Technical Report: BERT Fine-Tuning for Text Classification

Methodology Explanation

In this project, I have fine-tuned a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model for a text classification task. The model was trained on a labelled dataset and evaluated based on accuracy, F1-score, and confusion matrix metrics.

The fine-tuning process involved loading a pre-trained BERT model (bert-base-uncased) and adapting it to our specific classification task. We used the Hugging Face Transformers library for model loading and fine-tuning, alongside the PyTorch framework for training.

Architecture Decisions

1. **Pre-trained Model**: I have used BERT (bert-base-uncased) as the base model. The model was pre-trained on a large corpus of English text(Ag-news dataset).
2. **Data Preprocessing**: The text data was tokenized using the BERT tokenizer, which splits the text into subword tokens and converts them into embeddings. We also applied basic text cleaning functions, such as removing special characters and extra spaces.
3. **Training Strategy**: We employed transfer learning, leveraging the pre-trained weights of BERT and adapting them to the specific task at hand. The final layer of BERT was replaced with a fully connected layer for classification.
4. **Evaluation Metrics**: Accuracy, F1-score, and confusion matrices were used to evaluate the model's performance.

Hyperparameter Choices and Reasoning

Several hyperparameters were tuned to ensure optimal performance during fine-tuning:

1. **Learning Rate**: Set to 2e-5. This relatively low learning rate was chosen to ensure fine-tuning does not disrupt the pre-trained weights significantly. A lower learning rate helps achieve more stable convergence.
2. **Batch Size**: A batch size of 64 for training and 32 for evaluation was used. Larger batch sizes help in processing more data simultaneously but require more memory. A batch size of 64 was chosen for efficient memory usage. I have also tried with different batch size. Because of limited resources and time taken in training. I have finalized batch size 64.

3. **Number of Epochs**: Set to 4. Since BERT is a large model, training for many epochs may lead to overfitting. Four epochs were found to strike a good balance between training time and model performance.
4. **Weight Decay**: Set to 0.01 to avoid overfitting. Weight decay is a regularization technique that penalizes large weights, encouraging the model to learn more generalizable features.
5. **Gradient Accumulation**: Set to 2 steps. This helps in simulating a larger batch size by accumulating gradients over multiple mini batches, effectively reducing memory usage while training.

Performance Analysis

The model achieved strong performance in terms of classification accuracy and F1-score. The evaluation was performed on a test set, and key metrics are summarized below:

- **Accuracy**: ~94%
- **F1-score**: ~0.91
- **Training loss** : 0.012415256351232529

For more information Please visit: Output Report

These results suggest that fine-tuning a pre-trained BERT model can be highly effective for text classification tasks, delivering state-of-the-art results in a short amount of time.

Challenges and Solutions

1. **Overfitting**: One of the challenges faced during training was overfitting, especially with a limited dataset. This was mitigated by:
    - Using **early stopping** and **patience** parameters, ensuring that training stops once performance plateaus.
    - Applying **weight decay** to regularize the model.
2. **Memory Constraints**: BERT's large size required substantial memory during training.

    Solution: To address this, we utilized **gradient accumulation** to simulate larger batch sizes without overloading the GPU memory.

Potential Improvements

1. **Hyperparameter Optimization**: While we chose standard values for key hyperparameters, a more thorough search using methods like grid search or random search could help find even better values, improving both training speed and model performance.

2. **Fine-tuning Longer or Using Larger Models**: Extending the number of epochs or experimenting with a larger BERT model (e.g., bert-large-uncased) could improve accuracy, although this would come at the cost of more computational resources and training time.
3. **Model Variants:** Experimenting with more advanced BERT models like RoBERTa or Distil BERT might offer faster training times or better accuracy.