Fine-Tuning Transformer Based Models for Tweet Classification

Faizan Athar Waheed Khan
Registration Number: 2004470
Computer Science and Electronic Engineering
University of Essex
Colchester, UK
fw20657@essex.ac.uk

***Abstract*—Twitter has risen in popularity as a platform for people to express their opinions on a wide range of topics since its advent as a microblogging site. Often the nature of such tweets is insulting, derogatory, or negative, and it becomes particularly important to identify and, in some cases, block such tweets. Recently, the use of transformer-based pretrained Language models has gained tremendous momentum of their intrinsic ability to extract and learn contextual information that otherwise would have been challenging to identify. As a state-of-art model, BERT and RoBERTa have performed exceptionally well in the task of text classification. This research primarily investigates the use of pretrained BERT and RoBERTa models with various fine-tuning strategies and hyperparameters optimization, intending to optimize the performance in the tweet classification task. We gather the TweetEval dataset for three classification task: Hate language detection, Offensive language identification and Sentiment analysis. We then draw a distinction between the performances of various fine-tuning techniques and decide on an optimal strategy. We experiment with different learning rates for the convergence of the network and different batch sizes to optimally train the network. Next, we critically evaluate the performance of each fine-tuning strategy with the BERT and ROBERTA models and the performance based on the tuning of different hyperparameters. The experimental results indicate that the optimum combinations of fine-tuning strategy and hyperparameter tuning outperform the state-of-art on all three tasks of tweet classification.**

***Index Terms*—BERT, Fine-tuning, Tweet Classification**

I. Introduction

Online media networks have grown in popularity in recent years. However, freedom of speech may sometimes result in aggressive and undesirable behaviour. For instance, hate speech, bigotry, obscene words, doxing, and other forms of offensive speech have become a serious issue for all major online social media platforms. This behaviour is most prevalent on Twitter due to its brief messages and highly engaging nature. Due to the massive volume of user-generated content, manual analysis is impractical. Obligated by statute to eliminate hate speech from their platforms, internet media organisations have spent significant time and money automating the detection and classification of hate speech and offensive words. In order to address this need, the attention was directed to the task of tweet classification

Tweet classification as a natural language processing task has garnered much attention and interest from scholars and has been extensively researched and provided state-of-the-art results. Tweets are produced in large quantities and, in most cases, contain a wealth of information. However, when exposed to noisy user-generated text, existing natural language processing systems are usually ill-equipped.

Simple tasks such as emotion analysis have been deemed difficult to perform on Twitter data (Poria et al., 2020), partly as a result of the scarcity of contextual clues in short texts (Kim et al., 2014). The most difficult aspect of sentiment analysis is the ambiguity inherent in term meanings. A term can have a positive connotation in one context and a negative connotation in another. To make sense of such text, it becomes vital to extract the contextual meaning out of the sentence. This makes the use of pretrained models especially significant.

Extensive research has demonstrated that pretrained language models on vast corpora are advantageous for various natural language processing activities, avoiding the need to train from scratch. Pretrained models include word embeddings like word2vec [1] and GloVe [2], as well as contextualised word embeddings such as CoVe [[3] and ELMo [4]. Sentence level pre-training models are another form of pre-training model. Howard and Ruder (2018) introduce ULMFiT, a system for fine-tuning pre-trained language models that achieves state-of-art performance over six commonly used text classification datasets. BERT is trained on plain text for masked word prediction and next sentence prediction tasks.

While BERT has shown remarkable performance in a variety of NLP challenges, its potential has yet to be realised. Not enough research has been carried out when it comes to improving the performance of BERT using fine-tuning strategies. This paper investigates various fine-tuning strategies and their impact on the task of the tweet classification task. We also perform hyperparameter tunings and evaluate the performance of the model using different settings. Finally, we compare the performance of our model on the three tasks with the TweetEval benchmark.

II. Background Study

This section will briefly present some background study needed to comprehend the paper.

A. BERT

Prior to the introduction of transformer architecture, most of the Natural Language Processing task (NLP) tasks used bidirectional LSTM, which had a vanishing gradient problem. Based on the attention mechanism, learning contextual relations among words within a sentence [5], the transformer architecture processes the entire sequence at once, enabling it to train parallelly. Bidirectional Encoder Representations from Transformers (BERT) by Google Research [6], based on the transformer architecture, has given state-of-the-art results for various Natural Language Processing task (NLP) and is assumed to be the starting point of the modern NLP revolution. However, though based on the transformer architecture, the BERT model works without a decoder layer and instead has 12 encoder layers in the base version and 24 layers in the Large version, stacked on top of each other. The BERT base model has a feed-forward network of 768 hidden layers and 12 attention heads with 110M parameters.

