NLU HW1

Beisong Liu
UID: bl1986

March 16, 2025

Question 1

Part (b)

1. **input_ids:**
   These are the numerical representations of tokens. Each token from the input text is mapped to a unique integer based on the tokenizer's vocabulary. These IDs are used by BERT to look up embeddings from the embedding matrix, effectively conveying the textual content of the input.

2. **token_type_ids:**
   These indicate which segment or sentence a token belongs to. For tasks that involve sentence pairs (such as question answering or natural language inference), tokens from the first sentence are typically assigned a value of 0 and tokens from the second sentence a value of 1. This helps BERT understand sentence boundaries and the relationship between different segments.

3. **attention_mask:**
   This is a binary mask that distinguishes between actual tokens and padding tokens. A value of 1 indicates that the corresponding token is valid and should be attended to, while a value of 0 indicates padding. This mask ensures that the model's attention mechanism focuses only on the meaningful parts of the input.

Part (c)

The $\text{BERT}_{\text{tiny}}$ model was fine-tuned for exactly 4 epochs using each combination of the following hyperparameters:

- **Batch size:** 8, 16, 32, 64, 128
- **Learning rates:** 3e-4, 1e-4, 5e-5, 3e-5

For each GLUE task, a grid search was performed over these (batch size, learning rate) pairs to determine the best fine-tuning configuration.

Question 3

Part (a)

	Validation Accuracy	Learning Rate	Batch Size
Without BitFit	0.888	3e-4	64
With BitFit	0.624	3e-4	16

Table 1: Comparison of validation accuracy, learning rate, and batch size with and without BitFit.

Part (b)

	# Trainable Parameters	Test Accuracy
Without BitFit	4386178	0.876
With BitFit	3074	0.623

Table 2: Comparison of the number of trainable parameters and test accuracy with and without BitFit.

Question:Finally, please comment on your results. How do they compare to the results reported by Zaken et al. (2020)? What does this say about BitFit and its applicability to other pre-trained Transformers?

Performance Comparison

My results show a significant drop in accuracy (from 0.876 to 0.623) when using BitFit. However, in the results reported by Zaken et al. (2020), by using BitFit, they can retain much of a model's performance while drastically reducing the number of trainable parameters. I guess the reason is that:

- The base model I used is already quite small (BERT-Tiny).
- Freezing most parameters further reduces its flexibility to learn the classification task.

Implications for Other Pre-trained Transformers

- **Benefit of Fewer Trainable Parameters:**
  BitFit drastically reduces the memory and computational cost of fine-tuning, making it appealing for large models or constrained deployment environments.
- **Performance Trade-Off:**
  If the underlying model is very small (like BERT-Tiny), freezing all but the bias terms can lead to substantial accuracy drops. Larger models tend to tolerate BitFit better and often show results closer to full fine-tuning.
- **Task Dependency:**
  For tasks with simpler decision boundaries or abundant labeled data, BitFit can still work surprisingly well. However, tasks that require more nuanced understanding may see larger drops in performance.