

ABUSIVE LANGUAGE DETECTION

by

KAZA PHANI ROHITHA	411634
BURRE CHANDU	411613
NAMPALLY NIHAL	411654

Under the guidance of

Dr. K Hima Bindu



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY ANDHRA PRADESH
TADEPALLIGUDEM-534102, INDIA**

May - 2020

ABUSIVE LANGUAGE DETECTION

*Thesis submitted to
National Institute of Technology Andhra Pradesh
for the award of the degree*

of

Bachelor of Technology

by

KAZA PHANI ROHITHA	411634
BURRE CHANDU	411613
NAMPALLY NIHAL	411654

Under the guidance of

Dr. K Hima Bindu



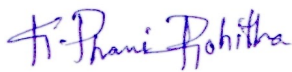
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY ANDHRA PRADESH
TADEPALLIGUDEM-534102, INDIA**

May - 2020


© 2018. All rights reserved to NIT Andhra Pradesh

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.



Kaza Phani Rohitha
Roll No: 411634
Date: 25-05-2020



Burre Chandu
Roll No: 411613
Date: 25-05-2020



Nampally Nihal
Roll No: 411654
Date: 25-05-2020

CERTIFICATE

It is certified that the work contained in the thesis titled “**Abusive Language Detection**” by “Kaza Phani Rohitha” bearing Roll No:411634, “Burre Chandu” bearing Roll No.411613 and “Nampally Nihal” bearing Roll.No.411654 have been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. K Hima Bindu
Dept. of Computer Science and Engineering
N.I.T. Andhra Pradesh
May, 2020

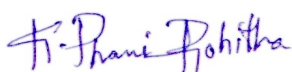
ACKNOWLEDGEMENTS

First and foremost, We would like to thank our guide Dr. K Hima Bindu for guiding us thoughtfully and efficiently through this project, giving us an opportunity to work at our own pace along our own terms. We are extremely grateful for your assistance and suggestions throughout our project.

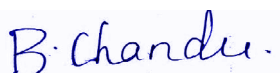
We would also like to thank you for making us aware about the PCCDS conference and motivating us to write a research paper by giving us quality inputs. We have learnt a lot while writing the research paper and we are glad that our paper has been accepted at the conference.

We avail ourselves of this proud privilege to express our gratitude to all the faculty of the department of Computer Science and Engineering at NIT Andhra Pradesh for emphasizing and providing us all the necessary facilities throughout the work.

We offer our sincere thanks to all our friends,fellow mates and other persons who knowingly or unknowingly helped us to complete this project.



Kaza Phani Rohitha



Burre Chandu



Nampally Nihal

LIST OF FIGURES

Figure	Title	Page No
1	Transformer Architecture	8
2	GPT-2 Model Architecture	10
3	Transformers for Language Modelling	10
4	Transformer-XL for Language Modelling	11
5	BERT Architecture	13
6	Permutation Language Modeling in XLNet	15
7	Sequential Transfer Learning in NLP	16
8	Model Architecture for Classification	17
9	BERT Input Formatting	18
10	LSTM Classifier Architecture	20
11	Attention Context based architecture	23

LIST OF TABLES

Table	Title	Page No
1	Results on OLID dataset-TASK A	30
2	Results on OLID dataset-TASK B	30
3	BERT BASE (OLID-TASK C)	31
4	BERT LARGE (OLID-TASK C)	31
5	BERT BASE Results on Toxic Comment Classification Dataset	31
6	BERT BASE Results on second approach	33
7	BERT BASE Results on Toxic Comment Classification Dataset	34
8	Comparison of LSTM and Attention results	35

NOTATIONS

NOTATION	MEANING
NLP	Natural Language Processing
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory Units
DL	Deep Learning
NMT	Neural Machine Translation
FFN	Feed Forward Neural Network
SGD	Stochastic Gradient Descent
LM	Language Modeling
AR	Auto Regressive
NLU	Natural Language Understanding
BERT	Bidirectional Encoder Representations from Transformers
XLNet	Generalized Auto-Regressive model for NLU
J	Used to represent cross entropy cost function
OLID	Offensive Language Identification Dataset
TPU	Tensor Processing Unit
GPU	Graphics Processing Unit
AUC	Area Under Curve
PR-Curve	Precision-Recall Curve

ABSTRACT

Massive amount of text content being generated in online social media platforms associated sadly it encompasses an abundance of offensive content. The abusive texts can be in the form of one-to-one private chat discussions, public posts about something or someone, a speech or a comment, etc. This downside ought to be addressed globally so as to access non abusive online content. The abusive language detection task is quite challenging because it negotiates with text data and the text data is so complex to handle because it comprises many languages (multilingual). The text still contains heaps of noise, and also the context is discourse. The problem with natural language processing is that there is an enormous vocabulary that the model must understand and at the same time it must capture the context when dealing with sequences of text. To extract the efficient features from text and to represent the text in a contextual form, we should train the model on an enormous corpus. As Deep Learning is greedy for huge data, and as it performs eminently by automatic feature extraction, we have a tendency to use Neural Language Models to extract the prime quality features from text and for its representation. This report presents the usage of two Natural Language Processing (NLP) models to spot and discover whether or not the text is abusive, threatening or targeting any individual or a group. The problem this project concentrates on is text classification and in particular multiclass multi-label text classification using deep learning models. BERT and XLNet are popular language models that have obtained state-of-the-art results on many NLP tasks. These two models are pre-trained models and this project makes use of them and fine tunes the models for abusive language detection. We have also proposed two architectures (one is LSTM based and other is attention based) where the features are extracted from BERT and given to those architectures to make predictions and achieved better results compared to the prevailing techniques.

TABLE OF CONTENTS

Content	Page No
Title	i
Declaration	ii
Certificate	iii
Acknowledgments	iv
List of Figures	vi
List of Tables	vi
List of Symbols and Abbreviations	vii
Abstract	viii
Table of Contents	ix

CONTENTS

1 INTRODUCTION	1
2 LITERATURE REVIEW	3
2.1 Language Modeling	3
2.1.1 Language Modelling in Sequence-to-Sequence Models	3
2.2 Attention Mechanism in Language Modelling	4
2.2.1 Bahdanau Attention	4
2.3 Transformer	5
2.3.1 Self-Attention	5
2.3.2 Key, Value and Quer	5
2.3.3 Encoder	5
2.3.4 Multi-Head Attention	7
2.3.5 Decoder	8

2.3.6	Dropout	9
2.3.7	Residual Connections	9
2.3.8	Layer Normalisation	9
2.4	Open AI's GPT-2	9
2.5	Transformer and Transformer-XL as Language Models	10
2.5.1	Using Transformers for Language Modelling	10
2.5.2	Using Transformer-XL for Language Modeling	11
2.6	BERT	12
2.7	XLNet	15
3	PROPOSED APPROACH	16
3.1	Problem Definition	16
3.1.1	Sequential Transfer Learning Language Models	16
3.1.2	Fine Tuning	17
3.2	Model Architectures	17
3.2.1	Input Formatting	18
3.2.2	Output from BERT encoder	18
3.2.3	Classifier	19
3.2.4	Softmax Layer as a task-specific layer	19
3.2.5	BERT Sequential Output with LSTM and weighted loss function	20
3.2.6	BERT Sequential output with Attention	23
3.3	Loss Functions	24
3.3.1	Weighted Cross Entropy for Binary Classification	24
3.3.2	Weighted Cross Entropy for Multi-Class Classification	24
3.4	Algorithm	25
4	EXPERIMENTAL PROCEDURE	26
4.1	Dataset Description	26
4.2	Feature Extraction	26
4.3	Implementation Details	27

	4.4 Training and Testing datasets	27
	4.5 Evaluation Metrics	28
	4.6 Evaluating the skill of the model	28
	4.7 Model Directory	28
	4.8 Miscellaneous experiments	29
5	RESULTS AND ANALYSIS	30
	5.1 Neural Network as a task specific layer	30
	5.2 Using Sequential Output, LSTM and Weighted Loss Function	33
	5.3 OLID 2020 Dataset	35
	5.4 Comparison of LSTM and Attention Context Vector approaches	35
6	APPLICATIONS	36
7	CONCLUSIONS AND FUTURE SCOPE	37
	REFERENCES	38
	APPENDIX	40

CHAPTER 1

INTRODUCTION

There is an exponential growth in the number of users using social media and it is estimated that there are around 330 million monthly active users and 145 million daily active users on twitter. On an average there are 500 million tweets sent out per day. So there is a lot of text content being generated online by the users. Unfortunately, the social media is victimizing the users in form of pornography, gambling, online scams, sexual abuses, women and child harassment etc. Such problems which are related to online abuse are hate speech, cyber bullying and offensive posts which are highly sensitive to children and few people. Most of the social media stars and celebrities deactivated their social accounts due to hurtful comments on their personal life which are posted on their social media posts. From facing bullying, harassment and messages from the haters on the internet, there are plenty of reasons why stars are going on a social media detox and staying off for a long duration of time. About 20% of the suicides among children are reported due to online or cyber bullying. It is also reported that every 30 seconds there is a toxic tweet on women and around 40% of internet users face online bullying.

Hence it is necessary to stop the abuse content being posted on social media and other forms of measures to mine and delete the toxic comments and posts. As the task is to deal with online social media the process is to be done automatically. Abuse language on social media can be in many forms such as sexism, racism, women abuse, child abuse, political criticizing, cyberbullying, identity hate, hate speech and threatening. So it is not only necessary to detect the abuse content but also to categorize them so that some political insights can be drawn from them and certain emergency steps can be taken in case of threatening like terrorists warning about bomb blasts etc.

But this task deals with natural abusive language detection which is critical because of the diverse characteristics of text. The text can be in an unstructured way and the words may be misspelled so there can be a lot of noise in the text. The context in the text can be discourse and text can be mixed up with other media like images, video etc.

The text can be multilingual with a combination of different languages. So all these diverse characteristics of text makes it difficult to perform automatic online abuse detection.

Therefore, the problem formulation for this task is defined as given a text, the text should be classified as either offensive or not offensive and upon detecting offensive, the text is targeted or not is to be identified and lastly if it is targeted text then whether it is targeted towards an individual, a group or any other is also to be found.

The field of artificial intelligence which learns patterns from text data is called Natural Language Processing (NLP). There are a lot of advancements in the field of NLP and a lot of research is happening in this field as it deals with natural human language and it is hard to identify patterns in natural language. There are statistical models initially dealt with NLP tasks. As today, due to the humongous amount of data being generated online and due to deep learning model's hunger for data, there is extensive research happening at a lightning speed in the field of deep learning (DL). So these models can be used for our task.

Hence, a NLP model is to be built which performs the abuse detection task. The model dealing with text has to address several problems. First the model has to preprocess the text data and then hidden patterns in the text are to be identified to extract numerical features which is called feature extraction. The features of text are in the form of vectors which are called word embeddings. Upon extracting features, the features are to be sent to the classifier to detect whether the text is abusive or not.

CHAPTER 2

LITERATURE SURVEY

The crucial part involving building the model is feature extraction and to extract those features there are many methods ranging from statistical models to deep learning neural network architectures. There are various ways of constructing word embeddings using n-grams, word2vec, glove etc. But as text is sequential in nature and there will be dependencies within words of sentence so the context of the whole sentence is to be captured. To get contextual word embeddings, there are language models which learn the embeddings using the language modeling task.

2.1 Language Modelling

In language modelling, the joint probability distribution for sequences of words or tokens is calculated and this is achieved by calculating distribution of one token given other tokens in a sentence. Hence, models are pre-trained with the language modeling objective that are capable of capturing enough information to predict what word comes in a sentence and can be applied to other NLP tasks like question-answering, text classification etc.

