Pruning and Quantizing BERT on GLUE Benchmark

Based on the Paper: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

Athina Stewart
as1896@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Souvik Dey
sd5223@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

1 TASK DEFINITION, EVALUATION PROTOCOL, AND DATA

1.1 The Natural Language Problem

Words are problematic in the sense that many are ambiguous, polysemous (having many meanings), and synonymous. Most words in the English language have multiple meanings and when it comes to spoken English specifically, the problem is made worse by homophones and prosody (the pattern of stress and intonation). A word has no meaning until use in a particular context and the meaning of a word can change as a sentence develops. For example in the sentence: "I like the way that looks like the other one", "like" has two meanings. Natural language understanding requires an understanding of context and common sense reasoning.

BERT (Bidirectional Encoder Representations from Transformers) was developed to address limitations in existing natural language processing (NLP) models. Traditional NLP models, especially those based on recurrent neural networks (RNNs) and convolutional neural networks (CNNs), had difficulty capturing the complex contextual relationships and nuances in language. BERT aimed to overcome these limitations and improve the representation of language by introducing bidirectional context understanding[2].

Traditional models processed text in a unidirectional manner (either left-to-right or right-to-left), which limited their ability to capture contextual information effectively. BERT introduced a novel bidirectional context understanding, allowing the model to consider both left and right context for each word, leading to better contextual representation.

In this report, we will carry out the modifications pruning and quantization to the BERT model, and then fine tune on datasets found in the General Language Understanding Evaluation (GLUE) benchmark. We hypothesize that a quantized or pruned model will result in an increase in the model's validation accuracy. The GLUE benchmark consists of nine sentence or sentence-pair language understanding (NLU) tasks built on existing datasets that are of varying sizes, genres and degrees of difficulty. This benchmark is a commonly used collection of resources for training, testing and analysing NLU systems. Of the entire benchmark set, we will be carrying out our experiment on both sentence pair tasks (MRPC) and single sentence classification tasks (SST-2 and CoLA).

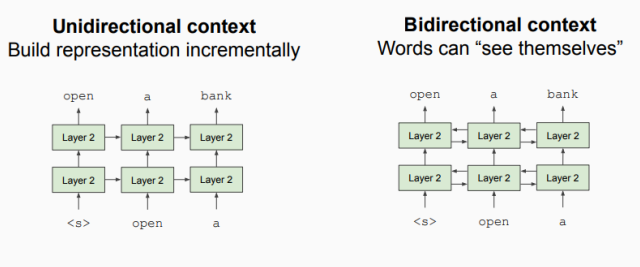

Figure 1: Unidirectional and Bidirectional Models[9]

1.2 Datasets

1.2.1 *MRPC.*

In 2005, the Microsoft research team created the Microsoft Research Paraphrase Corpus (MRPC). This corpus consists of 5,801 pairs of sentences. Each pair is

rated by a binary human-delivered judgement of whether the pair of sentences is similar in meaning.[3]. This dataset was born out of the need for a large enough corpus for training Statistical Machine Translation (SMT) models for paraphrase tasks. An initial set of 13M sentence pairs were extracted from the web over a two-year period. A classifier was then built, with feature classes such as string similarity and WordNet Lexical mapping, skewed more towards positive and plausible "near misses" as the performance of the classifier itself was not being evaluated. From the output, 5,801 pairs of sentences were randomly selected.

1.2.2 ***SST-2***.

The Stanford Sentiment Treebank (SST) is the first corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language (how the meaning of the sentence changes the more it is developed). It contains 11,855 single sentences extracted from movie reviews. In the case of the SST-2 dataset, labels are binary and either negative/somewhat negative or positive/somewhat positive. The full corpus has 5 labels: negative, somewhat negative, neutral, somewhat positive or positive. [7]

1.2.3 ***CoLA***.

