Comment Toxicity Classification Using BERT and GNN

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Candidate Name

Pavan Teja Reddy Kallam (AP20110010253)

Susmitha Dudipalli (AP20110010418)

Romith Bondada (AP20110010248)

Sukanya Karasala (AP20110010264)

Under the Guidance of

Prof. Ashu Abdul

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

December,2023

Certificate

Date: 28-Nov-23

This is to certify that the work present in this Project entitled **" Comment Toxicity Classification Using BERT and GNN"** has been carried out **Pavan Teja Reddy Kallam, Susmitha Dudipalli, Romith Bondada, Sukanya Karasala** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. Ashu Abdul

Department of CSE,

SRM AP, Amaravati.

Co-supervisor

(Signature)

Prof. Jatindra Kumar Dash

Head of the Department,

Department of CSE

SRM AP, Amaravati.

Acknowledgements

It is our privilege to express our sincere regards to our project mentor, Prof. Ashu Abdul, and our department HOD. Prof. Jatindra Kumar Dash for their valuable inputs, guidance, encouragement, whole-hearted cooperation throughout the duration of our project. We would like to express our special thanks to my mentor prof.Ashu Abdul again for his time and efforts he provided throughout the semester. I would also thank our friends for their cooperation and support.

Tables of Content:

Certificate

Acknowledgements

Table of Contents

Abstract

Abbreviations

List of Tables

List of Figures

1. Introduction
   1.1. Related Work
2. Methodology
   2.1. BERT
      2.1.1. Data Pre-processing
      2.1.2. Tokenization
      2.1.3. Padding and Truncation
      2.1.4. Word2Vec
      2.1.5. BERT
      2.1.6. Self-attention layer
      2.1.7. Feed Forward Neural Network
      2.1.8. Semantic Extraction
      2.1.9. Algorithm-1
   2.2. GNN
      2.2.1. Graph Representation
      2.2.2. Graph Convolutional Layers
      2.2.3. Loss Function
      2.2.4. Training
3. Discussion
      3.1.1. Precision
      3.1.2. Recall
      3.1.3. F1 Score
      3.1.4. Accuracy
      3.1.5. T test

Abstract

The major objective of this study is to develop a language model to recognize and categorize the toxic comments that might be found on social media. A challenge with classifying harmful remarks is that a single comment can belong to more than one category or label. Therefore, traditional approaches like Naive Bayes, SVM, or others cannot handle this type of categorization effectively because a model needs to comprehend language context in order to analyse poisonous comments. Therefore, the study included the use of BERT and GNN. BERT (Bidirectional Encoder Representation from Transformers), which can learn language in both directions, and GNN (Graph Neural Network) Model, which is a clever technique to train a graph model. The BERT is used to extract semantics and these semantics are fed to GNN. When the two models are trained together they give the best results for comment classification.

Abbreviations

BERT	Bidirectional Encoder Representations from Transformers
NLP	Natural Language Processing
GNN	Graph Neural Networks
OOV	Out of Vocabulary
CLS	Classification
SEP	Separator

List of Tables

Table 1. Metrics

List of Figures

Figure 2.1 Model Architecture…….……...……...……

Figure 2.2 Bert-Base-Uncased……

Figure 2.3 Text Preprocessing……

Figure 2.4 Bert Architecture……

Figure 2.5 GNN Architecture……..

1.Introduction:

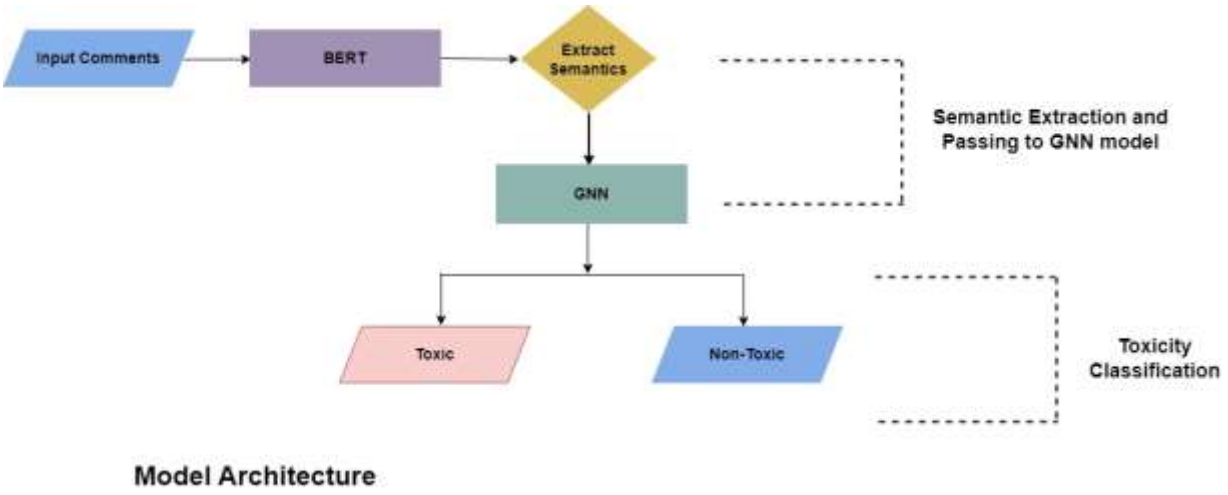
Toxicity in online communication has become a prevalent issue in recent years, with the increasing use of social media platforms and online forums. Toxic comments not only have the potential to harm individuals and groups, but also create a negative environment for discussion and debate. As a result, the automatic detection and classification of toxic comments has become an important area of research. It has a negative impact on user's mental health also and the participation of people from different backgrounds when hateful and insulting language is used on social media sites. Historically, categorical datasets with large amounts of data have been used extensively in automatic foul language identification. The severity of offensiveness varies, though. The proposed GNN and BERT architecture gives the accurate result. The toxicity score is calculated in the range of 0 to 1.

1.1 Related works

[1] R. Rivaldo, A. Amalia and D. Gunawan have introduced two pre-trained BERT models for harmful comment recognition and multi-label categorization were put forth in this paper. The reason of using BERT is that it is a cutting-edge model that learns the meaning of sentences in both directions, leading to a better grasp of language. ROC model evaluates the model performance in classification.[2] Danilo Croce, Giuseppe Castellucci, and Roberto Basili have expanded the capabilities of architectures based on transformations under challenging training scenarios. Then GAN-BERT was proposed, extending the fine tuning of BERT-like architecture with unlabeled data in a generative adversarial environment. In a number of tasks involving sentence classification, the model is performing well. [3] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf introduced DistilBERT, a pre-trained general-purpose version of BERT that is 40% smaller and 60% faster while retaining 97% of the language processing abilities, was introduced. With an ablation research, they demonstrated how a general-purpose language model can be trained successfully via distillation. [4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova have came up with with just one additional output layer, the pre-trained BERT model may be fine-tuned to produce cutting-edge models for a variety of tasks. On eleven natural language processing tasks, it achieves brand-new state-of-the-art results, boosting the GLUE score to 80.5%, MultiNLI accuracy to 86.7%, SQuAD v1.1 question answering Test F1 to 93.2, and SQuAD v2.0 Test F1 to 83.1%. [5] Welbl J, Glaese A, Uesato J, Dathathri S, Mellor J, Hendricks LA,Anderson K, Kohli P, Coppin B, Huang P-S discussed the difficulties of LM toxicity evaluation and the adverse consequences of automated toxicity mitigation have been investigated and described by the author utilising a variety of straightforward toxicity reduction techniques. At the end, detoxification raises LM loss and adds and amplifies social biases in subject and dialect coverage, possibly resulting in lower LM performance for marginalised groups. [6] Faal, F., Schmitt, K. & Yu, J.Y ,Proposed Reinforce-Detoxify, a method for mitigating toxicity in language models based on the proximal policy optimization from reinforcement learning that utilizes a reward model designed to mitigate unintended bias towards social identities in toxicity prediction. [7] R. Shounak, S. Roy, V. Kumar and V. Tiwari discussed about each comment receives a toxicity score to indicate its level of toxicity. They examined three natural language processing frameworks—Tfidf + Ridge Regression, Tfidf + Catboost, and Bert + Dense Layer—to predict the toxicity score. Following the experimentation, it was clear that the bert +dense layer had the best performance and provided the lowest mse of the three. [8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio have introduced the new framework, they simultaneously train two models—a generative model G that represents the data distribution and a discriminative model D that calculates the likelihood that a sample

