Fine Tuning BERT for Sentiment Analysis on Tweets

Rushabh K Patel

Submitted for the Degree of Master of Science in

Data Science and Analytics

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

December 06, 2021

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 8869

Student Name: Rushabh K Patel

Date of Submission: 06 December 2021

Signature:

Acknowledgements

I would like to express my gratitude towards my supervisor Dr Nicola Paoletti who has guided, motivated, and provided me with constructive feedback throughout all phases of the project. His guidance toward my experiments was immensely helpful and has made a major part in shaping the project to what it is today.

Abstract

Social media has become an important part of our lives. We have all used multiple different platforms and have voiced our opinions about certain things. In this project, we have chosen one such platform – Twitter and performed sentiment analysis on the data available on it. Twitter has one of the biggest presence for celebrities and businesses. We see a lot of engagement of celebrities with their fans, and business with their customers.

Our goal is to use a pre-trained model – in our case BERT, and to fine-tune to predict the sentiment of the tweets. BERT which was developed by Google and open-sourced in 2018, is a state-of-the-art model which broke several benchmark records on various NLP tasks. We have discussed a brief history and the motivation which ultimately led to the development of BERT. We discuss the pre-processing steps which must be performed before we feed the input to the mode. We compare the performances of our experiments with different set-ups and look at the shortcomings of BERT and talk about how these are being addressed by researchers.

Contents

1 Introduction

The internet has changed the way people communicate and express their opinions. There have been many instances where people have voiced their opinions about certain topics and have gotten real change. A big part is played by social media companies such as Twitter, Facebook, Instagram etc which give the users a platform to voice their opinions. The users may be praising a particular product or situation. Or it can be showing their displeasure in the performance of their favourite football team. Businesses are trying to leverage this data which is being generated on the platforms to try and sell more products and services. They are also trying to connect with their customers when they show displeasure at the product or service provided by them.

One such social media platform is Twitter. Currently, Twitter has more than 300 million active monthly users, in total, they generate approximately 500 million tweets daily. The sheer scale at which it operates is astounding. The amount of content generated by even a small percentage of users is hard to analyse by a normal person. Hence, there is a need to automate this task considering there is huge economic value for businesses.

Sentiment analysis can be defined as a process that mines attitudes, opinions, emotions from textual information using Natural Language Processing (NLP) techniques. Sentiment analysis involves classifying the text sample into either positive or negative. In certain cases, it can be even classified as neutral. With advances in Deep Learning and Natural Language Processing, the ability of the algorithm to analyse textual information has improved drastically.

There are many benefits to performing sentiment analysis. Businesses perform this to see how their products or services are performing in the industry. They may even do this before product launches to see what their customer base likes and what they dislike. Research and consulting organizations perform sentiment analysis to see how the markets are like. They perform this over time to see how the sentiments have changed over a period of time and predict what the future demand may be like. In our experiments, we perform sentiment analyses on tweets made by the users. We look at two datasets, the first one is the Offensive Tweets Dataset and the second one is the Airlines Complaint Dataset.

In the recent past, we have seen users tweet offensive and hateful speech on the platform. Social media has become an abstract layer where users have been misusing this abstraction to express their displeasure. The organizations are trying to flag and remove content which they find spreads hate and incite violence. They have gone so far as to permanently ban certain individuals from the platform for breaking their terms of use. The second dataset is about user complaints about the airline whose services they use. This is a classic example of a business trying to reach out and address the concerns of its customers. Twitter is a public platform, the tweets made on it, either positive or negative can be seen by anyone in the world. This means that for the businesses, it's their reputation at stake.

2 Background Research

Deep Learning is a subfield in machine learning which mainly deals with the use of Artificial Neural Networks. Neural networks are networks that try to mimic the behaviour of the human brain. They constitute many nodes or "*neurons*" which are linked together to form layers. Deep learning, or machine learning in general can broadly be divided into two: Supervised and Unsupervised learning. Supervised learning is where the dataset is labelled, which means that every sample in the dataset has a corresponding value. The job of the model is to approximate the function which maps the sample to the label value. Examples of this are regression and classification problems like linear and logistic regression, Support vector machine etc. Unsupervised learning is where algorithms discover hidden patterns in the datasets. Examples for this is K-means clustering, K-nearest neighbour etc. Sentiment analysis falls under supervised learning and the techniques we will be using broadly fall under the field of Natural Language Processing (NLP).

Natural Language Processing (NLP) is a subfield in Artificial Intelligence, which helps machines process and analyse human language. Human languages involve long sequences of words, and because of this, the earlier models were based on Recurrent Neural Networks (RNNs). RNNs are

the type of neural networks commonly used to process sequenced data. They are long blocks that cascade one after the other to process the sequence one step at a time. This leads to vanishing and exploding gradient problems. Long Short-Term Memory (LSTMs) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRUs) (Chung et al., 2014) improved on the concept and overcame the problem of vanishing and exploding gradients. However, LSTMs struggled to capture the contextual information whenever long sequences were involved. This led to the development of models based on the concept of attention, which we will discuss in further sections (Singh and Mahmood, 2021).

Since 2018 there has been amazing progress in the NLP community. With the release of large scale pre-trained models like BERT and GPT, we have achieved excellent performances on various NLP tasks. NLP is a diversified field with distinct tasks, and it is challenging to find a reasonably sized dataset for solving these task-specific problems. Deep learning based NLP models always benefited from a large amount of annotated data, but a large amount of annotated labelled data is hard to find. To bridge this gap, researchers came up with language representation models or simply *language models*. Language models are models which are pre-trained on large amounts of unannotated data, this helps the model learn basic linguistic features. Since there are no labels involved, they fall into the category of unsupervised learning. These models are trained on billions of words over hundreds of epochs. Once they have been trained, anyone can use these pre-trained models and fine-tune them to their specific task. These specific tasks can be anything simple from sentiment analysis to complex tasks like text summarization. We will see in our experiments how much time it takes to fine-tune a model and compare its performance in various set-ups.

2.1 Transformer and the attention mechanism

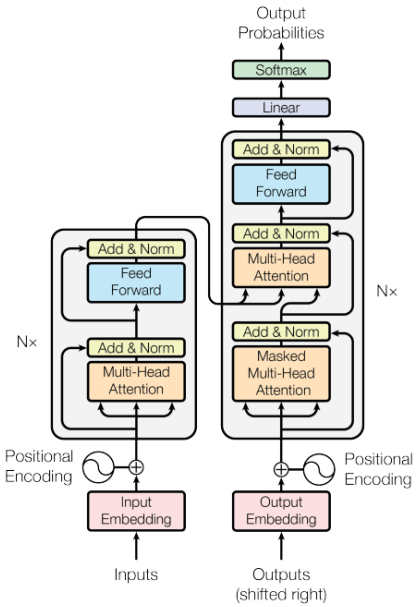


Figure 1: Transformer high-level design (Vaswani et al., 2017)

The Transformer architecture which was introduced in "Attention is all you need" paper (Vaswani et al., 2017) and was well received in the NLP space. It overcomes the LSTM's context problem by incorporation the attention mechanism. It aims to solve a lot of sequence-related tasks. Sequence tasks include language translation, text summarization, sentiment analysis etc. Sentences are sequences of words, and the order in which the words occur is crucial to making sense of the sentence.

