Transformers: Attention is Sometimes What You Need

An analysis of the BERT Natural Language Processing Model

UCR MSOL Capstone

Henry Sue

Student ID: 861164461

Specialization: Data Science

Abstract

Text understanding has always been an important problem space in the machine learning ecosystem. From fraud detection to chatbots, to autocorrect, natural language processing is a field where researchers continue to push forward for more complex and more accurate model architectures. Previously, natural language machine learning has been done by using statistical models, making classifications by most probabilistic. However, with the advent of stronger hardware, neural networks have become the dominant model type, learning deep contextual connections for words, rather than blindly outputting the most common word type.

This paper examines the usage of a traditional classification process using a baseline Naive Bayes probability model, and compares its accuracy to an implementation of BERT, a state of the art model. The BERT model was trained on commodity consumer hardware for 3 days and for 10 epochs in order to derive a “real-world” example of an application of the BERT architecture. The embedded word meaning from the pre-trained model was transferred over to our test application, where we classify the sentiment that tweets (short public text posts on Twitter.com) have.

We weigh the cost of training and implementing the model against the marginal accuracy increase from state of the art model to determine if and when it is appropriate to adopt and use the newer model type.

Introduction

The modern machine learning ecosystem features several different subcategories including, but not limited to, time series forecasting, image classification, and natural language processing. Natural language is a deeply complex subject, wherein machine learning engineers and scientists attempt to programmatically reach the innate human ability to understand the nuance of language. There are many difficulties in natural language processing that block the way for computers to process text. For example, understanding sarcasm is a common issue, or words that have multiple different uses and meanings.Thus, in order to continue to push State of the Art (SOTA), newer machine learning models utilize creative architecture that aims to better understand the meaning of text.

The current state of the art model is “BERT”, or Bi-directional Encoder Representations from Transformers *(Google Brain 2017)*. This architecture borrows from the attention model introduced in the paper “Attention is All You Need” (hence this paper’s title). Typically, SOTA language models rely on Recurrent Neural Networks (RNNs), a directed (non-acyclic) graph, which allows the output of previous nodes to be used as input nodes. By using RNNs, machine learning models can preserve contextual or long-term relations. Previous SOTA models include the Gated RNNs and Long-Short Term Memory models. The “Transformer” is a model that utilizes “encoder-decoder” attention layers. Essentially, it is made up of two connected RNNs that will read in a sentence or passage, and another will output a sentence. BERT, built on Transformers,

utilizes another innovation: the Masked Language Model. By randomly replacing words in training sentences with masks, the language model is able to generalize context awareness when words are missing. This mimics how humans derive meaning of passages, by identifying the context of the sentence structure or words.

BERT, a model based on the Transformers architecture, utilizes a directionally independent approach to training a neural network. The “Bi-directional” comes from the model’s training, which approaches sentence encoding from both the left side, as well as the right side. When using this pre-trained model, developers may transfer train the pre-conditioned model to generalize over multiple NLP applications. The BERT model is empirically strong, achieving a new General Language Understanding Evaluation (GLUE) score to 80.5%, 7.7% above the previous state of the art model. These pretrained models not only provide a greatly improved model for applications, but as they are based on neural networks, provide lightning quick eager executing models that can be trained on our modern day GPUs or TPUs. In fact, once the model is trained, we are able to convert the model into a graph executing model that performs computations even quicker.

Modern applications call for cutting edge machine learning models, and products that use these state of the art models require high accuracy, fault tolerance, and scalability. Neural networks such as the Transformer model have proven to be not only the most accurate, but widely transmutable to many different applications. In addition, neural networks have also proven to be extremely scalable. One example of this is the

new GPT-3 model by Open-AI, which is trained on 175 billion parameters and trained using the BERT model, reaches a level of accuracy in much fewer training cycles than previously, and has state of the art accuracy and generalization.

Project Overview

My project consists of a compositional study of the Transformer / BERT architecture, as well as comparing the result of the model on a toy dataset to a baseline machine learning model. First, I delve into the background for the transformer architecture, as well as touch on recurrent neural networks, as well as how BERT uses the transformer module to create a state of the art natural language model that generalizes well to many NLP applications. Then, using a common task, sentiment classification, I benchmark the performance of a hobbyist grade implementation of the BERT model against a baseline model, the Naive Bayes Classifier. By examining the results of the test, we can observe that the transfer-trained BERT model significantly outperforms the baseline model, despite only being trained using consumer hardware for a short period of time.