originated from the training data rather than G—in an adversarial process. The goal of the training process for G is to increase the likelihood that D will make a mistake. [9] Lucas Dixon, John Li, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman, in this paper they defined unintended bias for text classification and made a distinction between it and fairness in the use of ML. They have discussed ways to measure and reduce unintentional bias in datasets and the associated models. [10] Zhong, J., Liu, X., & Hsieh, C specifics of subtly hazardous online exchanges were examined. They used a public dataset with both healthy and unhealthy comments that were classified as having one of seven minor toxic traits: aggressive, antagonise, dismissive, generalise, generalise unfairly, or sarcastic. With a micro F1-score, macro F1-score, and AUC-ROC of 88.76%, 67.98%, and 0.71, respectively and machine learning models were able to identify between these remarks using a CNN-LSTM network with pre-trained word embeddings. Furthermore, they implemented a CNN-LSTM with pre-trained word embeddings. They used Tensorflow to implement CNN-LSTM deep model. [11] Minh Quan Do has concentrated on key ideas like machine learning, deep learning, and applications to challenges in the real world, especially natural language processing. A Deep Neural model that extracts the semantics of phrases and generates a score for inputs was also successfully implemented. [12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin they have introduced the Transformer, the first sequence transduction model based purely on attention, substituting multi-headed self-attention for the usual recurrent layers in encoder-decoder designs. [13] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell have examined the fairness measures in the context of adversarial debiasing. They consider supervised deep learning tasks in which the task is to predict an output variable Y given an input variable X, while remaining unbiased with respect to some variable Z, protected variable. [14] Martin Arjovsky, Soumith Chintala, and L´eon Bottou developed an algorithm that we called WGAN as a replacement for conventional GAN training. We demonstrated with this new model that it is possible to increase learning stability, eliminate issues like mode collapse, and produce informative learning curves beneficial to debug and hyperparameter searches. [15] Xu Chen, Jiang Wang, and Hao Ge, for generative adversarial learning, a primal-dual formulation was proposed. With a convergence guarantee, this formulation analyses GANs from the standpoint of convex optimisation and provides the best update to the discriminator and the produced distribution. The relevant training techniques can all be enhanced by directing the generated distribution in the ideal direction by framing various GAN versions within the convex optimisation framework. [16] Martin Arjovsky and Léon Bottou have discussed about advance theoretical understanding of the generative adversarial networks' training processes. They conduct focused experiments to confirm their presumptions, demonstrate the assertions, and quantify the occurrences in order to support theoretical analysis.

2. Methodology:



Model Architecture

Advancing to the proposed methodology, you can see the model architecture in fig(2.1). The model architecture reveals that we are working with generative AI models, specifically BERT (Bidirectional Encoder Representations from Transformers) and GAN (Generative Adversarial Networks). These models are referred to as generative models because they can anticipate and generate new words based on words they have previously observed. Now, by adding a small amount of labelled data to the equation, you can tune them to carry out common NLP tasks like classification or named entity recognition. This process is known as tuning, and it is essentially what we are doing when we use BERT to detect toxicity levels in the input comments. If they exceed the toxicity threshold (>0.5), they are then forwarded to GAN model for detoxification.

Figure 2.1: Model Architecture

2.1Data pre-processing:

Comments from the Jigsaw Unintentional Bias in Toxicology dataset are taken as input where $C=\{c_1, c_2, ..., c_k\}$ → Equation (1)

, where c_i , i belongs to n is a comment, and k be number of comments in the dataset. The input $c_i=\{s_1, s_2, ..., s_l\}$, where s_i is a sentence, is one comment, and each comment has the potential to include a number of sentences, say l. be $s_i=\{w_1, w_2, ..., w_m\}$ → Equation (2)

; where w_i is a word and m be the number of words in the sentence.

These input tokens go through the following steps before being fed to the BERT model

2.2 Tokenization:

To handle words that are not in the vocabulary (OOV), words are further tokenized into sub-words in trail of finding meaning. Let be the collection of words:

$$W = \{w_1, w_2, ..., w_m\} \rightarrow \text{Equation (3)}$$

$$\text{Tokenized as } W' = \{w'_1, w'_2, ..., w'_m\} \rightarrow \text{Equation (4)}$$