Sequence models normally comprise one or many encoder-decoder blocks. The input sequence is fed into the encoder block which maps it into an n-dimensional vector. The decoder takes this vector and produces the output. The output of the decoder typically is a vector which is a combination of

various aspects of sequences like context, meaning etc. It can be viewed as word embeddings with positional and contextual information added.

2.2 The Transformer Architecture

The Transformer is a stack of 6 Encoder-Decoder blocks. The encoder and decoder on high-level make use of self-attention, a concept we will discuss in a subsequent section and a feed-forward layer. The output of each encoder block is sent to the corresponding decoder block. Similar to other deep learning models having many parameters, some input features may be lost in the deeper layers. To overcome this problem, residual connections are made so that the input is explicitly added to the output of the current layer. Unlike many other RNN models, like LSTMs, Transformers do not make use of recurrent blocks, hence they cannot sample sequence distances. So how do Transformer learn which element came from which part of the sequence? The answer to that question is positional encoding shown in **figure 1.** Positional encoding is based on a sinusoidal function which are added to the word embedding even before they are fed into the encoder or the decoder blocks. This helps the model understand which token is part of which specific sub-sequence.

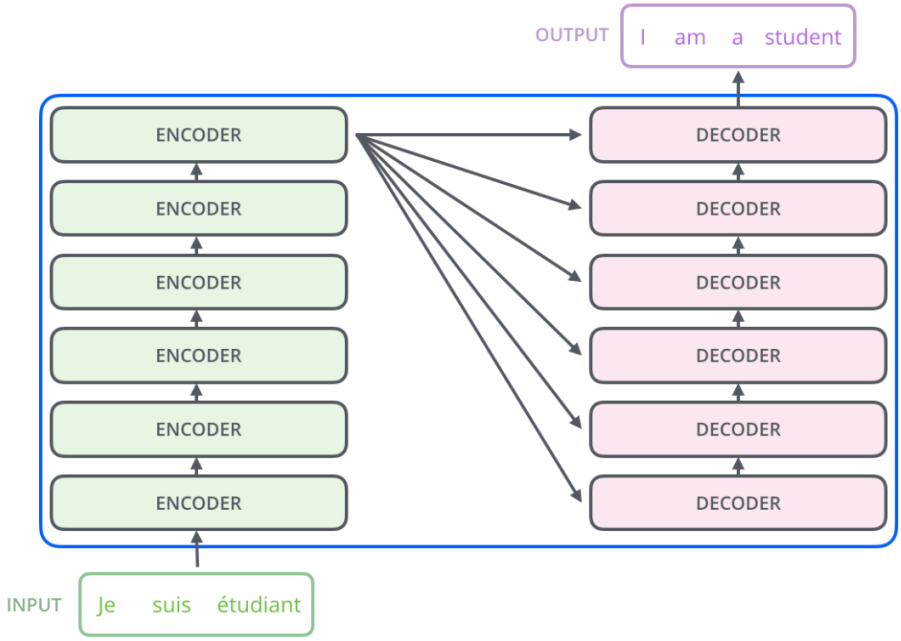


Figure 2: Transformer Architecture (Alammar, 2021)

2.2.1 The Attention Mechanism and Multi-Head Attention

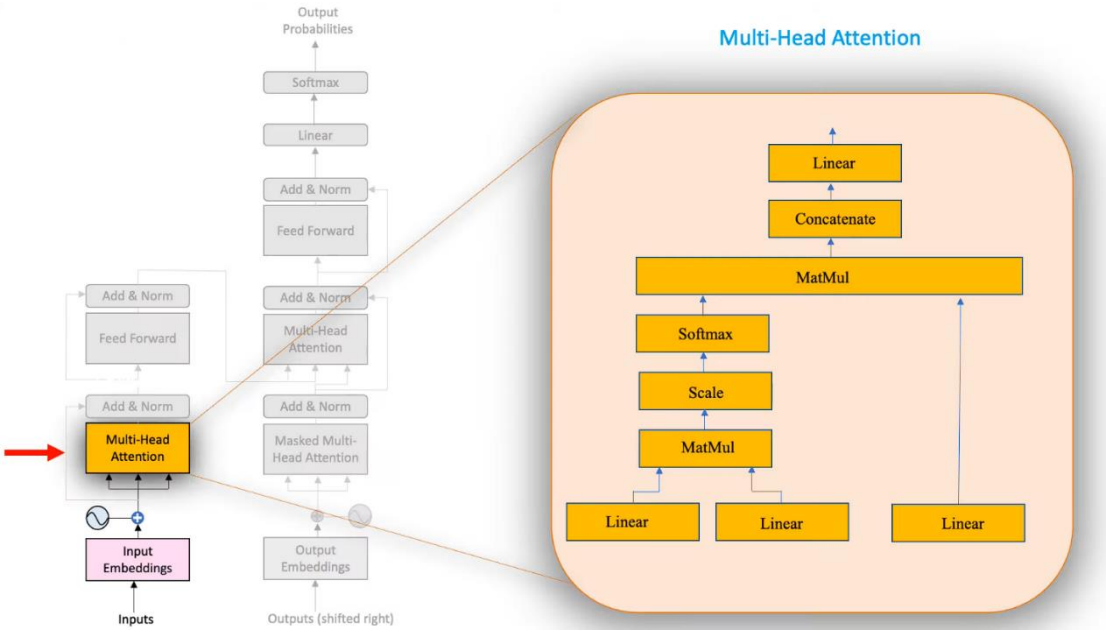


Figure 3: Multi-Headed Attention Block

In this section, we are going to break down the Multi-Head Attention block which is present both in the encoder and the decoder. To understand this, we will first have to understand what attention is and what mathematical operations are performed. Attention can be perceived as mapping a query and a set of key-value pairs to the output. Attention in the context of dealing with sequential data essentially gives us the relevance of a token with respect to a query or with respect to another sequence. Thus, the attention mechanism helps the model to focus on the important words in the sentence.

Transformer makes use of a type of attention mechanism called Self-Attention. Self-Attention is a method in which we calculate the relevance of a token with respect to all other tokens within the same sequence.

"He went to the bank to check his account balance, then he went to a river bank and cried"

In the above statement, the first occurrence of the word 'bank' represents a financial institution, and the second occurrence represents the physical area surrounding the river. The self-attention mechanism would give a high relevancy score to the words "account", "balance" and the first occurrence of "went" for the word "bank" in the financial institution context.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Mathematically attention can be calculated by the formula given below. The inputs to this are the query, key, and value vectors. In the first step, we calculate the dot the product of queries with the key, this can be done using matrix multiplication. The result of this is scaled down by dividing it with the square root of the vector dimension of the key vector. Now that the result is scaled down, we want to get the values between 0 and 1, to achieve this, we apply a softmax function. As a result of all these transformations, we get the attention filter matrix. Now the final step would be to multiply the attention filter with the value vector to get filtered attention scores.

Transformers further enhance the self-attention concept by implementing multiple attention mechanisms parallelly, each of which enables the model to attend to a different aspect of the sequence. The original transformer model makes 8 unique copies of the word embedding and each copy is sent to a different self-attention mechanism. This way the model, in each self-attention block, can focus on a different aspect of linguistic phenomena. In the end, the output of all 8 self-attention blocks is concatenated together. This is where the term 'Multi-Head Self Attention' (Vaswani et al., 2017) comes from.

