Deep Learning

- Marri Bharadwaj (B21CS045)

LSTM with ATTENTION

Binary Classification

Dataset

	review	CLASS
0	I've joined IMDb so people know what a great f...	1
1	This is a so called 'feel-good' movies, howeve...	1
2	Somehow they summed up the 60's, ten years tha...	0
3	OK..this movie could have been soooo good! All...	0
4	I don't know why I keep doing this to myself!!...	0

Data Preprocessing

For the binary sentiment analysis task using the above *IMDB* dataset, I began by processing the dataset to ensure it was ready for training and evaluation. Positive and negative reviews were extracted separately from their respective directories (*/pos* and */neg*) in the dataset. Each review was read from the files and stored in separate lists: *Positive_IMDB* for positive reviews and *Negative_IMDB* for negative reviews.

I combined the two lists into a single dataset and assigned labels to distinguish between positive and negative sentiments. Positive reviews were labelled as *1*, and negative reviews were labelled as *0*. The combined dataset was shuffled to ensure randomness and then converted into a Pandas DataFrame for easier handling and manipulation.

Next, I divided the data into three sets: training, validation, and testing. Using an *80-10-10* split:

- 80% of the data was allocated for training.
- The remaining 20% was split equally for validation and testing.

The training, validation, and testing data were prepared as separate DataFrames with columns renamed for clarity (*text* for reviews and *label* for sentiment). Additionally, all DataFrames were reset to ensure consistent indexing.

Input Features, Tokenisation and One-Hot Encoding

The preprocessing of the *IMDB* dataset for binary classification involved careful handling of tokenisation, vocabulary creation, and uniform sequence representation. To begin with, the training data was tokenised using the *spaCy* English tokeniser. This allowed me to break down the text into individual tokens and calculate their frequencies. To ensure a manageable vocabulary size, only words with a frequency of 5 or more were retained, while two special tokens, *"UNK"* and *"PAD"*, were added. The *"UNK"* token handled words that were not present in the vocabulary, while *"PAD"* was used for padding sequences to a fixed length. This step was critical for maintaining consistency during batching and for handling unseen words during testing and inference.

Once the vocabulary was established, the reviews were tokenised by mapping each word to its corresponding vocabulary entry. Words that were not part of the vocabulary were replaced with the *"UNK"* token. After tokenisation, I determined the average token length from the training data to set a maximum sequence length. Any reviews exceeding this length were truncated, while shorter reviews were padded using the *"PAD"* token to ensure uniform sequence lengths across all samples. This padding approach was applied consistently to all three dataset splits—training, validation, and testing—providing a standardised input format for the model.

```
Maximum length is 272

Displaying few training samples
                                                text  label  \
0  This review contains spoilers for those who ar...      1
1  This is by far one of the most boring and horr...      0
2  This movie starts out with an execution of a p...      0
3  I bought the DVD of Before Sunset and saw it f...      1
4  I am a sucker for films like this. Films that ...      1

                                              tokens  \
0  [this, review, contains, spoilers, for, those,...
1  [this, is, by, far, one, of, the, most, boring...
2  [this, movie, starts, out, with, an, execution...
3  [i, bought, the, dvd, of, before, sunset, and,...
4  [i, am, a, sucker, for, films, like, this, ., ...

                                       padded_tokens
0  [this, review, contains, spoilers, for, those,...
1  [this, is, by, far, one, of, the, most, boring...
2  [this, movie, starts, out, with, an, execution...
3  [i, bought, the, dvd, of, before, sunset, and,...
4  [i, am, a, sucker, for, films, like, this, ., ...
```

The processed datasets, including the tokenised and padded reviews, were then saved into pickle files to facilitate quick reloading during subsequent stages of the project. The vocabulary was serialised similarly to ensure consistency across training and evaluation phases. A custom dataset class, *dataset*, was implemented to manage this tokenised data. This class was designed to retrieve the processed tokens and their corresponding labels, converting the tokens into numerical indices using the vocabulary. It also supported

