Using GAN & BERT for Mitigating Comment Toxicity

K Pavan Teja Reddy
CSE Department
SRM University, Amaravati, AP
Guntur, India
Pavantejareddy_kallam@srmap.edu.in

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 GAN. BERT (Bidirectional Encoder Representation from Transformers), which can learn language in both directions, and GAN (Generative Adversarial Network) Model, which is a clever technique to train a generative model. The GAN Model frames the problem as a supervised learning problem with two sub-models: the generator model, which we train to generate new examples, and the discriminator model, which tries to classify examples as either real (from the domain) or fake (generated). When the two models are trained together in an adversarial, zero-sum game, the generator model produces instances that are plausible approximately half the time, fooling the discriminator model about half the time.

Keywords—toxic comments; BERT; GAN

I. INTRODUCTION

This paper, Using GAN & BERT for Mitigating Comment Toxicity, gives readers a detailed view into the working of BERT transformer, theory behind GAN and their combined use in this study. Starting with the GAN model architecture, diving into the working of Generator and discriminator. We learn the work flow from data preprocessing, feeding it to BERT transformer for classification using a Toxicity score (0.5), in the generator, and later passing toxic comments into BERT transformer for paraphrasing these paraphrased sentences are loaded to discriminator where the discriminator network aims to distinguish between the generated paraphrases and the original sentences from the dataset. It helps provide feedback to the generator network, enabling it to improve its ability to generate realistic and semantically equivalent paraphrases.

II. METHODOLOGY

Advancing to the proposed methodology, you can see the model architecture in Figure 1. The model architecture reveals that the study deals 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, they are then forwarded to GAN model for detoxification.

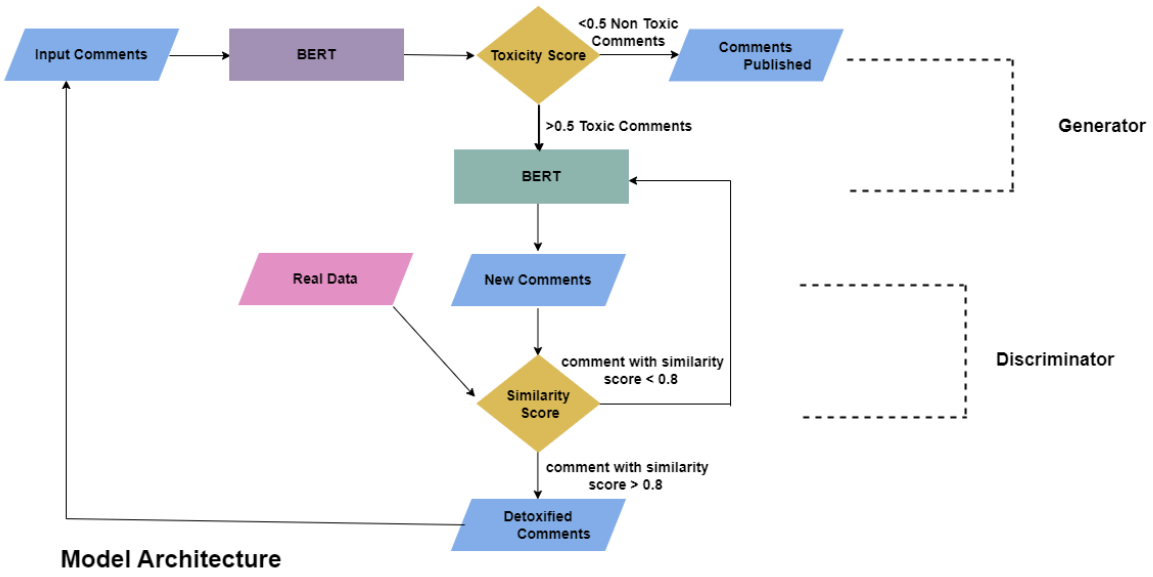


Figure 1: Model Architecture.

