Task 1 - Aspect Term Extraction

Introduction

Aspect term extraction (ATE): given sentence, identify all aspect terms present in the sentence.

Dataset Pre-Preprocessing

Since Aspect Term Extraction (ATE) is treated as a **sequence labeling task**, the dataset needs to be preprocessed using the **BIO tagging scheme** (B = Beginning, I = Inside, O = Outside). This helps train models to recognize aspect terms in text.

- The dataset is stored in **JSON format** and contains **sentences** with manually labeled **aspect terms**.
- Each sentence is processed individually.
- All text is converted to **lowercase** to ensure consistency and avoid case-sensitive mismatches.
- Punctuation marks (e.g., `. , ! ?`) and special characters are removed.
- The sentence is split into individual **tokens (words)**.
- Every token is initially assigned the label **'O'** (Outside), meaning it is not part of an aspect term.
- The **aspect terms** are extracted from the dataset and matched within the tokenized sentence.

Word Embeddings

GloVe (Global Vectors for Word Representation)

- **Learns word embeddings based on word co-occurrence in a corpus (global statistical information).**
- **Produces fixed-size dense vectors for words, capturing semantic meaning.**
- **Cannot handle out-of-vocabulary (OOV) words as it treats words as atomic units.**
- **Performs well for general NLP tasks but struggles with rare or misspelled words.**

FastText

- **Uses subword embeddings (character n-grams), allowing it to handle OOV words (e.g., misspellings).**
- **Better for morphologically rich languages due to its subword information.**
- **Can generate embeddings for unseen words, unlike GloVe.**
- **Generally more effective for text classification and noisy text data**

Models

ATE_RNN Architecture (Vanilla RNN)

- **Pretrained Embeddings**: Uses fixed word embeddings (e.g., GloVe, FastText).
- **RNN Layer**:
  - Processes input sequentially.
  - Struggles with long-range dependencies (vanishing gradient problem).
- **Fully Connected (FC) Layer**: Converts hidden states into output logits for classification.
- **Output**: Shape **(batch_size, seq_length, num_classes)** for BIO tagging.

ATE_GRU Architecture (Gated Recurrent Unit)

- **Pretrained Embeddings**: Same as ATE_RNN.
- **GRU Layer**:
  - Uses **Reset Gate** to forget past info.
  - Uses **Update Gate** to decide what info to retain.
  - Solves vanishing gradient problems, handles long dependencies better.
- **Fully Connected (FC) Layer**: Converts hidden states into classification output.
- **Output**: Same as ATE_RNN.

