Name: Rehaan Shaikh

Student no : C240824638

**Efficient Fine-Tuning of BERT for Word-in Context Classification**

**1) Introduction:**

Understanding word meaning variations across different contexts is a critical challenge in natural language processing (NLP). The Word-in-Context (WiC) task is designed to assess whether a target/specific word maintains the same meaning in two different sentence contexts. This report details the implementation of Fine-tuning BERT base to address this problem, emphasizing efficient fine-tuning strategies. To further improve the model, instead of full BERT fine-tuning, we explore Low-Rank Adaptation (LoRA) techniques to enhance training efficiency and model performance.

**2) Dataset Description:**

The Word-in-Context (WiC) dataset is designed to evaluate a model’s ability to determine whether a given word maintains the same meaning across two different sentence contexts. The dataset consists of three splits: train, validation, and test, with the training set containing 5,428 rows. Each data instance consists of a **pair of sentences**, a **target word**, and a **binary label** indicating semantic similarity. Unlike traditional word sense disambiguation, this task requires models to learn subtle semantic differences purely from context.

**Example:**

* **Sentence 1**: "The bat flew out of the cave."
* **Sentence 2**: "He swung the bat and hit a six."
* **Label**: 0 (Different meanings)

This task is particularly challenging because the meaning of many words depends on contextual clues rather than fixed dictionary definitions.

**3) Model Choice:**

We chose **BERT base** for this task because it generates **contextualized word embeddings**, meaning that the representation of a word is influenced by the surrounding text or the context behind it. Unlike traditional static embeddings such as **Word2Vec** or **GloVe**, which assign a fixed vector to each word regardless of context, BERT dynamically adjusts word representations based on their usage in a sentence. This makes BERT well-suited for the **Word-in-Context (WiC) task**, where distinguishing subtle shifts in meaning is crucial. Additionally, BERT’s **self-attention mechanism** enables it to capture long-range dependencies in text, allowing for a deeper understanding of semantic nuances necessary for word sense disambiguation.

However, it is important to note that Fine-tuning Large language models such as BERT can be computationally expensive and time consuming. Given the computational constrains, we have also

Implemented LoRA, a parameter-efficient fine-tuning method to address these challenges.

**3) Dataset Preprocessing:**

Each sentence is tokenized using bert-base-uncased, with **offset mapping** to correctly align target word positions with tokenized representations. A PyTorch Dataset class is implemented to automate preprocessing.

We preprocess the **WiC dataset** to ensure that it is properly formatted for training our BERT-based model. The pre-processed dataset consists of 3 key components:

* **Sentence Pairs**: Two sentences where the target word appears.
* **Target Word Indices**: The start and end positions of the target word in both sentences.
* **Labels**: : Binary classification label indicating whether the word meaning is the same (1) or different (0).
* **Steps in Preprocessing:**

To prepare the dataset, we implement a custom **WiCDataset** class using PyTorch's Dataset module. This class automates tokenization, target word alignment, and tensor conversion.

1. **Tokenization**: Each sentence is tokenized using **BERT’s tokenizer (bert-base-uncased)**, ensuring subword tokenization consistency.
2. **Offset Mapping**: Since BERT tokenizes words into subwords, we use **offset mapping** to correctly align the target word’s character positions with the tokenized representation.
3. **Target Word Masking**: We generate a **binary mask** that highlights the tokens corresponding to the target word. This is achieved using the find\_target\_token\_indices function, which assigns a value of **1** to relevant tokens and **0** to others.
4. **Padding and Truncation**: Sentences are **padded to a fixed length (128 tokens)** and truncated if necessary to ensure consistent input size.
5. **Batch Dimension Removal**: Since the tokenizer returns outputs in batch format, we remove extra dimensions before returning the processed data.
6. **Final Output Format**: Each processed sample is returned as a dictionary containing **input IDs, attention masks, target masks, and labels**, ready for training.

This preprocessing ensures that our model correctly learns contextual word representations while maintaining alignment between raw text and tokenized sequences.

**3) Model Implementation:**

We implemented two models for the **Word-in-Context (WiC) classification task**:

1. **Base Model** – A standard BERT-based classifier fully fine-tuned on the dataset.
2. **Improved Model** – A LoRA-optimized version of the base model, enhancing efficiency and accuracy.

The details of both these models are given below:

1. **Base Model:**

The base model is a BERT-based classifier fine-tuned for the WiC task. It extracts contextual embeddings from both sentences, computes the target word representation, and combines these embeddings using concatenation. A simple fully connected (FC) layer with ReLU activation classifies whether the word has the same meaning in both contexts.

* **Model Architecture:**
* **Encoder:** BERT-base-uncased (pre-trained BERT)
* **Embedding Size:** 768 (BERT hidden size)
* **Feature Combination:**
  + **Concatenation of target word embeddings** from both sentences
  + **Input to Classifier:** 2 × 768 = 1536
* **Classification Head:**
  + **FC Layer:** Linear(1536, 2)
  + **Activation:** ReLU()
* **Training Configuration:**
* **Optimizer:** AdamW (learning rate = 2e-5, weight decay = 0.01)
* **Batch Size:** 32
* **Loss Function:** CrossEntropyLoss (for binary classification)
* **Learning Rate Scheduler:** Linear decay with warm-up steps (10% of total steps)
* **Number of Epochs:** 5
* **Gradient Clipping:** 1.0 (to prevent exploding gradients)
* **Regularization:** No explicit dropout applied in the classifier head
* **Limitations of Base Model:**
* Treats word meaning comparison **only via concatenation**, lacking interaction-based features.
* Full **BERT fine-tuning is computationally expensive** as all parameters are updated.
* No **additional feature engineering** beyond target word embeddings.

