

Independent Study Project Report

A Report on the Use of Deep Learning Models in Natural Language Processing.

Ojo Fisayo

Department of Computer Science
University of Denver

Chapter 1. Problem Scope

1.1. Background

Natural Language Processing (NLP) is an important aspect of artificial intelligence and continues to find new methods to enable computers gather data, interpret and manipulate human languages. The language extracted are then used as data for sentiment analysis, information extraction, machine translation and many other technological advancements in our world today.

Like many other fields of computation, Natural Language Processing is not without its own fair of challenges. Human Languages remain a very complex subject area that changes with each generation. Language continues to evolve from Pictographs to Cuneiform, to the Phoenician Alphabet and later the Greek and Latin alphabets [1]. The changes in language also reflects in syntactic structures, idioms and how we express sarcasm. Earlier approaches to Natural Language Processing were mostly rule-based which meant they used manually coded linguistic rules that failed to capture the complexities in language structure and form. Thus, these traditional methods proved inefficient and were not successful at characterizing and inferring meaning from language data.

Advancements in computer science birthed improvements in the field of natural language processing particularly deep learning models. This is because deep learning models are able to learn features from data and their architecture is formed using several interconnected layers which make them suitable for complex learning tasks. The deep learning models include: Convolutional Neural Networks, Recurrent Neural Networks and Transformers. Convolutional Neural Networks(CNNs), for the longest time were known strictly for their image processing prowess, but more recently they are being used for Natural Language Processing tasks. This is because they can identify patterns in data which made them ideal for tasks like sentence classification and sentiment analysis. To make this possible, textual data is treated as a 1D image and Convolutional Neural Networks are able to establish patterns and extract semantic information from texts.

Recurrent Neural Networks(RNNs) on the other hand were designed and are trained to deal with sequential data inputs. Sequential data inputs include: words, sentences or time series data in which sequential components relate with one another based on complex semantics and syntax rules. Recurrent Neural networks attempt to mimic the way

humans would perform sequential data conversions, for example translating text across languages [2]. Recently, RNNs are getting exchanged in favor of transformer-based artificial intelligence and Large Language Models(the most popular of them would be chat gpt). This is because LLMs and transformers tend to be more efficient in sequential data processing and have brought about serious advancement and breakthroughs in the world of NLP such that it has necessitated further research in this field.

One of the reasons why the RNN shines with sequential text is because it was specially designed to handle sequences of data. It can also remember information from previous inputs in the sequence and this allows it make context relevant correlations with textual data. Despite all the great characteristics of RNNs, its downside is that it struggles with long term dependencies which is caused by the vanishing gradient problem. The vanishing gradient problem occurs during the training of deep neural networks and it occurs when the gradients used to update the network become extremely small or vanish as they are backpropagated from output layers to older layers.

In trying to solve the vanishing gradient problem, the Long Short Term Memory Networks were developed. It was found to be able to learn more complex and long range patterns. Transformers were eventually developed and found to be better than both LSTMs and RNNs due to their parallel processing abilities and their ability to capture long range dependencies. They revolutionized NLP because their parallel processing of sequences which also makes them significantly faster than LSTMs and RNNs. They also utilize the self attention mechanism that works by giving different priorities or importance to words in a sentence which helps it learn global dependencies. Transformers are what form the foundation of many renowned NLP models like GPT(Generative Pretrained Transformer) and BERT(Bidirectional Encoder Representations from Transformers).

Although the field has advanced throughout the years, deep learning models in NLP is still plagued with challenges associated with language evolving such as irony, sarcasm and cultural contexts that may exist depending on the use of language across regions. There is also the need to gather large amounts of data which is also a difficult problem and additionally there is the environmental concerns arising from substantial use of computational resources while training the models.

Regardless of the challenges the field of Natural Language processing is faced with, billions of people across the world benefit from the research and integration into so many applications we use today. A great example is chat GPT and how it has become a top choice for searching for information, learning support etc. Research in this field is key because its continual advancement will significantly improve our quality of life.

1.2. Description of the Project

The Project scope began with using the NLTK library and using it to carry out simple analysis on text data to determine the tone.

The project scope evolved to using neural networks to determine if two documents or texts are related and comparing their results when analyzing data. I utilized the Convolutional Neural Network, Recurrent Neural Network and Transformers and will go on to discuss my step by step process of how this was done:

1.2.1. Data Processing.

Finding good data that gave good results and was useful with respect to the project was difficult. I began by forming my own data by downloading texts on oil exploration. Then I carried out data preprocessing which involved cleaning the data to strip punctuation and labeling the data etc.

