., Johnson, J., & Yeung, S. (2017, April 13). *Stanford CS231 - Lecture 4: Backpropagation and Neural Networks.* Retrieved from Stanford CS231: http://cs231n.stanfoard.edu/slides/2017/cs231n_2017_lecture4.pdf

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning*, *30.*

Mcculloch, W. S., & Pitts, W. (1990). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology, 52*(1/2), 99-115.

McGonagle, J., Shaikouski, G., Williams, C., Hsu, A., Khim, J., & Miller, A. (n.d.). *Backpropagation*. Retrieved from Brilliant: https://brilliant.org/wiki/backpropagation/

Microsoft. (n.d.). *CORREL function*. Retrieved from Microsoft Support: https://support.microsoft.com/en-us/office/correl-function-995dcef7-0c0a-4bed-a3fb-239d7b68ca92

Microsoft. (n.d.). *Microsoft Support*. Retrieved from PEARSON function: https://support.microsoft.com/en-us/office/pearson-function-0c3e30fc-e5af-49c4-808a-3ef66e034c18

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *International Conference on Machine Learning*, (pp. 807–814).

Nayak, P. (2019, October 15). *Understanding searches better than ever before*. Retrieved from Google Blog: https://blog.google/products/search/search-language-understanding-bert/

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2*(11), 559-572.

Phi, M. (2018, August 26). Retrieved from Youtube: https://www.youtube.com/watch?v=LHXXI4-IEns

Phi, M. (2020, May 1). *Illustrated Guide to Transformers- Step by Step Explanation*. Retrieved from Towards Data Science: https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

Ravichandiran, S. (2021). *Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT.* Packt .

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 1. Retrieved from https://www.nature.com/articles/323533a0

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986, October 9). Learning representations by back-propagating errors. *Nature, 323*, pp. 533–536.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Kaiser, Ł. (2017). Attention Is All You Need. *arXiv*.

West, M. (2020, June 11). *Explaining Recurrent Neural Networks.* Retrieved from Bouvet: https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks

Wilson, B. (1998). Retrieved from The Machine Learning Dictionary: http://www.cse.unsw.edu.au/~billw/mldict.html#activnfn

Winston, P. H. (2015). *Lecture 12A: Neural Nets.* Retrieved from MIT OPEN COURSEWARE: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-12a-neural-nets/

Word Embeddings. (n.d.). Retrieved from Papers With Code: https://paperswithcode.com/task/word-embeddings/codeless

Appendix:

Appendix 1 – Diagram of how gradient descent works:

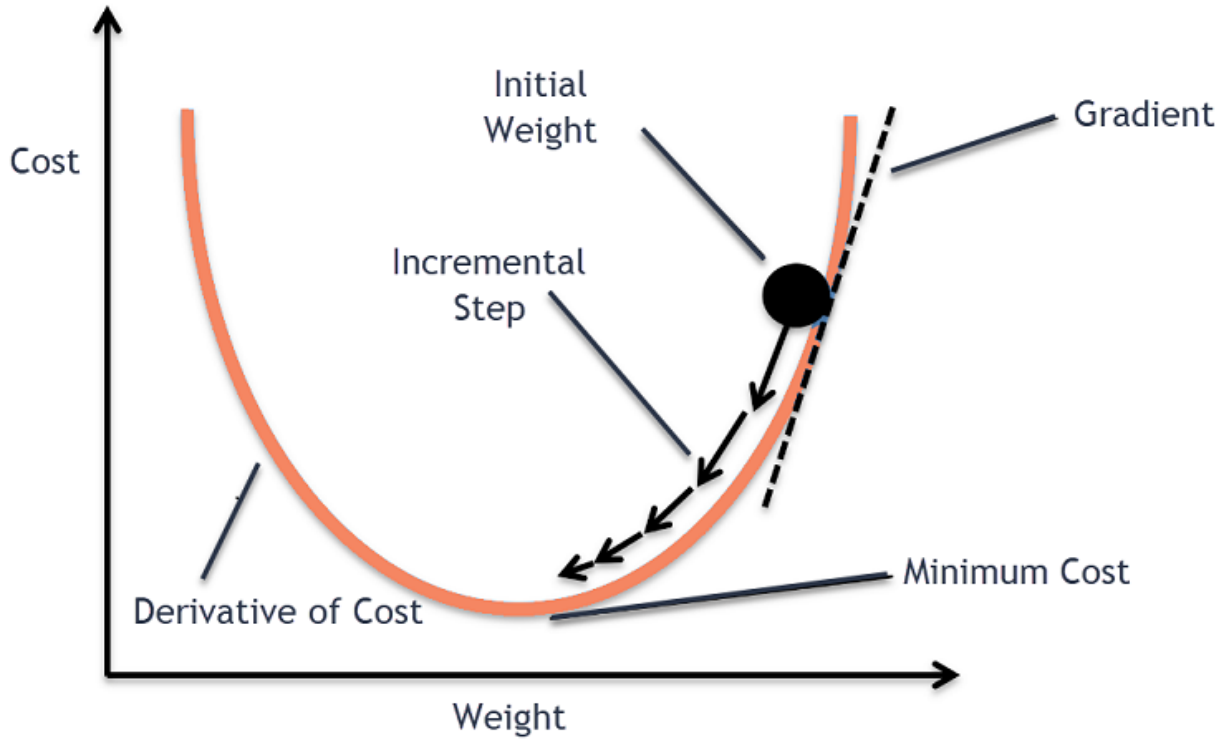

Figure 14: (Crypto1, 2020)