In a 2018 study, artificial neural networks (ANNs) were trained to better understand grammar like human beings. These are trained to make judgments about the grammatical acceptability of sentences. This means distinguishing between well and poorly structured sentences The Corpus of Linguistic Acceptability (CoLA) consists of 10657 sentences from 23 linguistics publications, annotated for acceptability (grammatically) by their original authors. CoLA is the largest corpus of its kind. Grammatic violations included morphological, syntactic and semantic violations

2 THE BERT MODEL

2.1 Model Definition

BERT-Base, is a state-of-the-art, pre-trained natural language processing (NLP) model developed by Google, with 12 transformers and 110M parameters. Some key features and concepts related to BERT include: (1) Transformer Architecture: BERT is built upon the Transformer architecture, which is a neural network architecture. (2) Bidirectional Training: Unlike earlier models that read text in a unidirectional manner (from left to right or vice versa), BERT uses bidirectional training to understand the meaning of each word in a sentence. This bidirectional training is crucial for capturing context-dependent meanings and relationships between words. (3) Pre-training: BERT is pre-trained on large amounts of text data in an unsupervised manner. During pre-training, it learns to predict missing words in sentences, which helps it capture contextual information and semantic relationships within the text. (4) Transfer Learning: After pre-training, BERT can be fine-tuned on specific downstream tasks with a smaller dataset. This transfer learning approach allows BERT to adapt its learned representations to a more specific NLP tasks, such as text classification, named entity recognition and question answering. (5) Contextual Embeddings: BERT produces contextual embeddings for each word in a sentence, taking into account the surrounding context. This leads to more accurate and nuanced representations of words based on their context in a given sentence.

2.1.1 ***Pre-training***.

(1) Transformer Architecture: BERT utilizes the Transformer architecture, which includes an encoder for processing input text. Unlike directional models, BERT's encoder reads the entire sequence of words at once, making it bidirectional. (2) Training Strategies: BERT uses two training strategies: Masked Language Model (MLM) and Next Sentence Prediction (NSP). (3) Masked Language Model (MLM): Before inputting word sequences into BERT, 15% of the words are replaced with a [MASK] token. The model predicts the original values of these masked words based on the context provided by the non-masked words in the sequence. The BERT loss function considers only the prediction of masked values, enhancing context awareness. (4) Next Sentence Prediction (NSP): BERT receives pairs of sentences as input during training and learns to predict if the second sentence follows the first in the original document. The input is first tokenized. A [CLS] token is added at the beginning of the first sentence, and a [SEP] token is added at the end of each sentence. Sentence embeddings and positional embeddings are used to distinguish between sentences. A sentence embedding indicating belonging to sentence A or B is added to each token. A positional embedding is added to each token to indicate its position in the sequence. (5) The embedded word sequences, including the [MASK] tokens, are processed through

BERT's Transformer encoder. Word embeddings are vector representations of words, contained in BERT's that the model can now do mathematical operations. BERT has a Word Embedding Lookup Table consisting of 30,000 tokens and each token has 768 features in its embedding. There is a dictionary that maps the raw word with the hashed word id, that then corresponds to a row in the lookup table. (6) Training Process: BERT trains on both MLM and NSP simultaneously, minimizing the combined loss function of the two strategies.

2.1.2 *Why Fine-Tune?*

Fine-tuning a model refers to using the weights of an already trained network as the starting values for training a new network. BERT is pre-trained on a corpus consisting of the BooksCorpus (800M words) and English Wikipedia (2,500M words) [2]. For this assignment, the volume of data that BERT is pre-trained on is prohibitively large to replicate. Instead, we have decided to fine-tune the BERT base model on various language datasets, detailed in Section 1, and report the performance.

In general, fine-tuning also offers the advantages of adding words and sentences more specific to a task into the vocabulary, if only a small training dataset is available, fine-tuning BERT on this training set will offering better accuracies and using a pre-trained BERT model can also allow for quicker model development times.

BERT can be specific to NLP tasks such as question-answer, named entity recognition, textual entailment, sentiment analysis, paraphrase detection, text summarization, conversational AI, information retrieval, document classification etc.