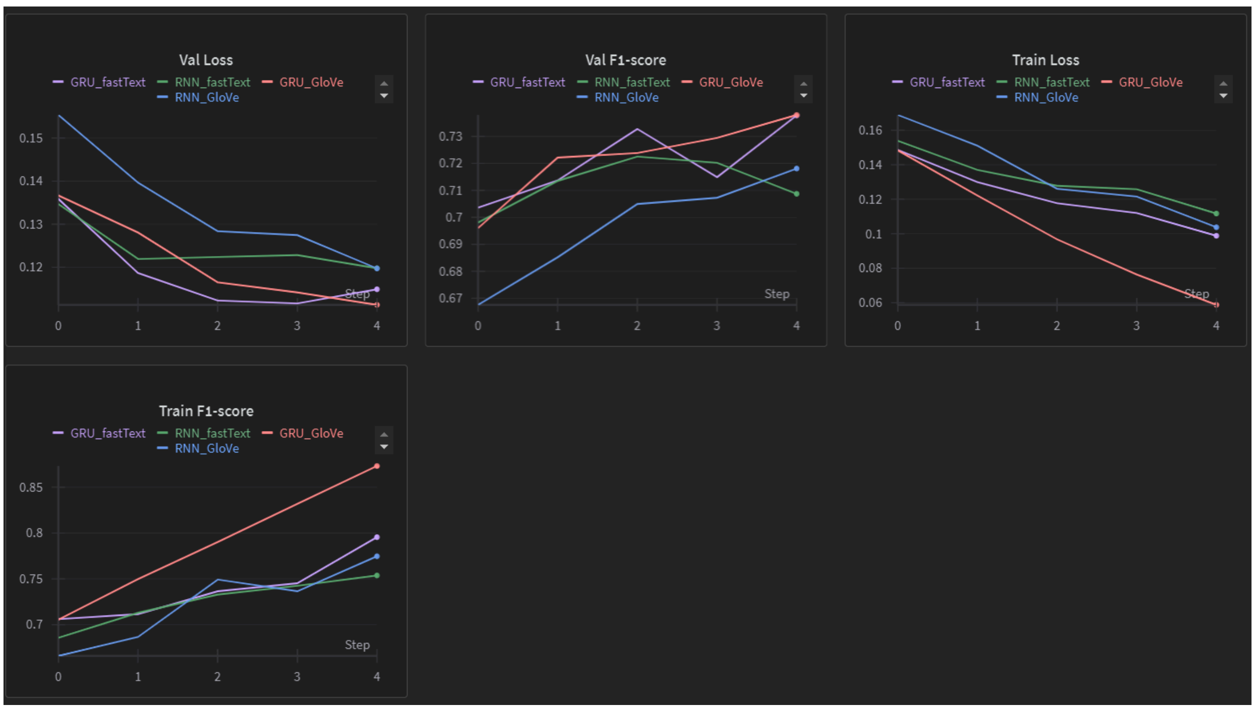
Model Training

```
------------------------------------- TRAINING RNN_GloVe -------------------------------------
Epoch [1/5] -> Train Loss: 0.1689, Train F1: 0.6660 | Val Loss: 0.1554, Val F1: 0.6675
Epoch [2/5] -> Train Loss: 0.1511, Train F1: 0.6867 | Val Loss: 0.1397, Val F1: 0.6852
Epoch [3/5] -> Train Loss: 0.1260, Train F1: 0.7492 | Val Loss: 0.1284, Val F1: 0.7050
Epoch [4/5] -> Train Loss: 0.1216, Train F1: 0.7365 | Val Loss: 0.1275, Val F1: 0.7072
Epoch [5/5] -> Train Loss: 0.1038, Train F1: 0.7747 | Val Loss: 0.1197, Val F1: 0.7181
-------------------------------------- TRAINING COMPLETED -------------------------------------
```

```
------------------------------------- TRAINING GRU_GloVe -------------------------------------
Epoch [1/5] -> Train Loss: 0.1483, Train F1: 0.7055 | Val Loss: 0.1367, Val F1: 0.6960
Epoch [2/5] -> Train Loss: 0.1221, Train F1: 0.7497 | Val Loss: 0.1281, Val F1: 0.7221
Epoch [3/5] -> Train Loss: 0.0968, Train F1: 0.7901 | Val Loss: 0.1165, Val F1: 0.7239
Epoch [4/5] -> Train Loss: 0.0763, Train F1: 0.8318 | Val Loss: 0.1142, Val F1: 0.7294
Epoch [5/5] -> Train Loss: 0.0587, Train F1: 0.8732 | Val Loss: 0.1113, Val F1: 0.7380
-------------------------------------- TRAINING COMPLETED -------------------------------------
```

```
------------------------------------ TRAINING RNN_fastText ------------------------------------
Epoch [1/5] -> Train Loss: 0.1539, Train F1: 0.6858 | Val Loss: 0.1347, Val F1: 0.6981
Epoch [2/5] -> Train Loss: 0.1371, Train F1: 0.7130 | Val Loss: 0.1219, Val F1: 0.7135
Epoch [3/5] -> Train Loss: 0.1278, Train F1: 0.7329 | Val Loss: 0.1224, Val F1: 0.7225
Epoch [4/5] -> Train Loss: 0.1258, Train F1: 0.7425 | Val Loss: 0.1228, Val F1: 0.7202
Epoch [5/5] -> Train Loss: 0.1117, Train F1: 0.7537 | Val Loss: 0.1198, Val F1: 0.7087
-------------------------------------- TRAINING COMPLETED -------------------------------------
```

```
------------------------------------ TRAINING GRU_fastText ------------------------------------
Epoch [1/5] -> Train Loss: 0.1486, Train F1: 0.7061 | Val Loss: 0.1359, Val F1: 0.7036
Epoch [2/5] -> Train Loss: 0.1299, Train F1: 0.7115 | Val Loss: 0.1187, Val F1: 0.7137
Epoch [3/5] -> Train Loss: 0.1177, Train F1: 0.7365 | Val Loss: 0.1123, Val F1: 0.7328
Epoch [4/5] -> Train Loss: 0.1120, Train F1: 0.7453 | Val Loss: 0.1116, Val F1: 0.7149
Epoch [5/5] -> Train Loss: 0.0988, Train F1: 0.7955 | Val Loss: 0.1149, Val F1: 0.7378
-------------------------------------- TRAINING COMPLETED -------------------------------------
```