1) Input to BERT: The very first input fed to a BERT model is a special token called [CLS], where CLS denotes Classification. Next, it takes in input as a sequence of words, where the first layer applies self-attention on it and outputs the result to the feed-forward network, which is then pushed to the next encoder layer. Essentially, BERT is fed with a sequence of tokens as input, which are then converted to vectors and processed in the feed-forward network. It gives output as a sequence of vectors, maintaining the input index.

2) BERT Training: The model is trained using two techniques: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP)

a) MLM: Each sequence that is given as input to BERT has 15 percent words substituted with a special MASK token. BERT learns the context from the remaining words in the sequence and tries to forecast the masked words. It achieves this by adding a fully connected classification layer at the output of the encoder and using softmax to compute the probability of words. However, the convergence of the model is relatively low as the loss function only takes into account the forecasting of masked words while ignoring the unmasked words.

b) NSP: The BERT model is trained by feeding it with two sentences at a time and it is made to learn and forecast whether the first sentence and second sentence are logically connected and the second sentence is supposed to be the subsequent sentence. It is achievied by training the model with 50 percent of the sentence pair that are contextually connected and the other 50 percent pair that are not connected. It is expected that the model learns the relationship between the connected sentences and is able to seggregrate the non related sentences.

III. Literature Review

Ever since the introduction of Transfomer based BERT architecture, its application has been extensive in the field of NLP. It can be used by tailoring it specifically to the task requirement. This technical terminology is referred to as fine-tuning of BERT. Several studies have fine-tuned BERT to achieve a state-of-the-art score in various NLP tasks.

The authors of [7] investigated different BERT fine-tuning strategies for the task of text classification. Their study presents some interesting finding; they concluded that (i) BERT's top layer is especially useful when the task is text classification. (ii) The fine-tuning can improve the accuracy of tasks with a smaller dataset.

The BERT model is based on the concept of Transfer learning. Resource-based transfer and model-based transfer are the two general paradigms for Transfer learning in NLP [8]. Our approach follows the Model-based paradigm, which leverages the resemblance between the task the model was trained and the task that is needed to be performed. [9] enhanced the performance of various NLP tasks by using a transfer learning framework that shared structural parameters among various tasks. [10] proposed a convolutional neural network catered explicitly for the task and that implemented simultaneous training to pass information from NER and POS tagging.

BERT is also extensively used to classify user sentiment across various social media platform including twitter. [11] built a BERT-based text classification model, they constructed auxiliary sentence to make text classification as the sentence-pair prediction task, which helped the model learn task-specific knowledge. Particularly in the field of tweet classification, there have been many exciting developments across varied disciplines. [12]] used BERT to classify medical and non-medical tweets; their model showed decent accuracy and was able to generate a vast database of medical tweets. Whereas [13] used BERT to predict sentiments of users of six different US Airlines in the field of aviation. Their findings suggest that lexical ambiguity in word representation cannot be tackled using traditional word vector generation models. Their result confirms the excellent performance of word representation by BERT.

Within tweet classification, the task of sentiment analysis has always been a topic of interest among researchers. Many have tried using it to understand the sentiment of users over social media. [14] developed a model to classify sentiments of users over Twitter. The model extracted semantical information from the tweets and classified them based on the extracted knowledge. To enhance the semantical information retrieval process from within the tweets, they used a knowledge enhancer. This made sure that no significant information was lost from the tweets.

However, the studies are not just limited to sentiment analysis; several studies have been done to detect offensive and hateful tweets on the platform. [[15] compared models based on feature engineering and word embedding based models and performed the task of hate and offensive language classification. They concluded that it is easier to perform classification on a binary task of hate speech or offensive content detection, but it becomes difficult to classify hate speech from offensive content.