Appendix 2 – Self-written Code for Data pre-processing:

```
import os

import csv

import re

fileNames = []


mainPath = '' ##directory containing all csv files

## get all filenames and add to fileNames array

for file in [f for f in os.listdir(mainPath) if f.endswith('.csv')]:
```

```
    fileNames.append(os.path.join(file))
print("filenames: ")
print(fileNames)


## for every file
for i in range(len(fileNames)):
    ## CSV filename
    fileName = fileNames[i]
    fileNameForWrite = fileName[:-4] + "Edited_25+.csv"
    print(fileNameForWrite)
    print("current filename: ")
    print(fileName)
    ## initialise
    rows = []
    sentenceLength = []
    ## read csv
    with open(fileName, "r") as csvfile:
        ## CSV reader
        reader = csv.reader(csvfile)
        ## for every row in the csv
        print(reader)
        count = 0
        for row in reader:
            ## make sure to include the headers row
            if ((count == 0) or ((25 < len(re.findall(r'\w+', row[0]))) )):
                ## append rows to rows
                rows.append(row)
            count += 1
        print(sentenceLength)
```

```
        ## get total number of rows
        print("Total number of rows: %d" % (reader.line_num))



    with open(fileNameForWrite, 'w') as writeFile:
        print(rows)
        writer = csv.writer(writeFile)
        writer.writerows(rows)
```

Appendix 3 – example of amended code:

```
import pandas as pd

import numpy as np

!pip install -e git+https://github.com/negedng/bert-
embedding#egg=bert_embedding

from sklearn.decomposition import PCA

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import LeaveOneOut

from sklearn import model_selection

import matplotlib.pyplot as plt

%matplotlib inline

url = "https://gist.githubusercontent.com/negedng/91c4cb1335a4b2bbc3fcf7ab
a3c6ecda/raw/3fc0b641caa95ebba986b1313522f59ecd757a2b/ducks2019type3.csv"

df1 = pd.read_csv(url)

from bert_embedding import BertEmbedding

bert_embedding = BertEmbedding(max_seq_length=35)

embs = bert_embedding(df1['Sentence'], filter_spec_tokens=False,)

duck_embs = []

for row in embs:

    try:

        duck_index = row[0].index('duck')

        duck_embs.append(row[1][duck_index])

    except ValueError:

        print(len(row[0]))
```

```
duck_embs = np.array(duck_embs)

duck_embs.shape

duck_pca = PCA(n_components=2).fit_transform(duck_embs)

duck_pca.shape

cdict = {0: 'red', 1: 'blue', 2: 'green'}

markers = {0: 'o', 1: '+', 2: 'v'}

labels = {0: 'bird', 1: 'verb', 2: 'fabric'}

scatter_x = duck_pca[:,0]

scatter_y = duck_pca[:,1]

fig, ax = plt.subplots(figsize=(10, 7))

for g in np.unique(df1.Type):

    ix = np.where(df1.Type == g)

    ax.scatter(scatter_x[ix], scatter_y[ix], c = cdict[g], label = labels[
g], s=100, marker=markers[g])

ax.legend(prop={'size': 12})

plt.show()

loocv = model_selection.LeaveOneOut()

model = KNeighborsClassifier(n_neighbors=8)

results = model_selection.cross_val_score(model, duck_embs, df1.Type, cv=l
oocv)

print("Accuracy: %.3f%% (STDev %.3f%%)" % (results.mean()*100.0, results.s
td()*100.0))
```

Appendix 4 – Table of data/results:

	Number of Sentences used as data						
Word	**$0 < x \leq 8$ words**	**$8 < x \leq 15$ words**	**$15 < x \leq 20$ words**	**$20 < x \leq 25$ words**	**$25 < x$ words**	**Average**	**Total**
address	17	20	22	15	15	**17.8**	**89**
bark	11	27	15	11	19	**16.6**	**83**
fall	19	30	15	9	22	**19.0**	**95**
feet	18	44	16	9	14	**20.2**	**101**
bat	11	34	25	10	10	**18.0**	**90**
date	16	42	17	10	16	**20.2**	**101**
right	20	22	17	9	21	**17.8**	**89**
tie	28	31	17	12	10	**19.6**	**98**
Average	**17.5**	**31.3**	**18.0**	**10.6**	**15.9**		
Total	**140**	**250**	**144**	**85**	**127**		

Figure 15 Coloured Table of Number of Sentences used as data