2.3 Padding and Truncation:

After tokenization, special tokens[*CLS*]&[*SEP*]are appended at the start and end of the word sequence,

$$s_i = \{w'_1, w'_2, ..., w'_m\} \rightarrow \text{Equation (5)}$$

$$s'_i = [CLS] + W' + [SEP] \rightarrow \text{Equation (3.6)}$$

, where *CLS* is acronym for classification and *SEP* for separation. *CLS* token is used to obtain a fixed-sized output vector from BERT, which can then be used for classification tasks, and *SEP* (separator) token, on the other hand, is used to separate two different sentences in a sequence, especially when multiple sentences are concatenated together. These basically indicate the start and end of a sentence. Fig(2.3)

2.4 Word2Vec:

These truncated sentences are converted to vectors using the embedding technique word2vec. $W_i = W_E = \{e(w_m) + e(w_m) + \ldots + e(w_n) + e(w_n)\} \rightarrow$ Equation (6)

where W_E represents sequence of embeddings and $E(w_i)$ is a word converted to vector.

Generating respective positional Embedding for the embedded text $P_E = \{P(e_m) + P(e_m) + \ldots + P(e_n) + P(e_n)\}$

$P_E = \{P(e_1) + P(e_2) + \ldots + P(e_m)\} \rightarrow$ Equation (7)

, where P_E is positional embedding of the embedded tokens, and $P(e_i)$ be positional vector of each word in sequence

To calculate positional embedding sinusoidal function is applied

$$POS_{E(pos,2i)} = \sin(pos/1000^{2i/d_{Model}}) \rightarrow \text{Equation (8)}$$

$$POS_{E(pos,2i+1)} = \cos(pos/10000^{2i/d_{Model}}) \rightarrow \text{Equation (9)}$$

The final embedded vector (E_v) is

Word Embeddings + Positional Embeddings + Attention mask

$$W_E + P_E = \{W_{e_1} + W_{e_2} + ... + W_{e_k} + P_{e_1} + P_{e_2} + ... + P_{e_k}\} \rightarrow \text{Equation (10)}$$

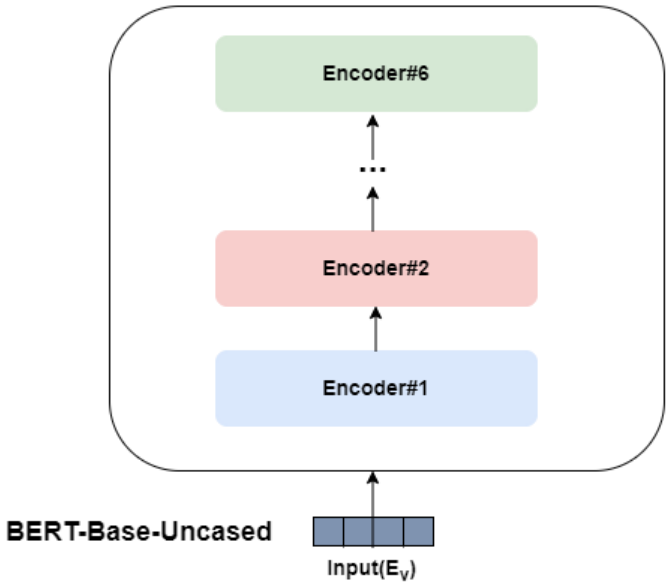
Adding Attention mask to differentiate between input & padding tokens

$$E_v = A_m + W_E + P_E = \{1,1,1,1,\ldots,0,0,0,0\} \rightarrow \text{Equation (11)}$$

, where '1' represents input tokens and '0' represents padding tokens, where E_v is embedding vector. These embedded vectors are now passed to the BERT model. Fig (2.2).

2.5 Bidirectional Encoder Representations from Transformers (BERT):

In order to calculate the toxicity scores of each comment, the encoder stack of BERT is in responsible for processing the input sequence of comments and extracting their



contextualized representations by employing self-attention and feedforward neural network layers fig(2.3). In this experiment we incorporated BERT base uncased with 6 encoder layers.