IV. Methodology

In this study, we aim to develop a model for the tweet classification and compare the results of various tasks with the Tweeteval benchmark [16]. We developed two different Language models for classification depending on the tasks. These models are based on transfer learning principles; we specifically use the Transformer model. We choose a pre-trained BERT base,uncased (henceforth BERT) model for Hate language detection, whereas we used a pre-trained RoBERTa based RobertaForSequenceClassification (henceforth ROBERTA) model for Sentiment analysis and Offensive language detection. The prototyping and development were carried out on Tesla 4 GPU provided by Google Collab. For our implementation, we used BERT and RoBERTa from the Hugging Face PyTorch-transformers package, based on the PyTorch framework. Whereas, for evaluation metrics, we used the sci-kit-learn libraryin python. Finally, we used matplotlib and seabor for data visualisation. The methodology consists of the following sections: Firstly, we represent the various datasets provided by Tweeteval and the classification tasks we perform. Then, we discuss in detail the text preprocessing that we perform on the tweets. Next, we go over the method of text tokenisation needed for performing classification. After that, we briefly discuss Transfer learning approaches and different fine-tuning strategies. Finally, we present a brief overview of the fine-tuning we performed on BERT and ROBERTA models for different classification tasks.

A. Dataset

The TweetEval repository consists of datasets for seven different tasks of multi-class tweet classification for twitter. However, out of the seven mentioned task, We shortlisted three tasks for the classification: Sentiment Analysis, Offensive Language Identification and Hate Speech Detection. The details of the dataset can be seen in Table I

TABLE I
TweetEval Tweet Classification Tasks

Classification Task	*Labels*	*Train Set*	*Validaton Set*	*Test Set*
Hate Speech Detection	Hateful / Not Hateful	9000	1000	2970
Offensive Language Identification	Offensive / Not Offensive	11916	1324	860
Sentiment Analysis	Negative / Neutral / Positive	45615	2000	12284

B. Proposed Architecture

C. Data Preprocessing

The Twitter data was noisy, inconsistent, and highly dimensional, requiring cleaning and preparation to make the unstructured data, relatively structured and meaningful before processing it further. As part of the dataset, we were provided seven files for each task, the first three files containing train texts, validation texts and test texts, respectively. The next three files have the ground truth for all three text files and finally a map file that maps the labelled values to the class; for example, for sentiment analysis, the labelled value maps to the class 'Negative'. As part of data preparation, we created three different Pandas DataFrames containing the tweets for training, validation and testing using the given files. After that, texts from tweets were extracted by removing URLs, usernames, special characters, including emojis and numbers. We also removed all the words with two or less than two character count. Furthermore, we created a list of words that we ignored; most of these were created because of the text cleaning we performed. For example, the word 'user' added to the list of 'ignore' words. Besides data preparation, we changed all the words to lowercase as uppercase or lowercase forms of words usually do not make much difference in classification. We performed the same preprocessing across the three datasets for training, validation and test, for all three classification tasks. The amount of preprocessing required is comparatively less for the Bert and Roberta transformer models as they are pre-trained on an entire sentence for a large corpus of text data.

D. Input to Models

After the initial text cleaning and data preprocessing, we need to tokenise the data before feeding it to a transformer model. To start with, we calculate the maximum length of sentences for our train and validation dataset and pass this length to the tokeniser. The sentences that are shorter than the specified maximum length are padded to the maximum length, whereas the sentences that are longer than the computed max length are truncated to a size of the maximum specified length.

Next, the tokeniser from the Hugginface transformer library is responsible for converting text into tokens. The tokeniser first splits a sentence text in words, referred to as tokens, and then converts these tokens to numbers. These numbers are later built as tensors and fed to the model.

For our experiment, we used two different pretrained tokenisers, each associated with the pretrained transformer model. Pretrained RobertaTokenizer was used with ROBERTA model whereas, BertTokenizer was used with BERT model. The reason for using an associated tokeniser is that it transforms texts into tokens in a similar format with which the model was pretrained. Similarly, it converts tokens into indices, maintaining the format of the pretrained model. Also, the pretrained models have a precise and fixed vocabulary, and these tokenisers handle out-of-vocabulary words in a specific manner.

After this, we get back a dictionary containing the list of input_ids and attention_mask from the tokeniser. This list is later converted to PyTorch tensors as they have the capability to run on GPUs.

Besides creating tensors, we create the PyTorch Dataloader object as well for our dataset. The use of dataloader saves memory while training and enhances training speeds.

E. Fine Tuning Strategies

Fine-tuning is a process of training a pretrained model further with a relatively small dataset. There are three ways with which we can fine-tune a model: (i) Training the entire architecture. (ii) Training some layers while freezing other layers. (iii) Freezing all the layers, adding a new layer and

training it. Since most of the information is already stored in the pretrained model, fine-tuning requires relatively less time and epochs to train the model.