Significance of Project

As told by Andrew Ng, professor of computer science at Stanford and founder of Cousera, “AI is the new electricity” *(Ng 2017)*. Machine learning is becoming more and more massively popular, with data science being touted as the new frontier of science. With that said, many organizations and scientists continue to push for state of the art for many different machine learning applications.

Natural Language Processing is an important sub-field of machine learning. There are many problems where NLP can be applied to, including but not limited to: fraud detection, spam / scam email filters, web searching, and language translation. With the continued development of stronger and stronger computer hardware, better and more accurate machine learning models are becoming more feasible to train and implement. One of these state-of-the-art models is called BERT, or Bi-directional Encoding Representations with Transformers.

The BERT model represents one of many huge improvements in the Natural Language Processing space. Additionally, BERT is so widely applicable to many different problems and is very easily transfer trained. BERT and GPT-3 *(OpenAI 2020)* are stepping stones in the journey for Artificial General Intelligence (AGI). My project also focuses on how even though these models are state of the art, they are immediately available for both consumer and enterprise level adoption. The level of accessibility from open-source organizations such as Huggingface allows many developers or corporations to utilize cutting edge models very shortly after they are developed. However, we are still a long shot away from language *understanding,* where machine learning models can digest meaning from context or language, rather than statistically or procedurally interpolating from historical data.

Literature Review

Previous Research

Natural language processing as a problem has been around since around the 1950s, beginning with automatic symbolic language translation or “machine translation”. In the 1980s to late 1990s, natural language processing was mostly focused on statistical models, such as bayesian or term frequency models *(Jurafsky and Martin 2014)*. Only until recently has there been an explosion in popularity of neural networks, mostly due to the massive increase in computer processing power, notably parallel and tensor processing.

Previous state of the art models include Enhanced Language Representation with Informative Entities (ERNIE) *(Zhang et al. 2019)* Long/Short Term Memory Networks *(Hochreiter & Schmidhuber 1997)*, Embeddings from Language Models (ELMo) *(Peters et al 2018)*. As an aside, machine learning researchers tend to have a sense of humour when titling their papers, as well as their models to conform to previous model naming conventions. For example, some state of the art models are named after sesame street characters (Bert, Elmo, Ernie) by clever acronym usage.

Findings and Unanswered Questions

By reading through related research, we are able to get a grasp of the state of the art models. However it is difficult to see how this directly translates to a concrete, real-world application. Indeed, typically machine learning models have a small lead time before newer models are implemented in enterprise settings. Sometimes, older models are preferred over SOTA, as simplicity and understandability are paramount, even at the

cost of model accuracy. It is understandable that as we push toward higher complexity models that perform marginally better, we have higher inertia to implement. However, it poses the question: why aren't there more widespread pretrained models? What are some ways in which developers or scientists may make these advances more accessible for the layman?

Preliminary Work

I am well versed in statistical modeling for natural language processing applications. For a previous class, I implemented a sentiment classifier from scratch using the naive bayes algorithm *(Sue 2021)*. There is significant crossover in that this project's comparison uses the naive bayes algorithm and compares the model accuracy against a similar task (sentiment classification). However, prior to this project, I had no exposure to neural network programming and modeling. Additionally, the model transfer training is novel for this application and dataset.

Remaining Questions

Another question that I would like to pose prior to this project is: how easy is the implementation, and how can we apply this to enterprise settings?

Technical Background

Baseline: Naive Bayes Classifier

One approach to Natural Language Processing is statistical machine learning. At a high level, the Naive Bayes Classifier is a machine learning model that tallies the number of previous occurrences of a word (the prior *probability)* for each target classification, then combines the weights for each feature, outputting the most likely label. In order for the machine learning model to interpret words, we tokenize or vectorize. This is the process in which each word in a passage is represented by: a unique 'token' in the form of a number, as well as the number of times it appears in the passage. As you feed the model more passages, each unique word adds to the corpus, as well as the count for each tokenized word.