2.1.1 Language Modelling in Sequence-to-Sequence Models

The seq2seq [1] model was born in the field of language modeling. It transforms an input sequence which is source text to a target text and both sequences can be of different lengths. The seq2seq model has an encoder-decoder type of architecture, composed of:

- An encoder with LSTM or RNN units, processes the input sentences and compresses the information into a fixed length context vector also known as a thought vector.
- Instead of only using the last state of the encoder network as the decoder initial state, the initial state to the decoder is given with the context vector or thought vector to emit the transformed output.

So each hidden representation of the Recurrent Unit (RNN or LSTM) is considered a feature vector of the word. But RNNs suffer from vanishing gradient descent and cannot be parallelized due to their sequential nature.

2.2 Attention Mechanism in Language Modelling

In order to address above problems first attention mechanism in NLP was introduced by Bahdanau [1]. Attention in encoder-decoder models refers to decoders having access to all of the encoder hidden states when predicting or inferring an output element. So attention in deep learning can be interpreted as a vector of importance weights in order to predict or infer one element, such as a pixel in an image or a word in a sentence, it is estimated using the attention vector how strongly it is correlated with other elements and take the sum of their values weighted by the attention vector as the approximation of the target.

Besides offering a performance gain, the attention mechanism can also be used as a tool for interpreting the behaviour of neural network architectures, which are difficult to understand. Interestingly, attention could provide a key to interpret and explain, at least to some extent, neural network behaviour.

2.2.1 Bahdanau Attention

The attention mechanism was born to help memorize longer source sentences in neural machine translation (NMT) [1]. In traditional sequence to sequence models the output of the last RNN is treated as a 'context' vector but that cannot capture representation of longer sentences. So unlike sequence to sequence models the technique that was followed in attention is to create short connections between the context vector and the entire source input. After all the hidden states are computed at the encoder RNN blocks then the decoder should predict output based on those hidden states. The attention mechanism computes the attention score for each hidden state based on the previous hidden state of decoder and present input of decoder. Softmax layer is applied on attention scores to get the attention weights for each encoder state and finally attention weights are multiplied with encoder hidden states to get the context vector. The weights are customizable for each output element. The alignment between the source and target is learned and controlled by the context vector. Essentially the context vector consumes three pieces of information: encoder hidden states, decoder hidden states and alignment between source and target. The encoder hidden states here act as the feature embeddings of the input words.

2.3 Transformer

“**Attention is All you Need**” [2] is one of the trend setting papers in the field of Natural Language Processing. It presented a lot of improvements to the soft attention. The “transformer” model is completely built on the self-attention mechanism without using sequential recurrent architecture. The further state-of-the-art models which are based on transformer architectures are BERT, XLNet, OPENAI-GPT.

2.3.1 Self-Attention

Self-attention, also known as intra-attention, is one type of attention mechanism in which different positions of words of a single sequence are related in order to compute a meaningful and contextual representation of the same sequence.

2.3.2 Key, Value and Query

The major component in the transformer is the unit of multi-head self-attention mechanism. The transformer views the encoded representation of the input as a set of key-value pairs, (K,V), both of same dimension(input sequence length); in the context of NMT, both the keys and values are the encoder hidden states. In the decoder, the previous output is compressed into a query (Q) and the next output is produced by mapping this query and the set of keys and values. The transformer adopts the scaled dot-product attention: “the output is a weighted sum of the values. The weight assigned to each value is computed by the dot-product of the corresponding query with all the respective keys”:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d}}\right)V \quad \text{---- (1)}$$

2.3.3 Encoder

Encoder has totally two stages:

- Self Attention
- Feed Forward Neural Network

The calculation of self-attention [18] has following steps:

1. The first and foremost step in calculating the self-attention is to create three vectors: a Query vector, a Key vector, and a Value vector. These vectors are created by multiplying the input embedding matrix three separate matrices that were trained during the training process.
2. The second step in calculating self-attention is to calculate a score for each word against remaining other words. The score is calculated by taking the dot product of the query vector and the key vector of the respective word that is currently being scored.
3. The third and fourth steps are to divide the scores that are calculated in step-2 by the square root of the dimension of the key vectors (64-used in paper). This helps in having more stable gradients. After dividing, these values are passed through a Softmax function for normalisation of scores.
4. The fifth step is to multiply each value vector by the softmax score calculated in step-4. The intuition here is to give weights to the values of the words to be focused on, and drop weightage for irrelevant words by multiplying them with very small numbers.
5. The final step is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position.

Calculation of Self-Attention with dimensions:

X - Input Matrix (**L x 512**) L - Maximum Sequence Length

$$1. Q = X.W^Q \quad (L \times 64)$$

$$V = X.W^V \quad (L \times 64)$$

$$K = X.W^K \quad (L \times 64)$$

$$2. score = Q.K^T \quad (L \times L)$$

$$3. SMAX = softmax(\frac{Q.K^T}{8}) \quad (L \times L)$$

$$4. SMAX.V \quad (L \times 64)$$

$$5. attention_{head1}(Z_0) = \frac{Q.K^T}{\sqrt{d_k}}.V \quad (L \times 64)$$

2.3.4 Multi-Head Attention [18]

The paper further advances the self-attention layer by adding a mechanism called “multi-headed” attention. This improves the performance of the attention layer in two ways:

- It expands the model’s ability to focus on different positions.
- It gives the attention layer multiple “representation subspaces”. With multi-headed not only one, but multiple sets of Query/Key/Value weight matrices (the Transformer uses eight attention heads, so it ends up with eight sets for each encoder or decoder).
- Each of these sets is randomly initialized. Then, after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.

$$Multihead(Q, K, V) = Concat(attention_{head1}, attention_{head2}, ..., attention_{head8}) \text{ ---- (2)}$$

6. Multihead(Q,K,V) (**L x 64 x 8 = L x 512**)

This last matrix (output) is given to **FEED-FORWARD NEURAL NETWORK** with W1 and W2 as the weight matrices.

$$FFN(x) = max(0, xW1 + b1)W2 + b2 \text{ ----- (3)}$$

7.Concat all Z0...Z7 and multiply that Z matrix with the W0 matrix learned during training. Z matrix is 512 x 512 matrix.

8. $Multihead(Q, K, V) (L \times 512) \times (512 \times 512) = (L \times 512)$. So the output from the encoder will be of dimension (**L x 512**)

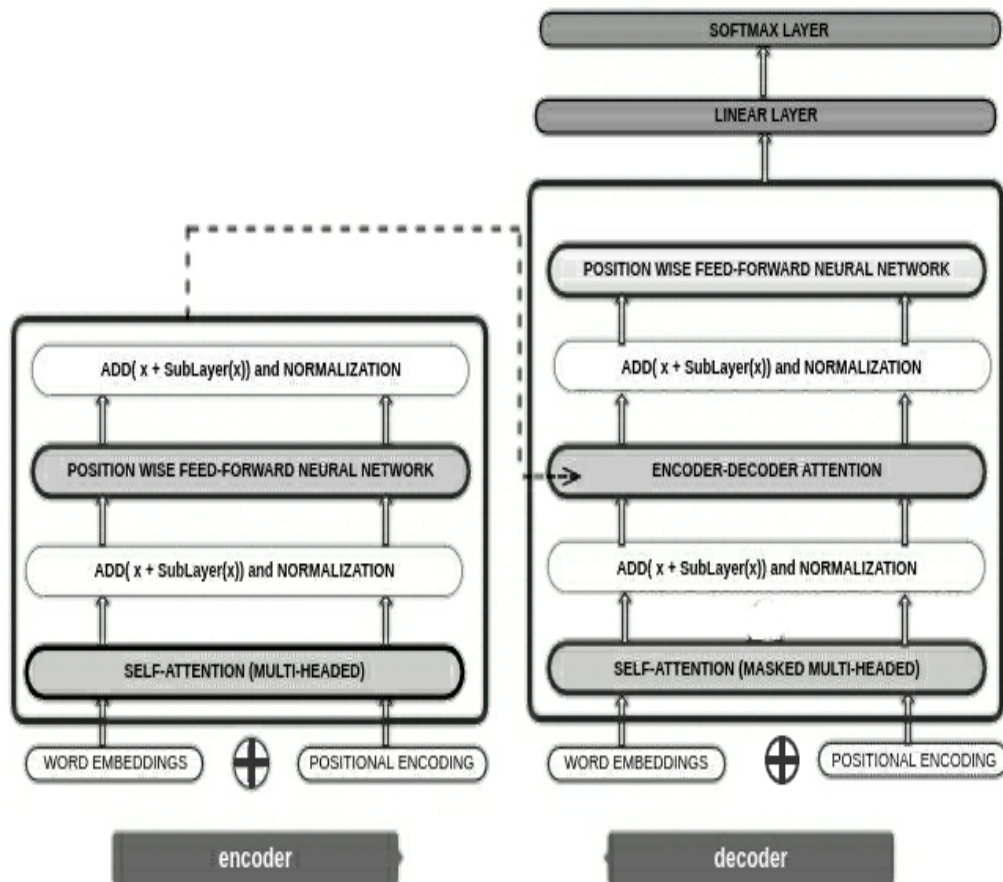


Fig.1 Transformer Architecture [2]

2.3.5 Decoder

Decoders use the same architecture as encoders but with one additional mechanism in the self attention layer called Masked Multihead Attention layer.

Masked Multihead Attention Layer

- When summing up word embeddings and position embeddings there is a problem of leaking "future" values when making predictions. To prevent the future leak of information, masking is used. This is done by performing a pointwise product of Q and K and the upper triangular matrix of ones.
- This method zeroes out the similarities between the word that is currently being processed and the words that appear after it. It prevents the model from memorizing the future words before training.

- Since such information is removed, it cannot be used by the model, and guarantee that only similarity to the preceding words is considered.

2.3.6 Dropout

Dropout is used in a few different places throughout the Transformer. In this technique, a random subset of neurons are not used in computation during each forward or backward pass to help prevent overfitting.

2.3.7 Residual Connections

If some function $f(x)$ is calculated, a “residual connection” produces the output $f(x)+x$. In other words, the original input is added back onto the output that was just calculated. Residual Connections are used in deep neural networks so that the information flows through all the layers.

$$H(x) = F(x) + x \text{ where 'x' is identity function}$$

Rather than learning $F(x)$ in a simple deep network which can have degradation problems it's helpful for the model to give accurate results. x is adding the deficiencies in $F(x)$ at each skip connection if it is less than $H(x)$ and absorbing all the errors in $F(x)$ is more than $H(x)$.

2.3.8 Layer Normalisation

This is a method that normalizes inputs across the features (as opposed to batch normalization which normalizes features across a batch). This method works better because unlike batch normalisation this computes mean and variance across features so dependencies between features are captured and not the dependencies between data points. Encoder and Decoder uses residual connections and layer normalisation.

2.4 Open AI's GPT-2

Language Models are Unsupervised Multi Task Learners [6] GPT-2 is used for language modelling just by using only the decoder blocks of transformer [3]. The transformer decoder blocks are stacked as high as practically possible, feeding them massive amounts of training text, and throwing vast amounts of compute at them for language modelling task. [6]

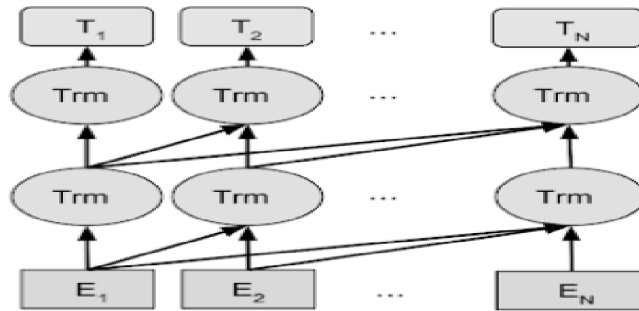


Fig.2 GPT-2 Model Architecture [6]

2.5 Transformer and Transformer-XL as Language Models

2.5.1 Using Transformers for Language Modelling

Vanilla Transformers [7] are capable and have a potential of learning longer-term dependency, but are limited by a fixed-length context in the setting of language modeling. Transformers also suffers from context fragmentation due to training on chunks of text. This architecture doesn't suffer from the problem of vanishing gradients. But the context fragmentation limits it to learn longer dependencies within text. During the evaluation phase, the segment is shifted to the right by only one position. The new segment has to be processed entirely from start (first transformer input) to end. This whole evaluation method is quite computationally intensive [19].