2.2 Fine-Tuning the Model

In order to ensure that we were able to run the model before carrying out modifications, We ran BERT-base on the several of NLP tasks mentioned in Section 1.1 with the goal of reproducing the results mentioned in the paper. To fine-tune on GLUE, we use a batch size of 32, and fine-tuned for 3 epochs over the validation data for the GLUE tasks. We made use of TPUs for QNLI, SST-2, CoLA, STS-B, MRPC, RTE, MNLI-(m/mm) and QQP datasets. We used a maximum sequence length of 128 to save substantial system memory and a learning rate of 2e-5.We will use a subset of the results obtained here (specifically the results obtained from fine-tuning on the MRPC, SST-2 and CoLA tasks) to serve as the baseline to which the modifications made to the model will be compared against.

2.2.1 ***Preprocess the text***.

We constructed a Keras model for preprocessing text data using the BERT (Bidirectional Encoder Representations from Transformers) model. The function takes a list of string-valued features and an optional parameter for the sequence length of BERT inputs. It creates Keras Input layers for each feature, tokenizes the input text using the BERT preprocessing model from TensorFlow Hub, and optionally trims the tokenized segments to fit the specified sequence length. The function then packs the tokenized segments into a format suitable for input to a BERT model using a Keras layer. The resulting Keras Model, when called with a list or dictionary of string tensors corresponding to the input features, produces a dictionary of tensors ready for consumption by the BERT model. This preprocessing is then applied to all the inputs in the dataset.

2.2.2 ***Define the model***.

We defined our model for sentence or sentence pair classification by feeding the preprocessed inputs through the BERT encoder and putting a linear classifier on top (or other arrangement of layers as you prefer), and using dropout for regularization.

2.2.3 ***Train the model***.

To distribute training onto TPU workers, we created and compiled our main Keras model within the scope of the TPU distribution strategy.

Preprocessing, on the other hand, runs on the CPU of the worker host, not the TPUs, so the Keras model for preprocessing, as well as the training and validation datasets mapped with it, are built outside the distribution strategy scope. The call to fit function takes care of distributing the passed-in dataset to the model replicas.

Fine-tuning follows the optimizer set-up from BERT pre-training (as in Classify text with BERT): It uses the AdamW optimizer with a linear decay of a notional initial learning rate, prefixed with a linear warm-up phase over the first 10% of training steps. In line with the BERT paper, the initial learning rate is smaller for fine-tuning (best of 5e-5, 3e-5, 2e-5).

2.2.4 ***Proposed Modifications to the BERT fine-tuning model - Quantization and Pruning***.

Despite its recent success and wide adoption, fine-tuning BERT on a downstream task is prone to overfitting due to overparameterization; BERT-base has 110M parameters and BERT-large has 340M parameters. The overfitting worsens when the target downstream task has only a small number of training examples. [Devlin et al. (2019)] show that datasets with 10,000 or less training examples sometimes fail to fine-tune BERT [2].

To mitigate this critical issue, multiple studies attempt to regularize BERT by pruning parameters or using dropout to decrease its model complexity. In an extension to this report, we will explore network compression using two methods: quantization and pruning. Quantization involves reducing the precision of the model's weights & activations and pruning involves removing certain connections (weights) from the model, typically those with small magnitudes, resulting in a sparser model. Pruning yields simple and explainable results and it can be used along with other regularization methods.

3 EXPERIMENT

3.0.1 *Hypothesis*.

Regularizing BERT through pruning will mitigate overfitting in downstream tasks and increase the validation accuracy of the BERT model. The quantization of model weights and activations will result in a sparser yet effective model, improving generalization performance.

Our experiment will therefore be measuring the accuracy of the model based on three conditions: (i) without pruning and quantization (as explained in Section 2.2) (ii) with pruning (iii) with quantization. In this section, we will formally define the processes of quantization and pruning, two types of network compression.

3.0.2 *Quantization*.

Quantization is largely recognized as one of the most effective approaches to mitigate against the extreme memory requirements that deep neural network models demand. Instead of adopting 32-bit floating point format to represent weights, quantized model representations store weights using more compact formats such as integers or even binary numbers. The most common lower precision data types are:

- float16, accumulation data type float16
- bfloat16, accumulation data type float32
- int16, accumulation data type int32
- int8, accumulation data type int32 [4]

The two most common quantization cases are float32 -> float16 and float32 -> int8. Despite a possible degradation in predictive performance, quantization provides a potential solution to greatly reduce the model size and the energy consumption [5]. There is of course is a trade-off with the amount of quantization needed to result in an acceptable level model accuracy.