Figure 2.2: BERT-Base-Uncased

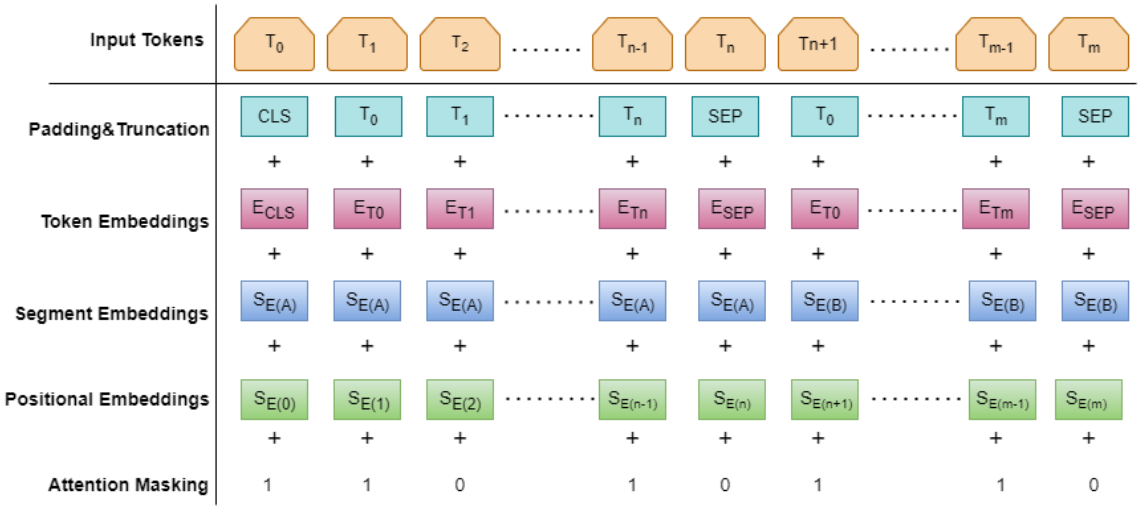


Figure 2.3: Text Pre-processing

2.6 Self-attention Layer:

When the embedded input sequences are provided, the self-attention layer's objective is to identify each word's meaning in connection to other words in the comment. In this experiment, the BERT model was trained using on a large corpus of textual data, from which it learnt the weights of each word's embedding as well as the weighted matrices for the self-attention layer.

The input sequence is initially split into three vectors, one for each input token: the *key*, the *value*, and the *query*. Linear matrices of the initial input embedding make up the query, key, and value vectors. These vectors are created by multiplying the embedding by these three matrices [12].

$$QueryVector(Q) = E_v \times M_q = Q\{q_1, q_2, \ldots, q_n\} \rightarrow \text{Equation (12)}$$

(The token for which the attention scores are being calculated)

$$KeyVector(K) = E_v \times M_k = K\{k_1, k_2, \ldots, k_n\} \rightarrow \text{Equation (13)}$$

(Contains encoded information)

$$ValueVector(V) = E_v \times M_v = V\{v_1, v_2, \ldots, v_n\} \rightarrow \text{Equation (14)}$$

(Represents the actual information contained in the tokens)

A $Score(S)$ which is the dot product of $q_i \times k_i^n$ calculated for each word in the embedded sequence with respective to key vector of every other word. This $Score(S)$ for each query and key pair is divided by the hyperparameter root over the value vector dimension $\sqrt{\dim}$ a *softmax* is applied to the result.

$$Attentioncore(A_v) = Softmax\ ((Q \times K^T) \div \sqrt{\dim}) \times ValueVector(V) \rightarrow \text{Equation (15)}$$

For each class or output label, *softmax* determines their decimal probabilities.

$$Softmax(L_i) = e^{Li} \div \sum_{i=0}^{6} (e^{Li}) \rightarrow \text{Equation (16)}$$

2.7 Feed Forward Neural Network:

This score is then passed to feed forward neural network for further processing Fig(2.4). The network's output is then passed through a dense layer with sigmoid activation to determine if the comment is likely to be toxic or not. The output nodes, or the classes of different toxicities are present in this final dense layer.

$$DenseLayer\_output = activation\ ((w_1 \times hidden\_state) + b_1)\ Final\ output(F) = w_2 \times Intermediate\_output + b_2 \rightarrow \text{Equation (17)}$$

$W_1(in,\ hid)$ is the weighted matrix with shape (size of the input hidden states, size of the hidden layer in the feedforward network).