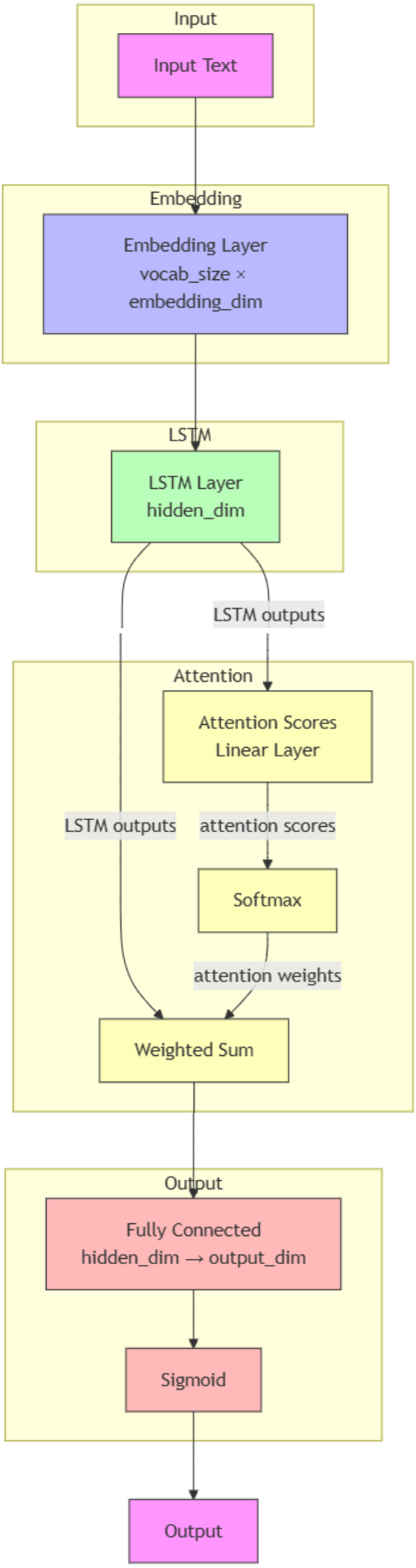
padding and ensured the data was returned in a tensor format suitable for *PyTorch*. Importantly, the class included a method to retrieve the original tokens, which would be useful for interpretability and debugging.

```
Number of training examples: 20000
Number of validation examples: 2500
Number of testing examples: 2500
```

To prepare the data for model training, *PyTorch dataloaders* were created by me for the training, validation, and test datasets. These loaders efficiently handled data batching, with a batch size of 64, and ensured that the training data was shuffled at each epoch.

```
Training data...
                                                text  label  \
0  This review contains spoilers for those who ar...      1
1  This is by far one of the most boring and horr...      0
2  This movie starts out with an execution of a p...      0
3  I bought the DVD of Before Sunset and saw it f...      1
4  I am a sucker for films like this. Films that ...      1

                                              tokens  \
0  [this, review, contains, spoilers, for, those,...
1  [this, is, by, far, one, of, the, most, boring...
2  [this, movie, starts, out, with, an, execution...
3  [i, bought, the, dvd, of, before, sunset, and,...
4  [i, am, a, sucker, for, films, like, this, ., ...

                                     padded_tokens
0  [this, review, contains, spoilers, for, those,...
1  [this, is, by, far, one, of, the, most, boring...
2  [this, movie, starts, out, with, an, execution...
3  [i, bought, the, dvd, of, before, sunset, and,...
4  [i, am, a, sucker, for, films, like, this, ., ...
```

Long Short-Term Memory (LSTM) architecture



For the binary classification task, I implemented the *LSTM* model with an attention mechanism to improve the model's ability to focus on the most relevant parts of the input sequence. The architecture was designed as follows:

1. *Embedding Layer:* The input sequences were first passed through an embedding layer, which transformed tokenised words into dense vector representations. This step allowed the model to capture semantic relationships between words effectively.
2. *LSTM Layer:* The embeddings were then processed by an LSTM layer, which is capable of capturing contextual dependencies in sequential data. This helped the model understand the relationships between words across different positions in the sequence.
3. *Attention Mechanism:*
    - The *LSTM* outputs were used to compute attention scores through a linear layer.
    - These scores were normalised using the softmax function to obtain attention weights, which represent the importance of each token in the sequence.
    - A weighted sum of the *LSTM* outputs was computed using these attention weights, effectively focusing on the most relevant parts of the sequence.
4. *Fully Connected Layer*: The attention-weighted representation was passed through a fully connected layer for binary classification. The output of this layer was processed by a sigmoid activation function to produce probabilities indicating whether a review was positive or negative.