After this I tried to use the processed data but it was not sufficient to train the model.

I then changed direction by going through datasets online and trying them out and observing things like accuracy rate and loss function to decide which was best to modify and fit my objective.

Some of the datasets I utilized in the process are:

- <https://www.kaggle.com/datasets/kanhataak/task-finding-semantic-textual-similarity>
- <https://www.kaggle.com/competitions/movie-review-sentiment-analysis-kernels-only/data>
- <https://github.com/facebookresearch/SentEval>
- <https://github.com/brmson/dataset-sts>
- <https://www.kaggle.com/c/learn-ai-bbc/data>

I also used GloVe (Gloval Vectors for Word Representation), which was created by a group from Stanford: <https://nlp.stanford.edu/projects/glove/> I used it because it had pre-trained word vectors and can capture semantic relationship between words which helps with more accurate predictions. When its combined with a different dataset in this case the data set I sourced from Facebook's research repo and modified. It improves model's performance because the model can leverage the semantic relationships in GloVe and also the patterns observed in the Facebook dataset. It also helps with some form of transfer learning across domains since the corpus on which GloVe was trained is different from the Facebook data set. Another reason its useful is because it is super efficient and provides me with a pre-trained layer that can be fine-tuned and adapted to my new task.

All of this made combining GloVe with my dataset the ideal thing to do in line with my objectives.

1.2.2. Data Modification/Wrangling

I ended up settling to use Sentence Evaluation data from facebook research, which is the third data set in my earlier list because it gave me the best results after modification and I could use it to determine if two sentences or texts have some relationship.

I modified the data by manually adjusting the `score` column, I also had to change the format significantly; I wrote Python code in the `"add_column.py"` file in my `More_IS` codebase for modifying the columns of the dataset. After which I had to go through it carefully and do a manual clean up for each item to ensure the format was right, I also populated with additional data of my own and took out data rows as needed. The process to shape up the data for use took a while, considering I did this for all the data sets I tried out too before I finally got great results with the Facebook data. I tried to comment out most of my attempts within the same file so that progress is visible.

1.3. Motivation behind my methodology

After a series of conversation with Dr. Scott last quarter I set out to learn more about NLP beyond readily available library Implementations. I really wanted to understand on a deeper level techniques that enable machines interpret and translate our human languages. How were computers able to observe the patterns through code and how do we currently teach them to do so. I then learnt about Convolutional Neural Networks being able to do so. I was further fascinated because I have always known CNNs to be used for image related tasks and finding out it could be used for Natural Language Processing deeply surprised me and after my check-in with Dr. Scott we agreed it was great to see how Neural Networks can be used in NLP with the initial topic around crude oil and its environmental impacts.

I set out on this path but was not getting great results using my own data since there was no publicly available dataset and I had to form my own data to train my CNN Model. I used PDFs which I converted to text files, labeled, and preprocessed on my own, but it was not sufficient to give me great results. The files related to this can be found in `CNN_NLP.py` in the `More_IS` repo of my work. I also decided to expand the scope of the project to using RNN and Bidirectional Encoder Representations from Transformer (BERT) and learning how they can be used for NLP. I changed the scope to using and learning about various neural networks and understanding how they can be used to find relationships between texts. Since the basis of my initial idea surrounding oil involved tracing relationships between press releases and citizen opinion about crude oil in Nigeria. I believe that my new project focus reflected my commitment to deeply understanding deep learning and ignited my desire to continue to learn beyond the independent study in order to contribute to the

evolution of NLP.

CNN was a great choice to explore because it could identify image patterns really well and I was curious to see how this played out with texts. RNNs on the other hand I was curious to see how they performed since they are lauded for their sequential data processing performance. Their ability to remember was also an added advantage and I was excited to explore by reading and also practicing code relating to this.

BERT on the other hand did not sound as traditional to me, but I really explored the Hugging face platform and read several articles about BERT and even tried out some demos, I wanted to really implement this too and see how it performs and also feed my curiosity by reading and watching videos around this. I was particularly marvelled by its ability to understand the different context of words in texts. I was glad to follow my curiosity and learn about the technologies that continue to push the boundaries of what we thought was possible in Natural Language Processing and in the future I hope to contribute in no small way to the field because I believe getting the computer to understand human languages and also produce human language in different form represents serious advancements in the field of computer science and will have a ripple effect on other fields and enhance research in other subject areas.