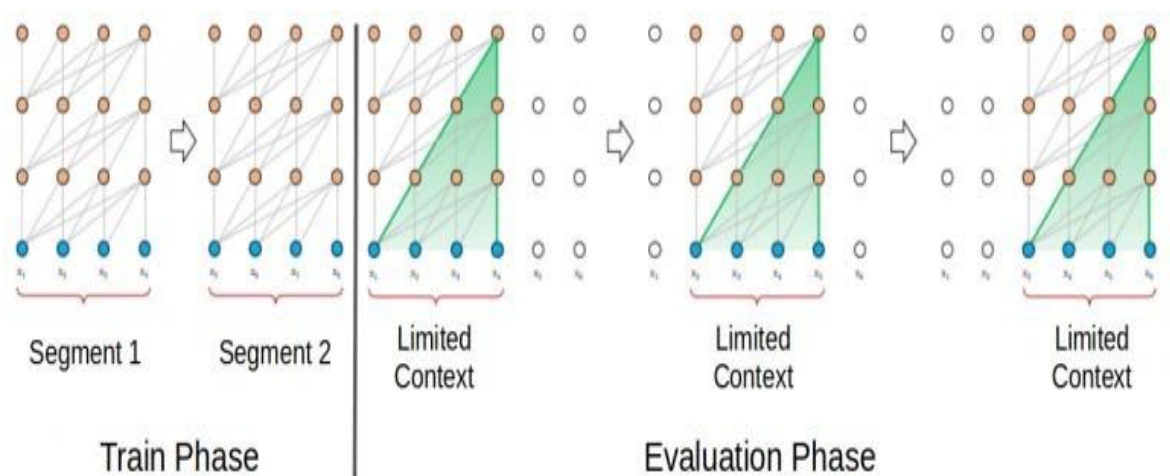


Fig.3 Transformers for Language Modelling [8]

2.5.2 Using Transformer-XL for Language Modeling

To overcome the problem of transformers for language modeling the transformer-XL-Attentive Language Models Beyond a Fixed-Length Context [8] used two methods: [19]

- Recurrence Mechanism
- Relative Positional Encoding

Recurrence Mechanism: The goal of the recurrence mechanism is to enable long-term dependencies by using information from previous segments. TransformerXL processes the first segment of tokens but stores the outputs of the hidden layers as a memory unit. When a segment is processed the output of the previous hidden layer of that segment and the output of the previous hidden layer from the previous segment are concatenated and then used to calculate the Key and the Value matrices of the current segment. This method reduces context fragmentation and the matrices can be used as embedding matrices.

Relative Positional Encoding: The recurrence mechanism also introduces a new challenge. The original positional encoding handles each segment separately and, as a result, tokens from different segments have the same positional encoding. Instead, the model learns a new positional encoding that is part of each attention module, as opposed to encoding position only before the first layer, and is based on the relative distance between tokens and not their absolute position.

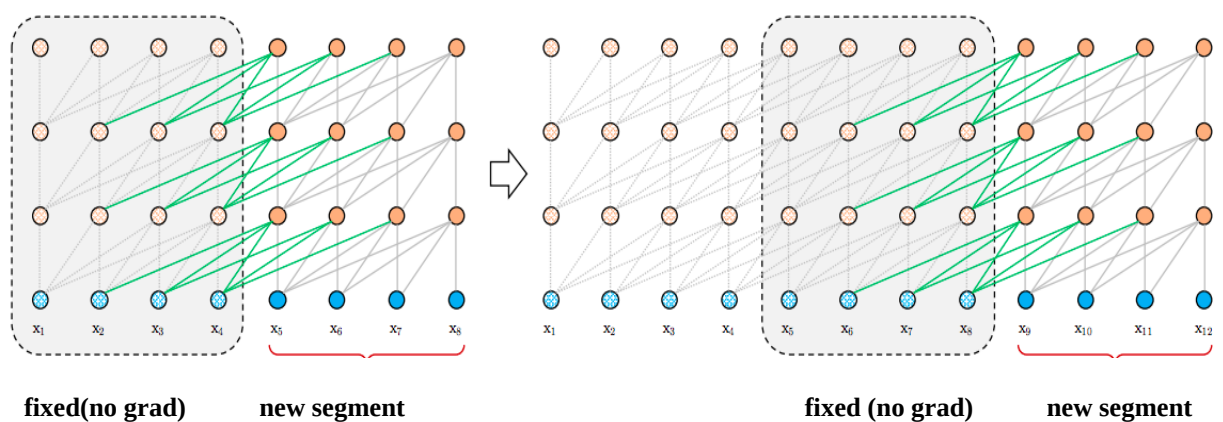


Fig.4 Transformer-XL for Language Modeling [8]

2.6 BERT

The key innovation of this paper (BERT) [4] is applying the transformer to language modelling but using the bidirectional mechanism. This is different from previous models where they looked at text sequences only in one direction i.e., from left to right or from right to left. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of natural language context and flow than a single-direction language model. BERT is a multi-layer bidirectional transformer encoder. There are two models in the paper.

- BERT Base – 12 layers (Transformer blocks), 12 attention heads, and 110 million parameters.
- BERT Large – 24 layers (Transformer blocks), 16 attention heads, and 340 million parameters.

BERT is pre-trained with above mentioned architectures on two NLP tasks:

- Masked Language Modeling
- Next Sentence Prediction

Masked Language Modelling:

BERT is designed as a deeply bidirectional attention model. This means the network effectively captures information from both the right and left context of a token from the first layer itself and all the way through to the last layer of the transformer blocks.

$$\max(\theta) \quad \log p_{\theta}(\bar{x} | \hat{x}) \approx \sum_{t=1}^T m_t \log p_{\theta}(x_t | \hat{x}) = \sum_{t=1}^T m_t \log \left(\frac{\exp(H_{\theta}(\hat{x})_t^T e(x_t))}{\sum_{x'} \exp(H_{\theta}(\hat{x})_t^T e(x'))} \right) \text{ ----- (4)}$$

The arrows in figure 5 indicate the information flow from one layer to the next layer. The boxes at the top indicate the final contextualized representation of each input word. The authors of BERT also include some caveats to further improve this technique:

- To prevent the model from focusing only on a particular position or tokens that are masked, the researchers randomly masked 15% of the words.

- The masked words were not always replaced by the masked tokens [MASK] because the [MASK] token would never appear during fine-tuning which causes the problem of fine tune discrepancy.
- So, 80% of the time the words were replaced with the masked token [MASK]. 10% of the time the words were replaced with random words from the vocabulary. 10% of the time the words were left unchanged.

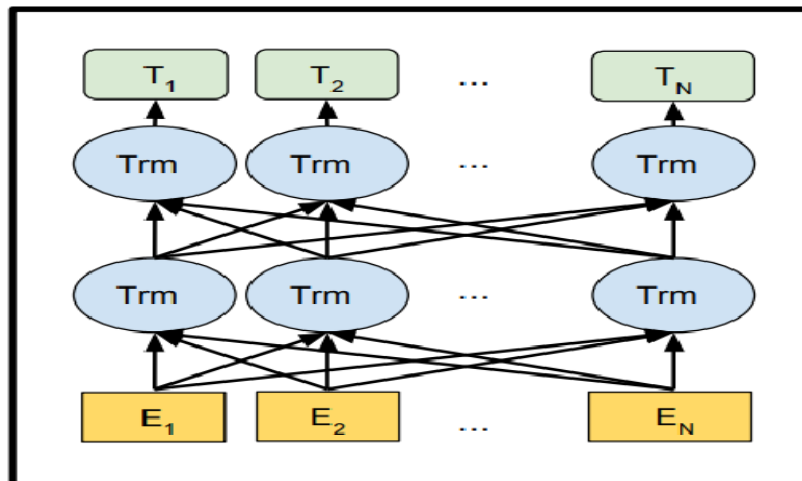


Fig.5 BERT Architecture [4]

Next Sentence Prediction:

Generally, language models do not capture the relationship between consecutive sentences i.e., Dependency between two immediately occurring sentences. Next sentence prediction task is a binary classification task in which, given a pair of sentences, it is predicted if the second sentence is the actual next sentence of the first sentence. BERT was pretrained on this task as well. For language model pretraining, BERT uses pairs of sentences as its training data. When training the BERT model, Masked LM and Next Sentence Prediction are trained together, to minimize the combined loss function of the two strategies.

Text Representations in BERT:

Every input embedding is a combination of 3 embeddings:

Position Embeddings: BERT learns and uses positional embeddings to capture the order of words in which they appear in a sentence. These are added to overcome the limitation of Transformer which is not able to capture sequential order of words.

Segment Embeddings: BERT can also take sentence pairs as inputs for tasks like Question-Answering. That is why it learns a unique embedding for the first and the second sentences to help the model distinguish between them.

Token Embeddings: These are the embeddings learned for the specific token from the WordPiece token vocabulary.

2.7 XLNet

Autoregressive (AR) Language Modelling is used for pre-training neural networks on large-scale unlabeled text corpora. AR language model is trained to encode a unidirectional context (either forward or backward). XLNet is based on Autoregressive Pre-Training. [5]

The effectiveness of XLNet is discussed with the following objectives:

1. Instead of using a fixed forward or backward factorization order as in conventional AR models, XLNet maximizes the expected log-likelihood of a sequence with respect to all possible permutations of the factorization order.
2. As a generalized AR language model, XLNet does not rely on data corruption. Hence, XLNet does not suffer from the pre-train-finetune discrepancy problem.
3. XLNet integrates the segment recurrence mechanism and relative encoding scheme of Transformer-XL into pretraining, which improves the performance especially for tasks involving longer text sequences and solving problems of context fragmentation.

XLNet also has two models one is with large parameters and the other is base model.

XLNet uses mainly two mechanisms:

- Permutation Language Modelling
- Two Stream Self Attention

Permutation Language Modelling

Permutation language modelling is a task of predicting a word in a sentence given any other number of words from a sentence in any random order. So the prediction takes among different

permutations of the order of the words in the sentence. The actual order of words in the input sentence are not changed, only the order in which they are predicted is changed. As long as the positional embedding is kept consistent, the model will see the tokens in the actual order that was present in the sentence.

$$\max(\theta) \quad E_{z \sim Z_T} = \sum_{t=1}^T \log p_{\theta}(x_{z_t} | x_{z < t}) \quad \text{----- (5)}$$

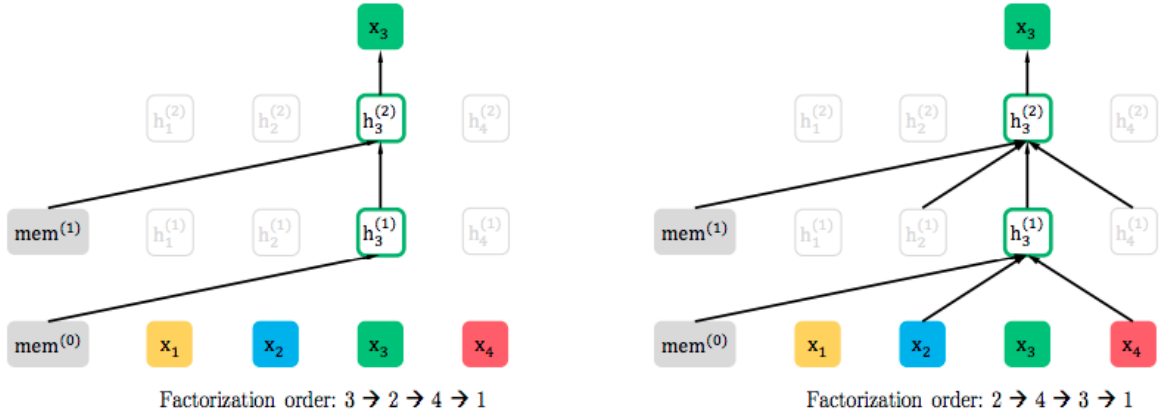


Fig.6 Permutation Language Modeling in XLNet

The permutation language model calculates key value vectors for the words that are before the token number that is going to be predicted. For example, in figure 6, for order 2-4-3-1 and when predicting token 3 the model just attends to token 4 and 2 and the previous stored memory. In figure 6, when predicting the third token, there are no token numbers before 3 in factorization order, therefore the model just attends to the memory from previous attention blocks.