We fine-tune the BERT and ROBERTA models for the downstream task of text classification. While fine-tuning a model, it particularly essential to look out for overfitting and losses during training. Thus, to avoid overfitting, minimising losses, and accelerating the model's convergence, we use the AdamW [17] optimiser. We also used a linear learning rate scheduler with warmup, which adjusts the learning rate while training. It achieves that by reducing the learning rate as per the predefined schedule. The rest of the section will briefly go over the fine-tuning we did for each of the three classification tasks.

1) BERT for Downstream Tasks:

a) Hate tweet Classification: For our setup, we used the pretrained BERT model to classify Hate tweets. To fine-tune, we added adding one fully-connected layer on top of BERT, froze all the existing layers in the architecture and trained only the new layer, retaining all the other layers' weights from the existing architecture. We applied this strategy since our training data is fairly similar to the data on which the model was pretrained. To elaborate, we introduced a dropout layer to enable regularisation, followed by a linear classification head and a subsequent Sigmoid layer. Also, we added an extra linear layer to make BERT output two classes (hate and not hate).

2) ROBERTA for Downstream Tasks:

a) Offensive Language Classification: : We used the pretrained Roberta for sequence classification as it has given state-of-the-art results for the task of tweet classification. For fine-tuning the pretrained ROBERTA model, we only retain the model's architecture and discard the weights of every other layer. Next, we train the entire model from scratch with our training data. We used this approach since our data is slightly different from the data on which the model was pretrained. During the model initialisation, we specify the number of desired class (offensive and not offensive) as two to be the output of ROBERTA's classification layer.

b) Sentiment Analysis: : We used the pretrained Roberta for sequence classification again for this task. For fine-tuning the pretrained ROBERTA model, we retain the model's architecture and weights of a few layers, i.e. we train it partially. We retrain only the upper layers of the model and freeze the weights of the initial layers. The number of layers to be frozen is task-dependent and can be tuned. Again we specify the number of desired class as three (negative, neutral and positive) to be the output of ROBERTA's classification layer.

F. Evaluation Metrics

To evaluate the performance of our model, we use different evaluation metrics depending on the classification task. We use macro averaged F1 for hate speech detection and offensive language detection, whereas for sentiment analysis, we use macro averaged recall.

V. Experiments

For the given three classification tasks, we fine-tuned different transformer models. We tried several transformer models, including the BERT model for all the classification tasks; the results were promising but had further room for improvements. However, based on the critical analysis and interpretation of the results, we chose two different models for the three classification task. This section presents details of the various settings and experiments that were needed for fine-tuning the models for each classification task.

A. Hyperparameters

As a standard-setting throughout all the experiments, we fixed the epsilon value to a minimal value of 1e-8 to avoid division by zero problems for the AdamW optimiser. Likewise, we perform tuning of two hyper-parameters for all the experiments: Learning rate and Batch size. The learning rate of a neural network model determines how quickly or slowly it learns a task. Accordingly, we tune the learning rate to a range of values suggested by authors of BERT, such as 1e-5, 1e-4, 5e-5 6e-5, 6e-6, and the batch size to recommended values of 16, 24, 32,64 ensuring we optimally utilise the GPU memory. When selecting a batch size, there is always a tradeoff between learning efficiency and development speed. A small batch size increases learning, whereas a bigger batch size increases training speed. The max sequence length was computed dynamically from training and validation datasets for hate and offensive language classification tasks, whereas it was set to 250 for the sentiment analysis task

B. Hate tweets Classification

Classification of tweets for this task was not straightforward and required much tuning. To begin with, we experimented with the BERT pretrained model by maintaining its original architecture and retraining it from scratch. Later, we tried freezing a few of the layers, retaining their original weights. However, none of these experiments resulted in good accuracy when compared to the Tweeteval benchmark. Eventually, we added an output classification layer on top of BERTS existing architecture and froze all the existing layers. The two dropouts were fixed to 0.1 and 0.17 after trying with several values. Next, we trained the model for three epochs with a range of learning rates and batch sizes discussed above.

C. Offensive language Classification

This task required fine-tuning of a pretrained ROBERTA model. During the prototyping phase, documented in report 1, it was evident from the series of experiments conducted that our data had significantly less similarity with the data with which the model was pretrained. We tried freezing several layers and only training a few layers, but none of the trials yielded a decent score. With these settings, we went ahead and trained the model for two epochs with different learning rates and batch sizes.