Training, Validation and Testing

- *Loss Function*: I used the *Binary Cross-Entropy Loss (BCE Loss)*, which is suitable for binary classification tasks, to measure the difference between the predicted and actual labels.
- *Optimiser*: The Adam optimiser was used for efficient parameter updates during training, ensuring faster convergence.

For training and evaluation, I structured the process to ensure robust monitoring of performance using metrics such as accuracy, precision, recall, F1 score, and loss for both training and validation phases. The metrics were tracked and stored for each epoch, allowing for a clear comparison of performance trends across the training process.

To begin, I initialised a metrics storage dictionary to record the aforementioned values for both training and validation. The training loop spanned 15 epochs, where the model was trained using the *Binary Cross-Entropy Loss* function. During each epoch, I calculated

and updated metrics for the training dataset by performing a forward pass, computing the loss, and then backpropagating to update the model's parameters. The predictions were thresholded at 0.5 to calculate binary metrics, ensuring consistent evaluation across all epochs.

```
Training completed...for 15 epochs!

Training Metrics...
Training Loss = 0.0056
Training Accuracy = 99.83 %
Training Precision = 0.9984
Training Recall = 0.9982
Training F1-Score = 0.9983
```

For validation, the model's evaluation mode was activated, and predictions were made without backpropagation to assess generalisation performance. Similar metrics were calculated on the validation dataset, and averages were computed at the end of each epoch. The best model was identified based on the highest validation *F1-score*, and its weights were saved for subsequent use.

```
Validation Metrics...
Validation Loss = 0.858
Validation Accuracy = 87.15 %
Validation Precision = 0.8756
Validation Recall = 0.8635
Validation F1-Score = 0.8554
```

Post-training, I evaluated the model's final performance on the testing dataset. Using the saved best model weights, I computed the metrics for accuracy, precision, recall, and *F1-score*, averaging them across all test batches. This ensured the reported results reflected the model's performance under optimal conditions.

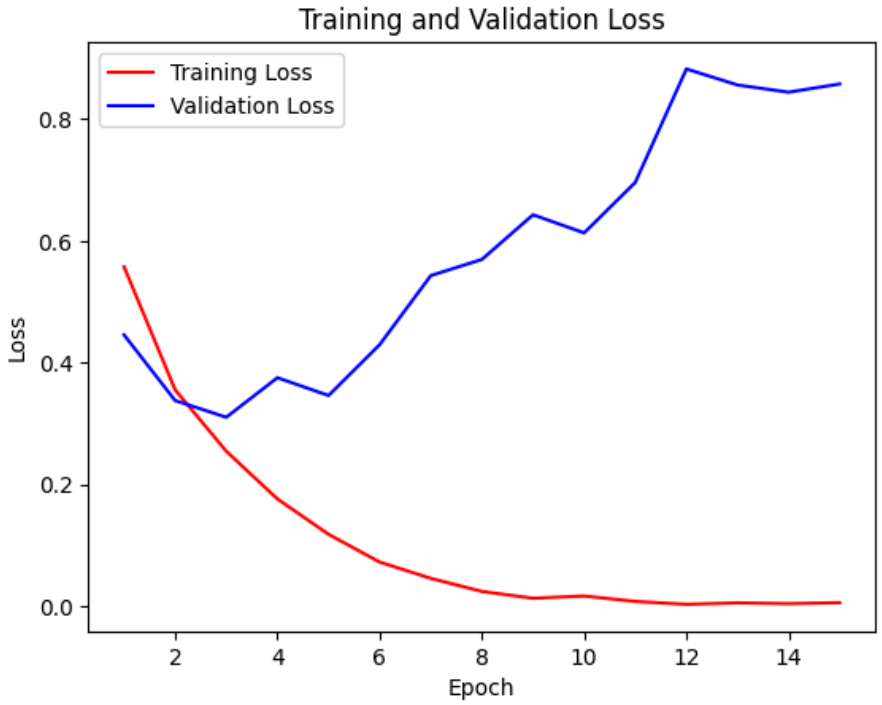
```
Testing...

Testing Metrics...
Testing Accuracy = 86.48 %
Testing Precision = 0.8665
Testing Recall = 0.8661
Testing F1-Score = 0.8637
```