2.2.2 Encoder block

The encoder is a stack of 6 identical layers. It processes the entire sequence at once. Each layer comprises 2 sub-layers, the first one consists of a multi-head self-attention mechanism, which we have discussed previously in detail and the second one is a vanilla feed-forward network (Vaswani et al., 2017). The feed-forward network is made up of 2 linear layers with rectified linear unit (ReLU) as the activation function (A Deep Dive Into the Transformer Architecture – The Development of Transformer Models | Exxact Blog, 2021). Both the sub-layers mentioned above employ residual connections (He et al., 2016) followed by layer normalization (Ba, Kiros and Hinton., 2016). The residual connections are necessary because deep into the network, the original word embedding meanings may be lost (Vaswani et al., 2017)

2.2.3 Decoder block

Like the encoder, the decoder is also a stack of 6 identical layers. The decoder consists of similar sub-layers as the encoder with an additional third sub-layer. This third sub-layer performs multi-head self-attention over the output coming from the encoder block. What this means is that the output coming from the encoder block is treated as the query vector going into the multi-head attention block. All the 3 sub-layers have residual connections with layer normalization for similar reasons as the encoder. One modification made in the first self-attention layer, which is called 'Masked Multi-Head Attention' is that the block has no clue of the token in the subsequent positions. Unlike in a normal multi-head self-attention block where all the tokens are processed parallelly which enables it to see the tokens further down the sequence. This means that at position i the predictions made for the masked token depend only on positions before i (Vaswani et al., 2017).

2.3 Bidirectional Encoder Representation from Transformers - BERT

Like Transformers, BERT, which stands for **B**idirectional **E**ncoder **R**epresentation from **T**ransformers, was developed by the Google Brain team in 2018 (Devlin et al., 2018). Common word embeddings like GloVe (Global Vectors for Word Representation) (Pennington, Socher and Manning, 2014) and word2vec (Mikolov et al., 2013) are non-contextualised which means that the word 'bank' in 'bank account' and 'river bank' will have the same embedding. BERT on the other hand is contextualised and was built on top of previous work done in pre-training contextual representations like ELMo (Embeddings from Language Models) (Peters et al., 2018) and GPT (Generative Pre-trained Transformer) (Radford et al., 2018).

One of the biggest problems in NLP is the shortage of labelled data for training. It is a bigger challenge for task-specific problems. To overcome this, researchers developed general-purpose language representation on large amounts of unlabelled and unannotated text data, this process is called pre-training. Now, the pre-trained model can be fine-tuned to specific tasks with fewer data samples and decreased processing time with significant improvements in accuracy. BERT is one such pre-trained model, some others are mentioned above. (Devlin and Chang, 2021)

s

Once we have a pre-trained model like BERT, there are two ways to use them on downstream tasks, they are, *feature-based* and *fine tuning*. ELMo makes use of the feature-based strategy, wherein the pre-trained features are included as additional features in the downstream task. In the case of the fine-tuning based models like GPT, the task-specific parameters are minimum and are trained on all pre-trained parameters too. In the case of GPT, the one drawback is that the model is unidirectional. It is left-to-right based; hence the model can only focus on previous tokens.

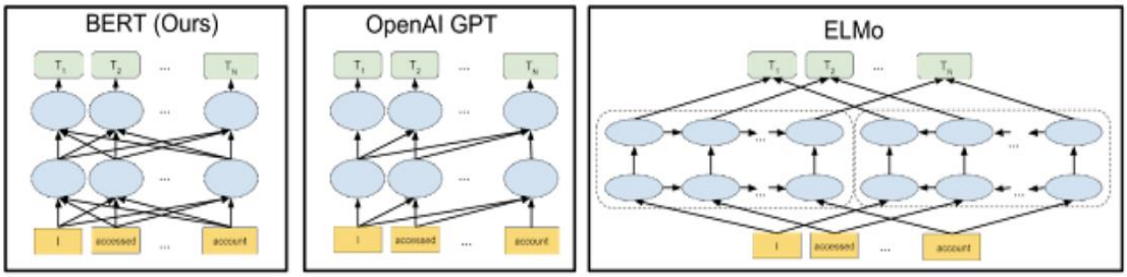


Figure 4: Architectures of BERT, GPT and ELMo (Devlin and Chang, 2021)

GPT is unidirectional and ELMo is shallowly bidirectional. BERT mitigates this unidirectional constraint by using two pre-training techniques, which are *masked language model* (MLM) and *next sentence prediction* (NSP). This makes BERT deeply bidirectional, although some would argue the term 'non-directional' to be apt.

2.3.1 Masked Language Model (MLM)

Masked language model is one of the techniques used to pre-train the BERT model. Intuitively, deep bidirectional models are better than both left-to-right models and shallow models which concatenate left-to-right and right-to-left models. Standard conditional language models can only be trained left-to-right or right-to-left. This is because the bidirectional conditioning would allow each word to indirectly "see itself", and the model could predict the target word in a multi-layered context. In order to "properly" train the model, some words are intentionally masked randomly, and the model is made to predict the masked words (Devlin et al., 2018).

While training the BERT model, 15% of the words are masked at random and the model is made to predict these masked words. There is one complication though, this process is carried out only during pre-training and not while fine-tuning. To alleviate this, not all the chosen 15% of the tokens are masked. Of the ones that were selected, 80% are replaced with [MASK], 10% are replaced by another random token and the rest 10% will remain unchanged (Devlin et al., 2018).

2.3.2 Next Sentence Prediction (NSP)

It is important for the model to understand the relationship between sentences. BERT does this by using a technique called next sentence prediction. Two sentences, A and B, are used as inputs, which are fed simultaneously to BERT. 50% of the time sentence B is actually the next sentence and hence labelled *IsNext* and the rest of the time it is just another randomly sentence from the corpus, which is labelled *NotNext*. While feeding the sentences, both the sentences are appended with a [SEP] token. The [SEP] token must be used both during the pre-training and fine-tuning.

2.3.3 Technical Details

BERT was developed by stacking encoder blocks from the original transformer model. A detailed explanation of the encoder block is explained in Section 2.2.2. Broadly there are two implementations of BERT; $BERT_{BASE}$ and $BERT_{LARGE}$. $BERT_{BASE}$ consist of 12 layers of the encoder block with a hidden layer of size 768 and 12 attention heads. There are a total of 110 million trainable parameters in $BERT_{BASE}$. Whereas $BERT_{LARGE}$ consists of 24 layers of the encoder block with a hidden layer of size 1024 and 16 attention heads. $BERT_{LARGE}$ has about 340 million trainable parameters. $BERT_{BASE}$ was designed to be at the scale of OpenAI's GPT model.