$b1$ is bias vector of shape (size of the hidden layer in the feedforward network)

$$H = [h_1,\ h_2, ..., h_n] \rightarrow \text{Equation (18)},$$

where h_i represents the sequence's i^{th} hidden state.

$$\text{Activation} = ReLU \text{ or } Sigmoid \rightarrow \text{Equation (19)}$$

$ReLU$ or *Sigmoid*

ReLU (Rectified Linear unit) function:

$$R(x) = max(0,\ x) \rightarrow \text{Equation (20)}$$

Sigmoid function:

$$S(x) = 1/(1 + e^{(-x)}) \rightarrow \text{Equation (21)}$$

$W_2(hid,\ out)$ is the weighted matrix with shape (size of the hidden layer in the feedforward network, size of output layer).

b_2 is bias vector of shape (size of output layer)

In this experiment, we used *BertForSequentialClassification*, which employees a single layer linear transformation i..e, maps the encoder output to output logits by performing a single matrix multiplication with a bias term.

$$OutputLogits(OL) = (h_f XW_m) + B \rightarrow \text{Equation (22)}$$

where, h_f *is the final hidden layer,* W_m *is the weighted matrix,* B *is the Bias term*

After passing these through the *Softmax* activation function, final probabilities are provided for each of the six labels used for this experiment. values between 0 and 1. Non-toxic values are near to 0, and very toxic values are close to 1.

$$ToxicityScores(TS) = Softmax(OL) \rightarrow \text{Equation (23)}$$

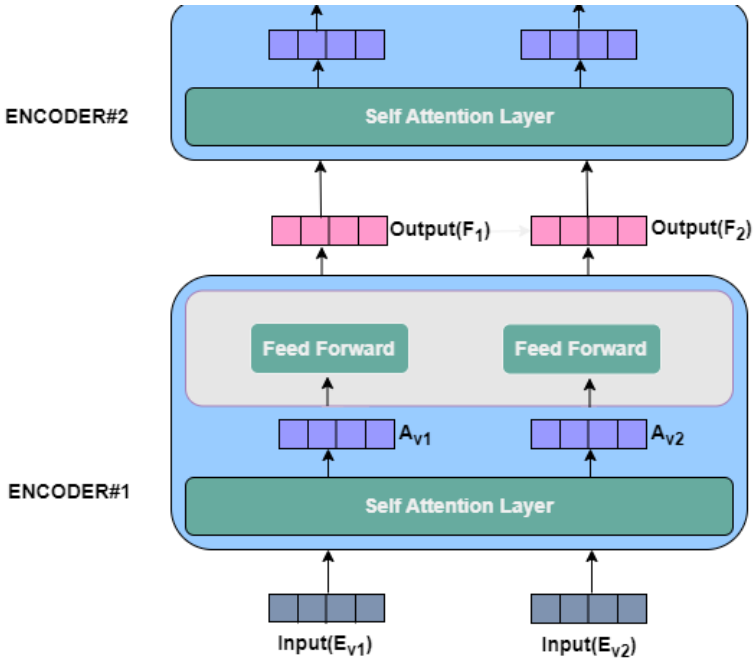


Figure 2.4: BERT Architecture

2.8 Semantics Extraction:

Algorithm-1

---

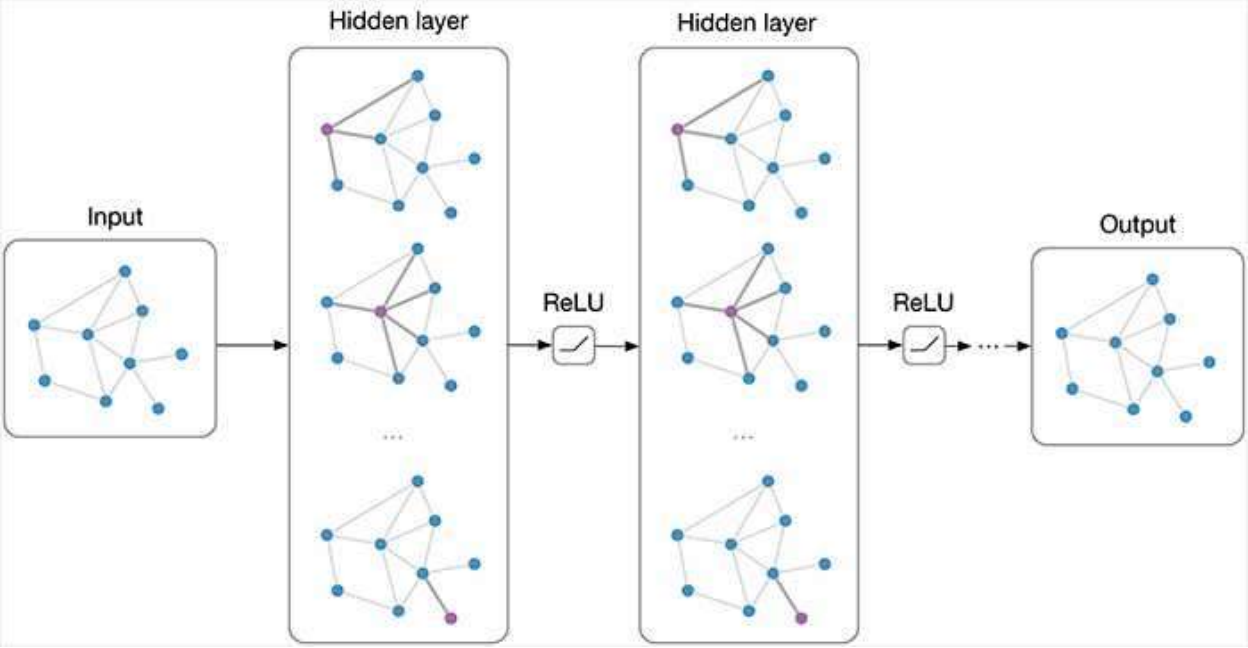
model = *BERT* (*bert* – *base* – *uncased*)

```
model, tokenizer = load_model_and_tokenizer("bert-base-uncased")
encoded_text = tokenizer(
text_to_process,
add_special_tokens=True,
max_length=128,
padding='max_length',
truncation=True,
return_tensors='pt',
)
device = 'cuda' if torch.cuda.is_available() else 'cpu'
inputs = {key: val.to(device) for key, val in encoded_text.items()}
with torch.no_grad():
outputs = model(**inputs)
logits = outputs.logits
embeddings = logits[:, 1]
```