We have chosen to implement dynamic quantization to reduce the precision of our model. Dynamic quantization quantizes the model's weights during inference dynamically, meaning that it quantizes the weights on-the-fly while the model is running. The weights are quantized to a lower bit precision (e.g., from 32-bit floating point to 8-bit integer) during inference, but the quantization is not performed during training. This allows for the use of quantized models without retraining. The advantages of this quantization model is a simple and effective way to reduce the memory footprint and computational cost during inference. Disadvantages include that the process is performed on each execution of the model, which can add some overhead.

Other methods of quantization include post-training static quantization and quantization aware training. Post-training static quantization precomputes activation ranges at quantization-time by using observers to record activation values during forward passes on a calibration dataset. Quantization-aware training, similar to static quantization, computes activation ranges at training-time using "fake quantize" operators that not only record values but also simulate quantization errors, allowing the model to adapt. While dynamic quantization is convenient, static quantization and quantization-aware training offer more control and optimization opportunities at the cost of additional computation during calibration or training[4]

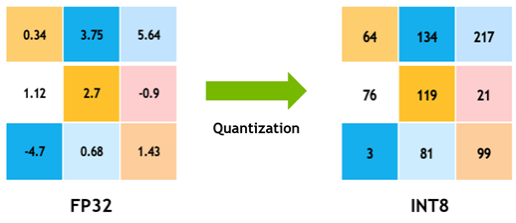

Figure 2: Image Depiction of Quantization[1]

3.0.3 *Pruning*.

Model pruning is used to reduce the size of a neural network by removing certain parameters (weights and connections) that are deemed less important. The goal of pruning is to create a more compact and efficient model without significantly sacrificing its performance. This can be particularly useful for deploying models on resource-constrained devices or reducing the computational cost of training and inference. Benefits of pruning include a more compact model with fewer parameters, smaller models that require less computational resources during both training and inference and potentially improving the generalization of the model.

In this experiment, we will be carrying out magnitude-based / weight-based pruning which means that parameters with small magnitudes will be removed. This is simple to implement and can lead to good results in terms of reducing the model size. However it can lead to a scattered distribution of zero-weights which may not significantly improve computational efficiency. Other pruning techniques include: (i) Neuron Pruning, where the least important nodes are removed from the network, though this can significantly alter the structure of the model and lead to lower accuracies than weight pruning (ii) Sparse Pruning, where the aim is to achieve and certain level of sparsity in the model without changing the overall architecture of the network. This can typically lead to significant reductions in model size but can also lead to an excessive performance drop (iii) Structured Pruning where entire neurons, layers, or filters are pruned instead of individual parameters. This preserves the otiginal structure of the layers and reduces the model size but it is not easy to implement without specialized knowledge of how the model works[8].

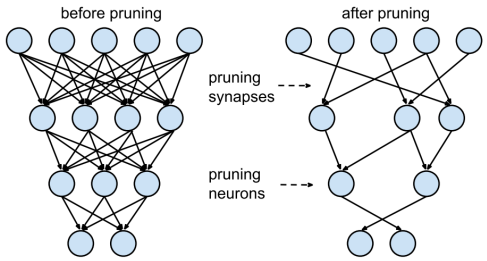

Figure 3: Image Depiction of Pruning[6]

3.1 Experiment Design

3.1.1 *Independent variables (Experimental Settings)*.

(1) Pruning Parameters
- Initial Sparsity: The proportion of weights that are initially pruned in the model.
- Final Sparsity: The target proportion of weights to be pruned by the end of the pruning process.
- Begin Step: The training step at which the pruning schedule begins.
- End Step: The training step at which the pruning schedule ends.

(2) Quantization Parameters
- Precision reduction amount for model weights and activations.

3.1.2 *Control variables (Biases and Modeling assumptions)*.