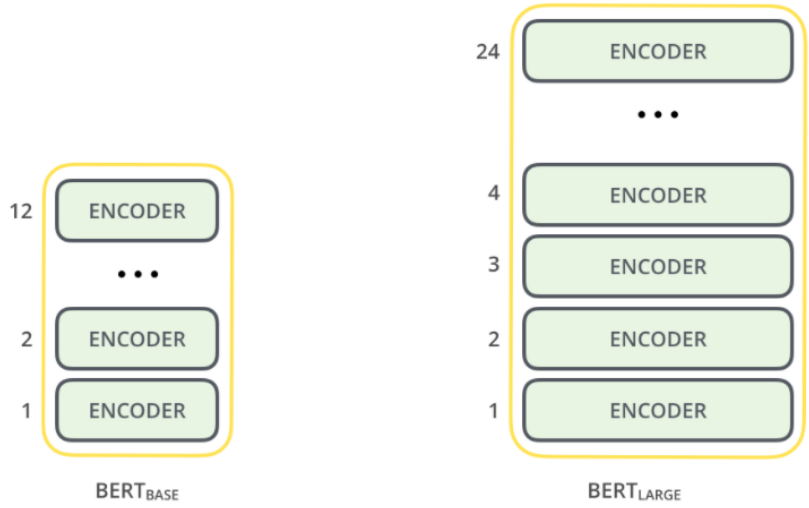


Figure 5: $BERT_{BASE}$ and $BERT_{LARGE}$

BERT was pre-trained on the BooksCorpus (Zhu et al., 2015) which has 800 million words and on English Wikipedia which has 2.5 billion words. For word embeddings, the WordPiece embedding {Wu et al., 2016) was used, which has a vocabulary size of 30,000 tokens. We can input two sentences simultaneously into BERT. Each sentence should be immediately followed by a [SEP] token and the first token in every sequence is a [CLS] token.

BERT achieved state-of-the-art performance on 11 NLP tasks, including on

- General Language Understanding Evaluation (GLUE) tasks
- Stanford Question Answering Dataset (SQuAD)
- Situations With Adversarial Generations (SWAG)

2.4 Generative Pre-Training Model - GPT

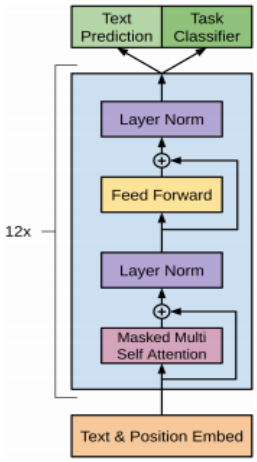


Figure 6: GPT Model architecture (Radford et al., 2018)

Prior to BERT, Generative Pre-Training (GPT) model (Radford et al., 2018) developed by OpenAI held state-of-the-art results for many NLP benchmark tasks. GPT was built by stacking 12 Transformer decoder blocks one on top of the other. As you can see from Figure 6, the input is first converted to positional embeddings which are then fed to the masked multi-head self-attention block. There are residual connections that add and normalize the input to the output of the first block. Finally, the output of this is passed to a feed-forward network with residual connections.

GPT unlike BERT is unidirectional which means it can learn the context from the previous words and cannot look at the next tokens. GPT was pre-trained on the BooksCorpus (Zhu et al., 2015), which is a collection of 7000 unpublished books from different genres. Pre-training GPT involved, incorporating the sliding context window mechanism. A context window of size k was chosen, say k = 3 with window: {T1, T2, T3}, the model was made to predict the next token T4. Subsequently, the windows were moved forward, {T2, T3, T4} and the model was made to predict T5 (Versloot, 2021). The authors of the paper used the Adam optimizer with a maximum training rate of 2.5e-4. It was trained for 100 epochs with a mini-batch of size 64 and each sample had a size of 512 tokens. The model itself has 117 million trainable parameters. (Radford et al., 2018)

2.5 Embeddings from Language Models – ELMo

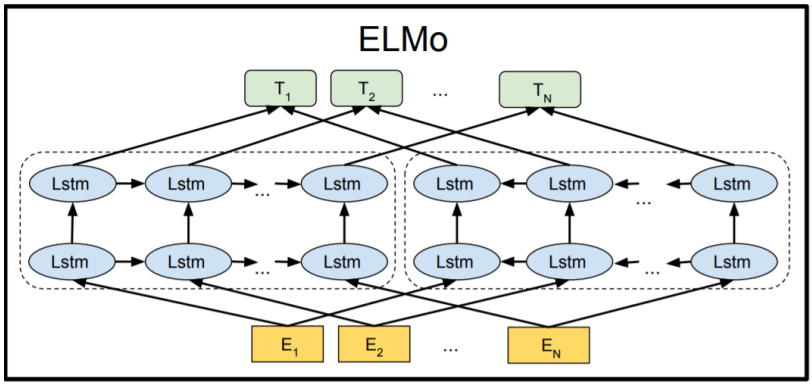


Figure 7: ELMo Architecture (Devlin and Chang, 2021)

Embeddings from Language Models (ELMo) was one of the first models developed to incorporate the concept of context into embeddings. Before ELMo, word embeddings such as GloVe and word2vec simply produced the embedding based on the word in isolation. They did not consider how the word was being used within the sentence. The foundation of ELMo is the bidirectional language model. First, the raw embeddings of the tokens are produced and are passed into the model. The language model learns to predict the probability of the next token given the history (Weng, 2021). The bidirectionality is achieved by making two passes separately, a forward pass and a backward pass. In the forward pass, the tokens before the target token are used, and in the case of the backward pass, the tokens after the target tokens are used.

The forward and the backward passes are modelled by multi-layer LSTMs. The contextualized embeddings are generated by performing a softmax normalization over the combination of the final layers of the LSTM networks. The original ELMo model was trained on a corpus of 5.5 billion words.

3 Experiment

In this section we will present the work done in selecting and downloading the pre-trained BERT model to fine tune it for our task. Although we have spoken about GPT and ELMo previously, we will be only focusing on BERT for our experiment. We will also discuss about the pre-processing steps which are required to be performed on the dataset before it is fed into the model. We will be experimenting with two separate datasets taken from Twitter.

3.1 Aim

The goal of this experiment is to highlight the ease and benefits of fine-tuning a pre-trained model on downstream tasks with minimal training time and with significant improvement in accuracy. We focused on the following areas:

- Correctly adopt specific pre-trained BERT model
- Pre-processing the input data to feed into the BERT model
- Train, evaluate and compare the performances for different set-ups.

3.2 Preparatory Work

3.2.1 Knowledge Acquisition

The first step was to read up about attention-based models, Transformers (Vaswani et al., 2017) and BERT (Devlin et al., 2018). We went deep in understanding how attention works and the mathematical operations being performed. We read up on the research papers where Transformers (Vaswani et al., 2017) and BERT (Devlin et al., 2018) were introduced. Finally, we went through a lot of blogs, tutorials and articles which spoke about fine-tuning the BERT model.

Choice of BERT model: For our experiment, we chose the pre-trained models offered by HuggingFace. HuggingFace is a large open-source community for pre-trained deep learning models specifically focusing on NLP. The detailed list of BERT models is listed in TABLE1.1. We selected the bert-base-uncased model for our experiment, as this would give us good results.

Model	Description
bert-base-uncased	12-layer, 768-hidden, 12-heads, 110M parameters.
	Trained on lower-cased English text.
bert-large-uncased	24-layer, 1024-hidden, 16-heads, 336M parameters.