D. Sentiment Analysis

For the given task, we fine-tuned a pretrained ROBERTA model. We have a large dataset for this task, so the optimal design involves freezing some of the initial layer s and retrain the remaining top layers. Similar to the above two tasks, we trained the model for different learning rates and batch sizes but only for a single epoch.

VI. Results

We ran our experiments for different batch sizes and Learning rates; however, we only describe the most significant findings.

A. Hate tweets Classification

Table II shows the result of Hate tweets classification—the experimental settings comprised a fixed batch size of 24 and different learning rates. Whereas Table III depicts the performance with a fixed learning rate of 6e-5 and different batch sizes.

The batch size of 24 and the learning rate of 6-e5 were chosen after conducting several rounds of experiments and tuning them in each iteration

TABLE II
HATE ANALYSIS WITH BATCH SIZE 24

Learning Rate	*Accuracy*
6e-4	54.31
6e-5	**60.15**
6e-6	36.36

TABLE III
HATE ANALYSIS WITH LEARNING RATE 6E-5

Batch Size	*Accuracy*
16	58.97
32	57.98
64	49.07

Table IV presents the performance of our model for the given task, achieving an accuracy of 60.15%, outperforming the Tweeteval leader board by a margin of almost four percent.

TABLE IV
HATE ANALYSIS WITH BEST ACCURACY

Batch Size	*Learning Rate*	*Accuracy*
24	6e-5	60.15

B. Offensive language Classification

Similar to Hate tweet analysis, we tested the performance of our model for offensive language classification by first by keeping the batch size constant and next by keeping the learning rate fixed.

Table V illustrates the Offensive language classification task results, where we kept the batch size fixed to 34 and varied the learning rates. On the contrary, table VI demonstrates the performance with a fixed learning rate of 5e-5 and different batch sizes.

Table VII shows the best accuracy of 82.11% for this task, beating the benchmark set by the Tweeteval leaderboard roughly by two percent.

TABLE V
OFFENSIVE LANGUAGE WITH BATCH SIZE 34

Learning Rate	*Accuracy*
5e-4	41.89
5e-5	**82.11**
5e-6	80.56

TABLE VI
OFFENSIVE LANGUAGE WITH LEARNING RATE 5E-5

Batch Size	*Accuracy*
16	80.57
32	81.72
64	81.48

TABLE VII
OFFENSIVE LANGUAGE WITH BEST ACCURACY

Batch Size	*Learning Rate*	*Accuracy*
34	5e-5	82.11

C. Sentiment Analysis

For the sentiment analysis task, we measure the performance of our model the same way as the other two tasks, i.e., by tuning the batch size and learning rate.

The performance of our model with the fixed batch size and different learning rates is highlighted, and vice-versa is highlighted in table VIII table IX, respectively. For the given task, we achieved the best results with a batch size of 15 and a learning rate of 1e-5.

TABLE VIII
SENTIMENT ANALYSIS WITH BATCH SIZE 15

Learning Rate	*Accuracy*
1e-4	33.33
1e-5	**73.66**
1e-6	70.97

TABLE IX
SENTIMENT ANALYSIS WITH LEARNING RATE 1E-5

Batch Size	*Accuracy*
16	73.53
32	73.45

The performance of our model on the Sentiment analysis task is portrayed in table X. we achieve an accuracy of 73.66%, marginally surpassing the Tweeteval leaderboard.

TABLE X
SENTIMENT ANALYSIS WITH BEST ACCURACY

Batch Size	*Learning Rate*	*Accuracy*
15	1e-5	73.66

Table XI compares the performance of our models for the given three tasks with the TweetEval leaderboard. It is evident from the table that we have outperformed the TweetEval leaderboard in all three tasks.

VII. Discussion

It is apparent from the results of all three classification tasks that varying the Batch size does not have a significant impact on the model's performance. Having said that, it was of umpteen importance for us to set an appropriate batch size, as a variation of 0.5% could be a deciding factor for a position on the leaderboard. We had to select the batch size empirically for all three tasks, and it was primarily dependent on the size of the data and the fine-tuning strategy that we chose. For

TABLE XI
CLASSIFICATION ACCURACY ON TEST DATASET

Model	*Hate*	*Offensive*	*Sentiment*
BERT $\|\|ROBERTA$	60.15	82.11	73.66
BERTweet	56.4	79.5	73.4
RoBERTa-Retrained	52.3	80.5	72.6
RoBERTa-Base	46.6	79.5	71.3

all three tasks, batch size between 15 to 34 gave a decent performance. Indeed, it is safe to say that a smaller batch size yield better results.