(1) BERT Architecture: The specific BERT architecture (BERT-base) to maintain consistency across experiments.
(2) Downstream Task and Dataset: The specific downstream task and dataset used for fine-tuning BERT to ensure a consistent evaluation environment. We will be a smaller subsection of tasks of those that we used to initially fine-tune the model. These are the MRPC, SST-2 and CoLA datasets.
(3) Evaluation Metrics: Standardized evaluation metrics for measuring model performance. This will be the validation loss, validation accuracy, training loss, training accuracy and model size.

3.1.3 *Dependent variables (Results Analysis)*.

(1) Sparsity level of the pruned model and the resulting model size after quantization.
(2) Performance metrics on the downstream task (accuracy, loss) after fine-tuning.

3.2 Methodology

The quantization process involves reducing the precision of model weights and activations, resulting in a more compact yet efficient representation. For this, we utilized PyTorch's dynamic quantization library, employing the `torch.quantization.quantize_dynamic` function. This method quantizes the model's weights and activations dynamically during runtime, reducing the precision to 8 bits (`torch.qint8`). Subsequently, we

measured the initial size of the BERT model in terms of memory consumption in the function `print_size_of_model(model)`. In the associated Google Colab notebook that illustrates the quantization process, the dataset is first downloaded and preprocessed. The SST-2 dataset was downloaded directly from the GLUE dataset present in the FBAI public files, Firebase AI's public datasets. The MRPC data was downloaded directly from the FBAI publicfiles' SentEval library, processed and saved into 'train' and 'test' tsv files. After preparing the dataset, the following steps were carried out:

(1) Model Configuration:
   - `configs.model_name_or_path`: Specifies the pre-trained BERT model to be used, set to "bert-base-uncased."
   - `configs.max_seq_length`: Sets the maximum length of an input sequence to 128 tokens.
   - `configs.task_name`: Specifies the GLUE task, set to "sst-2" for sentiment analysis.
   - `configs.processor`: Initializes the processor for the specified GLUE task.
   - `configs.output_mode`: Specifies the output mode for the GLUE task (e.g., classification).
   - `configs.label_list`: Retrieves the list of labels for the task.

(2) Device and Batch Configuration:
   - `configs.device`: Specifies the device to run the model on, set to "cpu" for CPU.
   - `configs.per_gpu_eval_batch_size`: Sets the batch size for evaluation on each GPU to 8.
   - `configs.n_gpu`: Specifies the number of GPUs used, set to 0 for CPU.
   - `configs.local_rank`: Specifies the local rank for distributed training, set to -1 for non-distributed training.
   - `configs.overwrite_cache`: Flags whether to overwrite cached features, set to False.

A random seed is set to 42 using the `set_seed` function for reproducibility.

When pruning the model, the BERT preprocessing model is created using TensorFlow Hub. This model tokenizes input sentences into word pieces and packs them into BERT inputs. The input sequence length is set to 128 tokens. The datasets are then loaded from TensorFlow Datasets (TFDS). The dataset is also cached and prefetched for better performance. The classifier model is built using a BERT encoder loaded from TensorFlow Hub. The output layer is a dense layer with a specified number of classes. The magnitude pruning technique is then applied to the dense layer using `tfmot.sparsity.keras.prune_low_magnitude`. The model is compiled with a sparse categorical cross-entropy loss and sparse categorical accuracy metric. It uses an AdamW optimizer with a learning rate schedule. The model is then trained using the training dataset and validated on the validation dataset. A pruning callback is employed during training to update pruning masks. Lastly, the BERT model is fine-tuned on the Cola and MRPC tasks.

4 EXPERIMENTAL RESULTS AND DISCUSSION

The experiment involved quantizing the fine-tuned BERT on model on the MRPC and SST-2 datasets, and additionally pruning the model and fine-tuning on the MRPC and CoLA satasets. Multiple challenges were encountered during the process, including issues with quantization in TensorFlow, the determination of optimal hyperparameters for pruning, and handling data preprocessing for the GLUE benchmark datasets.