A. **Data pre-processing**

Comments from the Jigsaw Unintentional Bias in Toxicology dataset are taken as input

$$C = \{c_1, c_2, \ldots\ldots, c_k\} - (1)$$

, where c_i, i belongs to n is a comment, and k be number of comments in the dataset. The input

$$c_i = \{s_1, s_2, \ldots\ldots, s_l\} - (2)$$

where s_i is a sentence, is one comment, and each comment has the potential to include a number of sentences, say l. be $S_i = \{w_1, w_2, \ldots\ldots, w_m\}$; 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.

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:

$$W = \{w_1, w_2, \ldots\ldots, w_m\} \ - (3)$$

$$\text{Tokenized as } \ W' = \{\mathrm{w}_1', w_2', \ldots\ldots, w_{\mathrm{m}}'\} - (4)$$

Padding and Truncation

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

$$s_i = \{\mathrm{w}_1', w_2', \ldots\ldots, w_{\mathrm{m}}'\} - (5)$$

$$s_i' = [CLS] + W' + [SEP] - (6)$$

, 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.

In this study the BERT model that was employed is $BERT - BASE - UNCASED$ which is trained on large corpus of data, through which it learned dense vector representations. This BERT model contains collection pre-trained word embedding in a matrix representation. Using the matrix, the model identifies pre-trained word embedding associated with each token, including the special tokens. These word embeddings capture the tokens' semantic content. Let S be the sentence representation after tokenization and S_E after word embedding:

$$S = \{s_1', s_2', \ldots\ldots, s_n'\} - (7)$$

$$S_E = \{S_{e_1}', S_{e_2}', \ldots\ldots, S_{e_n}'\} - (8)$$

Positional and Sentence Embedding

After tokenization, each token is mapped to a unique index from the BERT vocabulary. This index shows where the token is located within the vocabulary. Generating respective positional Embedding for each sentence:

$$p_e = \{Pos_{S_{e_1}'} + Pos_{S_{e_2}'} + \ldots. + Pos_{S_{e_n}'}\} - (9)$$

$$P = \{p_{e_1} + p_{e_2} + \ldots\ldots + p_{e_m}\} - (10)$$

, where p_e is positional embedding of the embedded tokens. To calculate positional embedding sinusoidal function is applied

$$POS_{E(pos,2i)} = \sin\left(\frac{pos}{1000^{\frac{2i}{d_{Model}}}}\right) - (11)$$

$$POS_{E(pos,2i+1)} = \cos\left(\frac{pos}{1000^{\frac{2i}{d_{Model}}}}\right) - (12)$$

The sinusoidal function assigns a unique position embedding to each place in the sequence. It uses sine and cosine functions of varying frequencies to encode position information. Each token now has a sentence embedding signifying Sentence *A* or Sentence B. Sentence embeddings are conceptually similar to token embeddings with a vocabulary indicating which sentence it belongs to.

$$Sen_E = \{P_A + P_A + \ldots\ldots + P_B + P_B\} - (13)$$

; where Sen_E represent the sentence embedding.

The combination of word embedding (S_E), positional embedding (P), sentence embedding (Sen_E) is later sent over to BERT's Self-attention layer.

$$E_v = S_E + P + Sen_E - (14)$$

; where E_v is final embedded vector.

B. **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. Figure 3. In this experiment we incorporated BERT base uncased with 12 encoder layers. Figure 2.

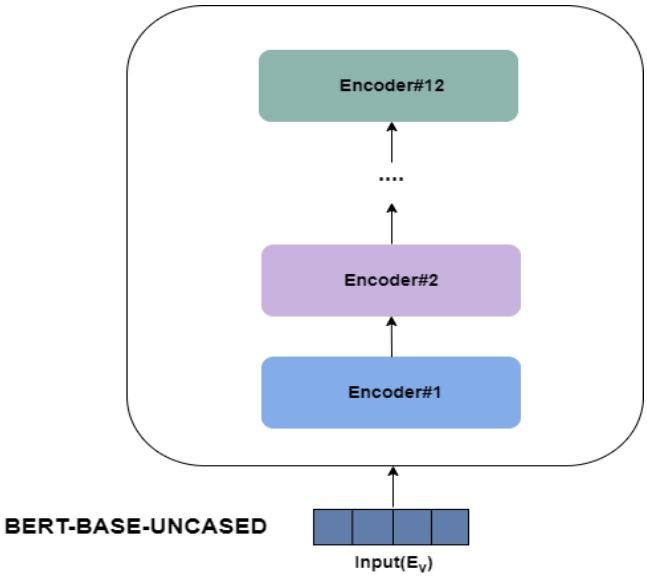


Figure 2: BERT-Base-Uncased

Self-attention Layer:

When the embedded input sequences (E_v) 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 [1].

$$QueryVector(Q) = E_v \times M_q = Q\{q_1, q_2, \ldots, q_n\} - (15)$$

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

$$KeyVector(K) = E_v \times M_k = K\{k_1, k_2, \ldots, k_n\} - (16)$$

(Note: Contains encoded information)

$$ValueVector(V) = E_v \times M_v = V\{v_1, v_2, \ldots, v_n\} - (17)$$

(Note: 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.

$$Attentioncore(A_v) = Softmax\left((Q \times K^T) \div \sqrt{\dim}\right) \times ValueVector(V) - (18)$$

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

$$Softmax(L_i) = e^{Li} \div \sum_{i=0}^{6}(e^{Li}) - (19)$$

Feed Forward Neural Network:

This score is then passed to feed forward neural network for further processing. 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 non-toxic. The output nodes, or the classes of different toxicities are present in this final dense layer.

$$DenseLayer\_output = activation\left((w_1 \times hidden\_state) + b_1\right) - (20)$$

$$Final\ output(F) = w_2 \times Intermediate\_output + b_2 - (21)$$

$W_1(in, hid)$ 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)

$$H = [h_1, h_2, \ldots, h_n], - (22)$$

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

$$\text{Activation} = ReLU \text{ or } Sigmoid$$

ReLU (Rectified Linear unit) function:

$$R(x) = max(0, x) - (23)$$

Sigmoid function:

$$S(x) = 1/(1 + e^{(-x)}) - (24)$$

$W_2(hid, out)$ 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.

$$Output_{Logits}(OL) = \left(h_f X W_m\right) + B - (25)$$

where, h_f is the final hidden layer, W_m is the weighted matrix *and,* 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 has values between 0 and 1. Non-toxic values are near to 1, and very toxic values are close to 1.

$$ToxicityScores(TS) = Softmax(OL) - (26)$$

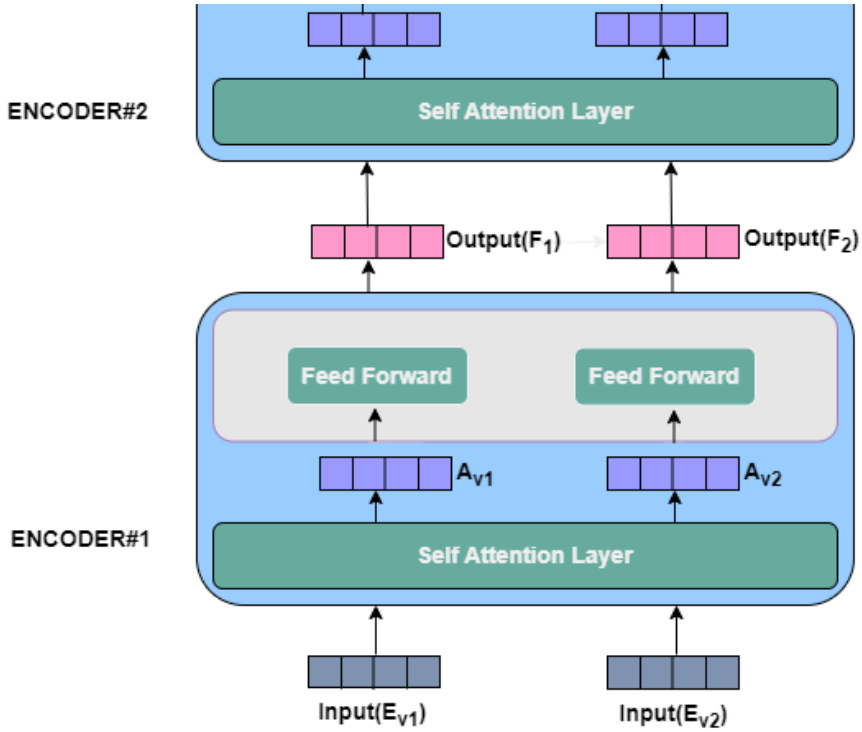


Figure 3: BERT Architecture

Training:

The training loop uses the $AdamW$ optimizer as its optimizer. We added weight decay to the optimizer to assist the model learn a more straightforward representation of the data, preventing overfitting. The hyperparameter $learning_{rate}(lr)$ is set to $2e-5$, usually lr varies between $1e-5$ to $2e-5$. We adjust the learning rate so that it is not too large where the optimizer makes huge steps leading to rapid convergence or too little making the optimizer take shorter steps leading to slower convergence. Below equations represent how weights of parameters are updated by $AdamW$ optimizer.

$$g_1 = decay_1 * m + (1 - decay_1) * G \quad -(27)$$

$$g_2 = deacy_2 * v + (1 - decay_2) * G^2 \quad -(28)$$

$$updated_{parameters} -= learning_{rate} * (\frac{g_1}{(\sqrt{g_2})} + epslion) + lambda * parameter_{values} \quad -(29)$$

G is gradient of the loss function. g_1 and g_2 are momentary gradients, and they monitor the direction and size of gradients. $decay_1$ and $decay_2$ are decay rates these are associated to adjust g_1 and g_2 overtime during training , $Epsilon$ added to avoid denomination by $zero$.We employed $BCEWithLogitsLoss$ which combines sigmoid activation function with binary cross entropy. To evaluate model predictions to the actual values.

$$BCEloss = -\left[target_{labels} * log\left(label_{probability}\right) + (1 - target_{labels}) * log\left(1 - label_{probabiliry}\right)\right] \quad -(30)$$

The $BCEloss$ with Logits function minimizes the measure of error during training of a multi-label classification model by comparing the predicted probability distribution and the ground-truth (original) probability distribution for each label. The loss function is given by

$$loss = \left(\frac{[\sum_0^n BCEloss\ ]}{n}\right) \quad -(31)$$ (Note: n here refers to no of labels)

Overall loss is taken by losses of all labels divided by number of labels. In order for the model to understand the patterns in the training data and generate precise predictions on new, untainted data, the loss function will be minimized during training. Based on the gradients calculated by the loss function, the $AdamW$ optimizer modifies the model parameters at the final stage.

Algorithm 1: BERT Algorithm

---

model = $BERT$ ($BERT - Base - Uncased$)

loss = $BCEWithLogitsLoss()$

optimizer = $AdamW$

for epoch in range (3):

for batch in data_loader:

- preprocess inputs: preprocess(batch)
- resetting gradients to Zero: $optimizer.zero\_grad()$
- Feeding data inputs to the model layers

$$Logits = model(input\_ids, attention\_mask)$$

- Compute Loss $(Logits, Labels)$
- Backpropagation: $loss.backward()$
- update the weights of the layers
- $optimizer.step()$

In the end, comments that are held below the toxicity level (< 0.5) are finally published, while those that are over the threshold (> 0.5) are considered $Toxic\_Comments(T_c)$ and are sent over to GAN for detoxification.

C. **Generative Adversarial Networks (GAN):**

The proposed GAN model, starts with a vector that randomly creates noise based on a custom dataset which contains toxic comments coupled with their non-toxic paraphrases. This noise serves as the input to the BERT generator, a neural network that uses the noise to produce fake non-toxic comments. Consequently, there are true data comments those are actual/real non-toxic comments. Figure 4. Note that the generator produced fake comments and original comments are of same dimension. [4]

Generator and Discriminator Working:

$RandomNoise\_input(I_n)$ generated based of the custom dataset, is passed through Generator neural network (Gen) to produce fake non-toxic comments $Gen(I_n)$ and the output $Gen(I_n)$ has the same dimension as the real non-toxic data (R_{NT}). $Gen(I_n)$ is sent to the discriminator which checks the similarity score of generated $Gen(I_n)$ and real data (R_{NT}).