Performance Comparison of All Models

1. Training and Validation Performance Overview

Name (4 visualized)	Runtim	Train F1-score	Train Loss	Val F1-score	Val Loss	batch_	epochs	learnin	loss_function	optimizer
GRU_fastText	40s	0.79551	0.098816	0.73779	0.1149	1	5	0.0005	CrossEntropyLoss	Adam
RNN_fastText	38s	0.75372	0.1117	0.70872	0.11978	1	5	0.0005	CrossEntropyLoss	Adam
GRU_GloVe	40s	0.8732	0.058742	0.73797	0.11128	1	5	0.0005	CrossEntropyLoss	Adam
RNN_GloVe	40s	0.77467	0.10376	0.71809	0.11974	1	5	0.0005	CrossEntropyLoss	Adam

2. Observations

- **GRU-based models outperform RNN-based models** in both training and validation.
- **GloVe embeddings lead to better performance** than FastText, especially in the GRU model.
- All Models have Roughly same **Tag F1 Score (Train: ~0.99, Val: ~0.95)**
- **GRU_GloVe achieves the highest chunk F1 score (Train: 0.8732, Val: 0.7380) and lowest loss**.
- FastText models perform slightly worse, likely due to differences in subword representation.

Best-Performing Model: GRU_GloVe

- **Final Validation Chunk Level F1 Score**: **0.7380**
- **Final Validation Tag Level F1 Score: 0.95387**
- **Final Validation Loss**: **0.1113**
- **Reason for Best Performance**:
    - GRU retains long-term dependencies better than RNN.
    - GloVe embeddings provide more accurate semantic representation.
    - Shows the best generalization with the lowest validation loss.

Task 2 - Aspect Based Sentiment Analysis

Introduction

Aspect Based Sentiment Analysis (ABSA): given sentence and its aspect terms, identify all sentiment of the aspects

Dataset Pre-Preprocessing

Since Aspect Term Extraction (ATE) is treated as a **sequence labeling task**, the dataset needs to be preprocessed using the **BIO tagging scheme** (B = Beginning, I = Inside, O = Outside). This helps train models to recognize aspect terms in text.

- The dataset is stored in **JSON format** and contains **sentences** with manually labeled **aspect terms**.
- Each sentence is processed individually.
- All text is converted to **lowercase** to ensure consistency and avoid case-sensitive mismatches.

- Punctuation marks (e.g., `.,!?`) and special characters are removed to match with Word Embeddings
- The sentence is split into individual **tokens (words)**.
- Aspect terms are also extracted, processed and find their position (index) in the sentence.
- Assign polarity labels (positive, negative, neutral, conflict) to aspect terms..

Word Embeddings

GloVe (Global Vectors for Word Representation)

- **GloVe Performed Better better in Task 1 so we are going with it.**

Model

ATAE-LSTM Architecture

1. Embedding Layer

- Uses **pretrained word embeddings** (GloVe).
- Loads embeddings with `nn.Embedding.from_pretrained()`.
- Aspect terms are also embedded using the same embeddings.

2. Aspect Representation

- Computes the **mean embedding** of the aspect term (ignoring padding).
- Expands the aspect embedding to match the sentence length.
- Helps the model understand aspect-specific sentiment.

3. LSTM Layer

- A **bidirectional LSTM** processes the combined input of:
  - **Sentence embeddings** (contextual understanding).
  - **Aspect embeddings** (focus on aspect-specific information).
- Outputs a **hidden representation** for each token.

4. Attention Mechanism

- **Learns importance scores** for each word in the sentence.
- Uses a linear transformation (`attention_M`) followed by `tanh()`.
- Computes **attention scores** (`attention_alpha`).
- Applies a **softmax function** to get attention weights.
- Masks **padding tokens** to prevent them from influencing attention.
- Computes a **context vector** as a weighted sum of LSTM outputs.

5. Fully Connected Layers

- First **fully connected layer (fc1)** reduces dimensions, followed by **ReLU activation**.
- Second **fully connected layer (fc2)** maps to the final output.
- Outputs **sentiment classification** (positive, negative, neutral, conflict).