1. **Improved Model (LoRA Fine-Tuned BERT):**

The improved model enhances the **base model** by integrating **Low-Rank Adaptation (LoRA)**, a parameter-efficient fine-tuning technique that significantly reduces computational cost while maintaining high accuracy. Instead of updating all of BERT’s parameters, **LoRA injects trainable low-rank matrices into the attention layers**, keeping most of the model frozen. This allows for **faster convergence, reduced memory usage, and better generalization**.

* **Model Architecture:**
* **Input to Classifier:** 4 × 768 = 3072 (concatenated + interaction-based embeddings)
* **FC Layer 1:** Linear(3072, 512)
* **Activation:** ReLU() (To introduce non linearity)
* **Dropout:** 0.4 (for regularization)
* **FC Layer 2:** Linear(512, 2)
* **Training Configuration:**
* **Optimizer:** AdamW (learning rate = 2e-5, weight decay = 0.01)
* **Batch Size:** 32
* **Loss Function:** CrossEntropyLoss (for binary classification)
* **Learning Rate Scheduler:** Linear decay with warm-up steps (10% of total steps)
* **Number of Epochs:** 5
* **Gradient Clipping:** 1.0 (to prevent exploding gradients)
* **LoRA Parameters & Model Configurations:**

| **Config** | **r** | | **LoRA Alpha** | **Target Modules** | **Dropout** |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 16 | | Query, Value | 0.1 |
| 2 | 12 | 15 | | Query, Value | 0.1 |
| 3 | 14 | 14 | | Query, Value | 0.1 |

* **Key Improvements Over Base Model:**
* **LoRA-Based Fine-Tuning:**
  + Only query and value attention layers are fine-tuned, keeping other BERT parameters frozen.
  + Reduces trainable parameters, making training more efficient.
* **Enhanced Feature Combination:**
  + Concatenation of target word embeddings from both sentences.
  + Element-wise multiplication to capture alignment.
  + Absolute difference to highlight semantic distinctions.
* **Deeper Classifier Head:**
  + Uses two fully connected layers instead of one.
  + Dropout (0.4) added for better regularization and to prevent overfitting.
* **Advantages of LoRA Model:**
* **Faster Convergence** – Since only a subset of parameters is updated, training is **more efficient** while maintaining accuracy.
* **Reduced Overfitting** – Freezing most of BERT’s layers ensures that fine-tuning does not distort pre-trained representations.
* **Improved Accuracy** – Interaction-based features (multiplication + difference) allow for better distinction between word meanings.
* **Best Performing Configuration**

Based on validation accuracy, **Config 3 (r=14, LoRA Alpha=14)** performed the best, achieving the highest **test accuracy(63.57%)** while keeping computational cost low.

**4) Results and Analysis:**

We compare the performance of the **base** and **improved** models based on training loss, validation accuracy, and test accuracy. The improved model shows significant enhancements over the baseline in terms of both convergence speed and classification accuracy.

| **Model** | **Validation Accuracy** | **Test Accuracy** |
| --- | --- | --- |
| Base Model | 57.68% | 55.07% |
| Improved Model | 64.73% | 63.57% |

**5.2 Training Loss Analysis**

Training loss shows steady convergence:

|  |  |  |
| --- | --- | --- |
| **Epoch** | **Base Model Loss** | **Improved Model Loss**  (Config 3) |
| 1 | 0.6705 | 0.6820 |
| 2 | 0.5434 | 0.6244 |
| 3 | 0.4069 | 0.5825 |
| 4 | **-** | 0.5636 |
| 5 | **-** | 0.5527 |

* The gradual decrease in loss for both these models loss indicates stable learning with no signs of overfitting.

**6) Conclusion:**

Compared to full fine-tuning, LoRA reduces computational requirement while maintaining strong performance. The best-performing model achieves **63.57% test accuracy**, outperforming baseline approaches.

* **Potential Reasons Why LoRA Outperformed Full Fine-Tuning/Base Model:**
* **Avoids Overfitting –** Full fine-tuning can overfit on a moderate-sized dataset, whereas LoRA prevents this by updating only attention layers.
* **Retains Pre-Trained Knowledge –** Freezing most of BERT allows it to preserve learned linguistic representations, leading to better generalization.
* **Efficient Parameter Updates** – LoRA fine-tunes only key layers, focusing learning on contextual nuances without excessive modifications.
* **Potential Improvements:**
* **Integrate Contrastive Learning** – Using contrastive learning can help the model **differentiate subtle word meaning variations** more effectively.
* **Leverage External Knowledge** – Incorporating **linguistic resources** like **WordNet** can improve polysemy handling.
* **Experiment with Larger Models** – Trying **RoBERTa or DeBERTa** could provide better contextual word representations and improved classification accuracy.
* This study demonstrates that parameter-efficient fine-tuning techniques like LoRA provide a promising direction for scaling NLP models while maintaining performance.