1.4. The Review of Existing Literature that Backs My Approach

Deep Learning Models have been researched recently because they have proved to be much more promising than previous methods. The deep learning models I looked into as mentioned earlier were the Convolutional Neural Network, Recurrent Neural Network and Bidirectional Encoder Representation from Transformers. This section looks into existing literature that supports the usage of the models I listed for sentence classification tasks.

Convolutional Neural Networks are well known for their ability to recognize patterns in data and in 2014 a researcher, Kim Yoon was able to demonstrate that a CNN with just a simple layer can actually produce great results when tested on sentence classification tasks [3]. I did read his paper and looked into his work and implementations and tried to implement it and observe myself how this worked with one of my initial data sets and the code depicting this is in the file named `cnn-for-sentence-classification`. It was able to show that Convolutional Neural Network actually had the ability to fully capture semantic and syntactic patterns in texts and thus was a good model for NLP tasks like sentence classification.

Recurrent Neural Networks on the other hand are well known to be very effective with texts and did not have the shock factor CNN gave me when I found out it could be used for text. There is notable research done by Lai et al in 2005 that demonstrated the ability of RNNs to capture long term dependencies in text and actually give even better results than previously known methods [4]. This pointer and similar articles made me realize that

it was also worth exploring in my studies.

BERT has changed the world of Natural Language Processing by enabling the pre training of language representations that can be adjusted for specific tasks. The model was introduced in 2018 by these three scientists from Cornell: Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. It was designed to allow the pre training of deep bidirectional representations from unlabeled text by conditioning together on both left and right context in all layers [5]. Also analysis point that BERT anages to perform previous models over time.

My research showed that all three models are useful and can be used for Natural Language Processing tasks. But the context of what you want to achieve with them have to be factored in. There has to be consideration on a case by case basis for the data, model complexity and computational resources. I will be talking extensively on my experience regarding computational resources with respect to one of the models in question.

There are also growing and very valid concerns regarding ethical considerations around bias and fairness. I did learn a lot about this in the Fall semester in the Introduction to Artificial Intelligence(AI) course and I am not surprised I did come across material discussing AI fairness with respect to NLP. I think although this was not the focus of my independent study but it is something always consider whenever innovating or working with technology that has serious impacts. In conclusion, there is a lot of research that shows the validity and importance of using CNN, RNN and BERT for sentence classification tasks in NLP.

Chapter 2. CONCLUSION

The repo for my work done can be found on github at:

2.1. Results of my project which involved using Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and the BERT model for sentence classification.

I was able to use all three deep learning models and learn about how they work on a deeper level, specifically the Convolutional Neural Network (CNN), transformers, and Recurrent Neural Networks (RNNs). I will be discussing the various challenges with them and the results produced by each of them. The libraries I used were: `keras`, `tensorflow`, `numpy`, `pandas`, `scikit-learn`, `pickle`, `nlTK`, `os`. The files that cover what I did were: `CNN_final`, `CNN_NLP`, `cnn-for-sentence-classification`.

I had issues earlier trying to put all my work on my personal GitHub, but I was able to move it to my school's GitLab account. The work for these projects can be found at https://git.cs.du.edu/fisojo/more_is and https://git.cs.du.edu/fisojo/more_is2. To get it to work, they should all be downloaded into the same folder after cloning the repos.

2.1.1. Convolutional Neural Networks.

Earlier in this report I already discussed the motivation for using the CNN and I will be sharing my results and observation.