	Trained on lower-cased English text.
bert-base-cased	12-layer, 768-hidden, 12-heads, 109M parameters.
	Trained on cased English text.
bert-large-cased	24-layer, 1024-hidden, 16-heads, 335M parameters.
	Trained on cased English text.
bert-base-multilingual-uncased	12-layer, 768-hidden, 12-heads, 168M parameters.
	Trained on lower-cased text in the top 102 languages with the largest Wikipedias
bert-base-multilingual-cased	12-layer, 768-hidden, 12-heads, 179M parameters.
	Trained on cased text in the top 104 languages with the largest Wikipedias
bert-base-chinese	12-layer, 768-hidden, 12-heads, 103M parameters.
	Trained on cased Chinese Simplified and Traditional text.
bert-base-german-cased	12-layer, 768-hidden, 12-heads, 110M parameters.
	Trained on cased German text by Deepset.ai
bert-large-uncased-whole-word-masking	24-layer, 1024-hidden, 16-heads, 336M parameters.
	Trained on lower-cased English text using Whole-Word-Masking
bert-large-cased-whole-word-masking	24-layer, 1024-hidden, 16-heads, 335M parameters.
	Trained on cased English text using Whole-Word-Masking
bert-large-uncased-whole-word-masking-finetuned-squad	24-layer, 1024-hidden, 16-heads, 336M parameters.
	The bert-large-uncased-whole-word-masking model fine-tuned on SQuAD
bert-large-cased-whole-word-masking-finetuned-squad	24-layer, 1024-hidden, 16-heads, 335M parameters
	The bert-large-cased-whole-word-masking model fine-tuned on SQuAD
bert-base-cased-finetuned-mrpc	12-layer, 768-hidden, 12-heads, 110M parameters.
	The bert-base-cased model fine-tuned on MRPC
bert-base-german-dbmdz-cased	12-layer, 768-hidden, 12-heads, 110M parameters.
	Trained on cased German text by DBMDZ
bert-base-german-dbmdz-uncased	12-layer, 768-hidden, 12-heads, 110M parameters.
	Trained on uncased German text by DBMDZ

Table 1: HuggingFace Pre-trained BERT models

3.2.2 Datasets

Offensive Tweet Dataset: Social media has become an abstract layer where users write about whatever they want and not face the consequences. This has changed in the recent past, Twitter has begun to automatically flag offensive and hateful content and also gone as far as to permanently ban certain users from the platform. This dataset was obtained from Kaggle, it contains about 27K tweets which are categorised into 3 classes – hate-speech, offensive language and neither. The dataset was split in a 3:1 ratio during training and evaluation.

Airline Complaints Dataset: Businesses are also using social media platforms to connect with their customers and address their concerns. This dataset was also obtained from Kaggle. The dataset is relatively small, it contains 3400 samples. All samples fall into "Complaint" or "Non-complaint" category. We intentionally selected a smaller dataset for two reasons. The first is to compare the results

obtained from the BERT on very small datasets, and the second, it would make error analysis easier for us.

3.3 Method

In the following few sections, we will discuss the step untaken as part of the experiment. Some of these steps must be performed specifically for fine-tuning BERT.

3.3.1 Pre-processing

In this section, we will discuss what are the pre-processing steps required to be performed on the dataset before it is fed into the model.

BERT Tokenizer: Tokenization is a process, where the input sequence is broken down into individual words (tokens). The tokens will then be mapped to their indices in the tokenizer vocabulary. For tokenization we must use the tokenizer included with BERT.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

# Print the original sentence.
print(' Original: ', sentences[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(sentences[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences[0])))
```

```
 Original:  !!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. &amp; as a man you shou
Tokenized:  ['!', '!', '!', 'rt', '@', 'maya', '##sol', '##ove', '##ly', ':', 'as', 'a', 'woman', 'you', 'shouldn', "'",
Token IDs:  [999, 999, 999, 19387, 1030, 9815, 19454, 21818, 2135, 1024, 2004, 1037, 2450, 2017, 5807, 1005, 1056, 17612,
```

This code shows the methods: *tokenizer.tokenize()* and *tokenizer.convert_tokens_to_ids()* and the corresponding outputs which they produce. Now, these two steps can be merged into one and the method *tokenizer.encode()* does it in a single step.

We still need to format the data, specifically, we need to add [CLS] token at the beginning and [SEP] token at the end. We also need to make sure all the sequences are of the same length and finally we need to come up with something called *"attention masks"*. Attention masks differentiate the real tokens from the padding tokens.

Now, the entire tokenization involves the below-mentioned steps in the same order:

- Split the sentence into tokens.
- Add the [CLS] and [SEP] tokens in the appropriate positions.
- Map the tokens to their token ids.
- Pad all sequences such that all of them have the same length.
- Create attention masks that differentiate between the actual tokens and the padding tokens.

The *tokenizer.encode()* method performs the first four steps, but in a recent update HuggingFace added the method *tokenizer.encode_plus()* which performs all the mentioned steps in one go. We have used this method in our experiment as shown below.

```
input_ids = []
attention_masks = []

encoded_dict = tokenizer.encode_plus(
                    sent,                      # Sentence to encode.
                    add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                    max_length = 64,           # Pad & truncate all sentences.
                    pad_to_max_length = True,
                    return_attention_mask = True,   # Construct attn. masks.
                    return_tensors = 'pt',     # Return pytorch tensors.
                )