6. Output

- Returns **final predictions** and **attention weights** (to interpret model focus).

🔹 **Key Strengths**:

- **Aspect-aware attention** improves sentiment prediction.
- **Bidirectional LSTM** captures context from both left and right.
- **Attention mechanism** helps model focus on relevant words

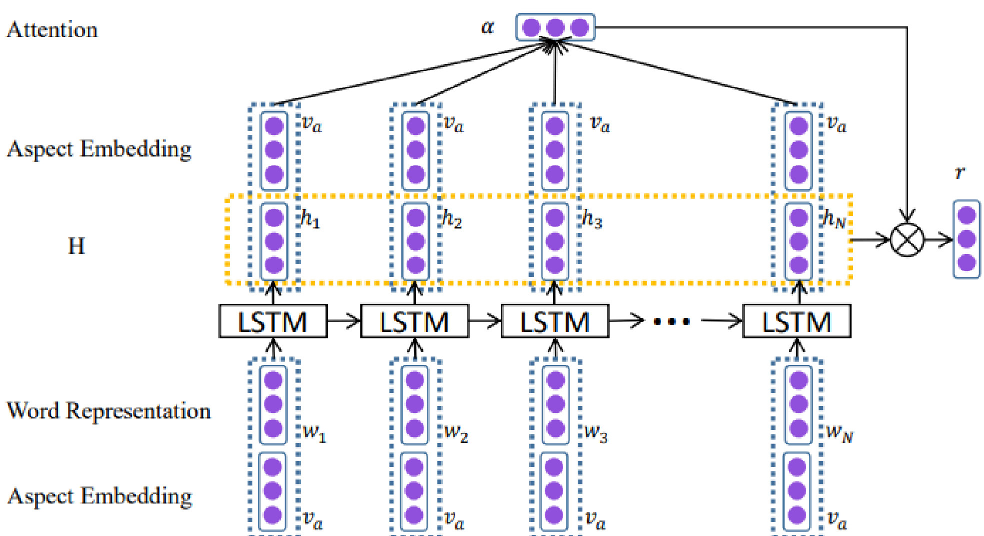
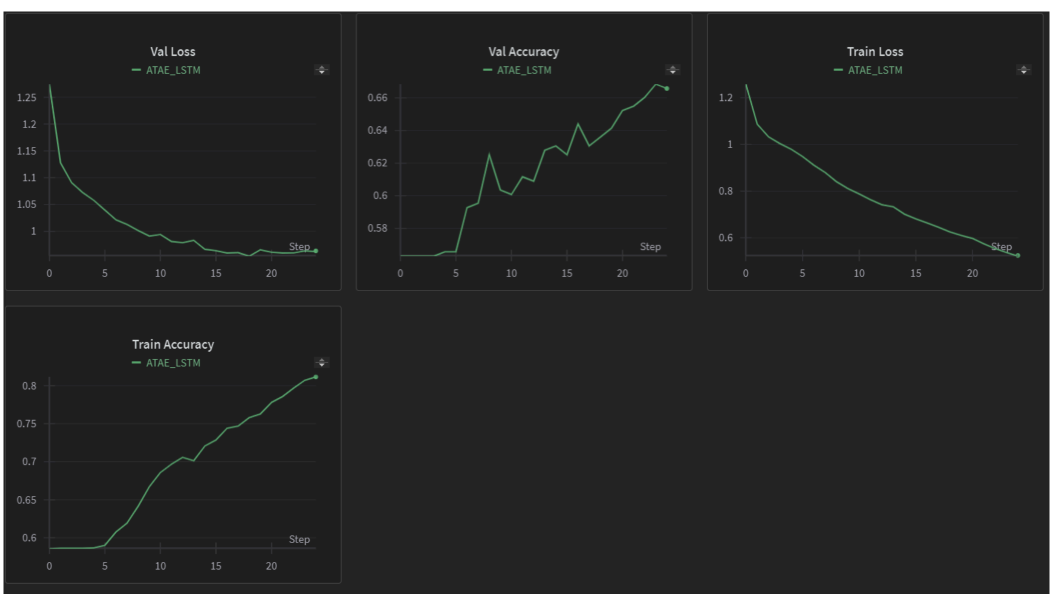
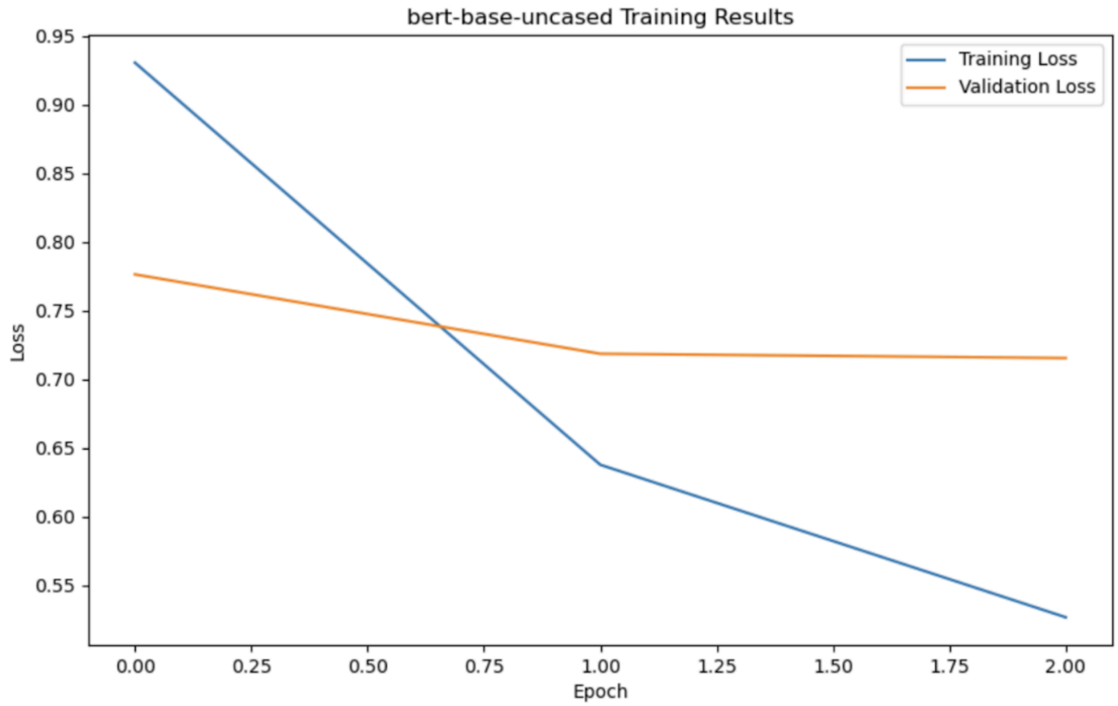


