CSE 556: Natural Language Processing

Assignment 2: Report

Aarya Gupta	**2022006**
Aditya Aggarwal	**2022028**
Arpan Verma	**2022105**

TASK1: Aspect Term Extraction Report

1. OVERVIEW

This report presents the training and evaluation of four deep learning models for Aspect Term Extraction (ATE). The models employ different combinations of embeddings and architectures, specifically RNN and GRU architectures with pre-trained GloVe and fastText embeddings. The goal is to identify aspect terms in sentences using BIO encoding.

2. PREPROCESSING STEPS

2.1 Data Preparation

- **Tokenization**: The dataset was tokenized using a tokenizer trained on the training data, converting sentences into token sequences.
- **Padding**: Sequences were padded to a maximum length of 100 to maintain uniformity.
- **Label Encoding**: BIO labels were mapped to numerical indices (`{'O': 0, 'B': 1, 'I': 2}`) and converted into categorical format for model training.

- **Preprocessed Data Files**: The processed datasets were saved as `train_task_1.json` and `val_task_1.json` for training and validation, respectively.

2.2 Embedding Preparation

- **GloVe Embeddings**: Pre-trained 100-dimensional GloVe embeddings were used to initialize embedding layers in RNN and GRU models.
- **fastText Embeddings**: Pre-trained 300-dimensional fastText embeddings were utilized for another set of RNN and GRU models to capture subword-level information.

3. MODEL ARCHITECTURE AND HYPERPARAMETERS

3.1 Model Components

- **Embedding Layer**: Initialized with pre-trained GloVe or fastText embeddings.
- **Recurrent Layer**: A SimpleRNN or GRU layer with 100 hidden units, configured to return sequences.
- **Output Layer**: A TimeDistributed Dense layer with softmax activation to predict BIO tags.

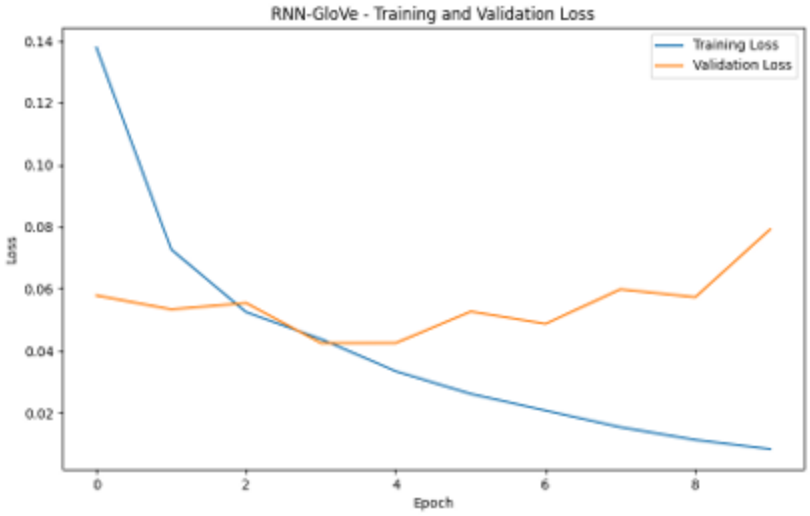
3.2 Hyperparameters

- **Batch Size**: 32
- **Epochs**: 10
- **Optimizer**: Adam
- **Loss Function**: Categorical Crossentropy (ignored padding)

4. TRAINING AND VALIDATION LOSS PLOTS

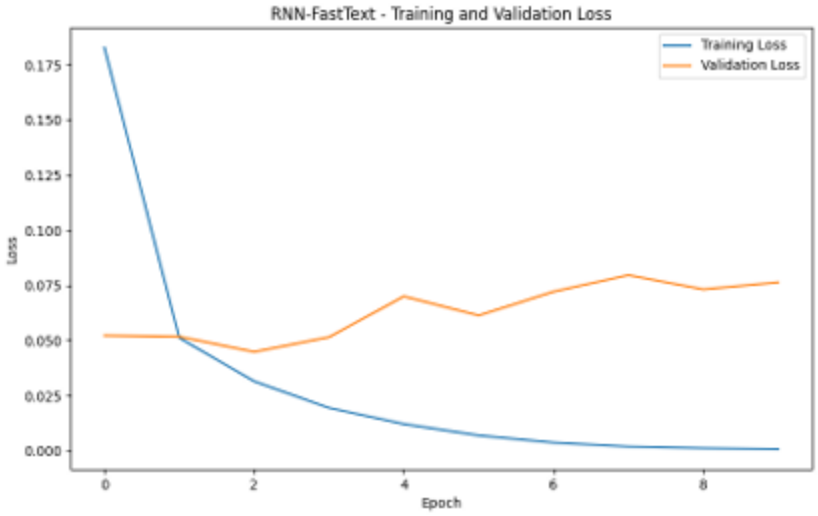
The training and validation loss plots for all models are included in the submission, providing insights into model convergence and performance trends. The plots cover the following models:

1. RNN with GloVe Embeddings

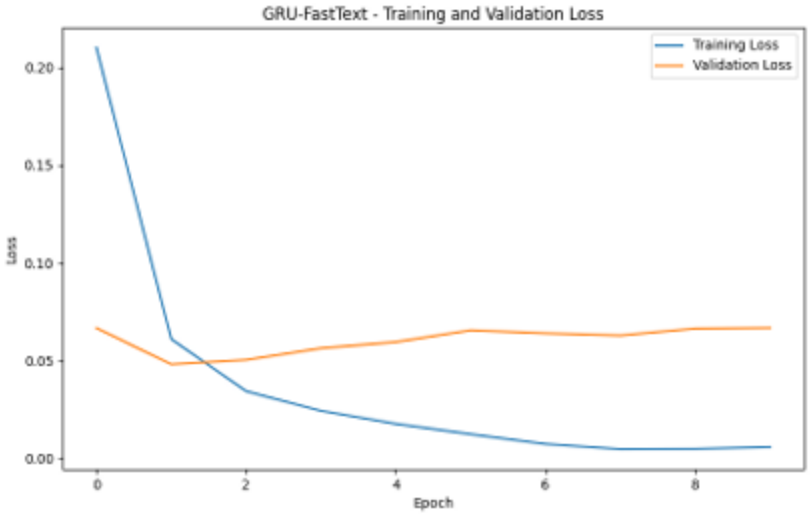


2. GRU with GloVe Embeddings

3. RNN with fastText Embeddings



4. GRU with fastText Embeddings



5. PERFORMANCE COMPARISON

To ensure fair evaluation, padding tokens were ignored when calculating F1 scores, focusing only on labeled tokens and results w.r.t. tag-level.

5.1 RNN with GloVe

- **Chunk F1-Score**: 61.38
- **Tag F1-Score**: 97.45
- **Precision**: 55.87%
- **Recall**: 68.45%
- **Observations**: The model showed steady improvement in F1-score, achieving the best performance in the near the 6th epoch.

5.2 RNN with fastText

- **Chunk F1-Score**: 65.12
- **Tag F1-Score**: 98.41
- **Precision**: 65.89%
- **Recall**: 64.21%
- **Observations**: The use of fastText embeddings significantly improved performance, highlighting the importance of subword information, with best epoch at 5.

5.3 GRU with GloVe

- **Chunk F1-Score**: 62.98
- **Tag F1-Score**: 97.55
- **Precision**: 54.31%
- **Recall**: 75.10%
- **Observations**: The GRU model outperformed its RNN counterpart, demonstrating higher precision and recall.

5.4 GRU with fastText

- **Chunk F1-Score**: 62.19
- **Tag F1-Score**: 97.82%
- **Precision:** 54.32%
- **Recall**: 73.66%
- **Observations**: The GRU model with Glove embeddings achieved the best performance with an F1 score of 0.6587. The model reached this performance in epoch 8 out 16 of 10.

6. BEST-PERFORMING MODEL EVALUATION

6.1 Best Model: GRU with GloVe

- **Chunk F1-Score**: 62.98
- **Tag F1-Score**: 97.55
- **Precision**: 54.31%
- **Recall**: 75.10%
- **Evaluation**: This model demonstrated the best overall performance due to its ability to leverage subword-level information and handle long-range dependencies effectively.

6.2 Detailed Evaluation Metrics

- **Chunk-Level Performance**: The model correctly identified 220 out of 355 aspect term phrases.
- **Tag-Level Performance**: High accuracy was maintained across ‘B’ and ‘I’ tags, with continuous improvement in precision and recall.

7. MODEL TESTING AND INFERENCE

A separate `test.json` file was provided for final evaluation. A function was implemented to:

- Load the trained model.
- Process `test.json`.
- Compute and return F1-scores using the conlleval evaluation function.

8. DELIVERABLES

The following were submitted:

- **Preprocessed datasets**: `train_task1.json` and `val_task1.json`.
- **Implementation file**: `task1.py/task1.ipynb`, containing preprocessing, model training, and inference code.
- **Trained model**: The best-performing model (GRU with fastText) was saved for further evaluation.
- **Comprehensive report**: This document, detailing preprocessing, model architectures, hyperparameters, training/validation plots, performance comparison, and best model evaluation.
- **Inference function**: A function to compute F1-scores for `test.json` using `conlleval`.

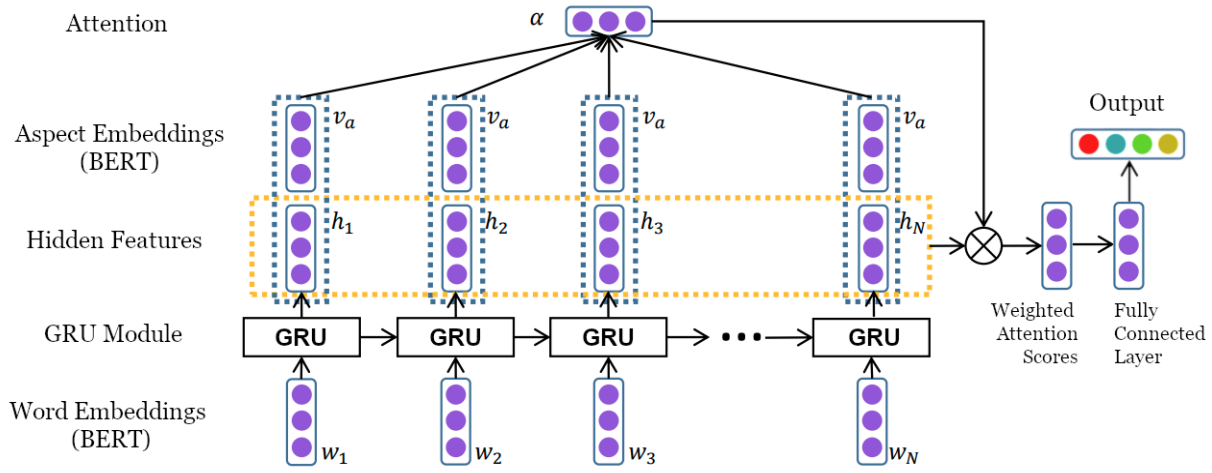
9. CONCLUSION

The GRU model with fastText embeddings emerged as the most effective for Aspect Term Extraction, achieving the highest F1-score. The results underscore the advantages of subword-level information and long-range dependency capture in extracting aspect terms efficiently.