# Add the encoded sentence to the list.
input_ids.append(encoded_dict['input_ids'])

# And its attention mask (simply differentiates padding from non-padding).
attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Print sentence 0, now as a list of IDs.
print('Original: ', sent)
print('Token IDs:', input_ids)
print('Attention Mask:', attention_masks)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `tru
Original:  !!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. &amp;
Token IDs: tensor([[  101,   999,   999,   999, 19387,  1030,  9815, 19454, 21818,  2135,
          1024,  2004,  1037,  2450,  2017,  5807,  1005,  1056, 17612,  2055,
          9344,  2039,  2115,  2160,  1012,  1004, 23713,  1025,  2004,  1037,
          2158,  2017,  2323,  2467,  2202,  1996, 11669,  2041,  1012,  1012,
          1012,   102,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0]])
Attention Mask: tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2218: FutureWarning: Th
  FutureWarning,
```

Here, we can observe Token ID 101 at the beginning, this is the Token ID of the [CLS] token and we make another observation that the last token has an ID 102 which is for the [SEP] token. The [SEP] token is followed by the [PAD] tokens which have an ID 0. If we take a closer look at the Attention Mask, we can see that the IDs corresponding to the [PAD] tokens are 0s and all others are 1s. This is how BERT can differentiate between the padded tokens and the actual tokens.

3.3.2 Batching and Data loading

In this section, we explain the process in which is split into batches and fed to the model for the forward pass in training. For this, we use PyTorch Dataset and DataLoader interfaces. The Dataset interface packs the train data, the attention masks, and the label together. The DataLoader is an elegant way of feeding the batches of training or test samples into the model. The Dataset object loads the data into memory and the DataLoader object fetches data from the Dataset object and serves the data in batches.

3.3.3 Training

In this section, we discuss what are the steps taken to fine-tune the model for the task. There are two ways to use the BERT model, we will discuss both and compare the performances. The first way is to use BERT embeddings. In this method, we freeze the BERT such that during backpropagation the gradients for the BERT model are not calculated and the parameters remain unchanged. Only the parameters of the augmented layers will be updated. In the second method, we do something similar, we stack a few layers on top of the model, the only difference is that we do not freeze any parameters. Hence during back-propagation, gradients of all the parameters, inside BERT and in the augmented layers are calculated and the parameters are updated.

There are advantages and disadvantages to using both methods. While using BERT as embedding, although it is computationally cheap and very fast to train, the accuracy is limited. Whereas while fine-tuning the entire model, the model yields very good performance. This improvement in accuracy comes at the cost of time and resources though.

Coming to some technical details, the authors of the original BERT paper (Devlin et al., 2018) recommend a batch size of 16 or 32. For our experiment, we choose a batch size of 32. For the optimizer, we choose the AdamW optimizer from HuggingFace. AdamW optimizer is a modified version of the original Adam optimizer with weight decay. This means when the optimizer is close to the minima, the learning rate is reduced so that the model does not shoot away from the optimal point.

3.3.4 Evaluation

Offensive Tweets Dataset				
	$BERT_{BASE}$		$BERT_{LARGE}$	
	without internal parameters	with internal parameters	without internal parameters	with internal parameters
Training time (in seconds)	187	460	766	1851
Test Accuracy	78%	90%	81%	91%

Table 2: Evaluation on Offensive Tweets Dataset

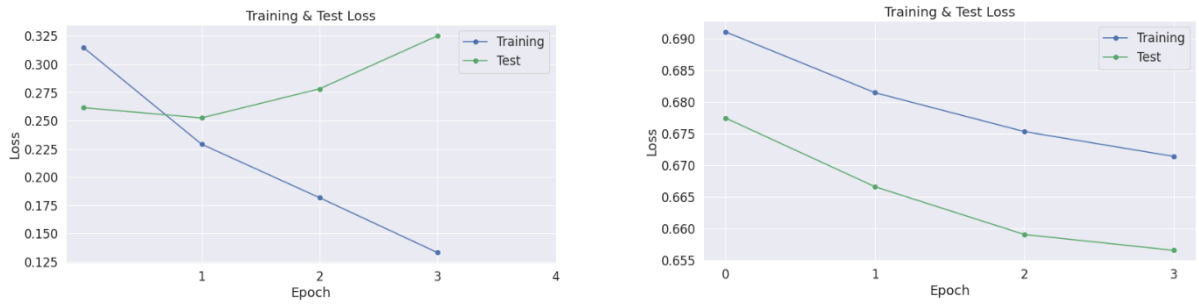
Airline Complaints Dataset				
	$BERT_{BASE}$		$BERT_{LARGE}$	
	without internal parameters	with internal parameters	without internal parameters	with internal parameters
Training time (in seconds)	26	70	108	230
Test Accuracy	65%	82%	82%	81%

Table 3: Evaluation on Airline Complaints Dataset

For our evaluations, we made use of two datasets. We used $BERT_{BASE}$ and $BERT_{LARGE}$ on both these datasets and compared the performances.

We first used the pre-trained BERT$_{BASE}$ model and added two linear layers on top of it. The output layer of the BERT$_{BASE}$ is of size 768. The size of the final linear layer is chosen based on whether it is a binary or a multi-class classification problem.

For our first experiment, we froze the internal BERT parameters and trained the model only on the augmented linear layers. As expected, since the model updated the parameters of the augmented layers only, the time to train was relatively less. For the Offensive Tweet Dataset, each epoch took on average about 187 seconds to train with a test accuracy of 78%. From Figure 8(a) we can see that the training loss is decreasing, and the test loss increases after the second epoch, which means the model is overfitting.



(a) **(b)**

Figure 8: Training and test loss curves

For the Airline Complaints Dataset, each epoch took about 26 seconds to train and achieved a test accuracy of 65%. The training graphs in Figure 8(b) correspond to the airline complaints dataset and we can see that the model is converging well.

In the second experiment, we did not freeze the internal BERT parameters. This means that during backpropagation the gradients of all the 110 million BERT$_{BASE}$ parameters along the augmented layer parameters will be calculated and the parameters would be updated. Intuitively, the training time would be more. With more parameters to update and fine-tune, it gives the model more flexibility to learn more complex functions, as a result, we can see a significant increase in accuracy.

For the Offensive Tweets Dataset, each epoch took on average about 460 seconds to train with a test accuracy of 90%. This is a 145% increase in training time with approximately 12% improvement in test accuracy. Although it still suffers from the same problem of overfitting as before.

For the Airline Complaints Dataset, each epoch took about 70 seconds to train and achieved a test accuracy of 82%. This is a 169% increase in training time with a 17% increase in accuracy. The model again overfits after two epochs.

We repeated the above two experiments with one change, replacing BERT$_{BASE}$ with BERT$_{LARGE}$. The output layer for BERT$_{LARGE}$ is of size 1024. For the Offensive Tweets Dataset with the internal BERT parameters frozen each epoch took about 766 seconds with a test accuracy of 81%. We noticed that the model overfits after two epochs. For the Airline Complaints Dataset, it took about 108 seconds for one epoch and gave a test accuracy of 82%.

Finally, we trained the entire model, that is, BERT$_{LARGE}$ with the augmented layers. For the Offensive Tweets Dataset, it took about 30 minutes for each to run and achieved a test accuracy of 91%. For the Airline complaints dataset, the model took 230 seconds to train and achieved a test accuracy of 81%.

3.3.5 Error Analysis

For simplicity, we performed error analysis on the airline complaints dataset, but the findings can be generalized to both datasets. Below we have mentioned a few wrongly classified samples and discussed what may have caused the model to misclassify these samples.

The way to obtain the misclassified samples turned out to be a challenge. The model produced a softmax output and we performed an argmax over this array to get the predicted class, this class was compared with the actual class to get the accuracy. We get the corresponding text input of the mismatched samples. These are in the encoded format, so we use the *tokenizer.convert_ids_to_tokens()* methods and use the python *join* function to decode the sample. As you can see from Table 4, the decoding is not perfect, there are some unwanted '#' symbols, but the string is good enough for our analysis.

Sl no.	Sample	Correct Class	Predicted Class
1	wait . . . is playing out ##kas ##t in its commercials . i am now a fan of both . # so ##fr ##esh ##ands ##oc ##lean	Complain	Non-complaint
2	you know who loves sitting on the tar ##mac and not moving , little kids .	Complain	Non-complaint