Moving to the discriminator neural network part of the model, input to the discriminator is the output of the generator i.e., fake non-toxic comments. The goal of the discriminator is to determine whether the provided input sample is real comment or fake generated comment. The Discriminator's task is to generate a single number that indicates the likelihood that the input is drawn from a real data distribution. Typically, we assign one if the sample is real and $zero$ if it is not. When a fake sample is sent to the discriminator it is denoted as $Dis\big(Gen(I_n)\big)$ this represents the discriminator estimate of probability that the fake sample is real and when real sample is sent it is denoted as $Dis(R_{NT})$ which represents the discriminator estimate probability that the real sample is real. The cost function for GAN could be given by

$$V(Dis, Gen) = E_{real}[log(Dis(R_{NT}))] + E_{noise}[log(1 - Dis(Gen(I_n)))] - (32)$$

(Note: $V(Dis, Gen)$ is the cost function. E_{real} represents the expectation over real data and E_{noise} represents the expectation over noise)

The cost function mentioned above is divided into two components. showing the discriminator predictions on real non-toxic comment data $Dis(R_{NT})$ first, and then showing the predictions on fake comments $Dis\big(Gen(I_n)\big)$. The cost function operates by giving the discriminator a high degree of confidence that the real comments $Dis(R_{NT})$ are, in fact, real. Expectation (E_real) is the average of the predictions. Although in training both the generator and the discriminator operate together, here the discriminator operates on its own.

The discriminator is trained on both $Dis(R_{NT})$ and $Dis\big(Gen(I_n)\big)$. $Dis\big(Gen(I_n)\big)$should be as little as possible for the discriminator in order to show confidence that the generated comments are indeed fake. The generator, however, deceives the discriminator by making the value as large as possible because our objective is to produce comments that are indistinguishable from actual comments. The $Dis\big(Gen(I_n)\big)$ is therefore highly confident. The adversarial game between the Generator and Discriminator is thus set up. The $\big[1 - Dis\big(Gen(I_n)\big)\big]$ is used in the equation because it aids in orienting the preferences of the generator and discriminator in a consistent manner between the two components. The $\big[1 - Dis\big(Gen(I_n)\big)\big]$ transforms the quantity from something the discriminator wants to minimize to maximize. Figure 5. So, in conclusion the discriminator maximizes the entire thing and the generator minimizes and we end up with optimization.

$$min - max\, V(Dis, Gen) = min\_\{Gen\}\, max\_\{Dis\}\, L(Dis, Gen) - (33)$$

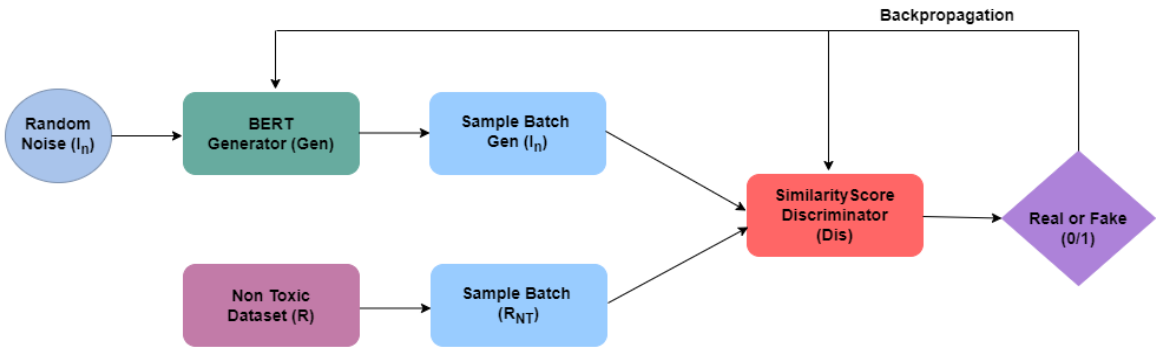
(Note: $min\_\{Gen\}$ represents the minimization of $\{Gen\}$ and $max\_\{Dis\}$ represents the maximization of $\{Dis\}$)

The adversarial loss function $L(Dis, Gen)$ is represented as

$$L(Dis, Gen) = E_{real}\big[\log\big(Dis(R_{NT})\big)\big] + E_{noise}\Big[\log\Big(1 - Dis\big(Gen(I_n)\big)\Big)\Big] - (34)$$

(Note: $L(Dis, Gen)$ represents loss function and measures similarity between Discriminator & Generator)

Generator minimizes the cost function and a discriminator maximizes the cost function.