---

3. Graph Neural Network (GNN):

With links represented as edges and entities as nodes, graph-structured data can be handled effectively with the help of Graph Neural Networks (GNNs). They are excellent at identifying complex relationships and patterns in relational datasets, which makes them useful for a variety of applications, including molecular chemistry, social network analysis, and recommendation systems. Fundamentally, GNNs use iterative information aggregation to update node embeddings by taking into account information from nearby nodes as well as local features. Convolutional operations on graphs are used by Graph Convolutional Networks (GCNs), a well-liked GNN architecture, to efficiently spread information. Managing graphs of different length and preventing over-smoothing are two problems with GNNs. Still, research is being done to improve GNNs and create variations such as Graph Attention Networks (GAT) to increase their capabilities, making them more and more



3.1 Graph Representation:

The nodes of the graph represent individual comments, and each node is associated with its semantic embeddings ($E_{Semantics}$) and bias indicators (F_{Bias}). Edges between nodes are defined based on semantic similarity, computed through arithmetic operations on semantic embeddings and the Nodes are enriched with semantic features, forming a comprehensive feature vector ($V_{Semantic}$) that combines semantic embeddings and bias indicators: $V_{Semantic} = [E_{Semantic}, F_{Bias}]$

3.2 Graph Convolutional Layers:

In each graph convolutional layer, nodes aggregate information through arithmetic operations. For node v, the updated representation $V_v^{(l+1)}$ is computed as a weighted sum of its neighbour's representations, considering both semantic and bias information:

$$V_v^{(l+1)} = \sigma\Big(\sum_{u \epsilon N(v)} \frac{1}{c_v}(W^{(l)} * V_u^{(l)} + B^{(l)} * V_u^{(l)})\Big)$$

Now the pooling operations, such as mean or max pooling, are applied hierarchically to down-sample the graph. Let $H^{(l)}$ be the node embeddings after the l_th layer of graph convolution. The hierarchical pooling operation involves down sampling the graph through mean or max pooling, capturing both semantic and bias-related information at different levels:

$$Pooling\big(H^{(l)}\big) = \frac{Mean}{Max\ Pooling(H^{(l)})}$$

After pooling, aggregate semantic and bias-related information across levels, resulting in the final representation

$$H_{final} = Aggregation(Pooling\big(H^{(l)}\big))$$

Later the final node embeddings are processed through fully connected layers, where semantic and bias-related features are combined using arithmetic operations:

$$O_{final} = \sigma(W_{final} * V_{final} + b_{final})$$

3.3 Loss Function:

The bias-aware loss function penalizes biases and unintended biases in predictions. Let O be the logits from the final layer and Y be the one-hot encoded ground truth labels. The bias term B represents the bias indicators. The loss is computed as follows:

$$Loss = \sum_{i=1}^{C}(Y_i * \log\left(\frac{e^{o_i+B_i}}{\sum_{j=1}^{C} e^{O_j+B_j}}\right))$$

In the above formula, C is the number of classes, Y_i is the ground truth label for class i, $O_i\ is\ the\ logit\ for\ class\ i, and\ B_i$ is the bias term for class i.