Two-Stream Self Attention:

When using Permutation Language Modeling, the information that is given to the model comprises the location and order of the words they appear in the sentence. So there is a high chance that the model sees the position of the future words when predicting and this causes the leakage of facts to the model. So, there need to be one of a kind random phrase embeddings for the words and the model learns these embeddings all through training. So, each token position within the sentence is related to two vectors (one actual word embedding and any other random word embedding) at each self-attention layer to keep away from the future leak of factual information.

CHAPTER 3

PROPOSED APPROACH

3.1 Problem Definition

A model for abuse language detection should be able to extract high quality features from text and feed it to the classifier. Here, the input data is just raw text and labels can be categorical variables depending on the sub task. In order to extract high quality features from text the models need to be trained on large amounts of corpus. But training a model end-to-end with large amounts of data requires unimaginable computing resources and time to train. So, here comes the concept of transfer learning to achieve this task.

3.1.1 Sequential Transfer Learning Language Models

Transfer learning is a technique used to extract knowledge from one setting which is called source and apply it to a different target setting. In the field of NLP transfer learning is applied by using various pre-trained NLP models for the downstream tasks like classification, sequence labelling etc.

Sequential transfer learning is the form that has led to the largest enhancements so far. The well known and widely used practice is to pretrain representations on a huge unlabelled text corpus and use any method of preference to adapt these representations to a supervised target task using labelled data. BERT [4], Open AI's GPT-2 [6] and XLNet [5] are such models that use sequential transfer learning to learn the representations using language modelling.

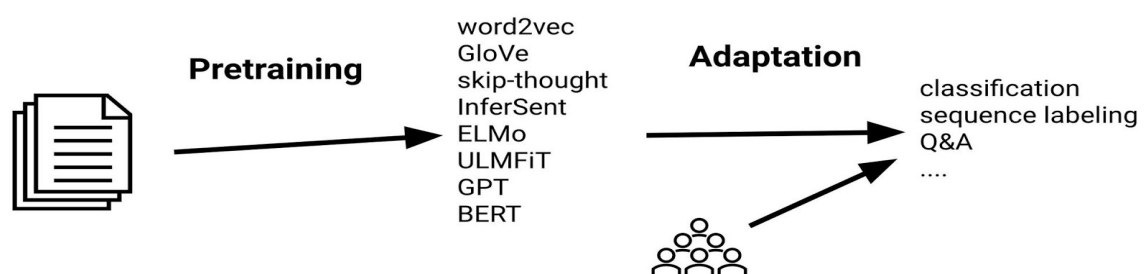


Fig.7 Sequential Transfer Learning in NLP [15]

3.1.2 Fine Tuning

In our approach, we used the state-of-the-art pretrained models like BERT and XLNet for the classification task. The features are extracted from these pretrained models and are fine tuned with task specific layers to perform predictions. The task of fine-tuning a network is to tweak the parameters of an already trained network so that it adapts to the downstream tasks. So the approach mostly focuses on fine tuning the BERT after extracting the features from it. When fine tuning the BERT model, the feature vectors which represent the words or tokens are also updated during the tuning process. This means that the feature vectors are modified to represent the words in a more meaningful way for the downstream task.

3.2 Model Architectures

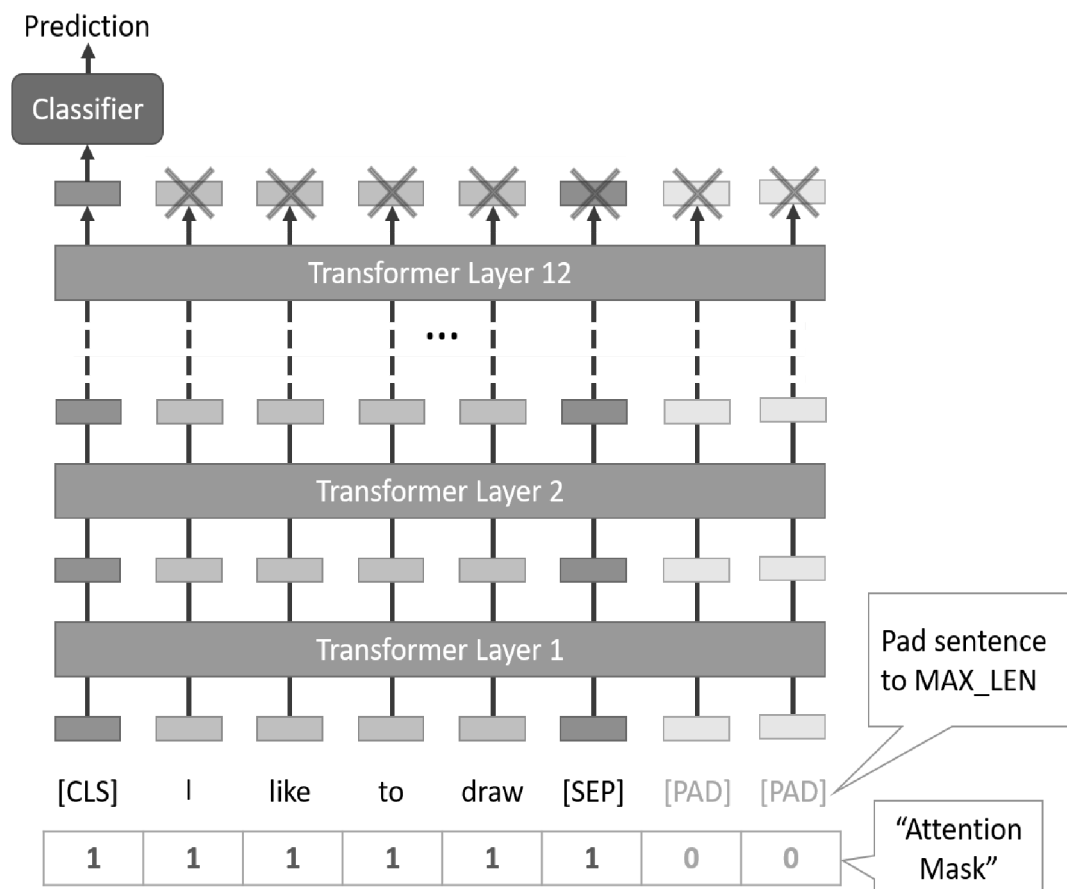


Fig.8 Model Architecture for Classification [16]

3.2.1 Input Formatting

As BERT is a pretrained model, the inputs to the model should be given in a certain format that the model will accept.

- Special tokens [CLS] to mark the beginning of the sentence and [SEP] to mark the end of the sentence.
- Tokens that conforms with fixed size vocabulary used in BERT.
- Token IDs from the BERT tokenizer which is unique for every token.
- Attention Mask IDs to indicate which elements of sequence are actual tokens and which are padded elements. The mask makes the model not to include padded elements in calculation of self attention.
- Positional embeddings to let the model know the position of the token in the sequence.

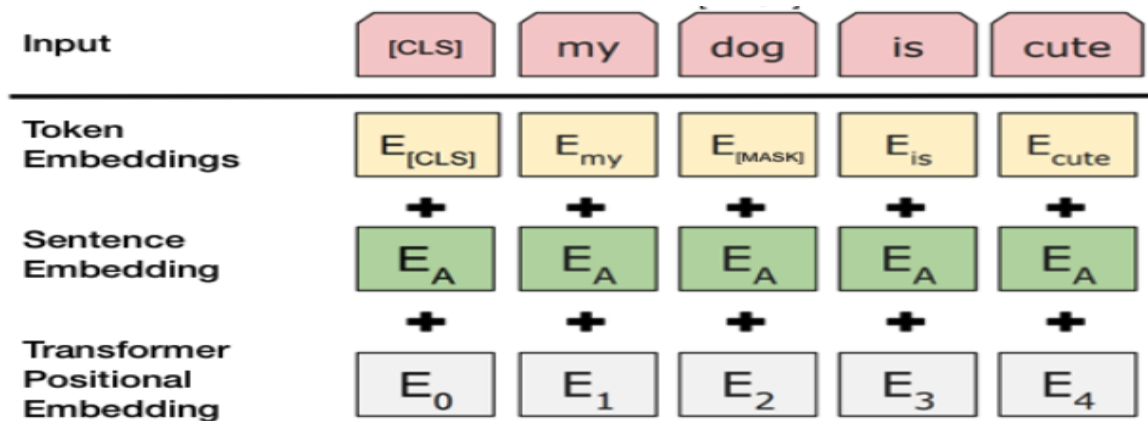


Fig.9 BERT Input Formatting [4]

3.2.2 Output from BERT encoder

The series of token embeddings pass via the variety of bidirectional transformers and number of self attention layers (12 for base and 16 for large) and finally for each token there might be a contextual word embedding coming out of the final transformer.

The embedding for class token [CLS] is treated as the representation of whole sentence input and is used to carry out further NLP tasks. Usually the output from the BERT consists of two kinds of output, one is pooled output and the other one is sequential output. The pooled output

is generally used and it depicts the representation of the whole sentence. The pooled output will be of dimension **[batch_size, 768]** and it is token embedding of the [CLS] token. The sequence output of the BERT is token embedding for each token and the dimension of sequence output is **[batch_size, max_seq_length, 768]**. In our first approach, we used only the pooled output and in the second approach we used the first 20 tokens combined with the pooled output to represent the sentence which brought better results compared to the first approach.

3.2.3 Classifier

The classifier module in the architecture takes the inputs from the BERT encoder and performs the classification task. When using a language model for other NLP tasks, a task-specific layer which is commonly a non-linear layer, is brought on top of these models. So the classifier in figure 8 can have different types of layers. Fine tuning replaces the output layer (that's firstly trained to apprehend a different quantity of instructions or a layer which is used for other NLP obligations) with a layer that recognizes the number of classes in our training data. The new output layer that is attached to the BERT model is then skilled to take the lower level features from the initial part of the network and map them to the desired output classes using Stochastic Gradient Descent (SGD) or any other loss optimization algorithm. Once this has been done, the extra layer jointly gets trained with the BERT model. Different types of task-specific layers are used for these tasks and the principal aim of the approach is to fine tune the BERT model for the downstream tasks with features from BERT as embedding inputs.

3.2.4 Softmax Layer as a task-specific layer for fine tuning

Softmax function, an *activation function* that turns numbers, usually logits into probabilities that sum to one. Softmax function outputs a vector that represents the probability distributions of a list of potential outcomes. Softmax layer is typically the last layer used in classification tasks.

The task specific layer is just simply a softmax layer. The input to the layer is 768 dimensional vector of [CLS] token and the softmax layer contains neurons equal to the number of labels in the data. Then the Softmax layer is fine tuned along with the BERT features and the weights in the Softmax layer are updated during fine tuning. The weight matrices are stored and are used in inference mode for predictions.

