Lab-06: AWS Machine Learning University Module 2 Lab 05 Fine Tuning Bert

1. Learning Insights

1.1 Transfer Learning with Transformers

Working through the BERT fine-tuning lab clarified how powerful transfer learning is in NLP. Instead of training a language model from scratch, I loaded bert-base-uncased via Hugging Face's Transformers library and repurposed its deep bidirectional encodings for sentiment classification. Fine-tuning involved only adding and training a lightweight classification head on top of BERT's pooled output, yet the pretrained weights carried rich contextual knowledge from 3+ billion words of Wikipedia and BookCorpus.

1.2 Tokenization and Input Formatting

I gained deep appreciation for subword tokenization (WordPiece). Converting raw text into input IDs, attention masks, and token-type IDs taught me how BERT handles out-of-vocabulary words by splitting them into known subwords (e.g., "unbelievable" → "un", "##believable"). I learned to pad sequences to a fixed max_length=128 and to batch examples efficiently, ensuring that the model sees uniform tensor shapes.

1.3 Optimization & Learning-Rate Scheduling

The lab guided me to use AdamW optimizer with a low learning rate (2×10^{-5}) and a linear warmup schedule over the first 10% of training steps. Observing training curves showed how warmup prevents large initial weight updates that could destroy pretrained features. I saw that even a small SOTA model requires careful tuning of optimizer hyperparameters to converge smoothly.

1.4 Evaluation Metrics

Beyond accuracy, I computed precision, recall, and F1-score on the test split. Fine-tuned BERT achieved ~92% accuracy and an F1-score of ~0.91 on a balanced reviews dataset. Tracking these metrics reinforced that accuracy alone can mask class imbalances and that F1-score is often a better indicator of real-world performance for binary tasks.

2. Challenges and Struggles

2.1 Resource Constraints & Batch Size

Fine-tuning BERT on a consumer-grade GPU (8 GB) quickly ran into Out-of-Memory errors

when using batch sizes > 16. I experimented with gradient accumulation (accumulating gradients over 4 steps with batch size 8 to emulate batch size 32) which allowed stable training without OOM.

2.2 Tokenizer Quirks

When I first created the dataset, I passed raw reviews directly to BERT's tokenizer.encode_plus. I forgot to set truncation=True, so some reviews exceeded the model's max_length and caused index errors. Adding truncation=True, padding='max_length' fixed it.

2.3 Overfitting & Early Stopping

After 4 epochs, training loss continued to drop but validation loss began to rise—a clear overfitting signal. Implementing early stopping (patience = 2 on validation loss) and saving the best checkpoint prevented performance degradation.

2.4 Understanding Model Outputs

Initially I treated BERT's logits as probabilities; only after applying softmax did the outputs sum to 1.0 and become interpretable. Plotting the softmax outputs for a few examples helped me debug why some confidently wrong predictions were being made.

3. Personal Growth

3.1 Confidence with Hugging Face APIs

Before this lab, I had read about Transformers but never used them hands-on. Now I feel comfortable instantiating BertTokenizer, BertForSequenceClassification, and defining a PyTorch Dataset that feeds input_ids, attention_mask, and labels into the model.

3.2 Pipelines for Reproducibility

I refactored training and evaluation steps into reusable functions—train_epoch(), eval_epoch()—and wrote a simple CLI argument parser so I can rerun experiments with different hyperparameters without rewriting code. This "production mindset" was new and immensely valuable.

3.3 Broader Perspective on NLP

Fine-tuning BERT drove home that modern NLP workflows revolve around large pretrained models and domain-specific adaptation. I now see how these techniques extend beyond sentiment analysis to question answering, named-entity recognition, and more—opening many avenues for future projects.

4. Critical Reflection

4.1 What I Would Do Differently

- **Mixed-Precision Training**: I would integrate NVIDIA's AMP (automatic mixed precision) to reduce memory usage and speed up training.
- **Dynamic Padding**: Instead of padding all sequences to the same length, I'd sort examples by length in each batch to minimize wasted tokens.
- **Cross-Validation**: I'd implement k-fold cross-validation to better estimate generalization performance on small datasets.

4.2 New Questions & Exploration

- **Model Size vs. Speed**: How much accuracy do I lose if I use a smaller DistilBERT or ALBERT?
- **Domain Adaptation**: Could further pretraining on in-domain text (e.g. Mars rover logs) improve performance?
- **Explainability**: What methods (e.g., attention-based heatmaps) can I apply to interpret which words most influence BERT's decisions?

4.3 Fit into the Broader ML Landscape

This lab cemented my understanding that transfer learning with large-scale pretrained models is the current frontier in NLP. It complements earlier sequence-based RNN work and points the way toward fully self-supervised, task-agnostic representations (the path toward GPT/LLM-style training).