3.4 Training:

During training, backpropagation is utilized to update model parameters. Adjustments to weights are made using arithmetic operations involving gradients and learning rates, ensuring that both semantic and bias-related information contribute to the learning process.

Training Loop:

- For each epoch:
    - For each mini-batch:
        - Forward Pass:
            - Perform graph convolutional layers with message passing.
            - Apply hierarchical pooling to down-sample the graph.
            - Aggregate information across levels.
            - Pass through fully connected layers and obtain logits.
        - Compute Loss:
            - Calculate the bias-aware cross-entropy loss using predicted logits and ground truth labels.
        - Backpropagation:
            - Compute gradients with respect to model parameters.
            - Update model parameters using gradient descent or an optimizer.

Discussion:

Metrics:

Epoch	Validation Accuracy	Precision	Recall	F1 Score
0	0.9205	0.4115	0.9205	0.9163
1	0.9170	0.4391	0.9170	0.9186
2	0.6710	0.4791	0.6710	0.7677
3	0.8955	0.2464	0.8955	0.9056
4	0.8940	0.4171	0.8940	0.9045
5	0.8130	0.2387	0.8130	0.8598
6	0.8645	0.4360	0.8645	0.8905
7	0.8945	0.2382	0.8945	0.9041
8	0.9265	0.2079	0.9265	0.9185
9	0.7130	0.2338	0.7130	0.7969
10	0.8525	0.1716	0.8525	0.8803
11	0.8240	0.2114	0.8240	0.8653
12	0.7655	0.1898	0.7655	0.8303
13	0.6425	0.2185	0.6425	0.7467
14	0.8035	0.2158	0.8035	0.8528
15	0.5870	0.1806	0.5870	0.7067
16	0.8110	0.2402	0.8110	0.8585
17	0.8205	0.2159	0.8205	0.8630
18	0.7440	0.2244	0.7440	0.8172
19	0.8315	0.2144	0.8315	0.8697
20	0.7250	0.1751	0.7250	0.8045
21	0.7320	0.1912	0.7320	0.8098
22	0.7905	0.2277	0.7905	0.8469
23	0.7695	0.2396	0.7695	0.8338
24	0.7455	0.1962	0.7455	0.8186
25	0.8570	0.2205	0.8570	0.8839
26	0.8325	0.2203	0.8325	0.8691
27	0.8090	0.1878	0.8090	0.8564
28	0.7270	0.1986	0.7270	0.8065
29	0.8130	0.1953	0.8130	0.8593
30	0.7020	0.1816	0.7020	0.7897
31	0.7215	0.1948	0.7215	0.8029
32	0.6360	0.1884	0.6360	0.7424
33	0.7305	0.1922	0.7305	0.8085
34	0.8140	0.2508	0.8140	0.8613
35	0.6380	0.1918	0.6380	0.7440
36	0.7260	0.1867	0.7260	0.8051
37	0.8090	0.1985	0.8090	0.8576
38	0.8455	0.1874	0.8455	0.8774
39	0.7365	0.1972	0.7365	0.8121
40	0.7010	0.1871	0.7010	0.7880
41	0.7380	0.1946	0.7380	0.8133
42	0.7875	0.1832	0.7875	0.8436
43	0.8065	0.2310	0.8065	0.8557
44	0.8680	0.2279	0.8680	0.8894
45	0.7255	0.2101	0.7255	0.8059
46	0.6715	0.2019	0.6715	0.7678
47	0.7485	0.1891	0.7485	0.8197
48	0.7575	0.1797	0.7575	0.8254
49	0.7655	0.2415	0.7655	0.8318

(Table-1)

T Test:

A t-test is a statistical method used to compare the means of two groups and determine if there is a significant difference between them. It is named after the t-distribution, which is a probability distribution that is critical for the test. Here we took one of the metrics which is recall for T testing and the rest is 0.04.

$$t = \frac{X_1^- - X_2^-}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

- Where:
  - $X_1^-,\ X_2^-$ are the sample means of the two groups.
  - $S_1,\ S_2$ are the sample standard deviations of the two groups.
  - n_1, n_2 are the sample sizes of the two groups.