First of all I started out with working with the movie reviews data set for learning purposes and here is what the first 30 epochs results look like:


```

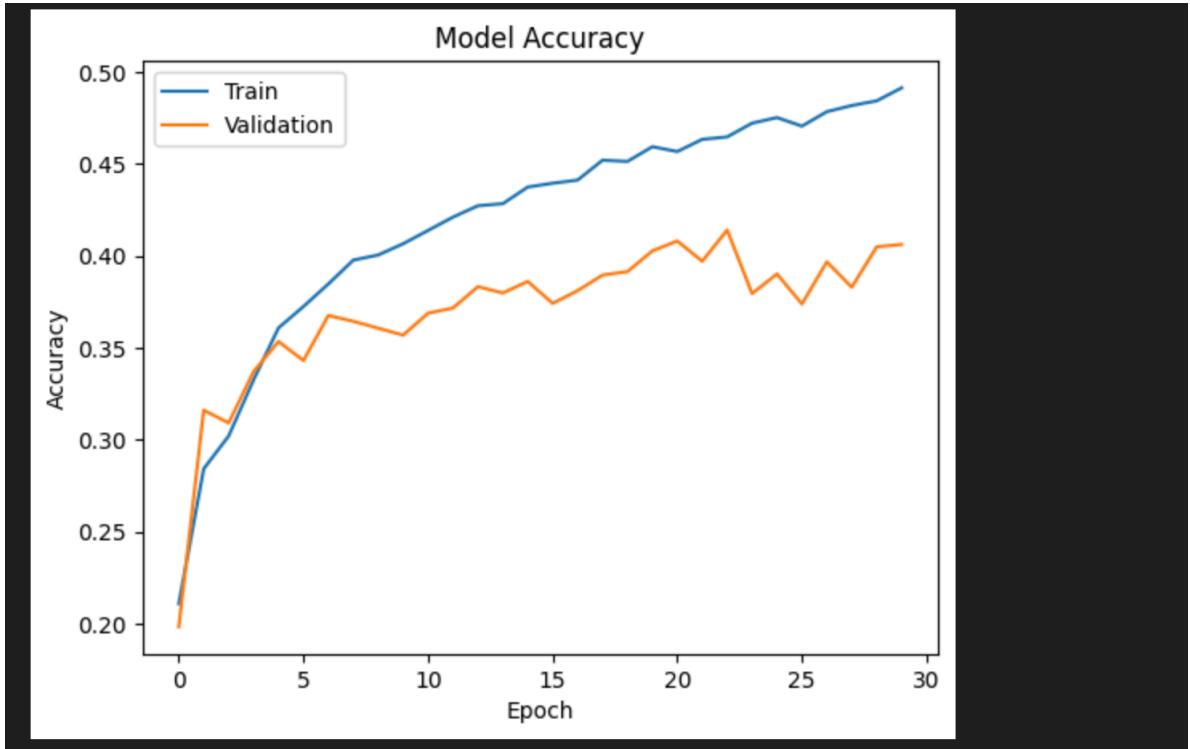
Epoch 1/30
896/896 ————— 12s 13ms/step - accuracy: 0.2021 - loss: 2.0656 - val_accuracy: 0.1982 - val_loss: 1.6294
Epoch 2/30
896/896 ————— 11s 13ms/step - accuracy: 0.2741 - loss: 1.6209 - val_accuracy: 0.3161 - val_loss: 1.6187
Epoch 3/30
896/896 ————— 12s 13ms/step - accuracy: 0.2941 - loss: 1.6013 - val_accuracy: 0.3091 - val_loss: 1.5633
Epoch 4/30
896/896 ————— 12s 13ms/step - accuracy: 0.3255 - loss: 1.5574 - val_accuracy: 0.3371 - val_loss: 1.5929
Epoch 5/30
896/896 ————— 12s 14ms/step - accuracy: 0.3561 - loss: 1.5171 - val_accuracy: 0.3534 - val_loss: 1.5392
Epoch 6/30
896/896 ————— 13s 14ms/step - accuracy: 0.3742 - loss: 1.4833 - val_accuracy: 0.3431 - val_loss: 1.5590
Epoch 7/30
896/896 ————— 13s 14ms/step - accuracy: 0.3879 - loss: 1.4540 - val_accuracy: 0.3676 - val_loss: 1.5360
Epoch 8/30
896/896 ————— 13s 14ms/step - accuracy: 0.3939 - loss: 1.4498 - val_accuracy: 0.3644 - val_loss: 1.5378
Epoch 9/30
896/896 ————— 13s 15ms/step - accuracy: 0.3980 - loss: 1.4509 - val_accuracy: 0.3607 - val_loss: 1.5751
Epoch 10/30
896/896 ————— 14s 15ms/step - accuracy: 0.4089 - loss: 1.4381 - val_accuracy: 0.3569 - val_loss: 1.5480
Epoch 11/30
896/896 ————— 13s 15ms/step - accuracy: 0.4180 - loss: 1.4112 - val_accuracy: 0.3688 - val_loss: 1.5651
Epoch 12/30
896/896 ————— 13s 15ms/step - accuracy: 0.4215 - loss: 1.4084 - val_accuracy: 0.3717 - val_loss: 1.5231
Epoch 13/30
...
Epoch 29/30
896/896 ————— 12s 13ms/step - accuracy: 0.4895 - loss: 1.3479 - val_accuracy: 0.4050 - val_loss: 1.6132
Epoch 30/30
896/896 ————— 12s 14ms/step - accuracy: 0.4946 - loss: 1.3481 - val_accuracy: 0.4062 - val_loss: 1.6425

```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings...](#)

First 30 epochs for CNN

and here is the graph that depicts that:



Corresponding graph plot.