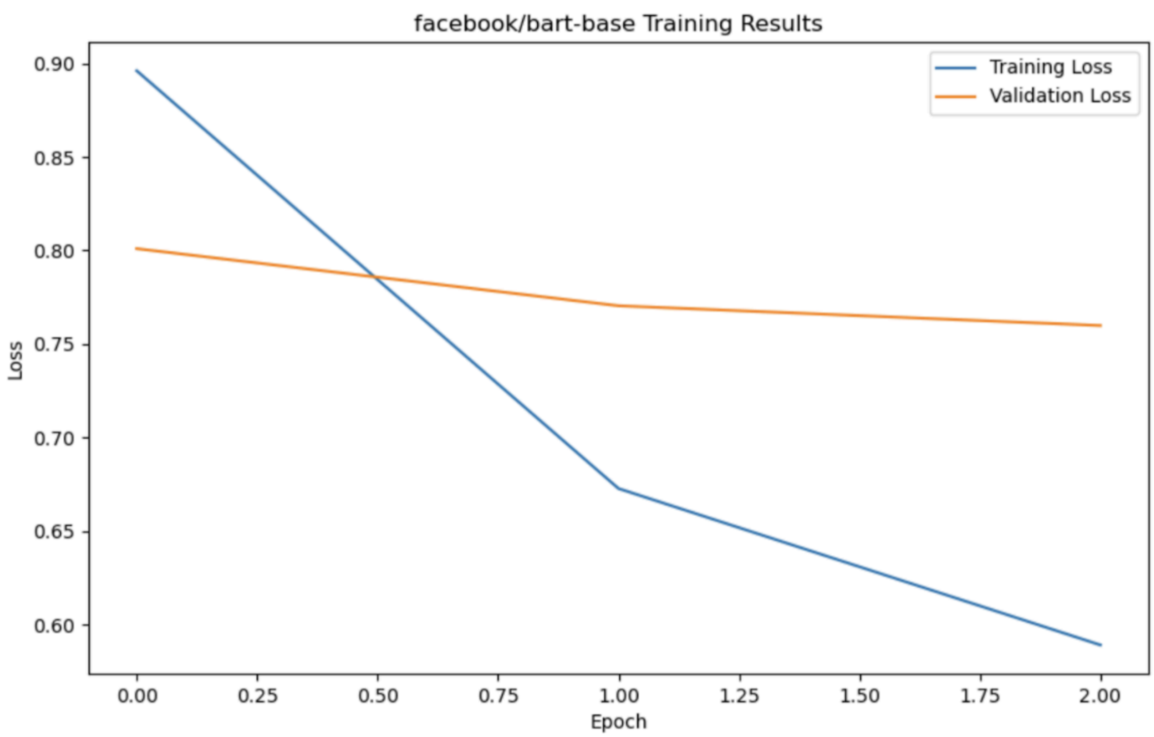
Figure 3: The Architecture of Attention-based LSTM with Aspect Embedding. The aspect embeddings have been take as input along with the word embeddings. $\{w_1, w_2, \ldots, w_N\}$ represent the word vector in a sentence whose length is N. v_a represents the aspect embedding. α is the attention weight. $\{h_1, h_2, \ldots, h_N\}$ is the hidden vector.