On the other hand, the learning rate had a colossal impact on the model's performance. As the result shows, we get great results when the learning rate is significantly less. Also, it is highly dependent on the fine-tuning task that we are performing. In addition to that, we observed that lower learning rates in the range [1e-5 – 6e-5] gives the best performance for all three classification tasks. Consequently, with a higher learning rate, the performance of Sentiment analysis and the offensive language task is worse than a random guess, and the most probable cause for this is gradient explosion during the training. Likewise, the hate analysis task performance reduced almost to half when the learning rate was decreased further.

We had to keep a very low learning rate for the hate analysis task as we retrain the model from scratch. Similarly, we retrain the Offensive language model with a small learning rate since we train it from scratch and discard the pretrained model weights. Likewise, we keep a low learning rate for sentiment analysis. Nevertheless, the learning rate is higher compared to the other two tasks, as we are only training the top few layers and freezing the initial layers.

VIII. CONCLUSION

In this study, we addressed several aspects of the Natural Language Processing task of tweet classification using a pretrained transformer based model. We performed a detailed and thorough analysis of all three datasets for the tweet classification tasks: offensive language analysis, hate speech analysis, and sentiment analysis. To take it a step further, we did extensive analysis and performed regression testing to draw a conclusion that the dataset needed minimal preprocessing whilst working with a pretrained model. Likewise, we thoroughly considered different fine-tuning techniques and devised an optimal strategy for all three tasks. We further demonstrated that there is no one-size-fits-all approach for the three tasks; we had to cater each task with a different fine-tuning strategy. Next, we showcased that fine-tuning the models alone resulted in a decent accuracy for all three tasks; however, further accuracy improvements could only be achieved by tuning the hyperparameters. Later we presented learning rate and batch size as the two hyperparameters that we optimized for both models. After that, we trained both the models for all three classification tasks with different learning rates and batch sizes. We then critically evaluated and showcased the performance of our model based on different learning rates and batch sizes, and it was highlighted that both the models generalized well with a low learning rate and a small batch size. To conclude, we were able to outperform the TweetEval leader board by appropriately fine-tuning the pretrained language models and optimizing the hyperparameters.

REFERENCES

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[2] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[3] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *arXiv preprint arXiv:1708.00107*, 2017.

[4] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[8] Zhilin Yang, Ruslan Salakhutdinov, and William W Cohen. Transfer learning for sequence tagging with hierarchical recurrent networks. *arXiv preprint arXiv:1703.06345*, 2017.

[9] Rie Kubota Ando, Tong Zhang, and Peter Bartlett. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(11), 2005.

[10] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

[11] Shanshan Yu, Jindian Su, and Da Luo. Improving bert-based text classification with auxiliary sentence and domain knowledge. *IEEE Access*, 7:176600–176612, 2019.

[12] Kevin Roitero, VDMSM Cristian Bozzato, and G Serra. Twitter goes to the doctor: Detecting medical tweets using machine learning and bert. In *Proceedings of the International Workshop on Semantic Indexing and Information Retrieval for Health from heterogeneous content types and languages (SIIRH 2020)*, 2020.

[13] Yungao Xie, Hong Wen, and Qing Yang. Ternary sentiment classification of airline passengers' twitter text based on bert. In *Journal of Physics: Conference Series*, volume 1813, page 012017. IOP Publishing, 2021.

[14] Rabia Batool, Asad Masood Khattak, Jahanzeb Maqbool, and Sungyoung Lee. Precise tweet classification and sentiment analysis. In *2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS)*, pages 461–466. IEEE, 2013.

[15] Naman Deep Srivastava, Yashvardhan Sharma, et al. Combating online hate: A comparative study on identification of hate speech and offensive content in social media text. In *2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pages 47–52. IEEE, 2020.

[16] Francesco Barbieri, Jose Camacho-Collados, Leonardo Neves, and Luis Espinosa-Anke. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. *arXiv preprint arXiv:2010.12421*, 2020.

[17] Loshchilov Ilya, Hutter Frank, et al. Decoupled weight decay regularization. *Proceedings of ICLR*, 2019.