Generative Adversarial Networks (GAN)

Figure 4: GAN Architecture.

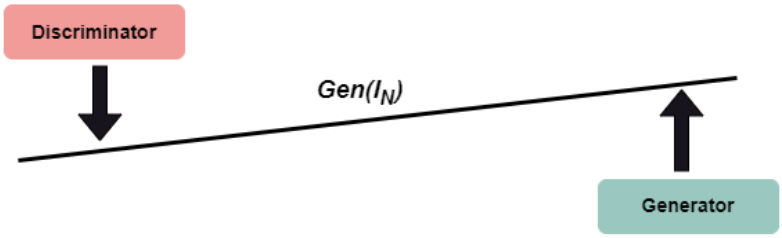


Figure 5: Adversarial Game Between Generator & Discriminator.

Algorithm 2: GAN Algorithm

---

for i in epoch:

for j in (3):

- Generate $Gen(I_N)$ comments $\{f_1, f_2, \ldots, f_n\}$ through Generator
- take a batch of R_{NT} comments $\{r_1, r_2, \ldots, r_n\}$ (same dimension as $Gen(I_N)$
- discriminator_maximize_gradient (R_{NT}, I_N)

$$\nabla_{\theta_{Dis}} \frac{1}{m} \sum_{i=1}^{m} [\log(Dis(R_{NT}^i)) + \log(1 - Dis(Gen(I_N^i)))]$$

(Emphasises the distinction between the discriminator's ratings assigned to real and fake data)

End

- generate $Gen(I_N)$ comments $\{f_1, f_2, \ldots, f_n\}$ through Generator
- update the discriminator parameters by minimizing the gradient

---

Training:

Step - 1: Generate a set of fake non-toxic comments using Generator with same dimensions as real data.

Step - 2: The discriminator is trained on a set of real comments as well as fake comments with the goal to classify whether the comment is real or fake. We calculate the gradient of the loss function with respective to discriminator labels to increase discriminator parameters.

Step - 3: Generate more fake data using random noise as input to generator.

Step - 4: Train the generator on the new fake data with the goal to create comments indistinguishable from the real data. We calculate the gradient of the loss function with respective to generator labels to increase generator parameters.

Step - 5: Repeat Step 1 to Step 4 till convergence.

Once BERT sends the $Toxic\_Comments(T_c)$ to pretrained GAN model they are detoxified and sent over to BERT again for toxicity evolution. This process repeats till the GAN model is trained until it gains the ability to detoxify every single toxic comment.

D. **Results**

Table 1: BERT Toxic Comment Classification Metrics on different parameters

Model	Parameters			Metrics			
	LearningRate(lr)	Batchsize	No of epochs	Accuracy	Precision	Recall	F1 Score
BERT	2e-3	32	3	0.9625	0.9925	0.9625	0.9441
	4e-1	16	3	0.9060	0.2845	0.9060	0.9103
	4e-1	16	4	0.9220	0.2295	0.9220	0.9281

The resulting scores for the BERT text classification model, which is designed for toxicity classification, are displayed in the above table for various parameters.

REFERENCES

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin (2017) Attention Is All You Need

[2] Minh Quan Do (2019), Jigsaw Unintended Bias in Toxicity Classification

[3] Lucas Dixon, John Li, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. Measuring and mitigating unintended bias in text classification. 2019.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[5] Xu Chen, Jiang Wang, and Hao Ge. Training generative adversarial networks via primal-dual subgradient methods: a lagrangian perspective on gan.arXivpreprint arXiv:1802.01765, 2018.

[6] R. Rivaldo, A. Amalia and D. Gunawan, "Multilabeling Indonesian Toxic Comments Classification Using The Bidirectional Encoder Representations of Transformers Model," 2021 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA), Medan, Indonesia, 2021, pp. 22-26, doi: 10.1109/DATABIA53375.2021.9650126.

[7] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862, 2017.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.