I then went on to change embedding layer and here are the results:

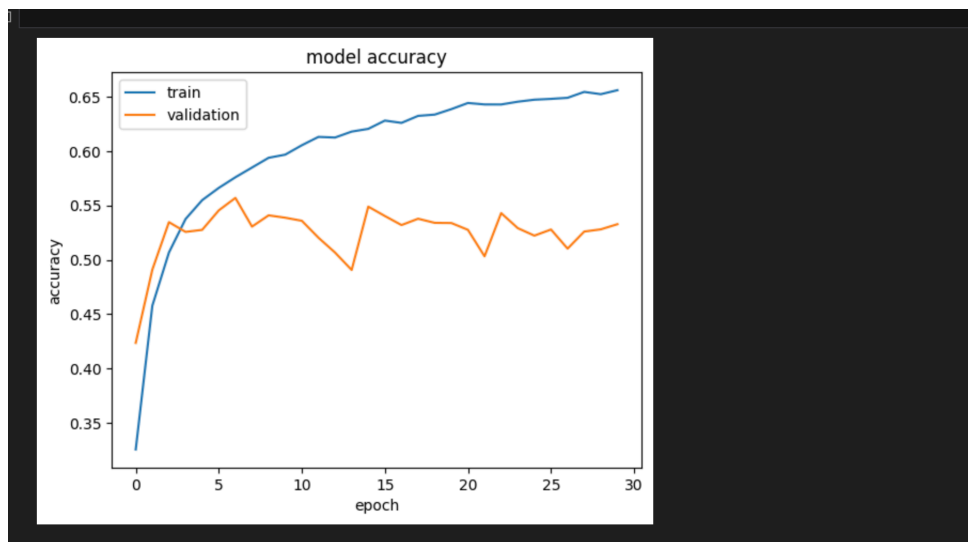
```

Epoch 1/30
796/796 ————— 15s 18ms/step - accuracy: 0.2878 - loss: 2.2601 - val_accuracy: 0.4236 - val_loss: 1.8221
Epoch 2/30
796/796 ————— 14s 17ms/step - accuracy: 0.4397 - loss: 1.8032 - val_accuracy: 0.4910 - val_loss: 1.7037
Epoch 3/30
796/796 ————— 13s 16ms/step - accuracy: 0.5074 - loss: 1.7072 - val_accuracy: 0.5346 - val_loss: 1.6395
Epoch 4/30
796/796 ————— 14s 17ms/step - accuracy: 0.5323 - loss: 1.6564 - val_accuracy: 0.5257 - val_loss: 1.6511
Epoch 5/30
796/796 ————— 13s 17ms/step - accuracy: 0.5583 - loss: 1.6108 - val_accuracy: 0.5276 - val_loss: 1.8408
Epoch 6/30
796/796 ————— 15s 18ms/step - accuracy: 0.5656 - loss: 1.5892 - val_accuracy: 0.5455 - val_loss: 1.6101
Epoch 7/30
796/796 ————— 15s 18ms/step - accuracy: 0.5769 - loss: 1.5397 - val_accuracy: 0.5570 - val_loss: 1.6103
Epoch 8/30
796/796 ————— 14s 17ms/step - accuracy: 0.5887 - loss: 1.5199 - val_accuracy: 0.5306 - val_loss: 1.6228
Epoch 9/30
796/796 ————— 13s 16ms/step - accuracy: 0.5981 - loss: 1.5087 - val_accuracy: 0.5409 - val_loss: 1.7022
Epoch 10/30
796/796 ————— 13s 16ms/step - accuracy: 0.5997 - loss: 1.4874 - val_accuracy: 0.5387 - val_loss: 1.7388
Epoch 11/30
796/796 ————— 13s 17ms/step - accuracy: 0.6110 - loss: 1.4839 - val_accuracy: 0.5359 - val_loss: 1.5701
Epoch 12/30
796/796 ————— 13s 16ms/step - accuracy: 0.6142 - loss: 1.4723 - val_accuracy: 0.5203 - val_loss: 1.6707
Epoch 13/30
...
Epoch 29/30
796/796 ————— 13s 17ms/step - accuracy: 0.6621 - loss: 1.3370 - val_accuracy: 0.5280 - val_loss: 1.7467
Epoch 30/30
796/796 ————— 13s 17ms/step - accuracy: 0.6655 - loss: 1.3380 - val_accuracy: 0.5328 - val_loss: 1.7727

```

30 epochs for CNN after changing embedding layer

Below is the corresponding graph:



Accuracy graph after changing embedding layer



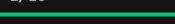







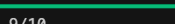

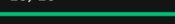
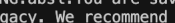
Loss graph after changing embedding layer

The observation is that accuracy values improve when we tweak the embedding layer and I applied this knowledge later on.

