Emotion Classification using BERT with Fine-Tuning Strategies

Melina Singh

June 28, 2025

1 Project Overview

This project performs **multi-label emotion classification** on the GoEmotions dataset using **RoBERTa** and evaluates multiple **fine-tuning techniques** to compare performance.

2 Dataset

- **Source:** Google's GoEmotions
- **Format:** TSV files with `Text`, `Class`, and `ID`
- **Classes:** 42 fine-grained emotion labels + `neutral`

3 Objective

Build and evaluate a robust transformer-based classifier that:

- Processes user-generated text (e.g., Reddit comments)
- Predicts multiple possible emotions per sentence
- Compares different fine-tuning approaches for transfer learning

4 Preprocessing Steps

1. **Load & Parse:** `train.tsv`, `dev.tsv`, and `emotions.txt`
2. **Label Expansion:** Map class indices to actual emotion names
3. **Ekman Mapping:** Aggregate 42 labels into 7 coarse categories (anger, disgust, fear, joy, sadness, surprise, neutral)
4. **Remove Noise:** Drop `neutral` and `disgust` classes
5. **Text Cleaning Pipeline:**
    - Emoji removal
    - HTML, URL stripping

- Punctuation/Contraction normalization
- Spelling correction

6. **Tokenization:** Performed with HuggingFace `AutoTokenizer` for RoBERTa

5 Model Architecture

The base model is **RoBERTa (roberta-base)** from HuggingFace Transformers.

Classification Head:

- `[CLS]` embedding → Dropout → Dense Layer → Sigmoid (for multi-label)

6 Training Setup

- **Loss Function:** `BCEWithLogitsLoss` (multi-label)
- **Evaluation Metrics:**
  - Accuracy
  - F1-Score (Micro, Macro)
- **Device:** GPU via Kaggle Notebook
- **Batch Size:** Tunable (e.g., 16)
- **Optimizer:** `AdamW`

7 Fine-Tuning Techniques Compared

7.1 Full Fine-Tuning

- All model weights are updated
- Baseline for comparison

7.2 Frozen Encoder (Train Classification Head Only)

- The base encoder is frozen
- Only the classification head is trained
- Used for fast, low-resource training

7.3 LoRA (Low-Rank Adaptation)

- Lightweight adapters are inserted into attention layers
- Only these adapters are updated
- Efficient and memory-saving

LoRA Config Used:

```
LoraConfig(
    r=8,
    lora_alpha=32,
```

```
    target_modules=["query", "value"],
    lora_dropout=0.1,
    task_type="SEQ_CLS"
)
```

7.4 Gradual Unfreezing

- Starts with all encoder layers frozen
- Unfreezes one layer at a time from top to bottom each epoch
- Prevents catastrophic forgetting and stabilizes early training

8 Results Summary

Method	Accuracy	F1 Micro	F1 Macro
Full Fine-Tuning	0.84	0.82	0.76
Frozen Encoder	0.78	0.75	0.69
LoRA	0.83	0.81	0.74
Gradual Unfreezing	0.85	0.83	0.78

Table 1: Performance Metrics

9 Model Checkpoints

- `model.bin` Base RoBERTa weights after initial training
- LoRA/Other variants are fine-tuned separately with isolated adapters or frozen weights

10 How to Run (in Kaggle Notebook)

1. Upload `train.tsv`, `dev.tsv`, `emotions.txt`
2. Run data preprocessing cells
3. Select and execute any fine-tuning strategy
4. Evaluate on validation set
5. Compare metrics

11 Dependencies

```
pip install transformers datasets torch scikit-learn peft
# Additional: bs4, emoji, tqdm, numpy, pandas
```

12 Future Work

- Try other base models
- Apply parameter-efficient tuning like **AdapterHub**
- Use a scheduler (e.g., linear decay LR)