3	i have switched flights 3 times in 40 minutes . - love you girl . let ' s do this . we can do this . hope . i live in a place called hope .	Complain	Non-complaint
4	ya ##y , my flight was cancelled ! ! ! i can sleep in a bit ! :)	Complain	Non-complaint

Table 4: Samples misclassified due to sarcasm

The above samples were misclassified due to the sarcastic tone of the sample. The sarcastic tone is hard to identify even for a human, let alone machines. The reason for this is that the enunciation effect is void in text and to know the context, the model needs to know some part of the conversation. Researchers are working on this problem, there were several papers published in 2020 on ways to address this problem. Previous work in this area by Baruah et al., 2020 and Srivastava et al., 2020 were done on the Twitter and Reddit datasets. Unlike traditional methods, there were two input parameters along with the label. The two inputs include a list of statements and a response over which the prediction is to be made. The list of statements helps the model understand a large context of the response statement.

Sl no.	Sample fed into BERT	Sample without any pre-processing	Correct Class	Predicted Class
1	stepped it up today huge . . . waiting on to do the same . especially considering they ' re the real problem here . . .	@AmericanAir stepped it up today huge...waiting on @united to do the same. Especially considering they're the real problem here...	Complain t	Non-complaint
2	mad props to for actually des ##ig ##ing their mobile app , the rest of y ' all airlines are nasty , 12 y ##r olds could do better	Mad Props to @AlaskaAir for actually desiging their mobile app, the rest of y'all airlines are NASTY, 12 yr olds could do better	Non-complaint	Complain t

Table 5: Samples misclassified due to ambiguity

Looking at the misclassified samples in Table 5, we notice that our model struggles to identify to whom the problem is addressed. Especially when the sample contains both positive and negative sentiments. Taking a look at sample 1 from Table 5, we see that the person is praising the efforts of American airlines and asking United airlines to follow suit. Now, we can even argue whether this is considered a complaint or not, because this depends on who's perspective you are talking from. If we were looking from the American airline perspective, then this would be considered a non-compliant (a positive sentiment). Whereas, looking from United airline perspective this would be considered a complaint (a negative sentiment). We see something similar for the second sample, where the person is praising Alaska airlines and complaining about all other airlines.

This problem is predominant on Twitter because if you see the pre-processed sample, we have removed named entities like @AmericanAir, @united and @AlaskaAir. It may also be due to the shortcomings of the BERT tokenizer, the word "designing" is broken into "des", "##ig" and "##ing". This changes the meaning completely. Researchers have been working to address the first problem, they have come up with Aspect Based Sentiment Analysis (ASBA) datasets and are designing models to correctly predict these kinds of samples. Some of the work done in this area using BERT include Xu et al., 2019, Sun, Huang and Qiu, 2019 and Hoang, Bihorac and Rouces, 2019. In ABSA the input is a sentence or a few sentences and the labels include aspect, (the entity being addressed) and the sentiment toward the aspect. Most of the time there is more than one aspect-sentiment pair for each sample as we can see from Table 5.

Most other samples which were misclassified were due to overfitting or the sample length being too small or they were just too ambiguous to understand. We have shown some of them in Table 6.

Sample	Correct Class
holy shit	Complaint
when is a cancelled flight not cancelled ?	Complaint
you sp ##n boys have the worst luck with air travel !	Non-complaint

Table 6: Samples misclassified

3.4 Conclusions

Looking at the performance of BERT and comparing them with previously available data of traditional NLP methods like CNN-based models, TF-IDF or bag-of-words, BERT unsurprisingly performs significantly better in terms of performance with very little or no increase in training time.

We can see that by using pre-trained models and fine-tuning them to specific tasks, we can achieve excellent results with very little computation power and training time. We also noticed that regardless of the size of the dataset and the choice of specific BERT model (base or large) the model overfits within the first 2 or 3 epochs. Especially for datasets with fewer training samples like the Airline complaints dataset. The reason for this is that the pre-trained models have already been trained on unannotated data for hundreds of epochs, so when we come to the fine-tuning stage, the model needs to just make minimal updates to all the parameters.

Broadly, we can see $\text{BERT}_{\text{BASE}}$ gives good performance within a relatively short time. We can say that $\text{BERT}_{\text{LARGE}}$ may be overkill for the size of the datasets used in our experiments. Looking at Table 2 and Table 3, we can observe that the improvement in the accuracy in $\text{BERT}_{\text{LARGE}}$ is extremely less given the time and computational power it requires. So largely $\text{BERT}_{\text{BASE}}$ should be the go-to model for training on any reasonably sized datasets.

With pre-trained models being popular than ever before, anyone with decent computation power can achieve close to state-of-the-art results on a large number of NLP applications. Since the introduction of Transformers (Vaswani et al., 2017) and consequently BERT (Devlin et al., 2018), there have been many such models some of which have been already discussed previously. These models have ushered in a new era in the NLP community.

4 Self-Assessment

In this section, we have discussed things that went right and the things that did not go well. The sections in which we struggled and how we overcame these issues. We also talk about the things we learned both during our experimentation and documentation.

4.1 Strengths

Reading research papers and blogs: We found the use of Google scholar immensely helpful. Up until our project work, we had always wondered why a lot of our professors and researchers used it. We found reading various research papers and other papers relating to the same topic easier than ever before. We did even try alternatives to it for curiosity sake, and some others which we found helpful were Microsoft Academic and Semantic Scholar.

Google Colab: Another Google service that makes it to the list would be Google Colab. Training out BERT models with millions of parameters on our local machine was very time consuming, especially without a GPU. We gave Google Colab a try and since we did, we have not looked back. Google Colab became our go-to platform, it offered us a free Tesla K80 GPU for training which was a game-changer. It is easy to use, especially for someone who is familiar with Jupyter notebook. It cut our training by minutes, looking back at it, it certainly would have been an uphill task without it.

4.2 Weaknesses

Scoping the project: One of the places where we struggled a bit was scoping out our project. Things like, experiments to include and whether there was sufficient time to carry out the analysis on these experiments turned out to be a challenge. Next time around, we would spend a couple of session early on, to nail down the expectations and the scope of our project.

Lack of resources in certain places: Working on models which were relatively newly introduced, it was hard to find code samples specifically using pre-trained models. Also, lack of a powerful machine to fine-tune the model. We overcame the latter by using Colab, although there were restrictions on daily usage.

4.3 Opportunities

Diving into Natural Language Processing: It is an exciting time for the NLP community, with so many different pre-trained models, one better than the other in different aspects. This renewed interest in the field can be hugely credited to the development of the Transformer (Vaswani et al., 2017), and subsequently BERT. Google also announced in 2019, that it had integrated BERT into Google search. Not only Google search, but Grammarly also makes use of a modified version of BERT.