Limitations:

- The contextual token of only [CLS] token is considered as the whole representation of a sentence which may not always be true.
- The architecture implicitly does not handle any class imbalance problem.

3.2.5 BERT Sequential Output with LSTM and weighted loss function

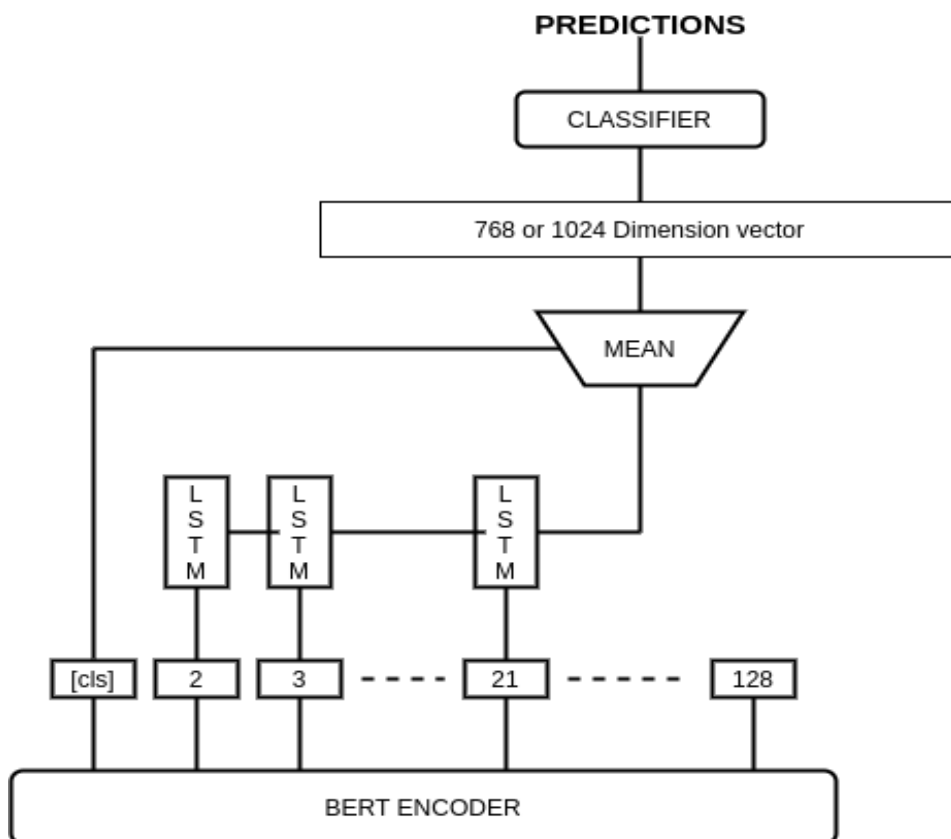


Fig.10 LSTM Classifier Architecture

The limitation of the previous approach is using pooled output which is representing the entire text in a single vector. To overcome this, we made use of another output from BERT which is sequence output i.e., After 12 layers of self-attention in BERT there is contextual representation for every word or token in the sentence including [CLS] and [SEP] token. So the approach is to represent the sentence by taking the token embedding of the useful words from the sentence.

In order to represent the sentence and preserve the context, the approach uses LSTMs. The sequence of token embeddings are given as input to the sequence of LSTM units. So the first embedding of sentence, excluding [CLS] token is given as input to the LSTM and in a sequential manner further LSTMs take output state of previous LSTM and next tokens as input and compute hidden representations and cell states. The outputs of all LSTMs are discarded and the cell state of the last LSTM is retained. The last hidden state is of 768 dimension which is the same as [CLS] token embedding dimension. The mean of two vectors is given as input to the Softmax layer for fine tuning. The equations for processing the token embeddings in stack of LSTM units is given below:

Let, $[x_1, x_2, \dots, x_{21}]$ be the token embeddings from sequence output of BERT

The equations of LSTM are follows:

$$\begin{aligned}
 i_t &= \sigma(w_i[h_{t-1}, x_t] + b_i) & i_t - \text{input gate, } f_t - \text{forget gate, } o_t - \text{output gate} \\
 f_t &= \sigma(w_f[h_{t-1}, x_t] + b_f) & w_x - \text{weights for respective gate neuron, } b_x - \text{biases} \\
 o_t &= \sigma(w_o[h_{t-1}, x_t] + b_o) & h_{t-1} - \text{output of previous LSTM block} \\
 \tilde{c}_t &= \tanh(w_c[h_{t-1}, x_t] + b_c) & \tilde{c}_t - \text{candidate for cell state at timestep } t \\
 c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t & c_t - \text{cell state (memory) at timestep } t
 \end{aligned}$$

The last cell state of the LSTM is used for computing input to the Softmax layer. To compute the last cell state of the input gate, the forget gate and the candidate for cell state (c_{21}) at the 21st LSTM is to be calculated. The mean of c_0 and c_{21} is given as input to the Softmax layer.

$$\hat{c}_{21} = \tanh(w_c[h_{20}, x_{21}] + b_c)$$

$$c_{21} = f_{21} * c_{20} + i_{21} * \hat{c}_{21}$$

$$c_0 = \text{token embedding of the [CLS] token}$$

$$c_{768} = \sum_{i=0}^{767} \frac{c_0[i] + c_{21}[i]}{2} \quad [\text{input to the Softmax layer}]$$

The main objective of this architecture not only limits to take the [CLS] token embedding as input but also the other token embeddings as well. The maximum length of the sentence used is 128 but in order to avoid taking padded tokens as input, which may induce noise, only the first 20 token embeddings are used in representation. The motive to take 20 embeddings is because the average length of sentences in the training dataset is found to be around 20. We can tweak this parameter but in the project, 20 tokens were used and brought better results.

The other problem with the previous approach is that it doesn't handle any type of class imbalance problem. To overcome this problem, there are many existing sampling techniques. But the problem with sampling techniques is that in oversampling minority samples are duplicated which leads to overfitting and in under sampling the majority samples are reduced to number of minority class samples which leads to great loss of information as minority samples number is very less.

Therefore, an alternative approach is to use the weighted cross entropy cost [17] when calculating the error. The weighted cross entropy uses a parameter called weights to adjust the loss in favour of minority class. In a broader view, the standard binary cross entropy cost is:

$$-\frac{1}{M} \sum_{i=1}^M [y_i * \log(\sigma(z_i)) + (1 - y_i) * \log(1 - \sigma(z_i))]$$

For example, suppose a training example has target 1, the output of the machine learning model is 0.6. From a Bayesian perspective, the model has 40% confidence in the wrong result or from a Frequentist perspective, the model will be wrong 40% of the time. The loss function penalizes this 40% by returning the value $-\log(0.6) = 0.22$. So this loss is incurred due to the model giving 40% support to the negative class, if the minority class is a positive class then we have to decrease this support and increase the loss. To increase this loss, the weighted cross entropy multiplies the term $y_i * \log(\sigma(z_i))$ with a positive value greater than 1. Conversely, if the minority class is a negative class, then this loss should be decreased then a positive value less than 1 is multiplied to the term. To handle the multi class classification, multiple classes need to have multiple weights so that every time an example of that class is processed the loss is increased or decreased according to the set of weights.

3.2.6 BERT Sequential output with Attention:

In the previous approach, the context feature vector for representing the sentence was extracted from the final cell state of the LSTM. As discussed, instead of focussing on too much data, the model can attend to only important parts of the data and can be used for making predictions. Therefore, in this approach a context vector is computed using the attention mechanism discussed by Badhanau [1].

$$\begin{aligned} \text{score}, S &= \tanh(W_1 \cdot O_{BERT} + W_2 \cdot C_{21}) && \text{Here, } O_{BERT} \text{ is the feature matrix} \\ \text{attention weights}, A &= \text{Softmax}(V \cdot S) && W_1, W_2, V \text{ are the trainable weights} \\ \text{context vector}, C &= A * O_{BERT} \end{aligned}$$

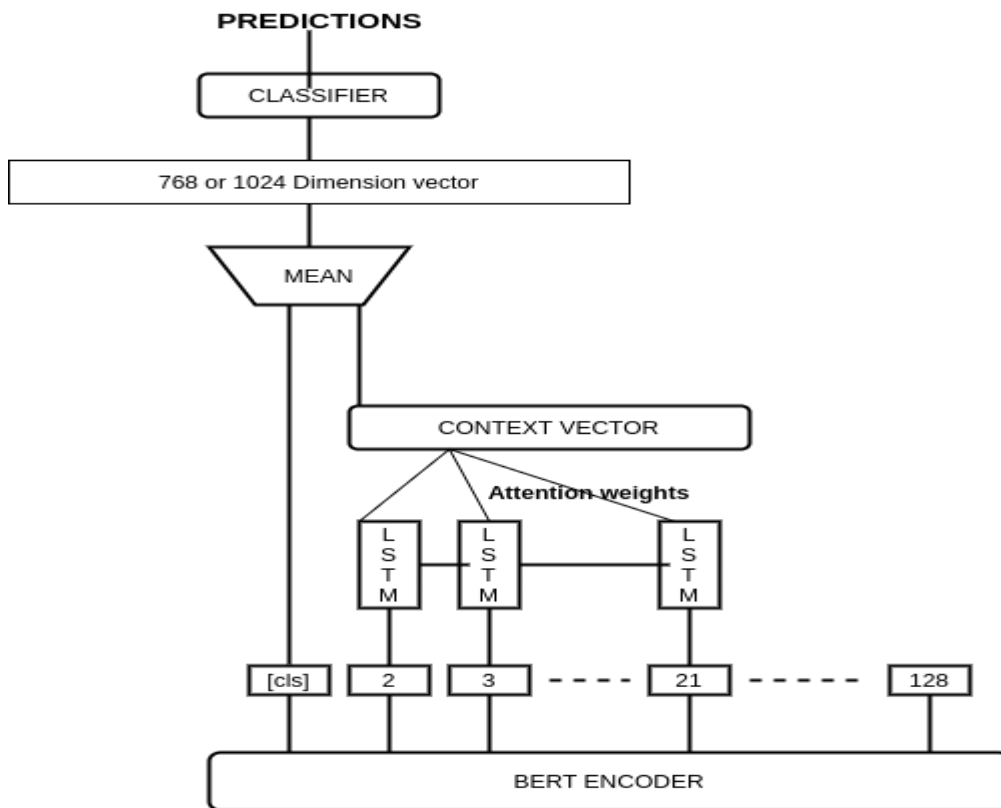


Fig.11 Attention Context based architecture

The Attention layer takes inputs as features (sequence output of BERT) and hidden cell state of the last LSTM and computes the attention weights and context vector. The attention weights are multiplied with the features to get the context vector. Then the mean of the context vector and the [CLS] token embedding is computed and given as input to the Softmax layer for getting

predictions. The skill of the model when used attention is slightly increased compared to previous LSTM approach.

3.3 Loss Functions

3.3.1 Weighted Cross Entropy for Binary Classification

X - Input to the Softmax or output layer $[x_1, x_2, x_3, \dots, x_n]$

W - Weights of the output layer

b - Bias for the output layer

Y - labels of the data $[y_1, y_2, y_3, \dots]$

α - Positive Weight

σ - Sigmoid Activation ($\sigma = \frac{e^{-z}}{1 + e^{-z}}$)

$$Z = W^T \cdot X + b \quad (\text{logits}) [z_1, z_2, \dots, z_n]$$

$$\text{Weighted Cross Entropy Cost (J)} = -\frac{1}{M} \sum_{i=1}^M [\alpha * y_i * \log(\sigma(z_i)) + (1 - y_i) * \log(1 - \sigma(z_i))]$$

Positive weight, [17] α allows one to trade off recall and precision by up or down, weighting the cost of a positive error relative to a negative error. A value $\alpha > 1$ decreases the false negative count, hence increases the recall. Conversely, $\alpha < 1$ decreases the false positive count and increases the precision. This means that positive weight, α is introduced as a multiplicative coefficient for the positive labels term in the loss expression.