The pseudocode for this is as follows (where vocabulary is the corpus of all previously tokenized words):

```
1: for document ← dataset do
2:     for all word ∈ document do
3:         if word ∉ vocabulary then
4:             vocabulary+ = word
5:             vector[word] = count(word)
6:         else
7:             vector[word] = count(word)
8:     output(vector)
```

An example of this is if we vectorize the sentence "dog sheep cat cat dog". In this example, we would go through the sentence word by word. If the word has not been

seen before, it will add another field to the word vector. If the word has already been seen, increment the word's corresponding count by one. Going through the example, the resulting word count-vector is:

| dog sheep cat |

| 2 1 2 |

Given that we can now represent each word passage as a vector of counts of each word, we are able to encode each of our data point, and begin to train our model. To train our model, we take each document (or passage) from each class, and calculate the log prior, which is the number of documents of each class C over the total number of documents in the training dataset. Then, for each word in the document, we count the total occurrences of each word in the class over the total occurrences in all classes in all documents. Then, we can calculate the log-likelihood, using the count of each word in each class.

In order to begin classification, you take the outputted log-prior and log-likelihood from the training step for each word, then apply it to each word in your document that you are classifying. After summing the value of each word, the class with the highest posterior likelihood is selected as the most likely class label. I go into the process more in depth in my previous work, where I create a Naive Bayes Classifier from scratch: (https://henrysue.github.io/projects/Naive_Bayes_from_scratch.pdf).

Recurrent Neural Networks

At the most basic level, BERT as well as the underlying transformer architecture are based on recurrent neural networks. Let us start by explaining a neural network. A neural network is a directed, fully connected graph of weighted nodes (hidden layers) that take in a number of inputs, process the inputs based on the weights of the nodes, then create an output. Usually, Neural Networks form DAGs (Directed Acyclic Graphs), where each node's outputs only go one way. Recurrent neural networks, however, attempt to capture and retain some information from prior inputs as outputs to use again as inputs for another cycle of node processing. Essentially, RNNs have loops in them that allow some information to be retained.

Transformer Architecture

Rather than using a gated recurrent neural network to selectively allow different pieces of embedded information through to subsequent neural networks, the Transformer architecture utilizes a process called "self-attention", where by using encoder-decoders, the neural network has access to different pieces of input data in the sequence. Pictured on the below, we have the Transformer architecture as depicted in the "Attention is All You Need" (Chang et al, 2019).

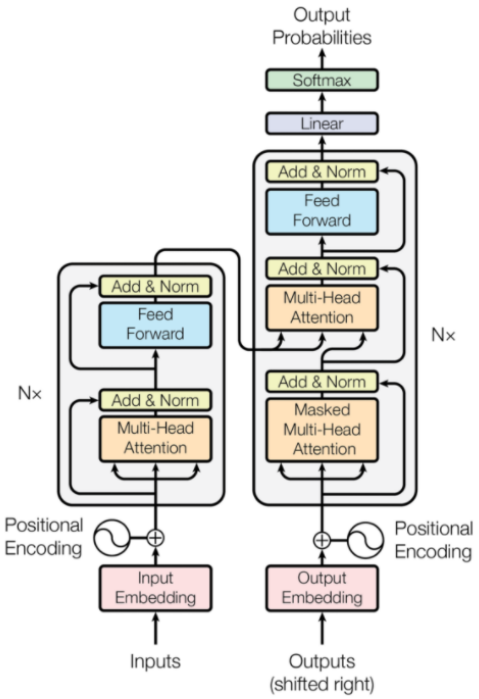


Figure 1: The Transformer - model architecture.

Additionally, the BERT model uses a novel concept in training, Masked language model. Essentially, words from sentences are randomly hidden with a “MASK” token, and the encoder-decoder architecture will predict the most likely candidate for the hidden masked word. This solves a problem where word embeddings only held meaning in one direction, whereas by using these mask tokens, context can be applied omnidirectionally. However, because language follows a sequence, context is preserved when reading from the left and the right, hence the “Bi-directional” part in BERT. Typically in training, MLM will mask a certain percentage of the words in order to obtain the word embeddings - Google arbitrarily uses 15% masked words.

One consequence of these recurrent architectures is that in order achieve state of the art performance, the number of parameters must be increased. As mentioned

previously, these models are enabled by our stronger processing power in processing units, as the training for these tensor neural networks require huge processing power.