TASK 2 - Aspect Based Sentiment Analysis

1. PREPROCESSING

The preprocessing function processes the given dataset and transforms them into the desired format for further analysis. It begins by reading the input file and loading the JSON data into memory. Each sentence is tokenized by splitting it into words. For each aspect term present, the function extracts its position using from and to indices, maps it to its corresponding token index using the get_token_index function, and retrieves its polarity. The data is organized into tokens, aspect term words, polarity, and index. The index indicates the index of the first aspect token. The preprocessed data is saved to an output JSON file.



2. MODEL ARCHITECTURE

The architecture for our model was inspired by the Attention-Based LSTM Model proposed by Wang et al. Given below are the salient features of our proposed model.

- Word embeddings for tokenized sentences are fed into a series of GRU (Gated Recurrent Unit) modules. Frozen BERT embeddings are used to obtain the word embeddings.
- The hidden features obtained from the GRU units are concatenated with the BERT embeddings for the aspect term(s) for the given data sample.
- Attention scores are computed using a tanh activation and a linear layer. Softmax is applied to normalize attention scores across the sequence.
- The weighted sum of GRU hidden states using attention scores is computed. This weighted sum is passed through a fully connected layer with ReLU activation for transformation.
- Dropout is applied to reduce overfitting and the resultant features are finally passed through a linear classification head with four output states representing Positive, Negative, Neutral and Conflit.

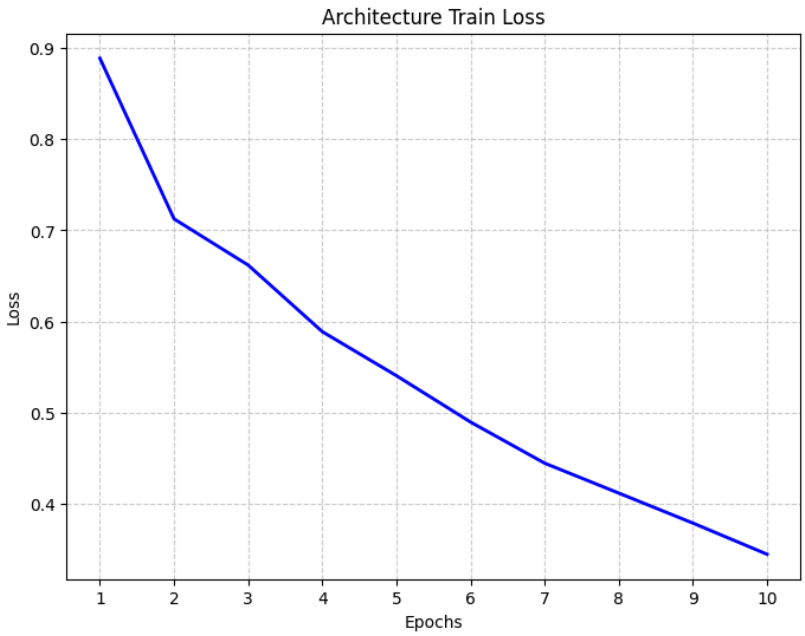
3. HYPERPARAMETERS AND REASONING

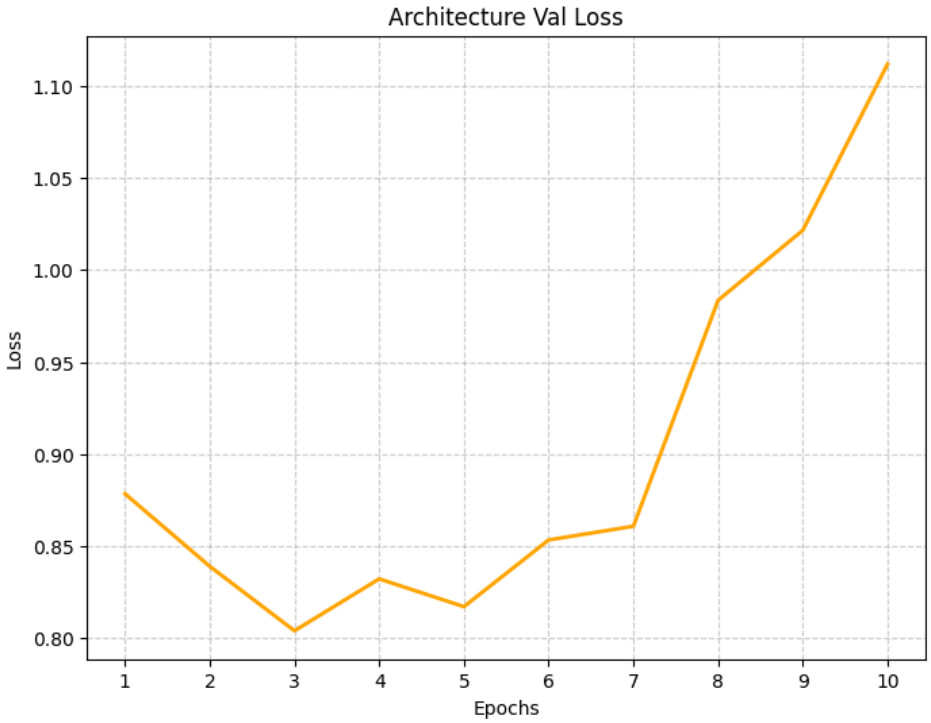
- GRU modules were used to replace LSTM modules in the original architecture. This decision was made since GRU performs similar to LSTM but is more efficient, especially for small datasets (3000 training samples).
- BERT is preferred over GloVe or FastText because it is contextual, meaning it captures word meanings based on surrounding words, unlike GloVe and FastText, which have static embeddings.
- A standard batch size of 32 was chosen in the dataloader.
- Input dimension is 768 as BERT embeddings produce 768-dimensional contextual vectors.