Finally, the best-performing model was saved for reproducibility and future use. This comprehensive approach ensured the model was trained and evaluated rigorously, providing reliable insights into its ability to classify binary sentiment data.

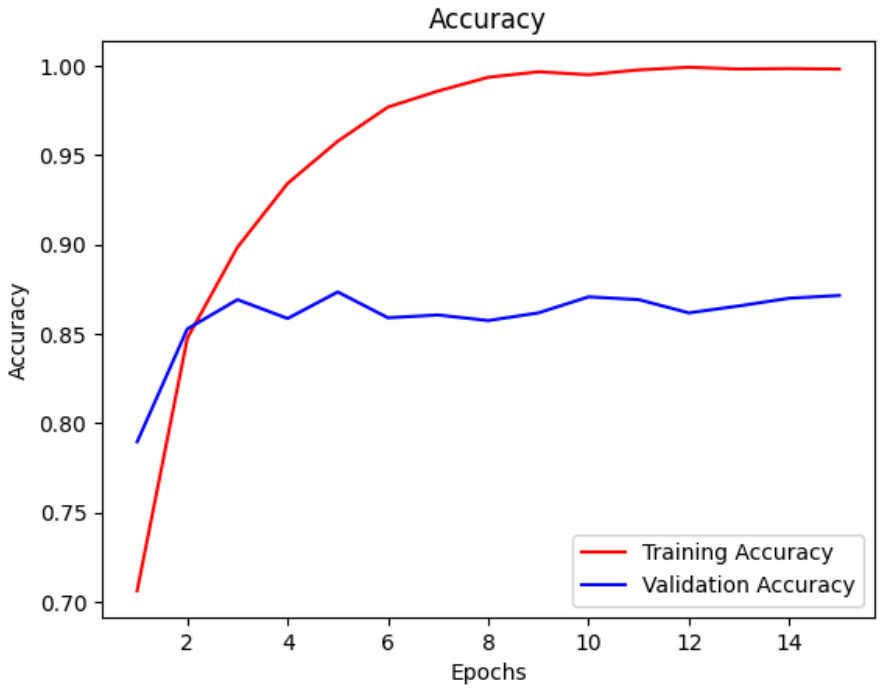
Plots and Heatmaps for Attention Weights

To track training and validation performance over epochs, I visualised the metrics:

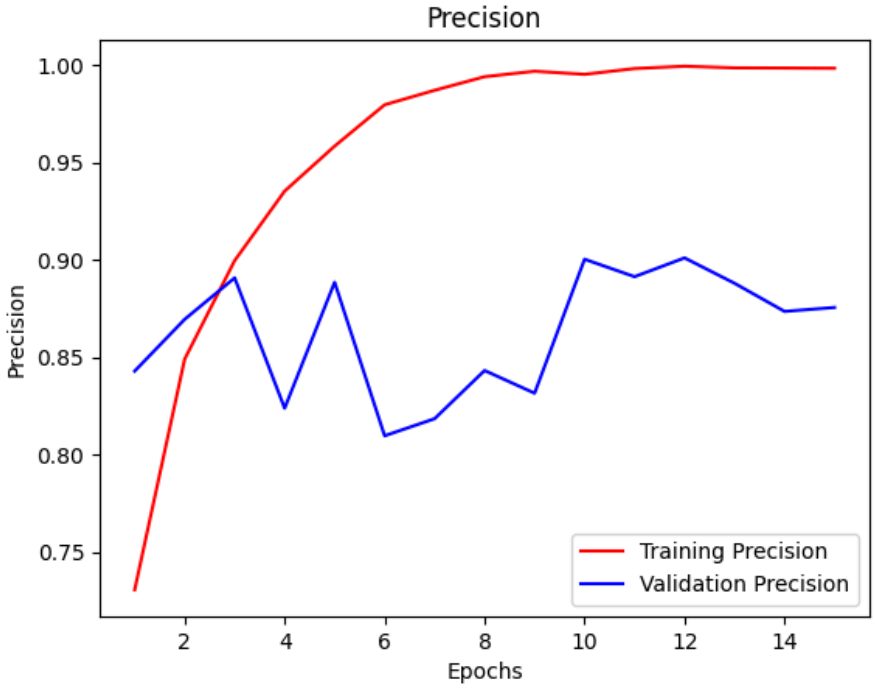
- **Loss** Plot



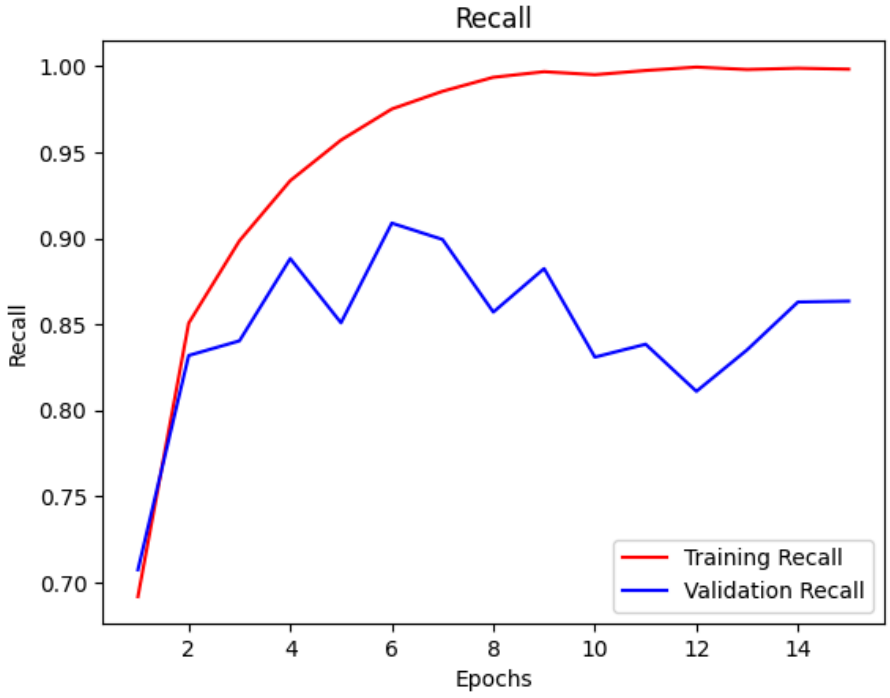
- **Accuracy** Plot



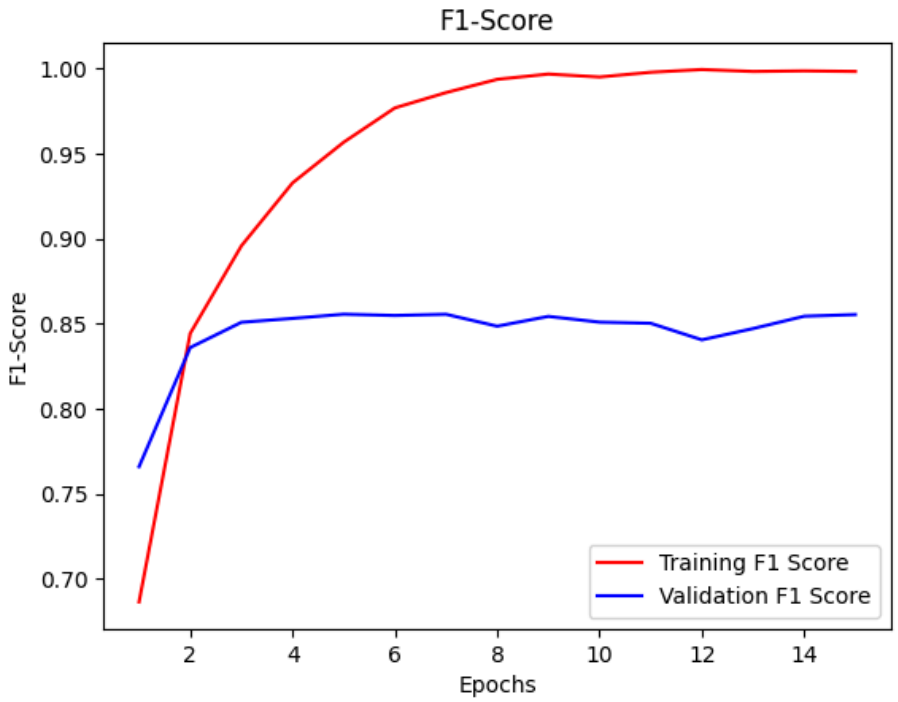
- **Precision** Plot



- **Recall** Plot



- **F1-Score** Plot



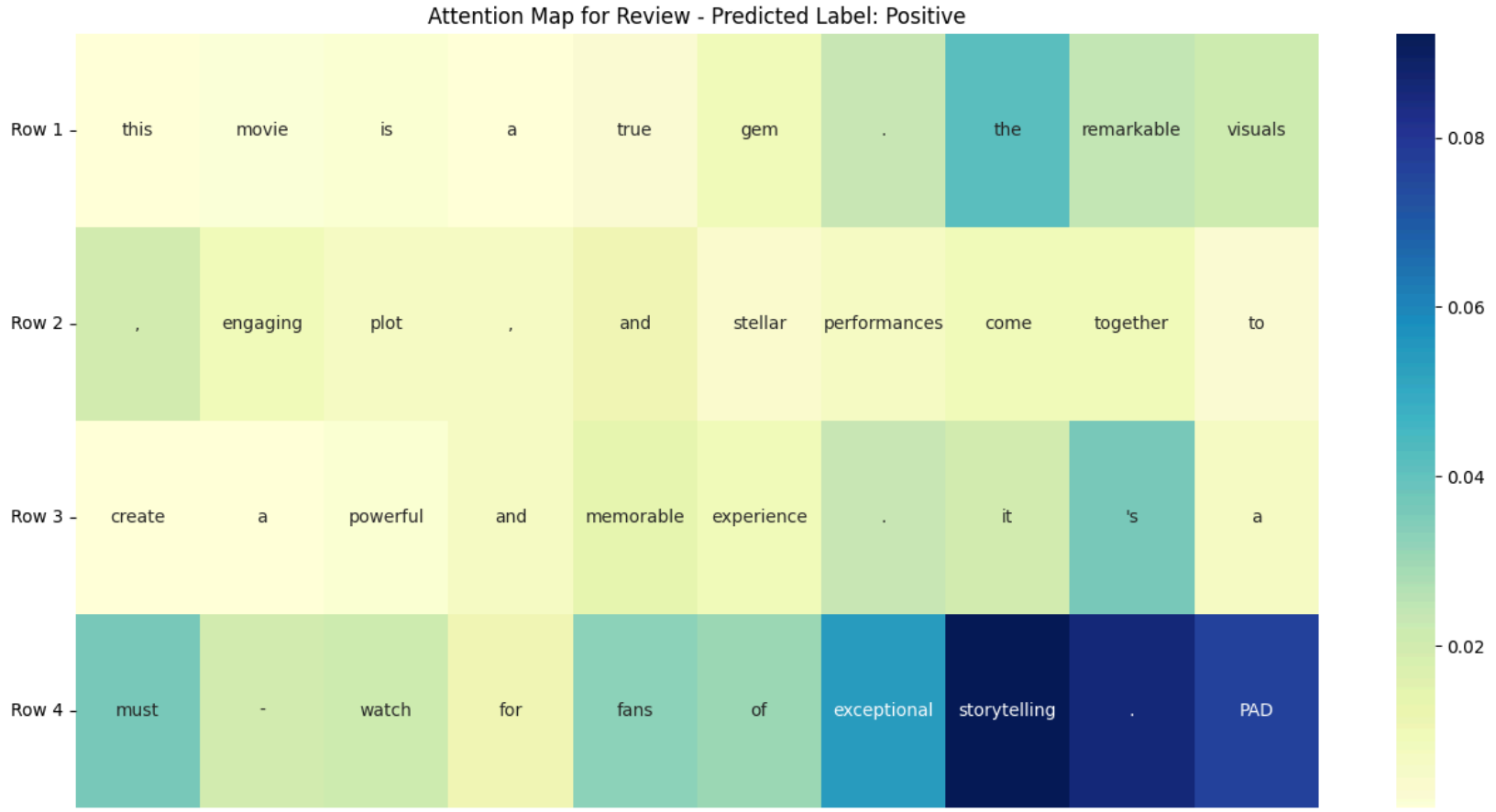
Attention Weights Heatmap Visualisation