One significant hurdle was faced in implementing quantization using TensorFlow. Due to the challenges encountered, we had to switch from TensorFlow to PyTorch for the quantization process. This switch allowed for more flexibility and ease of implementation. Identifying the right set of hyperparameters for the pruning schedule was another challenge. Determining the pruning parameters that enhanced the model's efficiency without compromising performance required extensive experimentation. In the end, the hyperparamters that we chose to manipulate included the initial sparsity, final sparsity, begin_step and end_step. Pruning layers without affecting the hidden attention head layers posed an additional challenge. We found that it was best to prune the final dense layer of the model instead of the encoder layer to prevent pruning these attention heads which could subsequently lead to a reduction in accuracy. Additionally, data preprocessing for GLUE benchmark datasets, such as MRPC and CoLA, was a complex task. Preparing the data appropriately to ensure compatibility with the BERT model architecture and GLUE benchmark requirements demanded careful attention and was a significant hurdle.

Metric		Pruning(%)	No Pruning(%)	Difference(%)
Training Loss	Epoch 1	67.1	68.1	-1.0
	Epoch 2	43.4	47.9	-4.5
	Epoch 3	30.2	35.2	-5.0
Training Accuracy	Epoch 1	62.5	63.4	-0.9
	Epoch 2	80.3	79.2	1.1
	Epoch 3	88.5	86.1	2.4
Validation Loss	Epoch 1	49.0	57.4	-8.4
	Epoch 2	44.7	48.3	-3.6
	Epoch 3	44.1	53.1	-9.0
Validation Accuracy	Epoch 1	77.9	75.5	2.3
	Epoch 2	81.0	79.7	1.3
	Epoch 3	82.6	80.2	2.3

Table 1: Performance Metrics of the Pruned BERT model, fine-tuned on the MRPC dataset

Metric		Pruning(%)	No Pruning(%)	Difference(%)
Training Loss	Epoch 1	53.2	56.9	-3.8
	Epoch 2	35.1	38.4	-3.3
	Epoch 3	24.5	27.1	-2.7
Training Accuracy	Epoch 1	76.4	72.4	4.0
	Epoch 2	87.0	85.6	1.4
	Epoch 3	91.8	90.7	1.0
Validation Loss	Epoch 1	44.5	49.2	-4.7
	Epoch 2	57.6	58.0	-0.5
	Epoch 3	61.9	81.1	-19.2
Validation Accuracy	Epoch 1	81.7	79.1	2.6
	Epoch 2	81.9	80.7	1.3
	Epoch 3	82.1	81.1	1.1

Table 2: Performance Metrics of the Pruned BERT model, fine-tuned on the CoLA dataset

Model	Accuracy (without quantization)	Accuracy (with quantization)	Model Size without quantization (MB)	Model Size with quantization (MB)
MRPC	80.21%	85.29%	438.000505	181.479765
SST-2	92.43%	91.40%	438.000505	181.479765

Table 3: Performance Metrics of the Quantized BERT model, fine-tuned on the MRPC & SST-2 dataset

Tables 1, 2 ad 3 present the findings from our quantization and pruning experiments. As stated earlier, we are comparing three scenarios: fine-tuning the BERT model without quantization or pruning, fine-tuning the model then pruning and fine-tuning the model then quantizing. Table 1 and Table 2 present the performance metrics of the pruned BERT model fine-tuned on the MRPC and CoLA datasets, respectively. Notably, the difference in training loss and accuracy between pruning and non-pruning scenarios is evident. Pruning resulted in a decrease in training loss and an increase in training accuracy across multiple epochs, showcasing the potential benefits of model compression.

Table 3 provides insights into the quantized BERT model's performance on the MRPC and SST-2 datasets. The quantized models demonstrated comparable accuracy to the respective non-quantized models, highlighting the effectiveness of quantization in reducing model size without sacrificing performance. We therefore believe that we were correct in our hypothesis that model quantization and pruning will lead to an increase in the model's accuracy. We were also able to observe a signifant reduction in the model's size with quantization (58.62%). A further extension to this project would be to explore the other pruning (neuron pruning, sparse pruning, structured pruning) and quantization (post-training static quantization, quantization-aware training) techniques mentioned in this paper, in order to determine which would be most effective for the BERT model. We could also compare the effectiveness of these network

compression strategies on other language models to provide a broader perspective on the generalizability of the techniques on different architectures.

Included in this results section are also our preliminary results from fine-tuning the BERT model pre-quantization and pruning.

