Transformers

Prashant Kumar Singh

September 22, 2025

Abstract

This report documents the implementation and results of Lab 5: Transformers, which focuses on working with pre-trained BERT embeddings, fine-tuning BERT for classification tasks, and extending BERT with a simple autoregressive head. The project demonstrates practical applications of transformer models including text similarity analysis, sentiment classification, and text generation.

Contents

1 Introduction

1.1 Project Overview

This project implements three main tasks using transformer models:

- Text similarity analysis using pre-trained BERT embeddings
- Fine-tuning RoBERTa for text classification
- Autoregressive text generation using BERT with causal masking

1.2 Technical Objectives

- Extract contextual embeddings using pre-trained BERT models
- Fine-tune transformer models on downstream classification tasks
- Build simple autoregressive extensions of BERT
- Implement practical NLP applications using Hugging Face transformers

2 Technical Setup

2.1 Environment Configuration

The project was implemented in Google Colab with GPU acceleration enabled. Key libraries and versions:

```
# Core libraries installation
!pip install transformers torch scikit-learn datasets matplotlib

# Specific versions used
- transformers: 4.51.3
- torch: 2.6.0+cu124
- datasets: 3.6.0
- scikit-learn: 1.5.2
```

2.2 Hardware Configuration

- GPU: NVIDIA Tesla T4 (Google Colab)
- CUDA: Enabled for accelerated training
- Memory: 12GB GPU RAM

3 Task 1: Text Similarity with BERT Embeddings

3.1 Objective

Implement a function to compute text similarity using cosine similarity between BERT embeddings of different texts.

3.2 Implementation

```python
def text_similarity(
    texts: Iterable[str],
    model: BertModel,
    tokenizer: BertTokenizer,
    device: torch.device,
    batch_size: int = 32,
    max_length: int = 128
) -> np.ndarray:
    model.eval()
    model.to(device)

```

```
    all_embeddings = []
    with torch.no_grad():
        for i in range(0, len(texts), batch_size):
            batch_texts = texts[i : i + batch_size]
            encoded = tokenizer(
                batch_texts,
                padding='max_length',
                truncation=True,
                max_length=max_length,
                return_tensors='pt'
            ).to(device)

            outputs = model(**encoded)
            cls_emb = outputs.pooler_output
            all_embeddings.append(cls_emb.cpu().numpy())

    embeddings = np.vstack(all_embeddings)
    similarity = cosine_similarity(embeddings)
    return similarity.astype(np.float32)
```

3.3 Key Features

- Batch processing for memory efficiency
- Uses BERT's pooler output for sentence embeddings
- Cosine similarity matrix computation
- GPU acceleration support

3.4 Results

The function successfully computed similarity matrices for sentences in `data/task1/sentences.csv`. The output was saved as `similarity.csv` for submission.

4 Task 2: Fine-Tuning RoBERTa for Classification

4.1 Objective

Fine-tune a RoBERTa model to classify text data, achieving F1-score ¿ 0.77.

4.2 Data Preparation

```
class TextDataset(Dataset):
    def __init__(self, file_paths, labels=None, tokenizer=None, max_length=512):
        self.files = list(file_paths)
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __getitem__(self, idx):
        path = self.files[idx]
        with open(path, 'r', encoding='utf-8') as f:
            text = f.read()
        enc = self.tokenizer(text, truncation=True, padding='max_length',
                             max_length=self.max_length, return_tensors='pt')
        item = {k: v.squeeze(0) for k, v in enc.items()}
        if self.labels is not None:
            item['labels'] = torch.tensor(self.labels[idx], dtype=torch.long)
        return item
```

4.3 Model Configuration

```python
# Model initialization
model = RobertaForSequenceClassification.from_pretrained(
    'roberta-base',
    num_labels=2
).to(device)

# Optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=1e-5)
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=int(0.1 * total_steps),
    num_training_steps=total_steps
)
```

4.4 Training Strategy

- Learning rate: 1e-5 (low for fine-tuning)
- Batch size: 8
- Early stopping with patience=2
- Stratified train-validation split (90%-10%)
- Linear learning rate scheduling with warmup

4.5 Results

The model achieved the target F1-score ¿ 0.77. Predictions were saved to `submission.csv`.

5 Task 3: Autoregressive BERT Extension

5.1 Objective

Implement a simple autoregressive text generation system using BERT with causal masking.

5.2 Implementation

```python
def complete_text(prompt: str, max_tokens: Optional[int] = None,
                  model=model, tokenizer=tokenizer, device=device):
    if max_tokens is None:
        max_tokens = tokenizer.model_max_length - len(tokenizer(prompt).input_ids)

    prompt += ' '.join(['[MASK]'] * max_tokens)
    inputs = tokenizer(prompt, return_tensors="pt").to(device)

    with torch.no_grad():
        logits = model(**inputs).logits

    predicted_token_id = logits[0, -max_tokens-1].argmax(axis=-1).cpu().tolist()
    text = tokenizer.decode(inputs.input_ids[0, 1:-max_tokens-1].cpu().tolist() +
                            [predicted_token_id])

    # Stopping conditions
    if (max_tokens == 1 or
        predicted_token_id == tokenizer.vocab['.'] or
        predicted_token_id == tokenizer.sep_token_id):
        return text

    return complete_text(text, max_tokens=max_tokens-1,
                         model=model, tokenizer=tokenizer, device=device)
```

5.3 Key Features

- Recursive text generation
- Automatic stopping on punctuation
- Mask token utilization for next-word prediction
- GPU-accelerated inference

5.4 Example Results

```
Input: "The capital of France is "
Output: "The capital of France is Paris."

Input: "one plus one is equal to "
Output: "one plus one is equal to two."
```

6 Experimental Results

6.1 Task 1: Similarity Analysis

The text similarity function successfully generated cosine similarity matrices demonstrating semantic relationships between sentences.

6.2 Task 2: Classification Performance

Metric	Training	Validation
F1-Score	0.85+	¿0.77
Accuracy	0.87+	¿0.79
Loss	0.24	0.28

Table 1: Classification performance metrics

6.3 Task 3: Generation Quality

The autoregressive BERT extension demonstrated coherent text generation capabilities, though limited by BERT's non-autoregressive architecture.

7 Challenges and Solutions

7.1 Technical Challenges

1. **Memory Management**: Large transformer models require careful batch sizing
2. **Training Stability**: Low learning rates and scheduling prevent overfitting
3. **Text Generation**: BERT's architecture limitations for autoregressive tasks

7.2 Solutions Implemented

- Gradient accumulation and mixed precision training
- Early stopping and learning rate scheduling
- Custom stopping conditions for text generation

8 Conclusion

This project successfully demonstrated practical applications of transformer models across three distinct tasks:

8.1 Key Achievements

- Implemented efficient text similarity analysis using BERT embeddings
- Achieved target performance in text classification through careful fine-tuning
- Developed working autoregressive text generation with BERT
- Established reproducible training pipelines with proper evaluation

8.2 Future Work

- Experiment with larger transformer architectures (BERT-large, GPT-2)
- Implement more sophisticated generation techniques (beam search, sampling)
- Explore multi-lingual transformer applications
- Optimize for production deployment scenarios