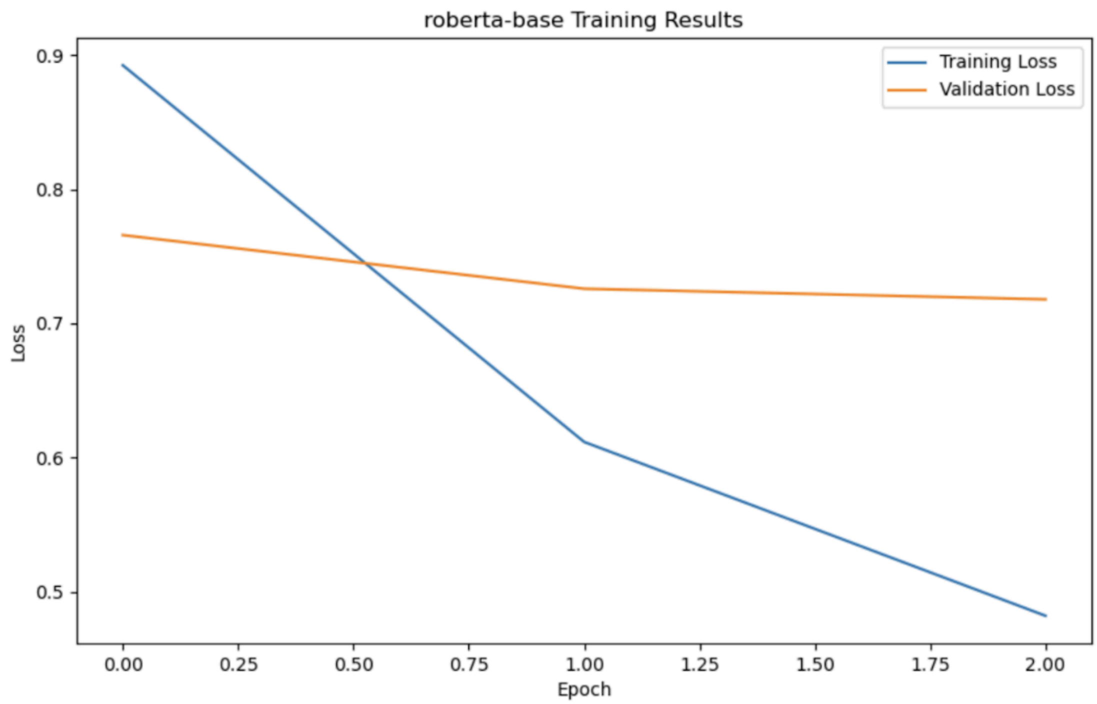
Model Training

```
---------------- TRAINING ATAE_LSTM ----------------
Epoch [1/25] -> Train Loss: 1.2587, Train Acc: 0.5863 | Val Loss: 1.2745, Val Acc: 0.5633
Epoch [2/25] -> Train Loss: 1.0876, Train Acc: 0.5870 | Val Loss: 1.1285, Val Acc: 0.5633
Epoch [3/25] -> Train Loss: 1.0339, Train Acc: 0.5870 | Val Loss: 1.0915, Val Acc: 0.5633
Epoch [4/25] -> Train Loss: 1.0049, Train Acc: 0.5870 | Val Loss: 1.0731, Val Acc: 0.5633
Epoch [5/25] -> Train Loss: 0.9807, Train Acc: 0.5876 | Val Loss: 1.0587, Val Acc: 0.5660
Epoch [6/25] -> Train Loss: 0.9499, Train Acc: 0.5910 | Val Loss: 1.0406, Val Acc: 0.5660
Epoch [7/25] -> Train Loss: 0.9122, Train Acc: 0.6086 | Val Loss: 1.0224, Val Acc: 0.5930
Epoch [8/25] -> Train Loss: 0.8814, Train Acc: 0.6201 | Val Loss: 1.0137, Val Acc: 0.5957
Epoch [9/25] -> Train Loss: 0.8421, Train Acc: 0.6424 | Val Loss: 1.0023, Val Acc: 0.6253
Epoch [10/25] -> Train Loss: 0.8128, Train Acc: 0.6677 | Val Loss: 0.9923, Val Acc: 0.6038
Epoch [11/25] -> Train Loss: 0.7896, Train Acc: 0.6863 | Val Loss: 0.9951, Val Acc: 0.6011
Epoch [12/25] -> Train Loss: 0.7654, Train Acc: 0.6974 | Val Loss: 0.9822, Val Acc: 0.6119
Epoch [13/25] -> Train Loss: 0.7440, Train Acc: 0.7062 | Val Loss: 0.9800, Val Acc: 0.6092
Epoch [14/25] -> Train Loss: 0.7352, Train Acc: 0.7018 | Val Loss: 0.9843, Val Acc: 0.6280
Epoch [15/25] -> Train Loss: 0.7034, Train Acc: 0.7210 | Val Loss: 0.9677, Val Acc: 0.6307
Epoch [16/25] -> Train Loss: 0.6833, Train Acc: 0.7288 | Val Loss: 0.9651, Val Acc: 0.6253
Epoch [17/25] -> Train Loss: 0.6656, Train Acc: 0.7443 | Val Loss: 0.9609, Val Acc: 0.6442
Epoch [18/25] -> Train Loss: 0.6474, Train Acc: 0.7470 | Val Loss: 0.9614, Val Acc: 0.6307
Epoch [19/25] -> Train Loss: 0.6276, Train Acc: 0.7582 | Val Loss: 0.9549, Val Acc: 0.6361
Epoch [20/25] -> Train Loss: 0.6128, Train Acc: 0.7629 | Val Loss: 0.9667, Val Acc: 0.6415
Epoch [21/25] -> Train Loss: 0.5997, Train Acc: 0.7781 | Val Loss: 0.9625, Val Acc: 0.6523
Epoch [22/25] -> Train Loss: 0.5769, Train Acc: 0.7859 | Val Loss: 0.9608, Val Acc: 0.6550
Epoch [23/25] -> Train Loss: 0.5566, Train Acc: 0.7970 | Val Loss: 0.9610, Val Acc: 0.6604
Epoch [24/25] -> Train Loss: 0.5399, Train Acc: 0.8072 | Val Loss: 0.9647, Val Acc: 0.6685
Epoch [25/25] -> Train Loss: 0.5242, Train Acc: 0.8116 | Val Loss: 0.9636, Val Acc: 0.6658
---------------- TRAINING COMPLETED ----------------
```




bert-base-uncased Training Results
Training Loss
Validation Loss
Loss
Epoch
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


facebook/bart-base Training Results
Training Loss
Validation Loss
Loss
Epoch
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



Fine-tuned model Val Accuracy

BERT - 0.71

BART - 0.661

RoBERTa - 0.682

Task 3 - Fine-tuning SpanBERT and SpanBERT-CRF

1. Dataset Description and Preprocessing

Dataset: SQuAD v2

The **Stanford Question Answering Dataset (SQuAD v2)** is a reading comprehension dataset that consists of:

- **Questions**: Natural language queries.

- **Context passages**: Paragraphs from which answers are extracted.
- **Answers**: Text spans within the context.
- **Unanswerable questions**: Introduced in SQuAD v2 to add complexity.

Preprocessing Steps:

1. **Data Loading and Splitting:**
   - The dataset is loaded using the `datasets` library.
   - Training data: 18,000 samples.
   - Validation data: 5,000 samples.
2. **Tokenization:**
   - Tokenization is performed using the `AutoTokenizer` from the **SpanBERT-base-cased** model.
   - The context and questions are tokenized together with a maximum sequence length of 512.
3. **Target Labeling:**
   - Labels are created by marking answer spans within the tokenized input.
   - If an answer is available, corresponding token indices are marked.
   - If no answer exists, the label is set to -1.
4. **Padding and Batching:**
   - Input sequences are padded to the maximum sequence length in the batch.
   - A **collate function** ensures proper alignment of input sequences, attention masks, and targets.
   - A **PyTorch DataLoader** is used to prepare mini-batches.

2. Model Choices and Hyperparameter Justification

SpanBERT-CRF Model

- **Encoder:** SpanBERT (Pretrained model fine-tuned on span-based tasks).
- **CRF Layer:** Conditional Random Field (CRF) is added on top of SpanBERT for structured prediction.
- **Classification Layer:** A linear layer transforms hidden states into emission scores for CRF.
- **Loss Function:** CRF negative log-likelihood loss for better sequence learning.

Justification:

- **SpanBERT** is optimized for span-based tasks like QA, making it a strong baseline.
- **CRF Layer** improves entity-level consistency in predictions.
- **Hyperparameters:**

- **Learning rate:** 5e-5 (Optimized for transformers using AdamW optimizer).
- **Batch size:** 8 (Balances memory efficiency and stable gradients).
- **Epochs:** 6 (Ensures sufficient learning without overfitting).

SpanBERT Baseline

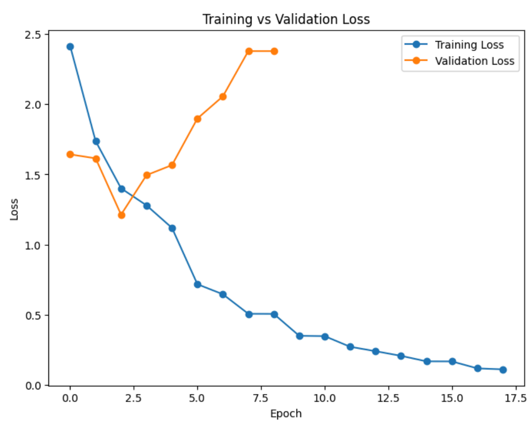
- **Same encoder** as SpanBERT-CRF.
- **Softmax classifier** instead of CRF.
- **Cross-entropy loss** for token classification.

3. Training and Validation Plots

The training and validation losses over epochs are shown in the plot below:

SpanBERT-CRF

SpanBERT



Epoch	Training Loss	Validation Loss	Exact Match
1	1.737500	1.642858	21.203722
2	1.279500	1.614799	29.023956
3	0.717100	1.213210	27.024352
4	0.505900	1.494799	27.162938
5	0.347300	1.566618	30.726589
6	0.239900	1.898837	26.608592
7	0.167600	2.056581	30.310830
8	0.110600	2.379298	29.934666

Training for SpanBERT model

4. Comparative Analysis: SpanBERT-CRF vs. SpanBERT

Model	Exact Match Score
SpanBERT	29.93%
SpanBERT-CRF	52.66%

Key Findings:

1. **SpanBERT-CRF achieves a higher exact match score (52.66%)** compared to SpanBERT (29.93%).
2. **CRF improves token alignment**, leading to better span extraction for question answering.
3. **SpanBERT alone struggles** with capturing interdependencies among tokens compared to the structured prediction approach of CRF.
4. **Training Complexity:**
   - SpanBERT-CRF is computationally heavier due to CRF decoding.
   - SpanBERT (softmax) is simpler but less effective in handling structured dependencies.

5. Conclusion

- SpanBERT-CRF improves **structured predictions** in extractive question answering.
- The addition of CRF enhances token-level consistency, leading to **higher exact match scores**.

```
Evaluation Results: {'eval_loss': 2.379298448562622, 'eval_exact_match': 29.934666402692535, 'eval
_runtime': 1443.264, 'eval_samples_per_second': 3.5, 'eval_steps_per_second': 0.219, 'epoch': 8.0}
```

```
Final Validation Metrics: {'exact_match': 52.66}
Model Saved Successfully !
```