Fine-tuning BERT Model for Sentiment Analysis

Overview

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based pre-trained model developed by Google. Due to its large parameter size, training BERT from scratch on small datasets leads to overfitting. Instead, fine-tuning a pre-trained BERT model allows us to transfer learning from a large corpus to a smaller sentiment classification task. This project fine-tunes BERT to classify sentences as POSITIVE or NEGATIVE.

Type of Fine-Tuning:

This is a **partial fine-tuning** strategy where:

- **BERT's pre-trained layers are frozen** (i.e., param.requires_grad = False)
- **Custom layers ("classifier head") are added on top** (dense layers + softmax)
- Only the **added layers are trained** (the BERT backbone is used for feature extraction)

Preparing the Dataset

The dataset contains two columns: 'sentence' (text) and 'label' (0 for negative, 1 for positive). The data is loaded and preprocessed using Pandas.

Data Splitting

The dataset is split into training (70%), validation (15%), and test (15%) sets using stratified sampling to preserve class distribution.

Loading Pre-trained BERT and Tokenizer

We load 'bert-base-uncased' from HuggingFace Transformers. The tokenizer converts text to input IDs and attention masks.

Choosing Padding Length

To avoid excessive padding or truncation, we plot sentence lengths and choose an average value (e.g., 17 tokens) as the maximum sequence length.

Tokenization

Using `batch_encode_plus`, text is tokenized into sequences with padding and truncation. The resulting input IDs and masks are converted to tensors.

Model Architecture

We freeze the BERT model parameters and add a custom classifier: a dropout layer, ReLU activation, two dense layers,and a LogSoftmax output.

Optimization

AdamW optimizer is used with a learning rate of 1e-5. Class weights are applied to the CrossEntropyLoss to manage data imbalance.

Training the Model

A custom training loop iterates through data in batches, computes predictions and loss, performs backpropagation, and updates model weights.

Validation

An evaluation function disables dropout and computes model performance on the validation set by calculating loss and collecting predictions.

Testing and Evaluation

After training, predictions on the test set are obtained and evaluated using sklearn's `classification_report` to assess precision, recall, and F1-score.

Conclusion

This project demonstrates effective fine-tuning of BERT for sentiment classification. It highlights the importance of tokenization, balanced data handling, and careful model design to adapt a powerful pre-trained model to a specific task.