3.3.2 Weighted Cross Entropy for Multi-Class Classification

Above loss function only works for binary classification. For multiclass classification softmax cross entropy should be used. Let α be set of weights for every class. $[\alpha_1, \alpha_2, \dots, \alpha_c]$

$$\frac{e^z}{\sum e^z} - \text{softmax function}$$

$$\text{Weighted Cross Entropy Cost(J)} = -\frac{1}{M} \sum_{j=1}^C \sum_{i=1}^M \alpha_i * y_i^j * \log\left(\frac{e^{z_i}}{\sum e^{z_i}}\right) [17]$$

3.4 Algorithm

Input:

D: The training samples $\{ (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \}$, where x is sentence and y is the label.

F: Bert_Feature_Extraction(D) (f_0, f_1, \dots, f_{127}).

Output:

W: Weights for the task-specific layer.

P: Predictions made by the classifier.

Bert_Feature_Extraction(D):

1. Preprocess the sentences in the dataset.
2. Load the BERT tokenizer.
3. Convert the sentences and labels to objects in format required by BERT.
4. Convert sentences into tokens using tokenizer which in turn use a vocab file.
5. Construct input masks for tokens. (1 for useful tokens , 0 for padding).
6. Load the pretrained weights from BERT.
7. Convert tokens and input masks to get the feature objects.
8. Input feature objects to BERT transformer and self-attention layers.
9. Returns a pooled output with representation of the entire sentence or sequence output with representation for each input sequence.

Fine_Tuning(F,D):

1. Get sequence output (F) from the BERT which contains 768 dimensions vectors for each token.
2. Input the sequence of contextual embeddings (f_1, f_2, \dots, f_{20}) to the sequence of LSTM and discard the output states but retain hidden states.
3. Compute the hidden state of the last LSTM which is of 768 dimension. (f_{lstm})
4. Compute mean of [CLS] token vector (f_0) and hidden state (f_{lstm}) or attention context (f_{att}) vector i.e., $\frac{f_0 + f_{lstm}}{2}$ or $\frac{f_0 + f_{att}}{2}$
5. Input the mean vector to the softmax layer and compute the logits
6. Compute the cross entropy cost or loss function (J).
7. Optimize the loss function (J) using AdamWeightDecayOptimiser.
8. Update the weights during optimization and evaluate on the test dataset.

CHAPTER 4

EXPERIMENTAL PROCEDURE

4.1 Dataset Description

The two datasets for this task are Offensive Language Identification Dataset (OLID) [9] and the other is Toxic Comment Classification dataset.

The toxic comment classification dataset has multiple (six) target features. The toxic comment classification dataset contains a text or tweet and contains six types of toxicity such as toxic, severe toxic, obscene, threat, insult and identity hate.

OLID contains a collection of annotated tweets using an annotation model that encompasses three sub tasks.

- **Task-A** Offensive Language Detection (whether the text is offensive (OFF) or not offensive (NOT)).
- **Task-B** Categorization of Offensive Language (whether the text is targeted insult (TIN) or Untargeted (UNT)).
- **Task-C** Offensive Language Target Identification (whether the text is targeting an Individual (IND), a Group (GRP) or others (OTH)).

4.2 Feature Extraction

In order to use the pretrained models like BERT and XLNet, the model parameters and weights are to be downloaded. For training BERT, the model has been made available as a tensorflow hub module which makes it easier to train with tensorflow which uses tensor programming and graph construction for every computation.

The whole package includes the model parameters, weights, saved model and also fixed size vocabulary file which is used by tensorflow to load and use it as a tokenizer.

4.3 Implementation Details

Software Specifications:

1. Programming language used : **Python3**
2. Framework used to build deep learning models : **Tensorflow version 1.15**
3. Visualisation tools : **Matplotlib, Tensorboard**
4. Other libraries and packages: **pandas, numpy, scikit-learn etc.**

Hardware and Storage Specifications:

1. Pretrained models take pretty much time to train because they have plenty of parameters. Therefore an AI accelerator application-specific integrated circuit (ASIC), Tensor Processing Units (TPU), developed by Google, specifically for neural network machine learning is used to train the models.
2. **Cloud TPU** is designed to run cutting-edge machine learning models with AI services on Google Cloud. Cloud TPU is available as a hardware accelerator in google colab.
3. TPUs are 110 times faster than CPU and 2-3 times faster than GPU. **TPU V2-8** is used in the project for training the model which is available for free to use in Colab.
4. GPUs are also widely used in training deep learning models but at a cost of time.
5. As model trained weights or checkpoints which are very large in size, are needed to be stored for every particular number of steps. So we used Google Cloud Storage buckets to store them which are usually tens of GB in size.

4.4 Training and Testing datasets

Dataset	Training examples	Testing examples
OLID Task A	13240	860
OLID Task B	4400	240
OLID Task C	3876	213
OLID 2020 TASK A	350000	350000
Toxic Comment Classification	143613	15957

4.5 Evaluation Metrics

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$F1 \text{ Score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{Accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{true positives} + \text{true negatives} + \text{false positives} + \text{false negatives}}$$

4.6 Evaluating the skill of the model:

ROC curves and AUC score can present an overly optimistic view of an algorithm's performance. Precision-recall curve plots can provide an accurate prediction of future classification performance due to the fact that they evaluate the fraction of true positives among positive predictions. The Precision-Recall Plot is more informative than the ROC Plot when evaluating binary classifiers on imbalanced datasets.

4.7 Model Directory

It contains various files about the model such as :

model.ckpt-1000.data-00000-of-00001:

It is a TensorBundle collection which saves the values of all variables that are used in that tensorflow session graph.

model.ckpt-1000.index:

It is a string-string immutable table. Each key is a name of a tensor and its value is a serialized BundleEntryProto. Each BundleEntryProto describes the metadata of a tensor.

model.ckpt-1000.meta:

Describes the saved graph structure.

events.out.tfevents:

It is a tensorflow log file which saves the following details:

1. storing a description of the tensorflow graph before training starts
2. writing a value of the loss function for every training step
3. storing a histogram of activations or weights for a layer once per epoch
4. storing average precision (or any other metric) for the whole validation set

This log file can be loaded with a tensorboard and can be used to visualise the histogram of weights and to trace the loss function in visual mode.

graph.pbtxt:

It contains the graph structure with all nodes and operations between them which can be visualized in tensorboard

eval_results.txt:

It contains the evaluation metrics on the test dataset which are defined in model function builder function.

eval folder:

It contains one more event log file created during evaluation which captures all the same type of information that was saved when training.

4.8 Miscellaneous experiments

As we are dealing with the transfer learning and fine tuning approach, the fine tuned model can be used to work on other domains as well. So we can train the model on one dataset and the model can be used to test samples from other abusive datasets. Therefore, a dataset which doesn't contain a few categories can be categorized into new classes using an already trained model which is a form of zero-shot learning.

A sample of 700000 rows from a huge dataset of 90 lakh examples is used. The train and test split is 50% i.e., 350000 for training and testing. The model trained on this dataset is used to test on other abusive datasets. Similarly, the model train on toxic comment classification is used to test on other samples and label them as one of the six categories in toxic comment classification.

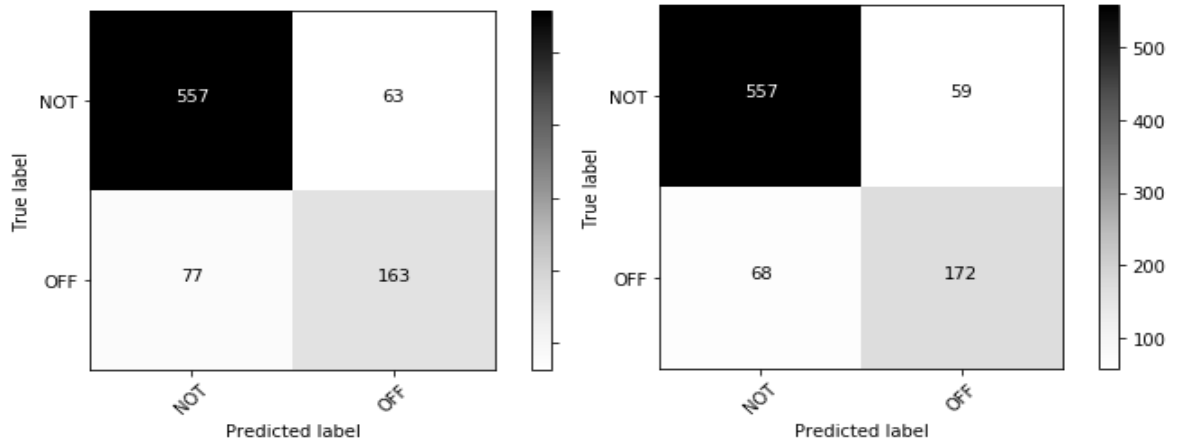
CHAPTER 5

RESULTS AND ANALYSIS

5.1 Neural Network as a task specific layer

Table 1. Results on OLID dataset-TASK A

Model	Precision	Recall	F1 Score	Accuracy
BERT (base)	89.8	87.9	88.8	84.1
BERT (large)	90.4	89.1	89.7	85.1
XLNet (base)	-	-	-	84.3
XLNet (large)	-	-	-	-

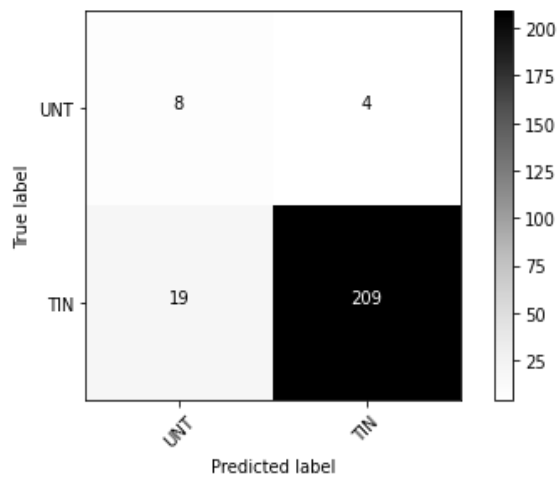


BERT BASE - TASK A

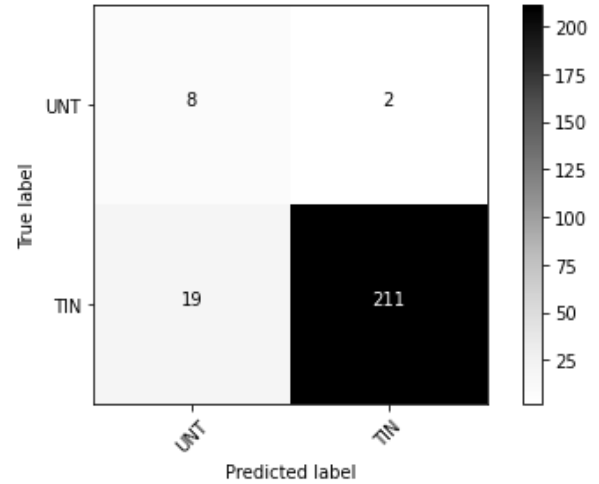
BERT LARGE - TASK A

Table 2. Results on OLID dataset-TASK B

Model	Precision	Recall	F1 Score	Accuracy
BERT (base)	67	30	41.4	90.4
BERT (large)	80	30	43.6	91.3
XLNet (base)	-	-	-	88.8
XLNet (large)	-	-	-	-