Fine-tuning pre-trained models with PyTorch: Fine-tuning BERT turned out to be a challenge as we found it hard to find resources and code samples. But once we did, we were really comfortable in using any other model in place of BERT. This was an amazing addition to our skillset. We even learnt new functions and cemented our understanding of already known functions in PyTorch.

4.4 Threats

The main obstacle that we faced is getting clean labelled datasets. The ones which we have used still have some issues they aren't perfect. We think that this, in general is the case for all NLP researchers. Although, pre-trained models have made it easier to train on smaller datasets there is a long way to go.

5 Professional Issues

5.1 Privacy and Informed Consent

While using social media data the biggest challenge is getting consent from the users of the platform. In the case of Twitter, working with large datasets, it would be difficult to obtain informed consent from all the users. There may be multiple reasons for this, the user may not reply to the request or simply the account might not be in use anymore. Apart from this difficulty, the process is also time-consuming and can cost a lot of money. Many ethical review boards consider publicly available data as unproblematic and hence not requiring user consent. Researchers think due to the above concerns there is a growing need for them to work alongside these social media companies. These social media companies can ask users for their consent during sign up phase or use pop-up dialogue box asking whether they are okay if their data is used for research purposes.

Researchers have also explored whether anonymizing and de-identifying the data could be used to avoid informed consent. They found that by just changing the account from public domain to private domain is not enough to maintain privacy. This was discovered in a study using Facebook data where the researchers anonymized the data by removing personal identifiers such as names and still using reverse engineering, were able to retrieve the original identifiers. This is not the only problem, let us say that, once the data is de-identified and cannot be reversed engineered to get the original identifiers, critical information may have been lost in the process, which can significantly affect the research.

5.2 Legal Issues

Twitter's Terms of Service and Privacy Policy are documents that govern how the users may access and use the Twitter platform (Weller et al., 2014). A justification often given in terms of ethical or legal implications in data used without informed consent is that the reuse of data is permitted by Twitter's terms of service. Additionally modifying tweets by removing user IDs will still violate Twitter's terms of service.

6 Future Directions

Something which we found exciting while doing this project is the ease in which we were able to fine-tune the model with excellent results. While carrying out background research we came across many more pre-trained models each with its unique set of attributes. We hope to learn and explore these models. Apart from this, we came across many research papers dealing with sarcasm detection and about Aspect-Based Sentiment Analysis while conducting error analysis. This is something we would read up on and want to pursue in our future experiments. Below we see can many state-of-art models being developed, pre-trained and open-sourced. We see these models with billions of parameters all pre-trained for anyone to download and fine in a couple of minutes.

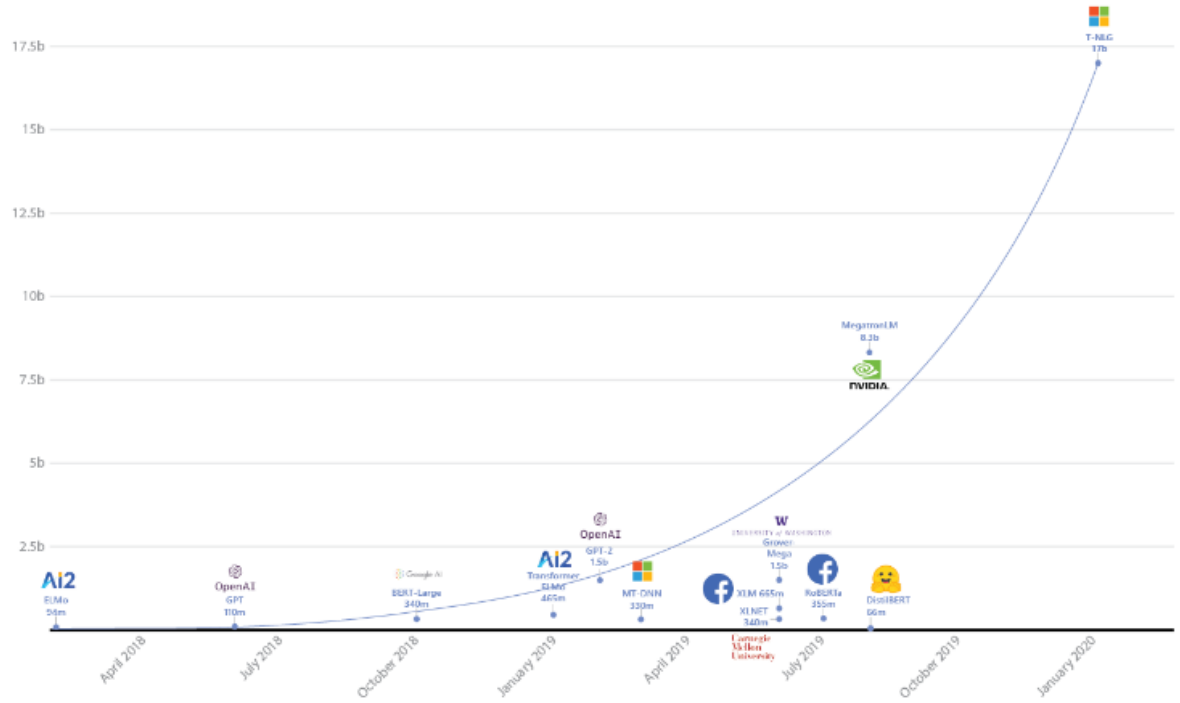


Figure 9: Number of parameters in the pre-trained model (Sanh et al., 2019)

7 How To Use My Project

The working copy of the code is stored in "MyProject.zip". After decompressing the file, you will see two files, the first one is the Thesis and the second is "BERT_NN_Layer.ipynb" file. This file contains the model implementation and can be run on your local machine or by uploading to Google Colab. The datasets have been uploaded to my drive, you can access them by clicking here. Once you have all the relevant files you can run the code in the Jupyter notebook one by one and see the corresponding output.

We have referred multiple code bases for us to write our code. We encourage you to have a read for yourself. The following blogs and their implementation were used while writing our code :

- SKIM AI : Tutorial : How to fine-tune BERT (Tran, 2021)
- Towards data science : BERT for dummies (Kana, 2019)
- Chris McCormick Blog : BERT Fine-Tuning with PyTorch (McCormick, 2021)

References

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, *9*(8), pp.1735-1780.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I., 2018. Improving language understanding by generative pre-training.

Weng, L., 2021. *Generalized Language Models: CoVe, ELMo & Cross-View Training*. [online] TOPBOTS. Available at: <https://www.topbots.com/generalized-language-models-cove-elmo/> [Accessed 2 December 2021].

Devlin, J. and Chang, M., 2021. *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing*. [online] Google AI Blog. Available at: <https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html> [Accessed 2 December 2021].

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K. & Zettlemoyer, L. (2018), 'Deep contextualized word representations' , cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.

Alammar, J., 2021. *The Illustrated Transformer*. [online] Jalammar.github.io. Available at: <https://jalammar.github.io/illustrated-transformer/> [Accessed 3 December 2021].

Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Exxactcorp.com. 2021. *A Deep Dive Into the Transformer Architecture – The Development of Transformer Models | Exxact Blog*. [online] Available at: <https://www.exxactcorp.com/blog/Deep-Learning/a-deep-dive-into-the-transformer-architecture-the-development-of-transformer-models> [Accessed 5 December 2021].

Pennington, J., Socher, R. and Manning, C.D., 2014, October. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In Proceedings of the IEEE international conference on computer vision, pages 19–27.

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. and Klingner, J., 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Ba, J.L., Kiros, J.R. and Hinton, G.E., 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Versloot, C., 2021. *Intuitive Introduction to OpenAI GPT – MachineCurve*. [online] MachineCurve. Available at: <https://www.machinecurve.com/index.php/2021/01/02/intuitive-introduction-to-openai-gpt/> [Accessed 3 December 2021].

Baruah, A., Das, K., Barbhuiya, F. and Dey, K., 2020, July. Context-aware sarcasm detection using bert. In *Proceedings of the Second Workshop on Figurative Language Processing* (pp. 83-87).

Srivastava, H., Varshney, V., Kumari, S. and Srivastava, S., 2020, July. A novel hierarchical BERT architecture for sarcasm detection. In *Proceedings of the Second Workshop on Figurative Language Processing* (pp. 93-97).

Xu, H., Liu, B., Shu, L. and Yu, P.S., 2019. BERT post-training for review reading comprehension and aspect-based sentiment analysis. arXiv preprint arXiv:1904.02232.

Sun, C., Huang, L. and Qiu, X., 2019. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence. arXiv preprint arXiv:1903.09588.

Hoang, M., Bihorac, O.A. and Rouces, J., 2019. Aspect-based sentiment analysis using bert. In Proceedings of the 22nd Nordic Conference on Computational Linguistics (pp. 187-196).

Weller, K., Bruns, A., Burgess, J., Mahrt, M. and Puschmann, C., 2014. *Twitter and society [Digital Formations, Volume 89]*. Peter Lang Publishing.

McCormick, C., 2021. *BERT Fine-Tuning Tutorial with PyTorch · Chris McCormick*. [online] Mccormickml.com. Available at: <https://mccormickml.com/2019/07/22/BERT-fine-tuning/> [Accessed 6 December 2021].

Kana, M., 2019. *BERT for dummies — Step by Step Tutorial*. [online] Medium. Available at: <https://towardsdatascience.com/bert-for-dummies-step-by-step-tutorial-fb90890ffe03> [Accessed 6 December 2021].

Tran, C., 2021. *Tutorial: Fine-tuning BERT for Sentiment Analysis - by Skim AI*. [online] Skim AI. Available at: <https://skimai.com/fine-tuning-bert-for-sentiment-analysis/> [Accessed 6 December 2021].

Sanh, V., Debut, L., Chaumond, J. and Wolf, T., 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.