Fine-Tuning BERT

Welcome to the BERT Project with Hugging Face! This document walks you through every step, from fine-tuning and debugging to comprehensive evaluation and a creative application. so you can reproduce and understand the full workflow.

Objective

You will learn how to:

- Fine-tune a pre-trained BERT model on a real dataset
- Identify and resolve common issues during training and inference
- Measure performance with structured metrics (accuracy, F1, exact match, MSE, log loss)
- Apply BERT creatively to a novel NLP task (named entity recognition)

Part 1: Fine-Tuning BERT

1. **Choose your task**
   We use IMDb sentiment analysis (binary classification).
2. **Environment setup**
   - Install Python 3.8+
   - Create and activate a virtual environment
     • Windows (PowerShell):
     • Run python -m venv venv
     • Then .\venv\Scripts\Activate.ps1
     • macOS/Linux:
     • Run python3 -m venv venv
     • Then source venv/bin/activate
   - Install libraries:
     • pip install transformers datasets torch scikit-learn evaluate scipy seqeval
3. **Data preparation**
   - Load the IMDb dataset via datasets.load_dataset('imdb')

- Tokenize text using BertTokenizerFast.from_pretrained('bert-base-uncased')
- Pad/truncate to uniform length, rename label column to “labels,” and cast to PyTorch tensors

4. **Model initialization and training**
    - Load BertForSequenceClassification with num_labels=2
    - Configure Trainer with TrainingArguments (output directory, batch size, learning rate, epochs, evaluation strategy)
    - Call trainer.train() and monitor training logs for loss and accuracy
    - Save the fine-tuned model in ./results

Part 2: Debugging Issues

1. **Detecting problems**
    - Review trainer.evaluate() metrics after each epoch
    - Watch for overfitting (training loss ≪ validation loss) or underfitting (high training loss)
    - Note any out-of-memory errors or excessively long epochs
2. **Common fixes**
    - **Learning rate**: reduce by a factor (e.g., from 5e-5 to 2e-5)
    - **Batch size**: lower to fit GPU memory, or use gradient accumulation
    - **Regularization**: add weight decay, dropout, or early stopping
    - **Data augmentation**: oversample minority cases or apply simple text augmentations
3. **Verification**
    - Re-train with adjusted hyperparameters
    - Compare pre- and post-debug metrics to confirm improvements

---

Part 3: Evaluating the Model

1. **Generate predictions**
    - Use trainer.predict(test_dataset) to obtain logits, labels, and probabilities
2. **Compute metrics**
    - **Accuracy**: proportion of correct predictions
    - **F1-Score**: harmonic mean of precision and recall
    - **Log Loss**: evaluates probabilistic confidence
    - **Exact Match (EM)**: for question-answering, using a QA pipeline and the SQuAD metric
    - **Mean Squared Error (MSE)**: for regression tasks (e.g., STS-B)
3. **Implementation details**
    - For classification: apply softmax to logits, then argmax for predicted class
    - For QA: use pipeline('question-answering') with your fine-tuned QA model, then compute EM and F1 via evaluate.load('squad')
    - For regression: run inputs through a regression-tuned BERT and compute MSE against ground truth
4. **Reflection and refinement**
    - If F1 ≪ accuracy, investigate class imbalance
    - High log loss suggests overconfident wrong predictions—consider label smoothing
    - For regression, a large MSE may indicate poor scaling—revisit data preprocessing

---

Part 4: Creative Application – Named Entity Recognition

1. **Task selection**
   We fine-tune BERT for NER on the CoNLL-2003 dataset.
2. **Data processing**
    - Load conll2003 via datasets
    - Use BertTokenizerFast with is_split_into_words=True

    - Align original word-level NER tags to token-level labels, marking sub-word tokens with -100

3. **Model and training**
    - Load BertForTokenClassification with num_labels equal to the number of entity classes
    - Configure Trainer with TrainingArguments (batch size 16, epochs 3, epoch-wise evaluation)
    - Use seqeval to compute precision, recall, F1, and overall token accuracy
4. **Evaluation**
    - Run trainer.evaluate() on a validation subset
    - Inspect per-entity and overall metrics to assess recognition quality

---

Conclusion

This end-to-end workflow demonstrates:

- Setting up your environment and data
- Fine-tuning BERT for classification, QA, regression, and token classification
- Debugging and refining your models
- Evaluating with a full suite of metrics
- Applying BERT creatively to real-world tasks

Future enhancements might include mixed-precision training, dynamic sequence lengths, or zero-shot transfer to new domains.