Methodology

Data

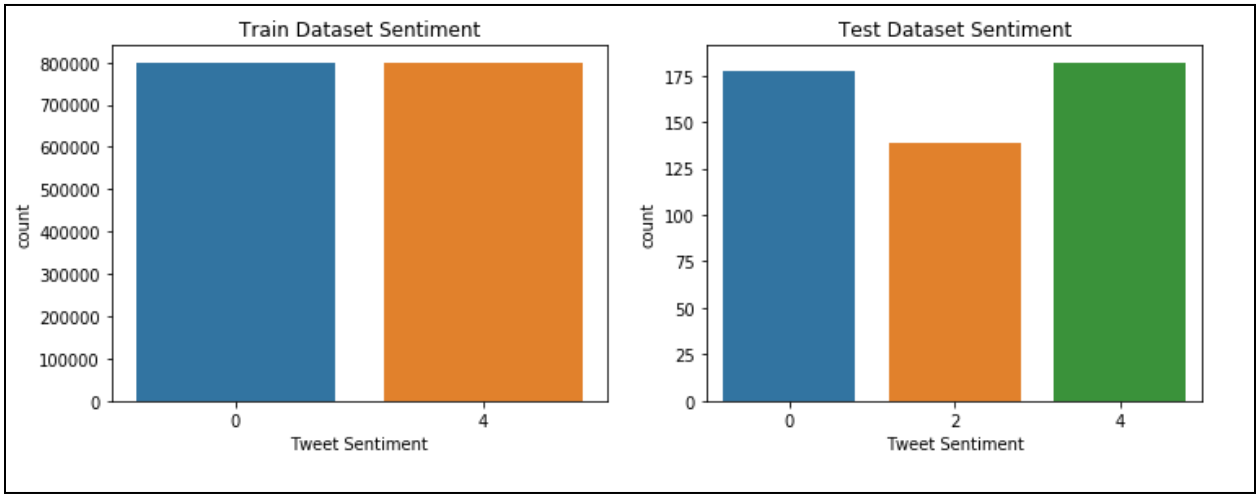
In order to build a robust model, we require an appropriate dataset. Because the goal of this project is to build a model to analyze tweets from twitter, an ideal dataset is Sentiment 140 *(Go et al 2009)*. Sentiment 140 started as a class project for Stanford's CS224 series, which focused on Natural Language Processing. The code base is not open source, but the dataset is free for academics. The dataset contains 16 million unique data points, or "tweets", gathered from twitter. The dataset features training and test data with the following parameters: tweet sentiment, tweet ID, tweet date, query used, tweet user, and tweet text.

	Tweet Sentiment	Tweet ID	Tweet Date	Query	Tweet User	Tweet Text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

For our use case, the only relevant features are the tweet text and the tweet sentiment, as user and query level information is not correlated with tweet sentiment, and tweet ID and tweet Date are only identifiers for the tweet.

The dataset was created by automatically scraping twitter for tweets and classifying the tweet's sentiment based on the presence of a happy smiley " :) " or a sad

smiley “ :(“. This approach to automatically creating a sentiment classification dataset is catalogued in their paper “Twitter Sentiment Classification using Distant Supervision” (Go, Bhayani, and Huang 2009). The training data consists of a balanced 50/50 split of 16 million data points. The polarity of the tweet is marked “0” for negative, and “4” for positive. The test set contains 175 positive and negative tweets, and 130 neutral tweets.



For simplicity, we drop the tweets with neutral sentiment in the test set. Then, to allow for our model to output positive or negative sentiments, we change the labels to “0” = Negative sentiment, and “1” = Positive sentiment. This binary classification makes it easier to distinguish whether

Classification

The first step to establish a baseline is to create a dummy classifier. In order to process text, we have to format it in a way that our code can recognize. By substituting each word in a sentence or paragraph with a token, we can represent the sentence or

paragraph with a matrix of vectors. For example, we can tokenize the sentence “I am so happy this happened” using scikit-learn’s CountVectorizer:

```
vect = vectorizer.transform(['I am so happy that this happened.'])
print(vect)
```

```
  (0, 64487)    1
  (0, 267224)   1
  (0, 267348)   1
  (0, 563436)   1
  (0, 600330)   1
  (0, 607680)   1
```

In this example, our vector matrix indicates a single count for each token, similarly, if we have repeated words, the count for each term-count pair will increase. Applying a dummy classifier to our dataset allows us to roughly estimate the accuracy of a model that randomly guesses a label. Our result is a model accuracy of 46.8%, a little less than just purely guessing 50%.

```
Dummy Classifier Evaluation without neutral sentiments in test data:

'Model Accuracy: 46.8%'
```

To establish a standard for our model, we evaluate the use of a Bernoulli Naive Bayes classifier on our dataset. A Bernoulli Naive Bayes Classifier assumes features as independent booleans that describe inputs. In our case, the presence of each token represents one feature. Let us try training the Bernoulli Naive Bayes Classifier on our dataset and testing it on our test dataset:

```
Bernoulli Naive Bayes Evaluation without neutral sentiments in test data:

'Model Accuracy: 79.67%'
```

Our Naive Bayes classifier performs significantly better than our dummy classifier, indicating that our model has some degree of success.