BERT BASE - TASK B



BERT LARGE - TASK B

Table 3. BERT BASE (OLID-TASK C)

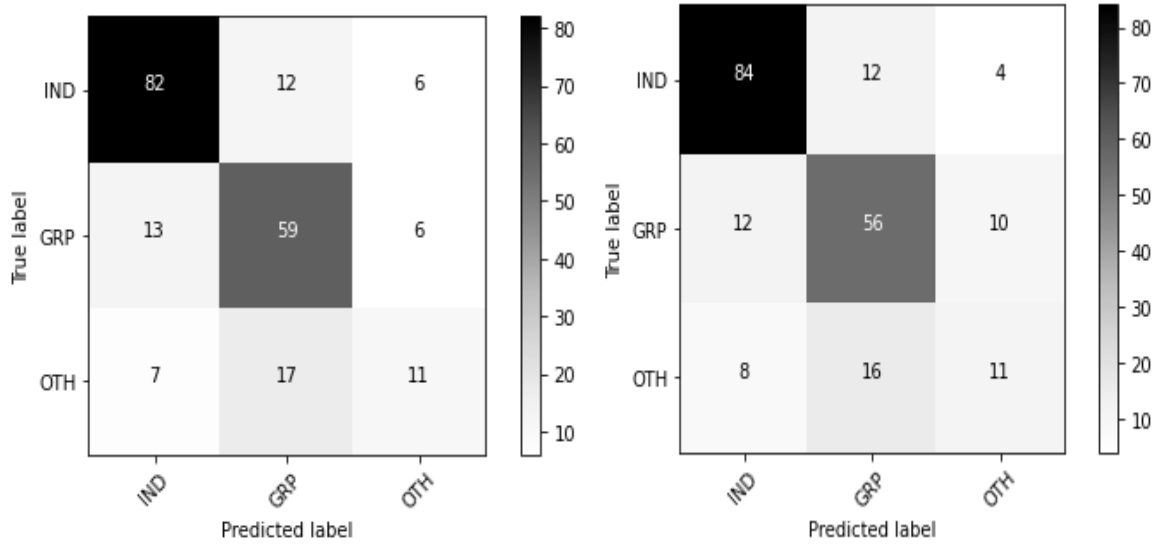
label	Precision	Recall	F1 Score
IND	80	82	81
GRP	67	76	71
OTH	48	31	38

Table 4. BERT LARGE (OLID-TASK C)

Label	Precision	Recall	F1 Score
IND	81	84	82
GRP	67	72	69
OTH	44	31	37

Table 5. BERT BASE Results on Toxic Comment Classification Dataset

Label	Accuracy
Toxic	98.6
Severe Toxic	99.1
Obscene	98.9
Threat	99.5
Insult	98.6
Identity Hate	99.2



BERT BASE - TASK C

BERT LARGE - TASK C

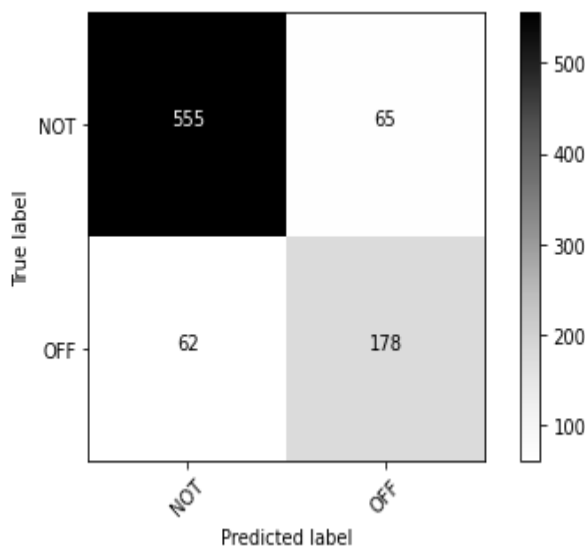
Analysis:

The embedding of [CLS] token of dimension 768 or 1024 is passed through a neural network hidden network to make predictions and the model achieved state-of-the-art results on the OLID dataset on task A. Both the labels of task A have achieved a decent F1 score on the test dataset. The focus is on OFF labels to predict whether the text is offensive or not and 168 of 240 were classified correctly by BASE model and BERT LARGE model achieved a minute improvement over BASE by classifying 172 of 240 correctly.

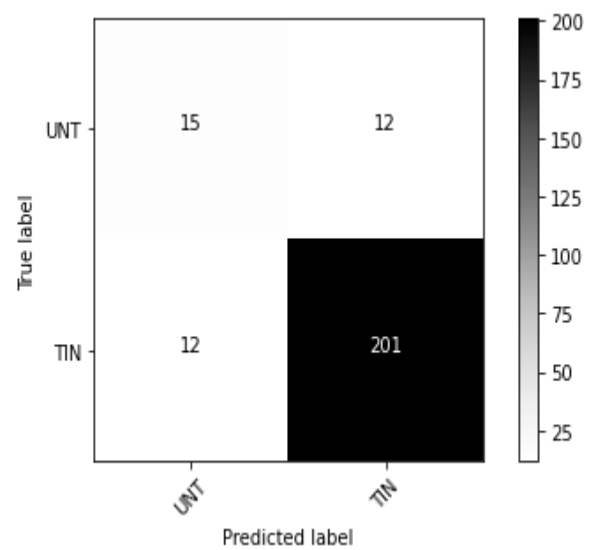
The results for task B are not satisfactory because of the class imbalance problem in the dataset and the imbalance ratio is almost 1:9. Therefore, the recall got affected a lot. But, the interest label is TIN, whether text is targeted or not, 209 and 211 out of 240 were classified correctly by the BASE and LARGE models respectively. So the model works well for the TIN class label which is a vital part of this task.

The results for task C are biased with the amount of training examples. Though F1 score is good for IND and GRP labels , F1 score gets affected for OTH labels as the number of training examples were very less compared to other labels. This approach was written into a research paper and was accepted at the **International Conference on Paradigms of Computing, Communication and Data Sciences (PCCDS-2020)**.

5.2 Using Sequential Output, LSTM and Weighted Loss Function



BERT BASE - TASK A



BERT BASE - TASK B

Table 6. BERT BASE Results on second approach -Task C

label	Precision	Recall	F1 Score
IND	78	85	81
GRP	70	71	70
OTH	52	37	43

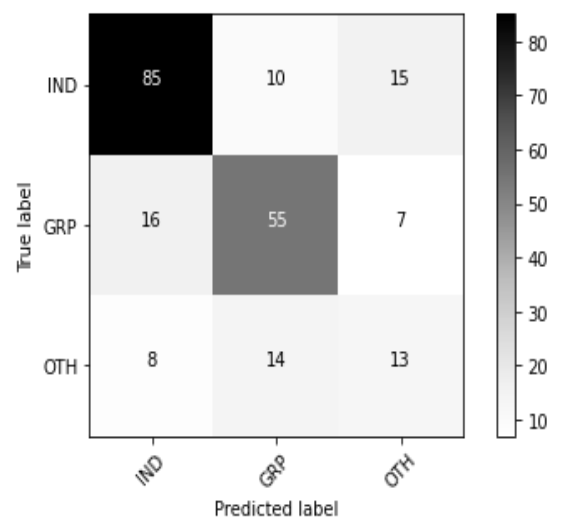


Table 7. BERT BASE Results on Toxic Comment Classification Dataset

Label	Accuracy
Toxic	98.4
Severe Toxic	99.2
Obscene	98.8
Threat	99.7
Insult	98.7
Identity Hate	99.4

Analysis and comparison

When used with weighted cross entropy cost and tokens other than [CLS] token as input to the task-specific layer, the model achieved greater performance compared to normal methods. The BERT base model on task, achieved accuracy of 85.23 which is greater than BERT base and large model in previous method and also achieved F1 score of 89.73 which is better than BERT base and large models in previous approach.

The problem with task B is high class imbalance, therefore weighted cross entropy is used but true positives although increased from 8 to 15, true negatives got reduced as less weight is given to the negative class compared to positive class. With trial and error methods, trying different weight ratios and epochs might increase the result.

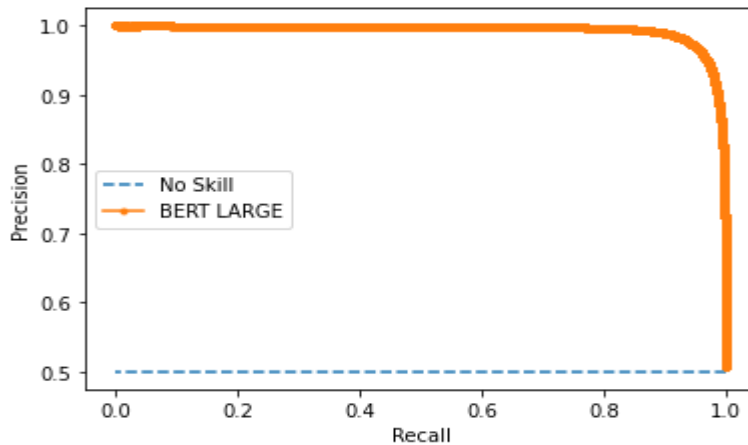
The total accuracy for task C is 1% more than the previous method which is 72 and the low ratio class , OTH label has increased its F1 score from 37 to 43. F1 score is also improved for the other two labels (GRP and OTH), which are less in number when compared to IND class. Therefore, these methods slightly improved the previous methods in almost three tasks.

The results on toxic comment classification were improved slightly with this architecture but here we did not use the weighted cross entropy loss function but we induced the contextual representation of first 20 tokens. The class label “threat” accuracy was increased by 0.2 percent and remaining labels accuracy was slightly increased by 0.1 percent.

So this approach brought better results compared to our previous publication and approach.

5.3 OLID 2020 Dataset

The model was trained on 350000 training examples and tested on 350000 test examples. The model achieved an accuracy score of 96% and AUC (area under the curve) of 99.3 which represent the model as highly skilled. The precision and recall are 97 and 95 respectively. The F1 score is 96. As this model is trained on large amounts of data, this can be used to predict samples from other smaller abusive datasets.



Precision-Recall Curve BERT Base on OLID-2020

5.4 Comparison of LSTM and Attention Context Vector approaches

Table.8 Comparison of LSTM and attention results

Metrics (MODEL - BERT BASE)	Our Approach-1 LSTM Method	Our Approach-2 Attention Method
F1 Score on OLID TASK A	89.73	89.9
AUC Score on OLID TASK A	93.8	94.1
F1 Score on OLID TASK B	48.8	50
AUC Score on OLID TASK B	50.9	50.1
F1 Score on OLID TASK C	64	62

The AUC value of LSTM stack based method is 93.8 and for attention mechanism it is 94.0 when trained on OLID TASK-A, which means the future performance of attention based method is highly acceptable compared to LSTM method. The two methods performed almost equally on the TASK B.

CHAPTER 6

APPLICATIONS

One major issue these days is harassment of women over electronic communication and posting abusive comments in well known people's social media accounts. Therefore, before causing such offensive comments, an automatic tool can be developed to find abuse, and to warn the maltreater or harasser. Rather than manually deleting comments or turning off comments on social accounts, an extra feature can be developed, which automatically removes the toxic comments from their accounts.

Abusive language detection can also be applied in many fields, like, online social media that contains posts, tweets, and also, on the content that is harmful for kids. So, supported age the social media platforms will hide such variety of content from an explicit group of individuals like kids, children etc. YouTube, Netflix, etc. have billions of videos and subtitles when streaming them, this technique can be accustomed to find abusive content and intimate the users that the video contains abusive or sensitive speech. This can also be used to test the movie subtitle files before adding to the video to check if there is any abusive content in the subtitle text, during every interval of time and further, the specific text can be hidden or masked when displayed in the video [13].

