DS-GA 1012: Natural Language Understanding and Computational Semantics (Spring 2025) Homework 2

Allen George Ajith (aa12938)

March 17, 2025

Problem 1b:

The input to BERT is structured as a dictionary with three keys: `input_ids`, `token_type_ids`, and `attention_mask`

1. `input_ids`: These are tokenized representations of the input text. BERT uses WordPiece tokenization to break input text into tokens, which are then mapped to their corresponding unique numerical IDs from BERT's vocabulary.

2. `token_type_ids`: They indicate the distinction between different segments in the input so they are also called segment IDs. In tasks involving sentence pairs, `token_type_ids` help the model differentiate between the first and second sentences. For example in a task involving two sentences, the tokens corresponding to the first sentence are assigned a segment ID of 0, and those corresponding to the second sentence are assigned a segment ID of 1.

3. `attention_mask`: This tells the model which tokens should be attended to and which should not. It differentiates between actual tokens and padding tokens added to maintain uniform input length across batches. Tokens with a mask value of 1 are attended to by the model, while tokens with a mask value of 0 are ignored.

The model internally also adds position-ids that indicate the position of each token in the input sequence. BERT uses these to incorporate positional information to understand the order of tokens.

Problem 1c

For each task in the GLUE benchmark, the best fine-tuning hyperparameters were selected from the following lists and trained for 4 epochs:

- **Batch Sizes**: `8, 16, 32, 64, 128`
- **Learning Rates**: `3e-4, 1e-4, 5e-5, 3e-5`

They probably tried all the different combinations of these and did a grid search (similar to how we have done in this assignment) to maximize a particular evaluation metric of the validation set like accuracy (in our case).

Problem 3a: Train Results

	Validation Accuracy	**Learning Rate**	**Batch Size**
Without BitFit	88.92	0.0003	32
With BitFit	63.44	0.0003	8

Standard hyperparameter tuning without Bitfit did much better than with Bitfit. This could be because for this particular dataset and model($\text{BERT}_{\text{tiny}}$) we need to fine-tune with all the parameters rather than only the bias ones to acheive best performance. Also we got the best accuracies with a smaller batch size in BitFit vs. without.

Problem 3b: Test Results

	# Trainable Parameters	**Test Accuracy**
Without BitFit	4,386,178	86.512
With BitFit	3,074	63.648

We achieve close to 64% accuracy even after only fine-tuning 3074 parameters of the tiny model using BitFit (0.07% of the total parameters). Fine-tuning only the bias parameters could be a good approach that may work better for bigger models which are generally over-parameterized. The $\text{BERT}_{\text{tiny}}$ model has much fewer parameters than the base model so not a lot of it's parameters are redundant which is why finetuning using all the parameters gives better performance.

		%Param	**QNLI**	**SST-2**
	Train size		105k	67k
(V)	Full-FT†	100%	**93.5**	**94.1**
(V)	Full-FT	100%	91.7±0.1	93.4±0.2
(V)	Diff-Prune†	0.5%	**93.4**	**94.2**
(V)	BitFit	0.08%	91.4±2.4	93.2±0.4
(T)	Full-FT‡	100%	91.1	94.1
(T)	Full-FT†	100%	**93.4**	**94.9**
(T)	Adapters‡	3.6%	90.7	94.0
(T)	Diff-Prune†	0.5%	**93.3**	94.1
(T)	BitFit	0.08%	92.0	**94.2**

Figure 1: Comparison of different fine-tuning methods from the BitFit paper

The data set used by Zaken et al. for a similar task is SST-2 (Stanford Sentiment Treebank) in which Full-FT achieves an accuracy of 94.1% in the validation set (V) and 94. 9% in the test set (T). BitFit demonstrates strong performance despite using only 0.08% of the parameters and achieves

an accuracy of 93.2% on validation set and 94.2% for test set. These are results for the $\text{BERT}_{\text{large}}$ model which may contain redundant parameters which is why BitFit achieves accuracies comparable to full finetuning.