- *Tokenisation*: The input review is tokenised and converted into indices using the vocabulary.
- *Model Forward Pass:* The tokenised input is passed through the model, obtaining predictions and attention weights.
- *Attention Visualisation:* The attention weights are split into manageable chunks (rows), which are plotted as a heatmap using *Seaborn*. The tokens are annotated for easy interpretation.

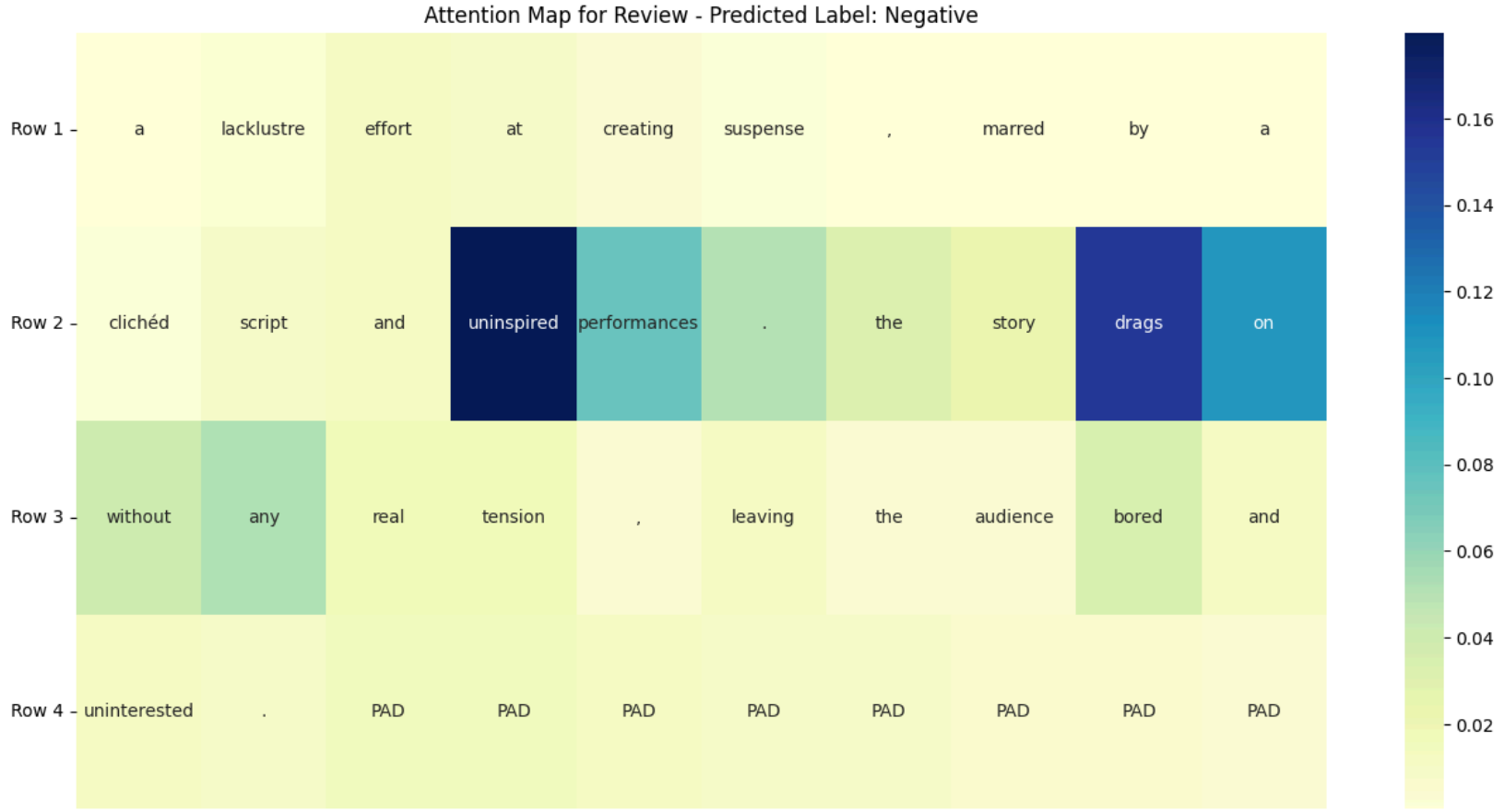
To understand how the attention mechanism works in predicting sentiment, I plotted heatmaps that visualise the attention weights for specific input sentences. These heatmaps indicate which parts of the input the model focuses on to make predictions.