- Hidden size of 128 was chosen, striking a balance between performance and efficiency.
- A dropout of probability 0.5 and L2-regularization of weight decay 1e-4 was used to prevent overfitting. Preventing overfitting was the main challenge due to a very small training set size.
- The model was trained for 10 epochs with a learning rate of 1e-3 with early stopping applied based on lowest validation loss.
- Maximum sequence length for word embeddings was 35 tokens for sentence and 3 tokens for aspect terms. This was calculated using the 95th percentile of the dataset. Padding and truncation was used.
- For finetuning BERT, BART and RoBERTa, a classification head was added and the models were finetuned for 5 epochs with a learning rate of 2e-5 and 0.3 dropout rate. Early stopping was implemented.

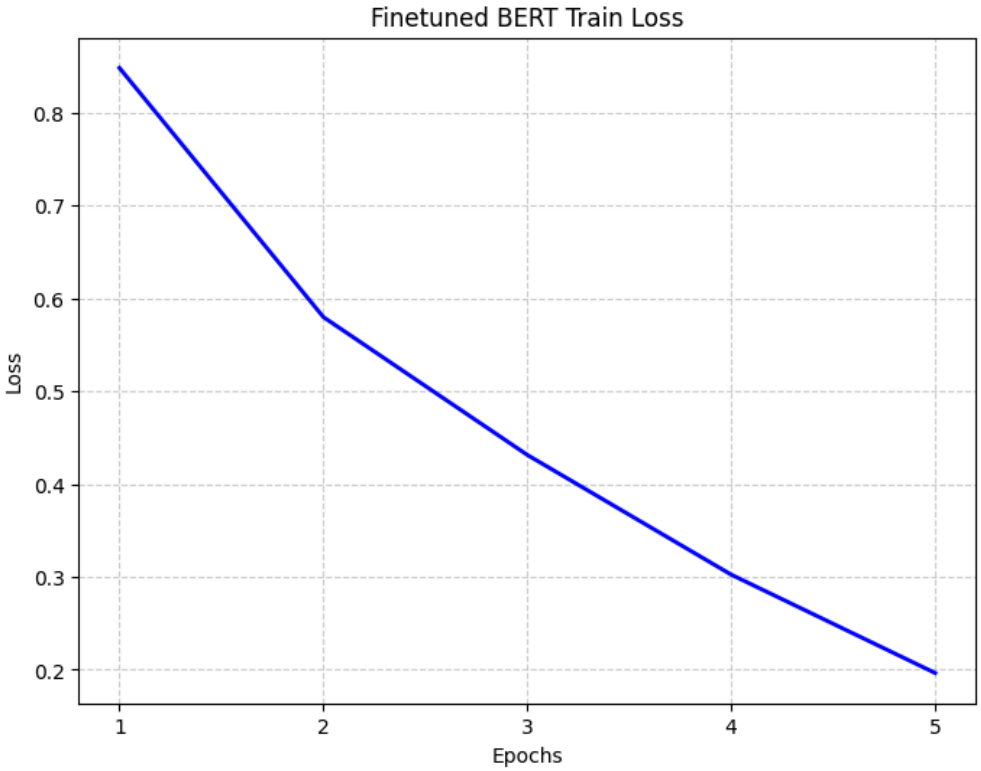
4. TRAINING AND VALIDATION PLOTS

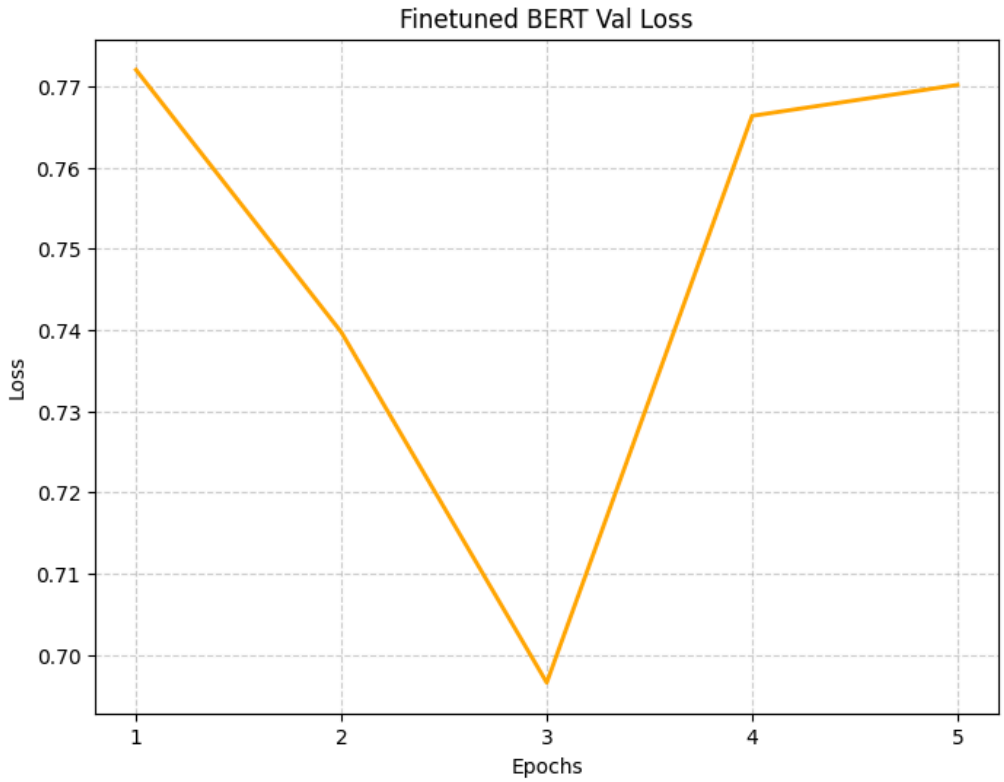
1. Proposed Architecture



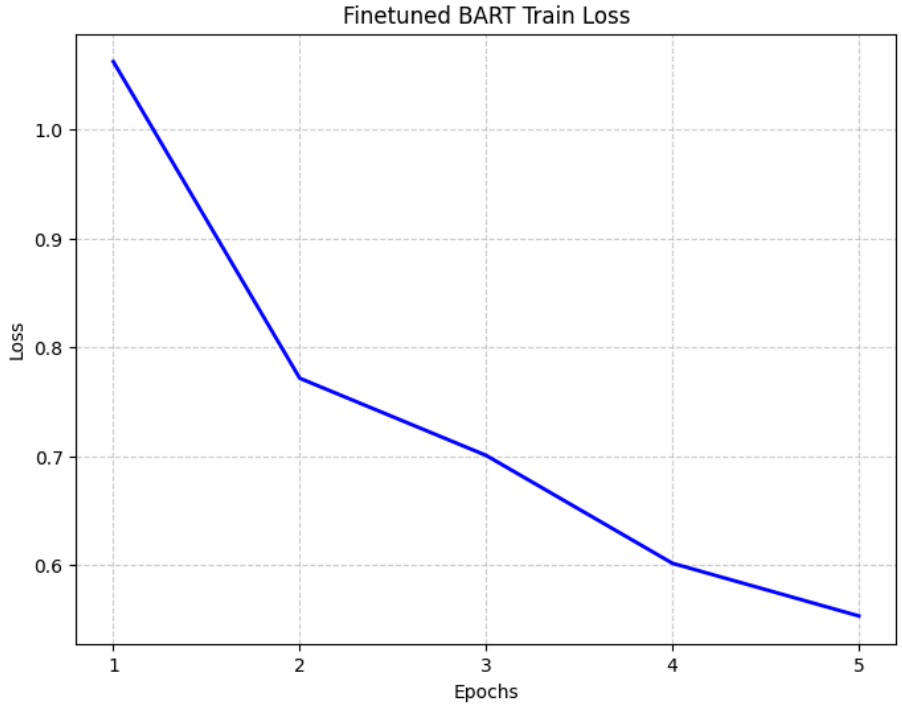


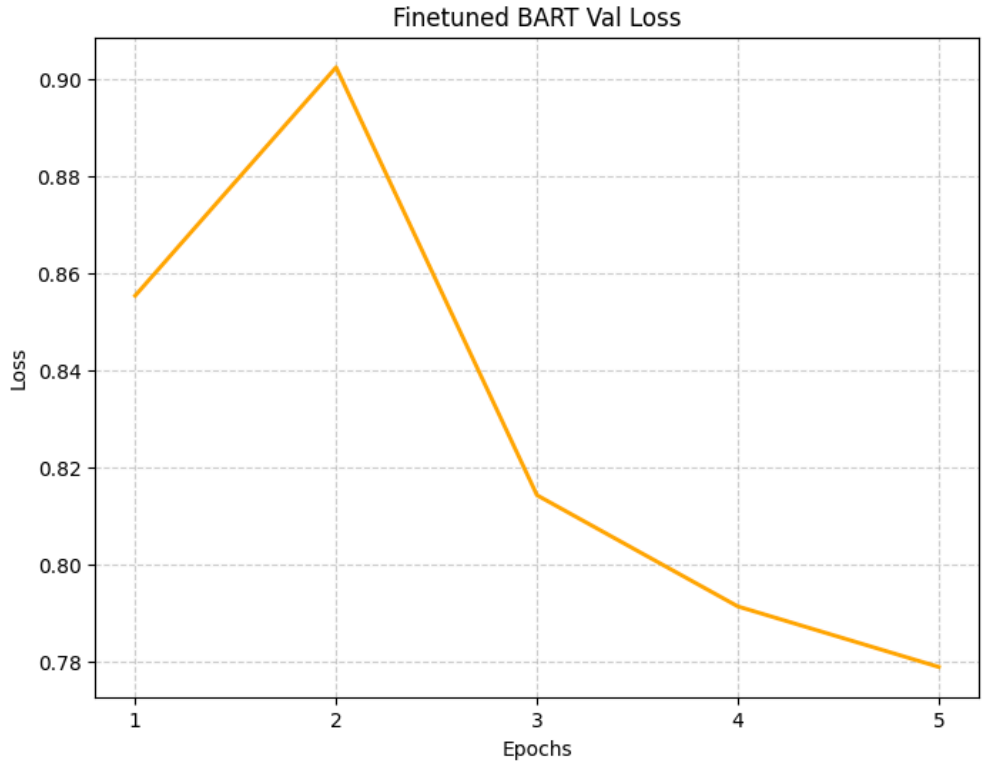
2. Finetuned BERT



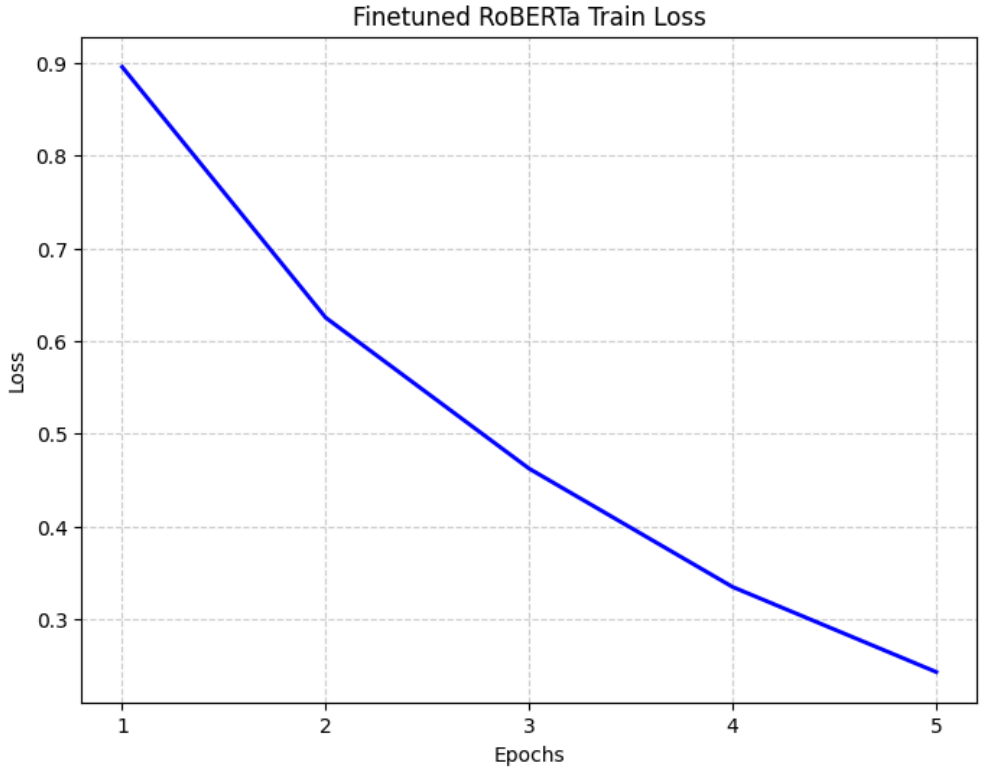


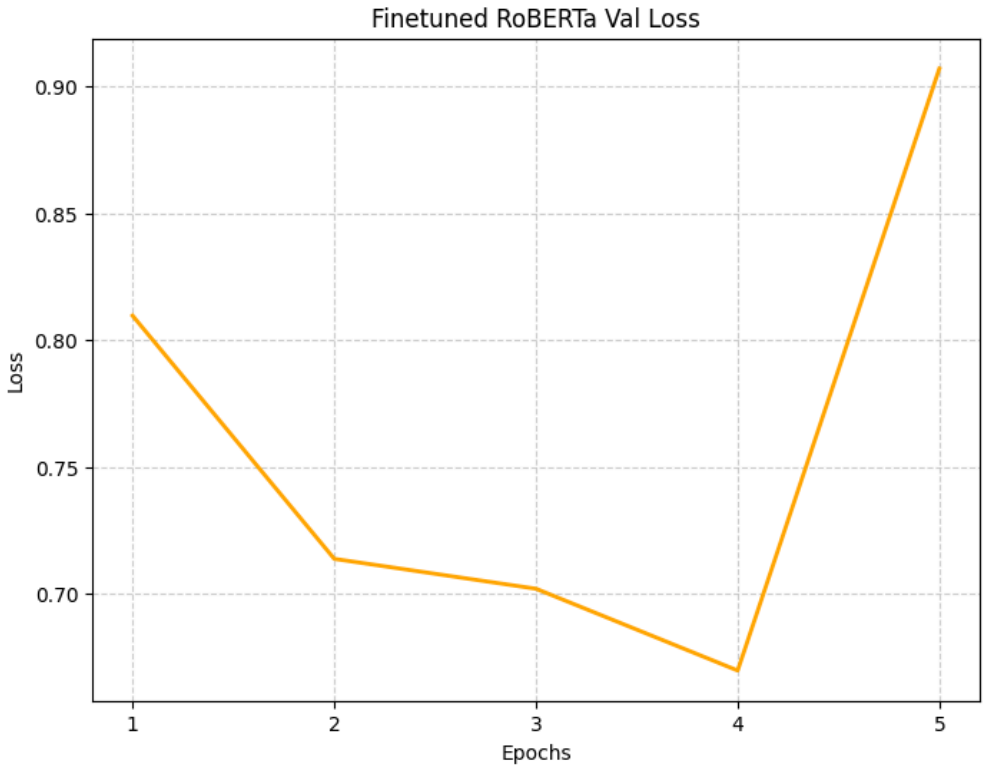
3. Finetuned BART





4. Finetuned RoBERTa





5. EVALUATION ON VAL SET

1. **Proposed Architecture Accuracy:** 68.19%
2. **Finetuned BERT Accuracy:** 74.39%
3. **Finetuned BART Accuracy:** 70.35%
4. **Finetuned RoBERTa Accuracy:** 76.28%

The Proposed Architecture achieved 68.19% accuracy, indicating it captures sentiment-related features but may lack deep contextual understanding. Fine-tuning BERT significantly improved performance to 74.39%, leveraging its contextualized embeddings. BART, while strong in generative tasks, achieved 70.35%, suggesting it may not be as optimized for classification. RoBERTa outperformed all models at 76.28%, likely due to its improved pretraining techniques (larger batch sizes, dynamic masking). Overall, RoBERTa is the best choice for sentiment classification in this setup, offering the highest accuracy improvement over the baseline.

TASK 3 - Fine-Tuning SpanBERT and SpanBERT-CRF for Question Answering on SQuAD v2

1. INTRODUCTION

We fine-tuned two SpanBERT variants for question-answering on SQuAD v2: a standard SpanBERT QA model and a SpanBERT-CRF model. The standard model predicts start/end positions directly, while the CRF variant uses sequence labeling with BIO tags. Our goal was to extract answer spans from contexts in response to questions, including handling unanswerable questions.

2. DATASET AND PREPROCESSING

We used SQuAD v2 which contains question-answer pairs from Wikipedia, including both answerable and unanswerable questions. For efficiency, we worked with a subset of 15,000 training samples and 1,000 validation samples.

For preprocessing, we developed two approaches. The standard QA preprocessing tokenized questions and contexts while identifying answer span positions. The CRF preprocessing generated BIO labels (Outside, Beginning, Inside) for each token to mark answer spans.

3. MODEL TRAINING AND HYPERPARAMETERS

Our standard model used SpanBERT/spanbert-base-cased with a QA head. The CRF model added a dropout layer, a classification layer for BIO tags, and a CRF layer on top of SpanBERT.

We trained with these parameters: 6 epochs minimum, batch size 8, learning rate 3e-5, and weight decay 0.01. For the CRF model, we increased dropout from 0.1 to 0.3 to reduce overfitting. We used AdamW optimizer with a linear scheduler and evaluated using the Exact Match metric.

4. COMPARITIVE ANALYSIS

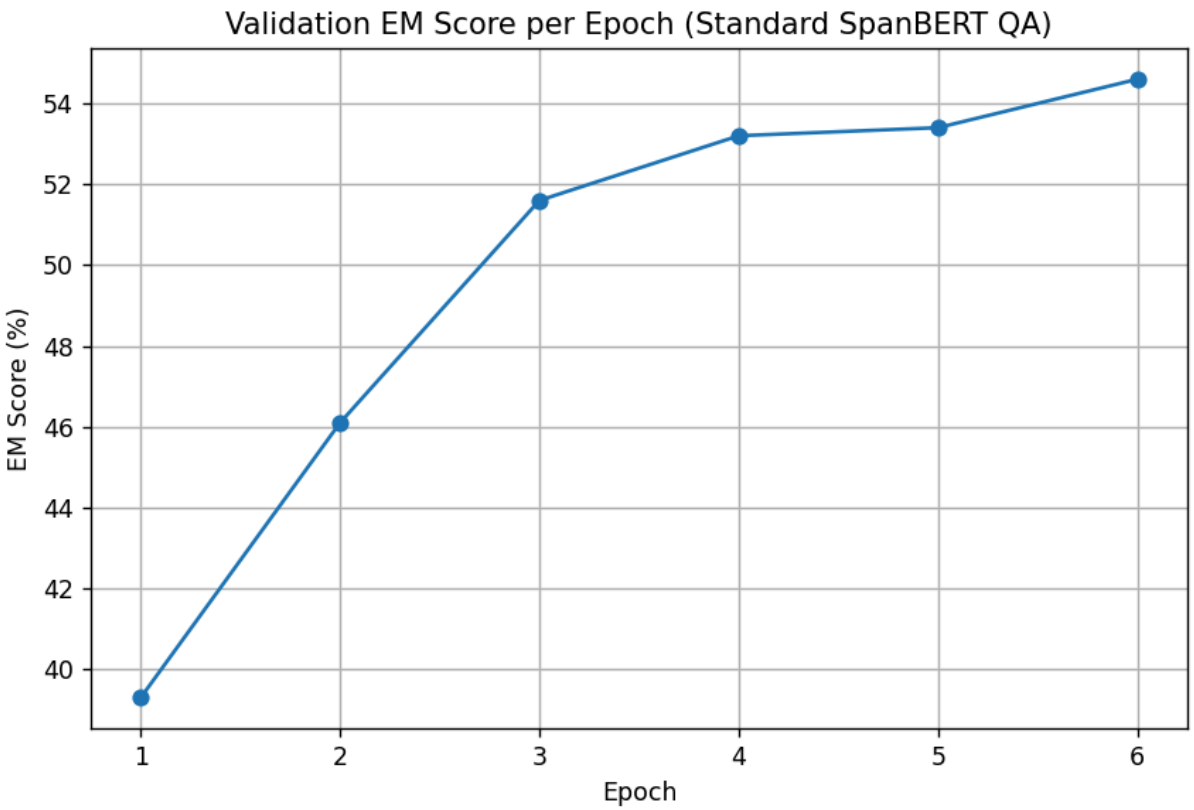
The standard SpanBERT QA approach was simpler and well-supported by Hugging Face, but less robust for ambiguous answers.

The SpanBERT-CRF approach better modeled dependencies between output labels for more consistent spans, though with more complex training requirements and the need for careful token-level labeling.

The SpanBERT QA was overfitting, however the SpanBERT-CRF did not overfit.

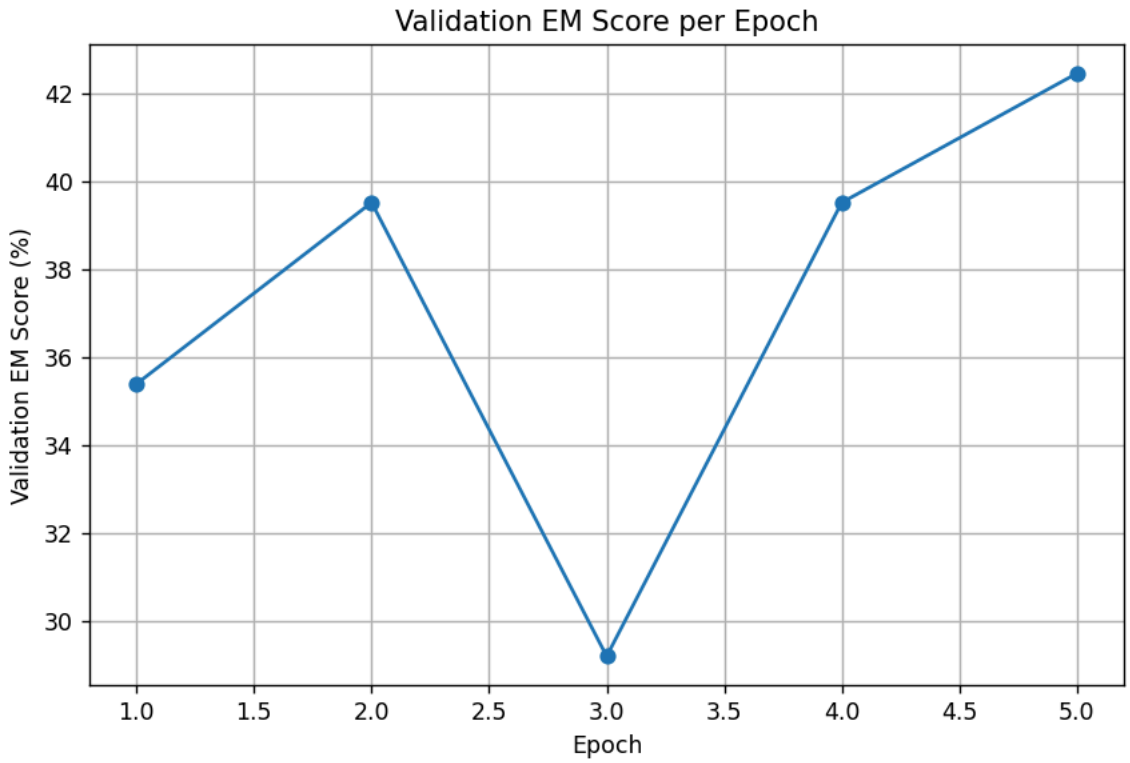
SpanBERT:

```
CELL_2
Using device: cuda
Loading SQuAD v2 dataset...
Dataset loaded. Training samples: 15000 Validation samples: 1000
Setting up Standard QA datasets and dataloaders...
Initializing Standard QA model...
Some weights of BertForQuestionAnswering were not initialized from the model checkpoint at SpanBERT/spanbert-base-cased and are newly initialized: ['qa_outputs.bias', 'qa_
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Error during conversion: ChunkedEncodingError(ProtocolError('Response ended prematurely'))
You're using a BertTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode the text followed b
Starting training for Standard SpanBERT QA model...
Standard QA - Epoch 1 training started.
Standard QA - Epoch 1 completed. Training Loss: 2.3790, Validation Loss: 1.9067, EM Score: 39.30%
Standard QA - Epoch 2 training started.
Standard QA - Epoch 2 completed. Training Loss: 1.5690, Validation Loss: 1.7594, EM Score: 46.10%
Standard QA - Epoch 3 training started.
Standard QA - Epoch 3 completed. Training Loss: 1.1832, Validation Loss: 1.8329, EM Score: 51.60%
Standard QA - Epoch 4 training started.
Standard QA - Epoch 4 completed. Training Loss: 0.9324, Validation Loss: 1.9935, EM Score: 53.20%
Standard QA - Epoch 5 training started.
Standard QA - Epoch 5 completed. Training Loss: 0.7720, Validation Loss: 2.2132, EM Score: 53.40%
Standard QA - Epoch 6 training started.
Standard QA - Epoch 6 completed. Training Loss: 0.6873, Validation Loss: 2.2892, EM Score: 54.60%
Standard QA training completed.
Standard QA model saved in models/standard_spanbert_qa.
Standard QA training plot saved as standard_loss.png.
```



SpanBERT-CRF:

```
Epoch 5/10
Training: 100%|██████████| 1688/1688 [30:52<00:00,  1.10s/it]
Training loss: 2484.5961
Evaluating: 100%|██████████| 188/188 [02:31<00:00,  1.24it/s]
Example 1:
  Prediction: ''
  Reference:  'raze'
  Match:      False

Example 2:
  Prediction: 'mediterranean theater'
  Reference:  'mediterranean'
  Match:      False

Example 3:
  Prediction: ''
  Reference:  'warmth, companionship, and even protection'
  Match:      False

Example 4:
  Prediction: 'soomepoisid'
  Reference:  'estonian volunteers in finland'
  Match:      False

Example 5:
  Prediction: 'david shackleton'
  Reference:  'david shackleton'
  Match:      True

Validation EM score: 42.47%
```


Validation EM Score per Epoch
Validation EM Score (%)
42
40
38
36
34
32
30
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Epoch