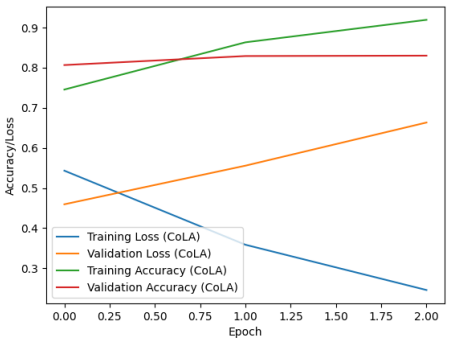

Figure 4: Fine-Tuning the BERT model on CoLA dataset without quantization or pruning

The CoLA dataset shows considerably higher validation accuracy results that the original paper. There is also a gradual increase in validation loss signaling loss in model's confidence as the epoch progresses. The trends in accuracy and loss seen here remain consistent with and without pruning.

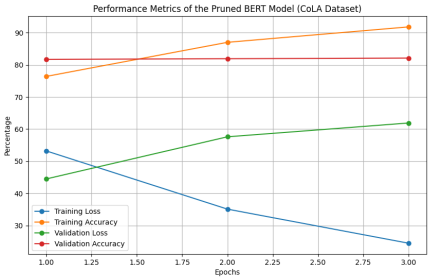

Figure 5: Fine-Tuning the BERT model on CoLA dataset with pruning

The MRPC dataset shows a decrease in validation loss but then after the first epoch, it slightly increases. Correspondingly, the validation accuracy decreases after the first epoch as well, signaling misclassifications. With pruning, there is a slight change in that the training and validation accuracy is nearly the same at the second

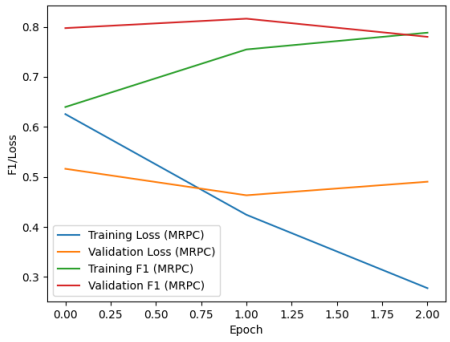

Figure 6: Fine-Tuning the BERT model on MRPC dataset without quantization or pruning

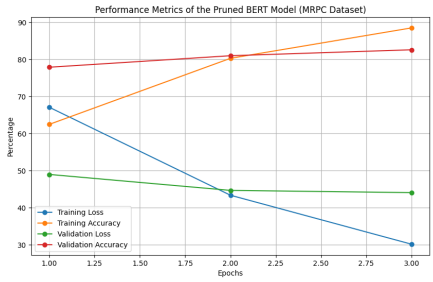

Figure 7: Fine-Tuning the BERT model on MRPC dataset with pruning

epoch and the validation loss is more signifcantly less than the training loss at the third epoch

REFERENCES

[1] NVIDIA Developer. [n. d.]. https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[3] William B. Dolan and Chris Brockett. 2005. Automatically Constructing a Corpus of Sentential Paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*. https://aclanthology.org/I05-5002

[4] Hugging Face. [n. d.]. https://huggingface.co/docs/optimum/concept_guides/quantization

[5] Yunhui Guo. 2018. A Survey on Methods and Theories of Quantized Neural Networks. *CoRR* abs/1808.04752 (2018). arXiv:1808.04752 http://arxiv.org/abs/1808.04752

[6] Souvik Paul. [n. d.]. https://medium.com/@souvik.paul01/pruning-in-deep-learning-models-1067a19acd89

[7] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard (Eds.). Association for Computational Linguistics, Seattle, Washington, USA, 1631–1642. https://aclanthology.org/D13-1170

[8] Yifei Wang. [n. d.]. https://www.red-gate.com/simple-talk/homepage/decoding-efficiency-in-deep-learning-a-guide-to-neural-network-pruning-in-big-data-mining/#:~:text=Comparison%20of%20Different%20Pruning%20Techniques,on%20model%20performance%20and%20structure

[9] Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. arXiv:1704.05426 [cs.CL]