For my first self-created dataset with oil data gotten from the web here was my result

flatten_1 (Flatten)	(None, 1444736)	0	max_pooling1d_1[0][0]
concatenate (Concatenate)	(None, 2889472)	0	flatten[0][0], flatten_1[0][0]
dense (Dense)	(None, 10)	28,894,730	concatenate[0][0]
dense_1 (Dense)	(None, 1)	11	dense[0][0]

Total params: 29,154,405 (111.22 MB)
 Trainable params: 29,154,405 (111.22 MB)
 Non-trainable params: 0 (0.00 B)

Epoch 1/10
 8/8  36s 3s/step - accuracy: 0.5869 - loss: 55.5486 - val_accuracy: 0.3833 - val_loss: 0.6936
 Epoch 2/10
 8/8  32s 4s/step - accuracy: 0.3767 - loss: 0.6937 - val_accuracy: 0.3833 - val_loss: 0.6935
 Epoch 3/10
 8/8  33s 4s/step - accuracy: 0.3762 - loss: 0.6935 - val_accuracy: 0.3833 - val_loss: 0.6932
 Epoch 4/10
 8/8  42s 5s/step - accuracy: 0.5313 - loss: 0.6931 - val_accuracy: 0.6167 - val_loss: 0.6927
 Epoch 5/10
 8/8  32s 4s/step - accuracy: 0.6172 - loss: 0.6925 - val_accuracy: 0.6167 - val_loss: 0.6922
 Epoch 6/10
 8/8  33s 4s/step - accuracy: 0.6233 - loss: 0.6919 - val_accuracy: 0.6167 - val_loss: 0.6916
 Epoch 7/10
 8/8  33s 4s/step - accuracy: 0.5973 - loss: 0.6916 - val_accuracy: 0.6167 - val_loss: 0.6910
 Epoch 8/10
 8/8  38s 4s/step - accuracy: 0.6285 - loss: 0.6905 - val_accuracy: 0.6167 - val_loss: 0.6904
 Epoch 9/10
 8/8  40s 5s/step - accuracy: 0.5953 - loss: 0.6907 - val_accuracy: 0.6167 - val_loss: 0.6899
 Epoch 10/10
 8/8  35s 4s/step - accuracy: 0.6561 - loss: 0.6883 - val_accuracy: 0.6167 - val_loss: 0.6893
 WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
 2/2  1s 655ms/step - accuracy: 0.6507 - loss: 0.6879
 Test Accuracy: 61.67%
 1/1  0s 78ms/step
 The documents are related with a confidence of 0.51

Epochs for CNN on self-created data.

The result was not so great and after having a conversation with other people with deeper expertise in ML, we concluded that the issue was that my data was not sufficient enough to train the model and it was best to try out different data set and manipulate them by adding and removing whatever I needed. So I did look into the datasets I mentioned earlier in this report and the final one I tweaked was the facebook research data set. Here is what my results looked like and the code file is named *CNN_{final}* :

```

print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')

[51] ✓ 8.4s
... Epoch 1/10
64/64 ————— 2s 16ms/step - accuracy: 0.5769 - loss: 0.6856 - val_accuracy: 0.5703 - val_loss: 0.6766
Epoch 2/10
64/64 ————— 1s 12ms/step - accuracy: 0.6240 - loss: 0.6350 - val_accuracy: 0.5820 - val_loss: 0.6533
Epoch 3/10
64/64 ————— 1s 11ms/step - accuracy: 0.6845 - loss: 0.5782 - val_accuracy: 0.6035 - val_loss: 0.6775
Epoch 4/10
64/64 ————— 1s 13ms/step - accuracy: 0.7767 - loss: 0.4805 - val_accuracy: 0.5938 - val_loss: 0.7014
Epoch 5/10
64/64 ————— 1s 11ms/step - accuracy: 0.8467 - loss: 0.3850 - val_accuracy: 0.6035 - val_loss: 0.7566
Epoch 6/10
64/64 ————— 1s 11ms/step - accuracy: 0.8728 - loss: 0.3002 - val_accuracy: 0.5840 - val_loss: 0.8608
Epoch 7/10
64/64 ————— 1s 11ms/step - accuracy: 0.9023 - loss: 0.2480 - val_accuracy: 0.5938 - val_loss: 1.0111
Epoch 8/10
64/64 ————— 1s 11ms/step - accuracy: 0.9188 - loss: 0.2173 - val_accuracy: 0.5801 - val_loss: 1.0330
Epoch 9/10
64/64 ————— 1s 11ms/step - accuracy: 0.9083 - loss: 0.2088 - val_accuracy: 0.5859 - val_loss: 1.1176
Epoch 10/10
64/64 ————— 1s 11ms/step - accuracy: 0.9447 - loss: 0.1630 - val_accuracy: 0.5723 - val_loss: 1.2064

```

Epochs for CNN on modified facebook research data

Here is what similarity scores also looked like:

```

def predict_similarity(text1, text2, model, tokenizer, sequence_length):
    seq1 = tokenizer.texts_to_sequences([text1])
    seq2 = tokenizer.texts_to_sequences([text2])

    data1 = pad_sequences(seq1, maxlen=sequence_length, padding='post', truncating='post')
    data2 = pad_sequences(seq2, maxlen=sequence_length, padding='post', truncating='post')

    prediction = model.predict([data1, data2])

    return prediction[0][0]

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'^a-zA-Z\s', '', text)
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [w for w in tokens if not w in stop_words]
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(w) for w in filtered_tokens]
    preprocessed_text = ' '.join(stemmed_tokens)
    return preprocessed_text

text1 = "Reading daily broadens minds and worlds."
text2 = "Books: gateways to wisdom and dreams"
sequence_length = 52
similarity_score = predict_similarity(text1, text2, model, tokenizer, sequence_length)
print(f"Similarity score: {similarity_score}")

✓ 0.0s
1/1 ————— 0s 12ms/step
Similarity score: 0.748603105545044

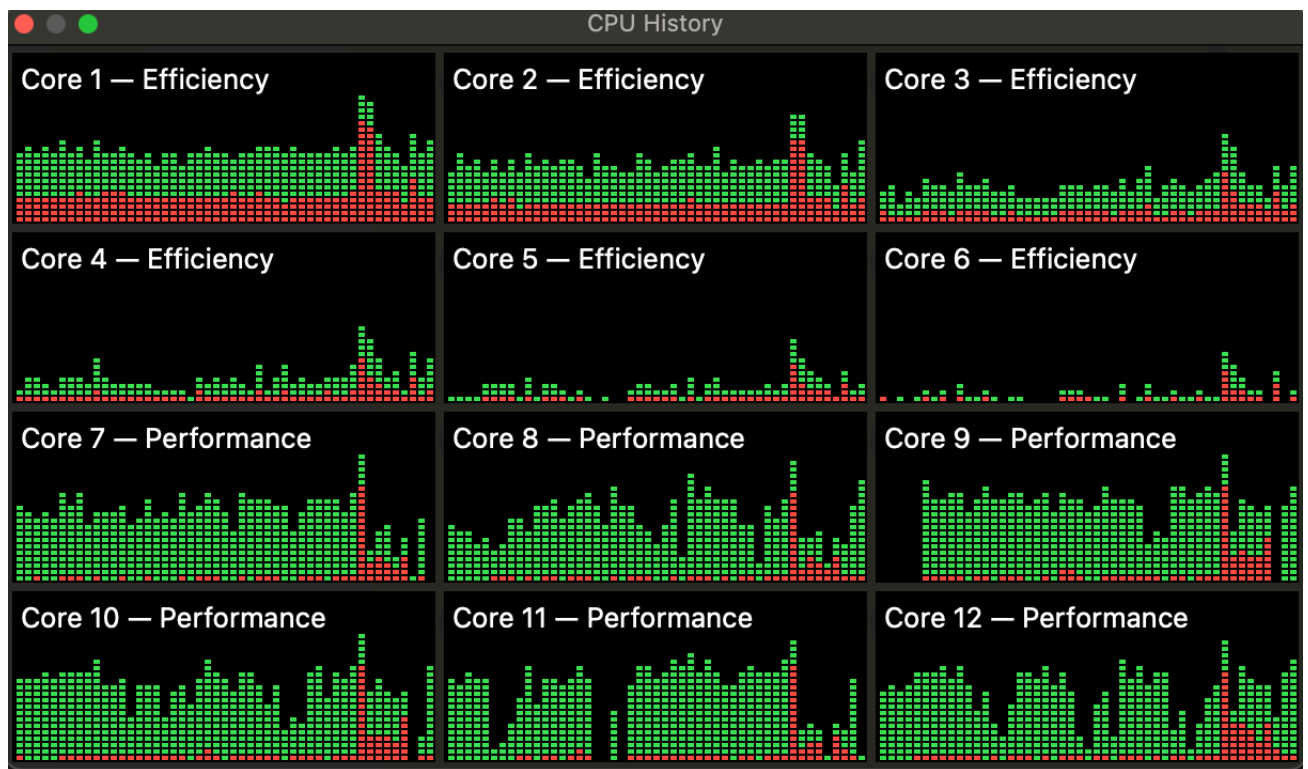
```

Similarity Scores

2.1.2. Recurrent Neural Networks.

For the recurrent neural network, I did some reading and studying about it and tried implementing it to compare and the major issue was that it consumed just too many computational resources. I used mostly the Tensorflow library and the RNN code is in the `RNNNLP.py` file.

Here is what that looked like when I tracked it it crashed my computer, I did further try to run it on google collab, both gpu and CPU and collab also complained about memory allocation:



CPU utilization for RNN.

2.1.3. Bidirectional Encoder Representations from Transformers

The BERT model was really great and I can see why research shows that it really represents a grand breaking breakthrough in the field of Natural Language Processing. It was a marvel to work with and implement and I was really impressed considering I did not have to do any extra training for it and it still gave really great results. I utilized the all-MiniLM-L6-v2 model and the SentenceTransformer library. The work for this is in the `Transformers.py` file.

I did this in context of data on oil and its environmental effect which is still the same

idea as sentence classification and the BERT Model worked really well in telling me if documents have the same idea or similar information regarding environmental impacts of oil in Nigeria, I also introduced unrelated documents just to see how the model was going to respond to unrelated data and so far it gave me great results evident in the image below:

```

articles_text/Effects of Oil Spillage (Pollution) on Agricultural Production in Delta.txt      articles_text/EFFECTS OF OIL SPILL
AGE ON FISH IN NIGERIA.txt                      Score: 1.0000
articles_text/Causes and Terrain of Oil Spillage in Niger Delta.txt      articles_text/Deficient legislation sanctioning oil spill.
txt                      Score: 0.8775
articles_text/Effects of Oil Spillage (Pollution) on Agricultural Production in Delta.txt      articles_text/ENVIRONMENTAL IMPACT
S OF OIL EXPLORATION.txt                      Score: 0.8584
articles_text/EFFECTS OF OIL SPILLAGE ON FISH IN NIGERIA.txt      articles_text/ENVIRONMENTAL IMPACTS OF OIL EXPLORATION.txt
Score: 0.8584
articles_text/Environmental Consequences of Oil Spills on Marine Habitats and the Mitigating Measures-The Niger Delta Perspective.txt
articles_text/NIGERIA OIL POLLUTION, POLITICS AND POLICY.txt      Score: 0.8145
articles_text/Effects of oil spills on fish production in the Niger Delta.txt      articles_text/Environmental Consequences of Oil Sp
ills on Marine Habitats and the Mitigating Measures-The Niger Delta Perspective.txt      Score: 0.7833
articles_text/Effects of oil spills on fish production in the Niger Delta.txt      articles_text/ENVIRONMENTAL IMPACTS OF OIL EXPLORA
TION.txt      Score: 0.7778
articles_text/Causes and Terrain of Oil Spillage in Niger Delta.txt      articles_text/Effects of oil spills on fish production in
the Niger Delta.txt      Score: 0.7740
articles_text/Effects of oil spills on fish production in the Niger Delta.txt      articles_text/NIGERIA OIL POLLUTION, POLITICS AND
POLICY.txt      Score: 0.7736
articles_text/Environmental Consequences of Oil Spills on Marine Habitats and the Mitigating Measures-The Niger Delta Perspective.txt
articles_text/ENVIRONMENTAL IMPACTS OF OIL EXPLORATION.txt      Score: 0.7622
(nlp) fisayo_ojo@luwafiyomisMBP More_IS %

```

Results from BERT model

Chapter 3. References

- [1] Mandy Gardner. 2023. Definition of Money. <https://medium.com/@Mandy-Gardner/a-brief-history-of-language-a-timeline-b767f9ae2c35>
- [2] Amazon Web Services. 2024. <https://aws.amazon.com/what-is/recurrent-neural-network/>
- [3] Kim, Y. (2014) Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, 1746–1751. arXiv: 1408.5882 <https://doi.org/10.3115/v1/D14-1181>
- [4] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. Retrieved from <https://ojs.aaai.org/index.php/AAAI/article/view/9513>
- [5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arxiv.org/abs/1810.04805 <https://doi.org/10.48550/arXiv.1810.04805>