CHAPTER 7

CONCLUSIONS AND FUTURE SCOPE

The project report mentioned various neural attentive language models and the technical details of how the features can be extracted using language modeling. This report conjointly mentioned two of the foremost and widely used state-of-the-art models in the NLP field and their application towards abusive detection tasks. The state-of-the-art models, BERT and XLNet are fine tuned to extract the prime features from text and use them for abuse detection tasks.

Based on the results, it ascertained rather than just using [CLS] token embedding alone as sentence illustration, the other token's contextual embeddings along with [CLS] token as input achieved better results. To extract the context vector LSTMs are used in this approach.

The extension to our approach, one more attention based context vector is used to make predictions which also performed better than the standard Softmax layer approach and LSTM based approach. It is also observed that using weighted cross entropy cost and giving weightage to the low class label weights handled the class imbalance problem to some extent. Although with trial and error methods, setting up different hyper learning parameters and training the model would have discovered high quality patterns and eventually brought better results.

The work can be extended to deal with speech data and the model can be customized to be able to work with the multi-lingual data. The model can be modified to train huge documents or websites which contain a lot of text and dependencies and hence can be used to identify whether any website, blog or social media content contains abuse language.

We are also planning to write another research paper as an extension to our previous publication, by including the methods we proposed and the future scope of work in this task.

REFERENCES

1. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv prints, abs/1409.0473, September 2014. <https://arxiv.org/abs/1409.0473>.
2. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In Advances in NIPS 2017. <https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>.
3. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016. <https://arxiv.org/abs/1607.06450>.
4. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. <https://arxiv.org/abs/1810.04805>
5. Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. <https://arxiv.org/abs/1906.08237>
6. A.Radford, J.Wu, R.Child, D.Luan, D.Amodei, I.Sutskever. Language Models are Unsupervised Multitask Learner.
7. Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, Llion Jones. Character-Level Language Modeling with Deeper Self-Attention. <https://arxiv.org/abs/1808.04444>
8. Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed length context . <https://arxiv.org/abs/1901.02860>
9. Marcos Zampieri, Shervin Malasi, Preslav Nakov, Sara Rosenthal Noura Farra ,Ritesh Kumar. Predicting the Type and Target of Offensive Posts in Social Media. arXiv preprint arXiv:1902.09666, 2019. <https://arxiv.org/abs/1901.09666>.

10. Papegnies, Etienne Labatut, Vincent Dufour, Richard Linarès, Georges. (2017). Impact Of Content Features For Automatic Online Abuse Detection https://doi.org/10.1007/978-3-319-77116-8_30
11. Bourgonje, Peter and Moreno Schneider, Julian and Srivastava, Ankit and Rehm, Georg. Automatic Classification of Abusive Language and Personal Attacks in Various Forms of Online Communication. https://doi.org/10.1007/978-3-319-73706-5_15
12. Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. <https://arxiv.org/abs/1211.5063>
13. Victor R. Martinez, Krishna Somandepalli, Karan Singla, Anil Ramakrishna, Yalda T. Uhls, Shrikanth Narayanan. Violence Rating Prediction from Movie Scripts. <https://doi.org/10.1609/aaai.v33i01.3301671>
14. Liu Ying and Loh, Han Tong and Kamal, Youcef-Toumi and Tor Shu Beng. Handling of Imbalanced Data in Text Classification: Category-Based Term Weights.
15. SEBASTIAN RUDER (2019, August 18). The State of Transfer Learning in NLP. <https://ruder.io/state-of-transfer-learning-in-nlp/>
16. Chris McCormick. (2019, July 22). BERT Fine-Tuning Tutorial with PyTorch <https://mccormickml.com/2019/07/22/BERT-fine-tuning/>
17. Ho, Yaoshiang, and Samuel Wookey. “The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling.” IEEE Access 8 (2020): 4806–4813. Crossref. Web.
18. Jay Alammar. (2018, June 17). The Illustrated Transformer. <http://jalammar.github.io/illustrated-transformer/>
19. <https://towardsdatascience.com/transformer-xl-explained-combining-transformers-and-rnns-into-a-state-of-the-art-language-model-c0cfe9e5a924>.

APPENDIX

PCCDS-2020 Publication:

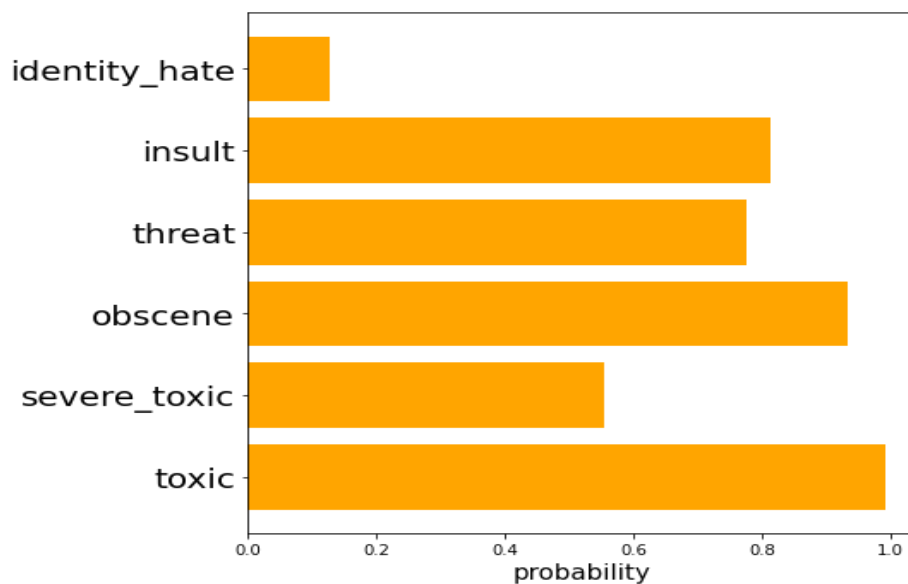
The results that are mentioned in section 5.1 are obtained during the period of writing our research paper for the **International Conference on Paradigms of Computing, Communication and Data Sciences (PCCDS-2020)**. Some conclusions and concepts are drawn from our research work and are emphasized in this report. So any content or experimental results that are found online similar to content in this report is because of our publication of a research paper. The title of the research paper we have written is “**Abusive or Offensive Language Detection using fine-tuned BERT and XLNet**”.

Toxic Comment Classification example:

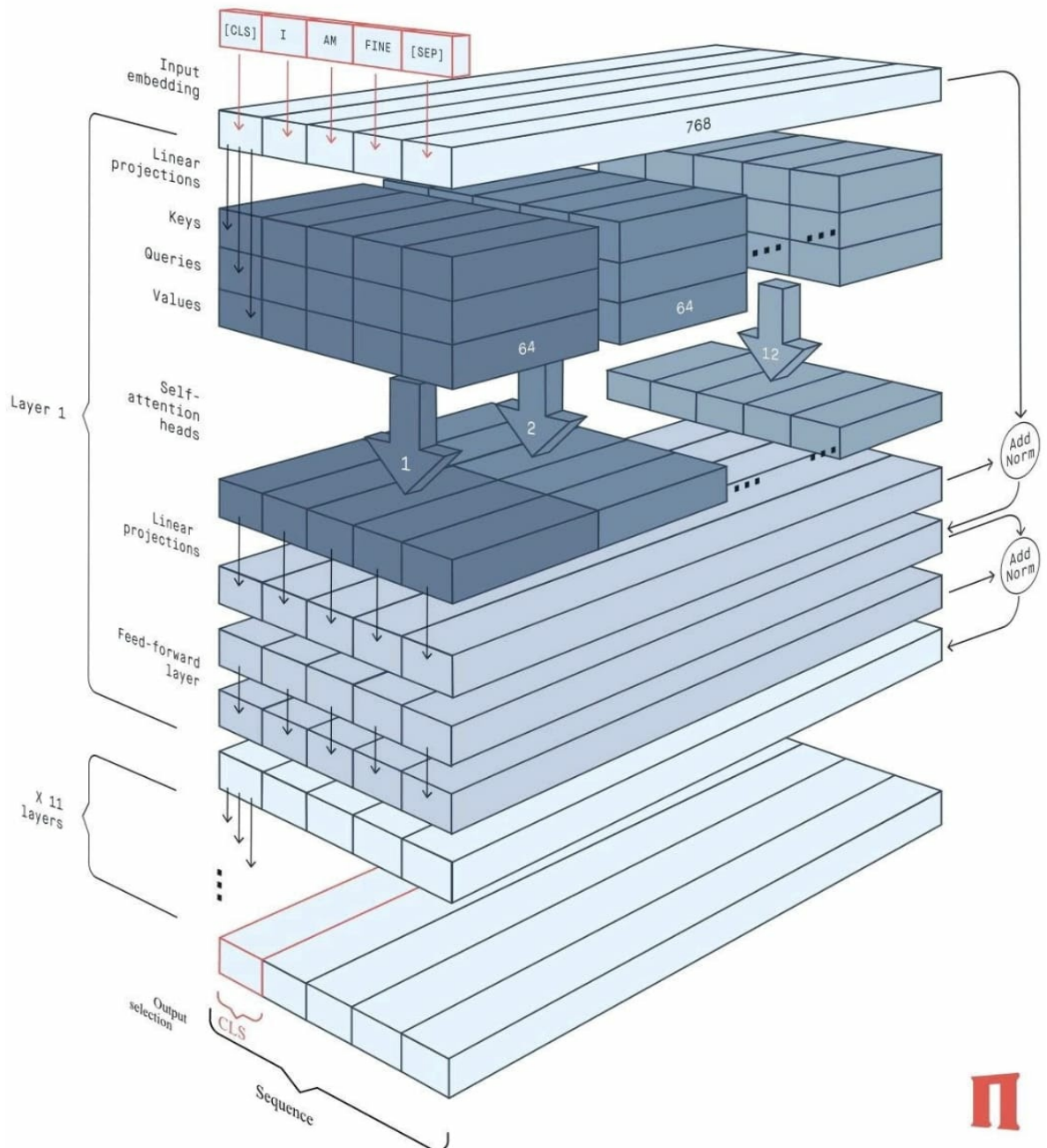
Abusive text:

*BLOCKING SCHNITZELMANGREEK FOR NO REASON!GET A LIFE YOU LOSER OR I WILL KILL YOU, YOU WIFE, AND CHILDREN!!!!BURN IN HELL!!I LIGHT YOUR CORPSE LIKE FIREWOOD! EWARE-YOU AND YOUR FAMILY. I DARE YOU TO TALK BACK TO ME AND SEE WHAT HAPPENS SO WHAT IS IT GOING TO BE F**kin Nig*a*

Model probability predictions of above text for each Class:



BERT 3D Architecture:



BERT Pretrained Weights:

https://storage.googleapis.com/bert_models/2018_10_18/cased_L-12_H-768_A-12.zip

https://storage.googleapis.com/bert_models/2018_10_18/cased_L-24_H-1024_A-16.zip

XLNet pretrained Weights:

https://storage.googleapis.com/xlnet/released_models/cased_L-12_H-768_A-12.zip

https://storage.googleapis.com/xlnet/released_models/cased_L-24_H-1024_A-16.zip

Toxic Comment Classification:

<https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge>

Offensive Language Identification Dataset:

<https://competitions.codalab.org/competitions/20011>

BERT Repository:

<https://github.com/chandu7077/mybert>

XLNet Repository:

<https://github.com/chandu7077/transformers>

Google Colab:

- In order to access cloud TPU, the code is to be run on the google colaboratory notebooks with access to Google cloud buckets to store checkpoints.
- Data is to be loaded either from Cloud buckets or can be manually uploaded every time into temporary storage of colab.
- GPU are also available in Colab and we used GPUs to train the toxic comment classification task.

Tensorboard:

