NLP Assignment 2

Manan Aggarwal 2022273
Shobhit Raj 2022482
Souparno Ghose 2022506

March 16, 2025

Task 1

Pre-processing Steps Applied

The preprocessing involves the following steps:

1. **Tokenization:** The input sentence is split into tokens using whitespace as a delimiter.
   - Input: `"All the money went into the interior decoration, none of it went to the chefs."`
   - After processing: `["All", "the", "money", "went", "into", "the", "interior", "decoration,", "none", "of", "it", "went", "to", "the", "chefs."]`
2. **Finding Token Positions:** Character positions of tokens are recorded to map them accurately with aspect terms.
   - Input: `["All", "the", "money", "went", "into", "the", "interior", "decoration,", "none", "of", "it", "went", "to", "the", "chefs."]`
   - Length prefix: `[0, 4, 8, 14, 19, 24, 28, 37, 49, 54, 57, 60, 65, 68, 72]`
3. **Aspect Term Alignment:** The provided "from" and "to" character indices are used to locate aspect terms in the tokenized sentence.
   - "interior decoration" (Character positions 28–47)
   - "chefs" (Character positions 72–77)
4. **BIO Encoding:** Each token is labeled as:
   - "B" (Beginning) if it is the first token of an aspect term.
   - "I" (Inside) if it is a subsequent token in the aspect term.
   - "O" (Outside) if it is not part of an aspect term.

   - Input: `["All", "the", "money", "went", "into", "the", "interior", "decoration,", "none", "of", "it", "went", "to", "the", "chefs."]`
   - Labels: `['O', 'O', 'O', 'O', 'O', 'O', 'B', 'I', 'O', 'O', 'O', 'O', 'O', 'O', 'B']`
5. **Final Output Formatting:** The transformed data, including tokens, BIO labels, and aspect terms, is stored in JSON format.

This preprocessing step effectively converts raw text into a structured format, preparing it for deep learning-based Aspect Term Extraction models.

Model Architectures and Hyperparameters Used

We trained four different models for **Aspect Term Extraction**:

- **Recurrent Neural Network (RNN)**
- **Gated Recurrent Unit (GRU)**

Each model was trained separately with `GloVe` (300d) and `fastText` (300d) embeddings, resulting in four trained models. The following hyperparameters were used:

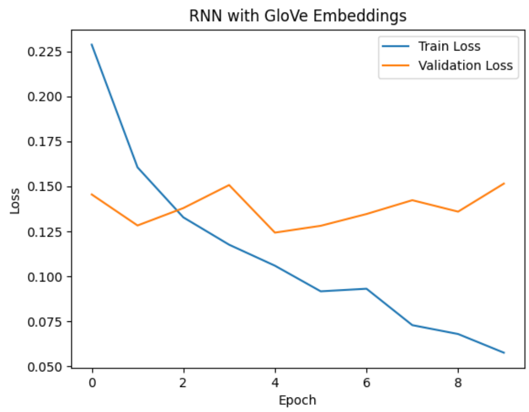
Model	Embedding	Hidden Dim	Learning Rate	Batch Size	Optimizer
RNN	GloVe	128	0.01	32	Adam
RNN	fastText	128	0.01	32	Adam
GRU	GloVe	128	0.01	32	Adam
GRU	fastText	128	0.01	32	Adam

Table 1: Hyperparameters used for different models

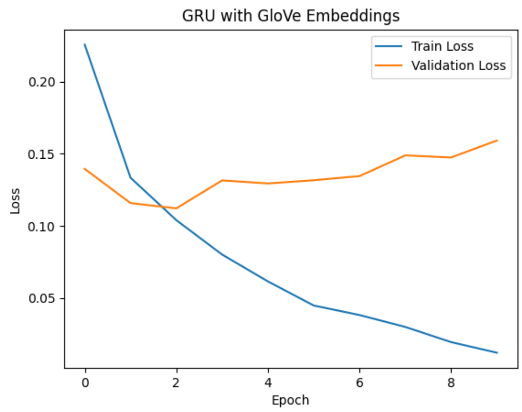
The models were trained with `Cross-Entropy Loss`, using `ignore_index=-100` to ignore padded tokens. Training was conducted for **10 epochs** using the `Adam` optimizer. Additionally, the models were check-pointed on the basis of **Harmonic Mean of Chunk level and Tag level F1 Scores**, helping the model grow in both the aspects. Thereby, giving a more robust performance estimator.

Training and Validation Loss Plots

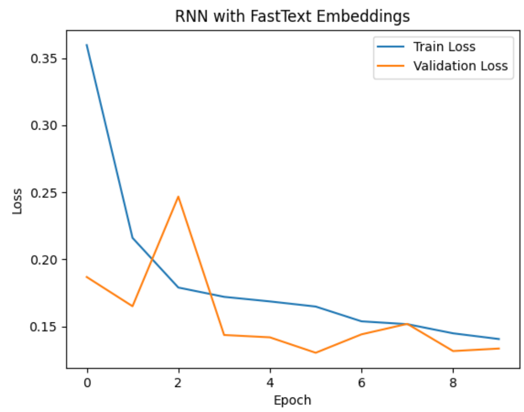
The training and validation loss curves for each of the four models are shown in the figure below. These plots illustrate how the models converge over **10 epochs**.



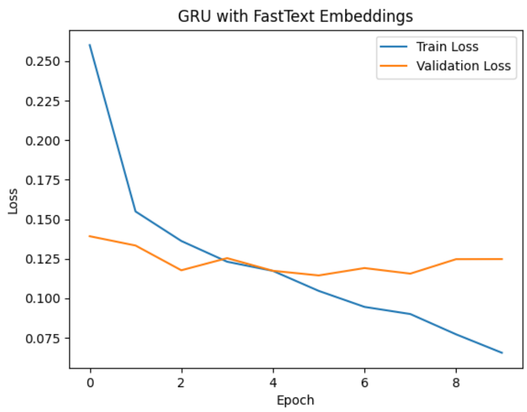
(a) `RNN` with `GloVe` embeddings



(b) `GRU` with `GloVe` embeddings



(c) `RNN` with `fastText` embeddings



(d) `GRU` with `fastText` embeddings

Figure 1: Training and validation loss curves for different models

Performance comparison of all models

We trained four models (`RNN` and `GRU` with `GloVe` and `fastText` embeddings) and evaluated their performance using **F1-score** at chunk and token levels.

- `GRU` models outperformed `RNN` models, indicating `GRU`'s superior ability to capture sequential dependencies.
- `fastText` vs. `GloVe`: `GloVe`-based models performed better, likely due to their robust word vector representation.
- `RNN` models had higher validation loss fluctuations, showing potential overfitting or weaker generalization.
- **GRU with GloVe** achieved the highest **F1-score**, making it the best-performing model.

Model	Train F1 (Chunk)	Val F1 (Chunk)	Train F1 (Token)	Val F1 (Token)
`RNN (fastText)`	0.7289	0.7239	0.9495	0.9521
`RNN (GloVe)`	0.8761	0.7425	0.9821	0.9587
`GRU (fastText)`	0.8627	0.7549	0.9786	0.9579
`GRU (GloVe)`	**0.9676**	**0.7797**	**0.9949**	**0.9619**

Table 2: Performance comparison of all models using F1-score at chunk and token levels.

Best-performing model and its evaluation

The best-performing model was **GRU with GloVe**, achieving the highest validation **F1-score**:

- **Chunk-level**: 0.7797
- **Token-level**: 0.9619
- Demonstrated the best generalization ability, as seen from its stable loss and F1-score progression.
- Low training loss and high token-level accuracy suggest optimal feature extraction for sequence labeling.

To further assess its performance, a function was implemented to load the best model and compute F1-scores on `test.json`.

Task 2

Pre-processing Steps Applied

The preprocessing involves the following steps:

1. **Tokenization:** The sentence is split into individual tokens based on spaces and punctuations are removed.

   - Input: `"All the money went into the interior decoration, none of it went to the chefs."`
   - After processing: `["All", "the", "money", "went", "into", "the", "interior", "decoration", "none", "of", "it", "went", "to", "the", "chefs"]`

2. **Finding Token Positions:** Character positions of tokens are recorded to map them accurately with aspect terms.

   - Input: `["All", "the", "money", "went", "into", "the", "interior", "decoration", "none", "of", "it", "went", "to", "the", "chefs"]`
   - Length prefix: `[0, 4, 8, 14, 19, 24, 28, 37, 49, 54, 57, 60, 65, 68, 72]`

3. **Aspect Term Alignment:** The provided "from" and "to" character indices are used to locate aspect terms in the tokenized sentence.

   - "interior decoration" (Character positions 28–47)
   - "chefs" (Character positions 72–77)

4. **Final Formatting:** Each aspect term is stored as a separate instance with its tokens, polarity, aspect_term, and index.

This preprocessing ensures that every aspect term is mapped to the correct index in the tokenized sentence, allowing for accurate aspect-based sentiment analysis.

Model Architectures and Hyperparameters

Our best-performing model is an **Enhanced LSTM with BERT embeddings**, which integrates several advanced components to enhance sentiment classification accuracy. Below are the key elements of our model:

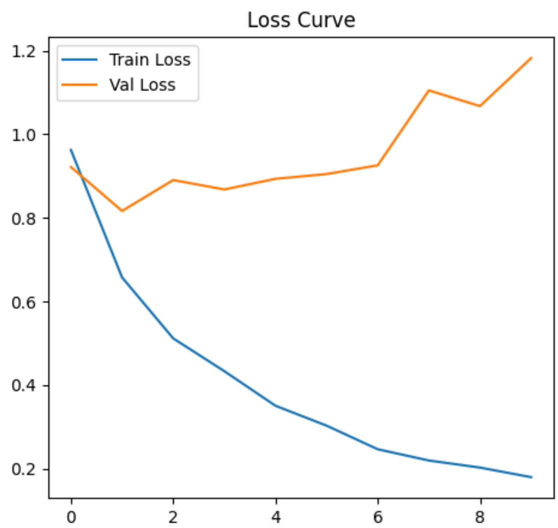
- **BERT Embeddings:** Provide contextualized word representations, improving sentiment analysis accuracy.
- **Bidirectional LSTM:** Captures both left and right context for a more comprehensive feature extraction.
- **Self-Attention Mechanism:** Focuses on crucial parts of the sentence related to the aspect.
- **Residual and Highway Connections:** Improve gradient flow, prevent vanishing gradients, and enable better feature transformation.
- **Batch Normalization:** Stabilizes training, accelerates convergence, and enhances generalization.
- **Aspect Projection:** Aligns aspect representation with LSTM outputs for better contextual understanding.
- **Dropout Regularization:** Reduces overfitting and enhances model robustness.

Hyperparameters:

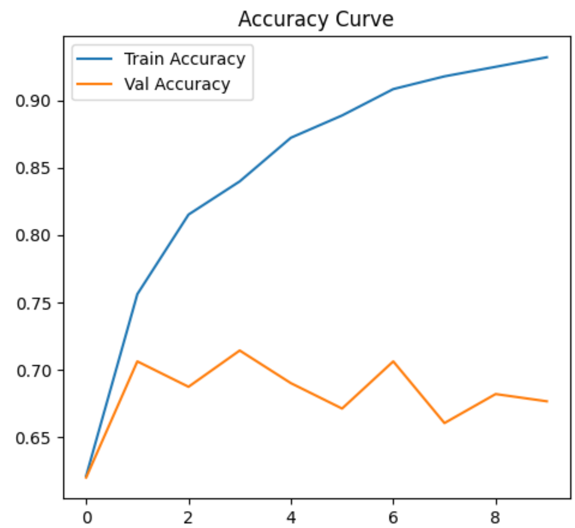
- **Hidden Dimension:** `128`
- **Batch Size:** `64`
- **Epochs:** `10`
- **Learning Rate:** `0.0001`
- **Optimizer:** `Adam`
- **Loss Function:** `CrossEntropyLoss`

Training and Validation Loss Plots

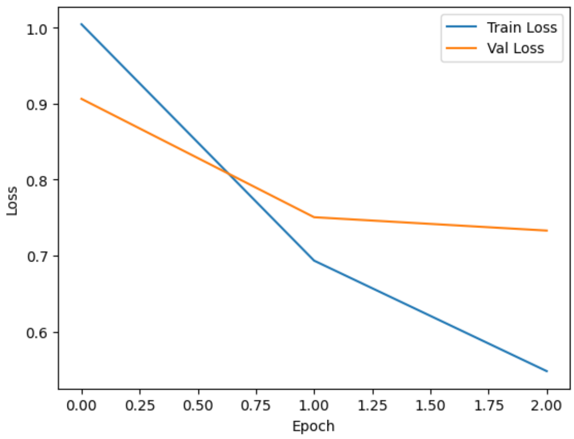
The following plots illustrate the training and validation loss curves, as well as accuracy curves, for all four models: **Enhanced LSTM with BERT Embeddings**, **BERT Fine-Tuned**, **BART Fine-Tuned**, and **RoBERTa Fine-Tuned**.



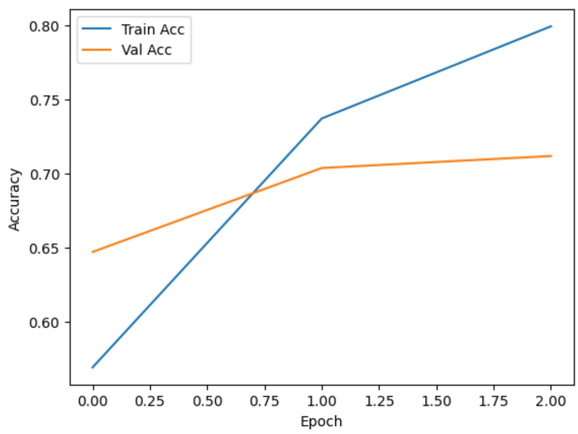
(a) Training and Val Loss for `LSTM + BERT`



(b) Training and Val Accuracy for `LSTM + BERT`

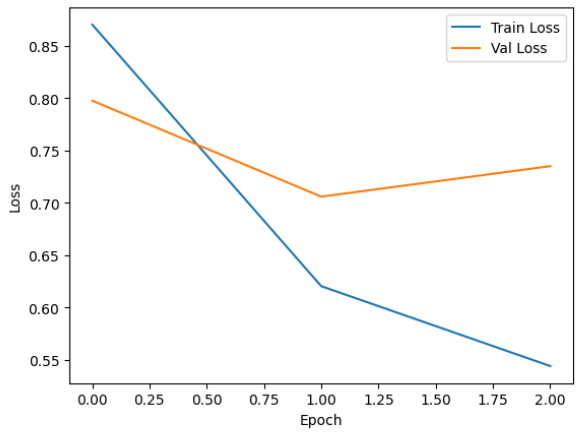


(c) Training and Validation Loss for `BERT Fine-Tuned`

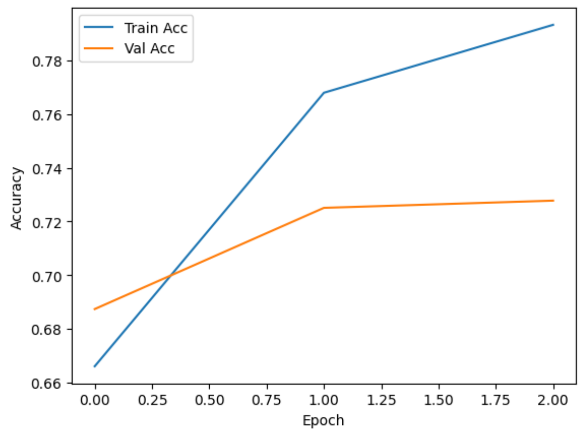


(d) Training and Validation Accuracy for `BERT Fine-Tuned`

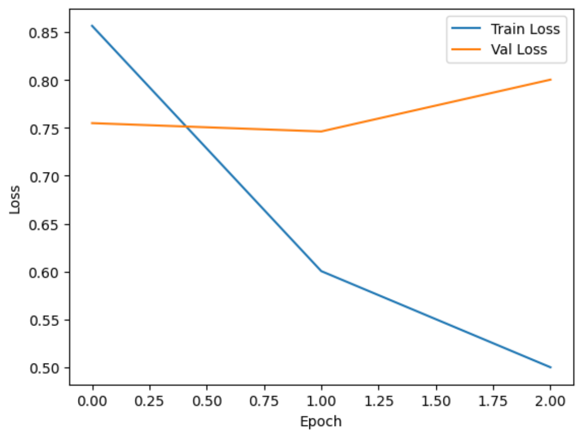
Figure 2: Training and validation loss/accuracy plots for `LSTM + BERT` and `BERT Fine-Tuned`



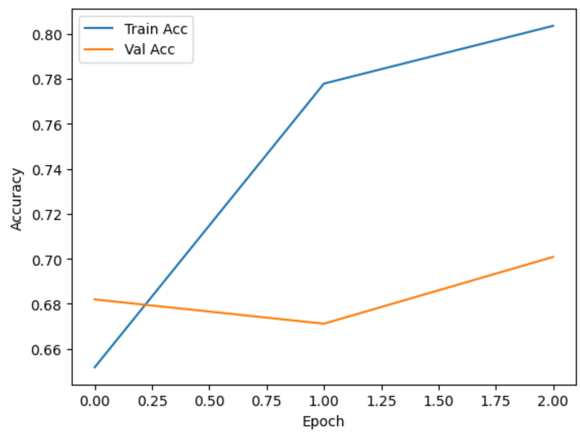
(a) Training and Validation Loss for `BART Fine-Tuned`



(b) Training and Validation Accuracy for `BART Fine-Tuned`



(c) Training and Validation Loss for `RoBERTa Fine-Tuned`



(d) Training and Validation Accuracy for `RoBERTa Fine-Tuned`

Figure 3: Training and validation loss/accuracy plots for `BART Fine-Tuned` and `RoBERTa Fine-Tuned`

Evaluation Metrics on Validation Set

The table below summarizes the training and validation accuracy for each model:

Model	Training Accuracy (%)	Validation Accuracy (%)
`Enhanced LSTM + BERT`	83.99	71.43
`BERT Fine-Tuned`	79.91	71.16
`BART Fine-Tuned`	79.33	72.78
`RoBERTa Fine-Tuned`	80.34	70.08

Table 3: Training and validation accuracy for different models

Analysis:

- `BART Fine-Tuned` achieved the highest validation accuracy (**72.78%**), showcasing its strong sequence-to-sequence modeling capabilities.
- `Enhanced LSTM + BERT` demonstrated strong training accuracy (**83.99%**) but with validation accuracy (**65.50%**), indicating better generalization while still benefiting from further regularization.
- `BERT Fine-Tuned` and `RoBERTa Fine-Tuned` performed similarly, with `BERT` slightly outperforming `RoBERTa`.

To further assess its performance, a function was implemented to load the best model and compute accuracy on `test.json`.

Task 3

Dataset and Preprocessing Steps

The **Stanford Question Answering Dataset v2 (SQuAD v2)** was used for fine-tuning. This dataset consists of question-answer pairs, including cases where no valid answer exists in the given passage. Due to computational constraints, a subset of **15,000** samples from the training set was used, while the entire validation set was retained.

Preprocessing steps included:

- **Tokenization** using `SpanBERT/spanbert-base-cased` tokenizer with truncation, padding, and a stride of **128**.
- Mapping tokenized outputs to the original dataset indices.
- Handling unanswerable questions by assigning `(0,0)` for start and end positions.
- Converting character-level answer spans to token-level positions.

Model Choices and Hyperparameters

SpanBERT:

- As pre-trained on span-based objectives, SpanBERT is suitable for extractive QA tasks.
- Fine-tuned using the `Hugging Face Trainer API`.

Hyperparameters:

- **Learning rate:** `3e-5`
- **Epochs:** `6`
- **Batch size:** `16` (train), `8` (eval)
- **Weight decay:** `0.01`
- **Best model selection based on** `Exact Match (EM)` score.

SpanBERT-CRF:

- Added a **CRF layer** to the SpanBERT model to better capture dependencies in answer spans.
- Loss function combines **SpanBERT's QA loss** and **CRF loss**.
- Custom training loop implemented using the `Trainer API`.
- **Same hyperparameters** as SpanBERT.

Training and Validation Plots

SpanBERT:

Epoch	Training Loss	Validation Loss	Exact Match Non-Empty	Exact Match
1	2.3459	2.1815	46.30	26.49
2	1.6314	2.1227	50.51	34.84
3	1.3089	2.0452	50.24	37.66
4	1.1167	2.2077	51.61	41.02
5	0.9635	2.2647	50.00	40.11
6	0.8608	2.4478	50.40	41.54

Table 4: Training and Validation Loss for SpanBERT

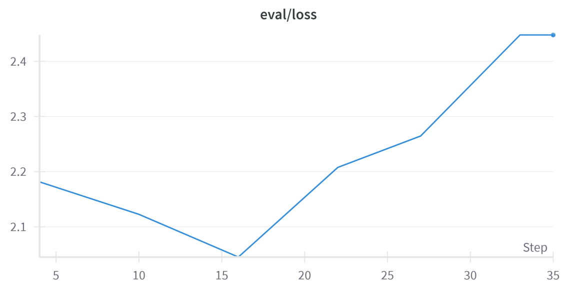
SpanBERT-CRF:

Epoch	Training Loss	Validation Loss	Exact Match Non-Empty	Exact Match
1	14.0960	12.7828	39.23	19.11
2	9.2472	10.7294	43.01	25.61
3	6.6786	10.3721	46.30	29.39
4	5.2687	10.6434	49.55	30.45
5	4.1172	11.1052	46.00	28.84
6	3.4708	11.9017	47.13	28.34

Table 5: Training and Validation Loss for SpanBERT-CRF

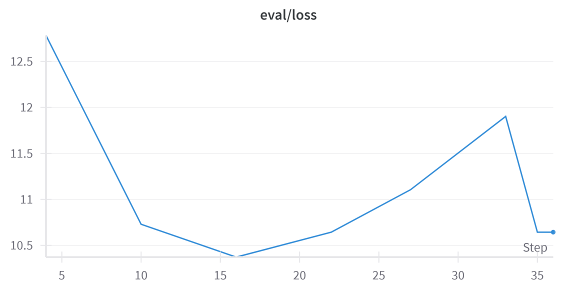
The training and validation plots of both models are given below :

(a) `SpanBERT` Training Loss



(b) `SpanBERT` Validation Loss

(c) `SpanBERT-CRF` Training Loss



(d) `SpanBERT-CRF` Validation Loss

Figure 4: Training and validation loss curves for `SpanBERT` and `SpanBERT-CRF` models

Observations:

- **Training loss** steadily decreased across epochs for both models.
- **Validation loss** initially decreased but later showed a slight increase, indicating possible **overfitting**.

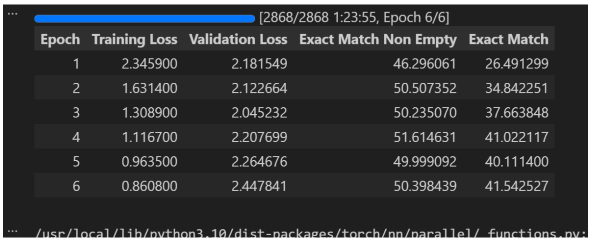
Comparative Analysis

- **SpanBERT** outperformed **SpanBERT-CRF** in terms of `Exact Match (EM)` scores for both **all examples** and **non-empty answers**.
- While **SpanBERT-CRF** showed improvements in `loss reduction`, it did not translate to a better `EM score`, likely due to increased model complexity and possible **overfitting**—suggesting that **CRF did not generalize well** in this task.
- **CRF-based models** may require additional `hyperparameter tuning` and `regularization` to enhance performance in extractive **QA tasks**.
- It is worth noting that **CRF-based models** achieved `EM` scores comparable to those of **Non-CRF-based** models on examples with valid answers. This indicates that both approaches are similarly effective when a valid answer is present. However,**CRF-based models** models tend to **under-perform** in scenarios where **no valid answer exists**, likely due to their reliance on structured prediction mechanisms.

Exact-Match Scores on Validation Set

- **SpanBERT:**
  - `Exact Match (All Examples)`: **41.54%**
  - `Exact Match (Non-Empty)`: **50.39%**
- **SpanBERT-CRF:**
  - `Exact Match (All Examples)`: **30.45%**
  - `Exact Match (Non-Empty)`: **49.54%**

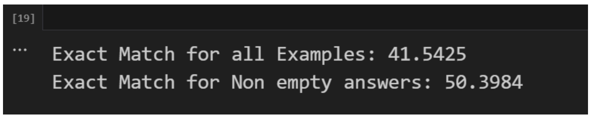
Screenshots from output cells in `task3.ipynb`, showing per-epoch metrics and final exact-match results for `SpanBERT` and `SpanBERT-CRF`, are given below.



[2868/2868 1:23:55, Epoch 6/6]

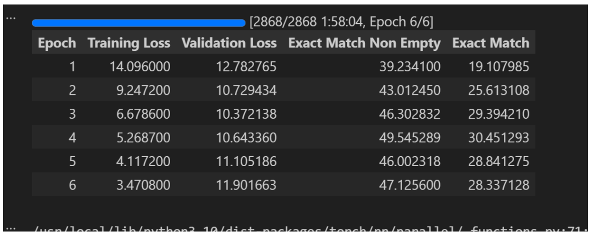
Epoch	Training Loss	Validation Loss	Exact Match Non Empty	Exact Match
1	2.345900	2.181549	46.296061	26.491299
2	1.631400	2.122664	50.507352	34.842251
3	1.308900	2.045232	50.235070	37.663848
4	1.116700	2.207699	51.614631	41.022117
5	0.963500	2.264676	49.999092	40.111400
6	0.860800	2.447841	50.398439	41.542527

(a) Per-epoch metrics for `SpanBERT`



```
Exact Match for all Examples: 41.5425
Exact Match for Non empty answers: 50.3984
```

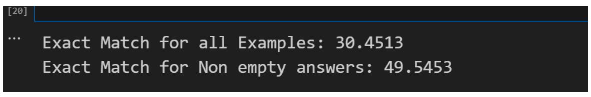
(b) Final `Exact Match` results for `SpanBERT`



[2868/2868 1:58:04, Epoch 6/6]

Epoch	Training Loss	Validation Loss	Exact Match Non Empty	Exact Match
1	14.096000	12.782765	39.234100	19.107985
2	9.247200	10.729434	43.012450	25.613108
3	6.678600	10.372138	46.302832	29.394210
4	5.268700	10.643360	49.545289	30.451293
5	4.117200	11.105186	46.002318	28.841275
6	3.470800	11.901663	47.125600	28.337128

(c) Per-epoch metrics for `SpanBERT-CRF`



```
Exact Match for all Examples: 30.4513
Exact Match for Non empty answers: 49.5453
```

(d) Final `Exact Match` results for `SpanBERT-CRF`

Figure 5: Screenshots of per-epoch metrics and final `Exact Match` results for `SpanBERT` and `SpanBERT-CRF`.

Individual Contribution

- **Task 1:** Shobhit Raj
- **Task 2:** Manan Aggarwal
- **Task 3:** Souparno Ghose
- **Report Writing:** Everyone contributed equally

References

1. Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. Retrieved from `https://nlp.stanford.edu/projects/glove/`
2. fastText. (n.d.). English word vectors. Retrieved from `https://fasttext.cc/docs/en/english-vectors.html`
3. Sighsmile. (n.d.). CoNLL evaluation script for sequence tagging tasks. Retrieved from `https://github.com/sighsmile/conlleval`
4. PyTorch Documentation. (n.d.). `torch.nn.RNN`. Retrieved from `https://pytorch.org/docs/stable/generated/torch.nn.RNN.html`
5. PyTorch Documentation. (n.d.). `torch.nn.GRU`. Retrieved from `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`
6. Rajpurkar, P., Jia, R., & Liang, P. (2018). Know What You Don't Know: Unanswerable Questions for SQuAD. Retrieved from `https://arxiv.org/abs/1806.03822`
7. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is All You Need. Retrieved from `https://arxiv.org/abs/1706.03762`
8. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780.
9. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved from `https://arxiv.org/abs/1810.04805`
10. Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., & Levy, O. (2020). SpanBERT: Improving Pre-training by Representing and Predicting Spans. Retrieved from `https://arxiv.org/abs/1907.10529`
11. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. Retrieved from `https://arxiv.org/abs/1910.03771`
12. Hugging Face. (n.d.). SpanBERT Model. Retrieved from `https://huggingface.co/SpanBERT/spanbert-base-cased`
13. Stanford Question Answering Dataset (SQuAD). (n.d.). Retrieved from `https://rajpurkar.github.io/SQuAD-explorer/`
14. PyTorch Documentation. (n.d.). Conditional Random Fields (CRF). Retrieved from `https://pytorch-crf.readthedocs.io/en/stable/`