BERT Model

BERT - or Bidirectional Encoder Representations from Transformers is a state-of-the-art model that is trained on the Transformers network architecture *(Vaswani et al 2017)*, reading all the tokens at once to remove directionality. Additionally, BERT is trained by masking 15% of training text and asking the model to guess what words are masked. In our project, we use the pretrained base model from the Hugging Face library "Transformers". We will use the case-sensitive version "Bert-Base-Cased" in order to distinguish between words and letters that have been capitalized.

In order to use our pretrained BERT model, we must first create a tokenizer in order for our model to process our data. In our project, we use the pretrained BERT tokenizer. Here we can see how the tokenizer breaks down a sentence into specific words and modifiers for use in the tokenizer:

```
sample_text = 'This text is a test to visualize what happens using the tokenizer.'

tokens = bert_tokenizer.tokenize(sample_text)
print(tokens)
```

```
['This', 'text', 'is', 'a', 'test', 'to', 'visual', '##ize', 'what', 'happens', 'using', 'the', 'token', '##izer', '.']
```

Then, the tokenizer changes the tokens into their corresponding token IDs in order for the model to recognize the token for each corresponding word:

```
token_ids = bert_tokenizer.convert_tokens_to_ids(tokens)
print(token_ids)
```

```
[1188, 3087, 1110, 170, 2774, 1106, 5173, 3708, 1184, 5940, 1606, 1103, 22559, 17260, 119]
```

We include special tokens to denote separators, classification, unknown, and padding tokens to allow our model to adequately process our data. We also set the max length of our token matrix to 140, as twitter tweets are limited to 140 characters, therefore the maximum number of tokens is also 140.

In order to transfer train the pretrained BERT model on our tweet dataset, we must split our dataset into a smaller subset to allow iteration through epochs in a reasonable training time. We split the training dataset into a much smaller subset of 64,000 tweets for our training data, and 8000 tweets for our validation and test datasets.

```
df_train.shape
(64000, 6)
```

```
df_test.shape
(8000, 6)
```

Then, the BERT model was trained for 3 days on an Nvidia RTX 2060 Super for 10 epochs, with a batch size of 8, using Pytorch. We can see that BERT's pretrained model is already fairly accurate before transfer learning in the first epoch:

```
Epoch 1 / 10
----------
Train loss 0.5088364473022521 accuracy 0.798578125
Val loss 0.49937918430566786 accuracy 0.809125
```

The model begins to show diminishing returns at the last few epochs:

```
Epoch 8 / 10
----------
Train loss 0.43110237485170366 accuracy 0.88153125
Val loss 0.4950049730539322 accuracy 0.81575

Epoch 9 / 10
----------
Train loss 0.42648462515324354 accuracy 0.886203125
Val loss 0.5054766044020653 accuracy 0.805125

Epoch 10 / 10
----------
Train loss 0.4244859441109002 accuracy 0.888171875
Val loss 0.4964141429960728 accuracy 0.81625
```

After training, we test our transferred model on our test dataset:

```
test_acc
```

```
tensor(0.8205, device='cuda:0', dtype=torch.float64)
```

We see that our transferred model is 82.05% accurate on the test data of approximately 8000 tweets. Let us compare to our Bernoulli Naive Bayes classifier on our test dataset:

```
Bernoulli Naive Bayes Evaluation on NN Test Data:

'Model Accuracy: 76.98'
```

If we perform a t-test to determine statistical significance, we find that our truncated p-value is much less than 0.01, indicating that our model performs better than the Bernoulli Naive Bayes with a confidence level of greater than 99%. Additionally, we find that the BERT model performs better than the dummy classifier with a confidence level of greater than 99%.

```
print('p value:' + p_val)
```

```
p value:0.0000
```

Results

As we can see from our test, the BERT model has a higher accuracy when classifying text sentiment on the twitter dataset. The baseline naive bayes classifier reached an accuracy of 76.98%, whereas the BERT model reached an accuracy of 82.05%, a 6% increase in accuracy. In the natural language processing applications, this jump in accuracy is very large, as the difficulty of classification for edge cases creates a logarithmic accuracy-difficulty curve, with diminishing accuracy returns the better a model becomes. By running a two tailed t-test, we can reject the null hypothesis that the model's increase in accuracy is due to random occurrence with 99%+ statistical significance.

Discussion and Conclusion

Let us start by answering the question: *are state of the art models better than statistical modeling?* Yes, not only do state of the art models attempt to encode the latent meaning in language, as well as taking into account word context, but pretrained models can generalize well to almost any natural language problem with little issue. However, neural network architecture is still in its infancy, and the margin of improvement is very slight, so the cost of implementation may outweigh any

improvements in accuracy. As well, the closer the models reach toward perfect accuracy, the more diminishing returns we see from model development.

Neural networks require powerful and expensive tensor processing or graphics processing hardware in order to train, and are sometimes time-intensive to reach the desired training level. At an enterprise level, there are solutions such as cloud computing, where companies such as Google, Amazon and Microsoft will rent computing power for enterprise level equipment, as well as taking care of end to end deployment. Although convenient, this approach relies on using third party providers, and may not always be the correct solution, especially in academic environments. Neural networks are also fairly cumbersome to train, requiring massive datasets to encode the meaning they are looking for. This is less of a problem as the world becomes more data mature.

Baseline models are still prevalent today, as they are simple to implement, cost effective, and the models are usually robust enough to be accurate enough for most tasks. In fact, many problems, such as classification (reviews, customer history targeting, spam detection, fraud detection), are not sensitive to false negatives, but have great returns on true positives. Thus, baseline models such as the Naive Bayes Classifier are still used today.

It is important to choose the correct type of machine learning model for your application. In most hobbyist and even smaller enterprise level machine learning tasks,

often the baseline model is enough, at least for proof of concept, before diving into more computationally expensive models such as BERT. Additionally, in order to achieve state of the art model accuracy, the models are becoming larger and larger, which poses another issue of diminishing returns. Attempts have been made to curb this issue by optimizing model size by pruning. The provider for the open-source BERT architecture that this paper is based on, Huggingface, also focuses on research to optimize state of the art models so that they are much more computationally efficient. There will always be room for pushing the state of the art. However, for most uses, this new architecture is unnecessary.

Computer Code is available from Google Drive / Google Colab:

https://colab.research.google.com/drive/1yMdQ0mxUTCdH6HRQNptZftb18x46-ERR

References

1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Arxiv*. https://arxiv.org/pdf/1810.04805.pdf.
2. Go, A., Bhayani, R., and Huang, L. 2009. Twitter Sentiment Classification using Distant Supervision. CS@Stanford (2009). http://help.sentiment140.com/
3. Google Brain. (n.d.). Attention is All You Need. https://arxiv.org/abs/1706.03762.
4. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
5. *Hugging Face – The AI community building the future.* Hugging Face –. (n.d.). https://huggingface.co/.
6. Jurafsky, D., & Martin, J. H. (2014). *Speech and language processing*. Pearson Prentice Hall.
7. Koehn, P. (2014). *Statistical machine translation*. Cambridge University Press.
8. Lai, G., Chang, W.-C., Yang, Y., & Liu, H. (2018). Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. *Arxiv*. https://doi.org/https://doi.org/10.475/123_4
9. Lynch, S. (2017). Andrew Ng: Why AI Is the New Electricity. *Insights by Stanford Business*. https://www.gsb.stanford.edu/insights/andrew-ng-why-ai-new-electricity.
10. OpenAI. (2020). Language Models are Few-Shot Learners. https://arxiv.org/pdf/2005.14165.pdf.
11. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *Arxiv*. https://arxiv.org/pdf/1802.05365.pdf.
12. Sue, H. (n.d.). (rep.). *Implementation of the Naive Bayes Classifier as a Tweet Sentiment Classifier*. Mar. 2021
13. Uszkoreit, J. (2017, August 31). https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html.
14. Zhang, Z., Han, X., Liu, Z., Jiang, X., Sun, M., & Liu, Q. (2019). ERNIE: Enhanced Language Representation with Informative Entities. *Arxiv*. https://arxiv.org/pdf/1905.07129.pdf.