Custom Sentences and Attention Visualisation

➢ For the ***positive review***, my custom sentence *"This movie is a true gem. The remarkable visuals, engaging plot, and stellar performances come together to create a powerful and memorable experience. It's a must-watch for fans of exceptional storytelling."* was predicted to be *positive*.



➢ For the ***negative review***, my custom sentence *"A lacklustre effort at creating suspense, marred by a clichéd script and uninspired performances. The story drags on without any real tension, leaving the audience bored and uninterested."* was predicted to be *negative*.



The attention weights in these cases reveal the key phrases the model paid attention to while predicting the sentiment of each review.

Comparison between the Models for Binary Classification

Comparing the performances of my *Feed Forward Neural Network (FFNN)*, *Recurrent Neural Network (RNN)*, from *Assignment-III*, with my *Long Short-Term Memory (LSTM) with Attention model* from above:

Phase	Metric	FFNN	RNN	LSTM
Training	***Loss***	0.30	0.69	0.01
	Accuracy (%)	87.05	51.91	99.83
	Precision	0.87	0.52	0.99
	Recall	0.87	0.41	0.99
	F1-Score	0.86	0.45	0.99
Validation	***Loss***	0.36	0.69	0.86
	Accuracy (%)	84.30	52.66	87.15
	Precision	0.81	0.52	0.88
	Recall	0.83	0.35	0.86
	F1-Score	0.82	0.40	0.86
Testing	***Accuracy (%)***	86.48	53.75	86.48
	Precision	0.84	0.56	0.87
	Recall	0.90	0.40	0.87
	F1-Score	0.87	0.46	0.86

🟢→ best of the three models

🟡→ moderate of the three models

🔴→ worst of the three models

Multi-Class Classification

Dataset

	label	text
0	1	One Night like In Vegas I make dat Nigga Famous
1	1	Walking through Chelsea at this time of day is...
2	-1	looking at the temp outside....hpw did it get ...
3	-1	I'm stuck in London again... :(I don't wanna ...
4	1	It's 2.32 a.m. here in Italy now :) Tomorrow o...

Data Preprocessing

For the multiclass classification task, I started by loading the training dataset, skipping any malformed lines. I renamed the combined column containing labels and text for easier processing and split it into separate columns for labels and text using the tab delimiter. The original combined column was then removed.

The data was shuffled and indexed before separating the features (text) and target labels. I then split the dataset into 80% training and 20% validation sets, ensuring reproducibility with a fixed random state. Labels were adjusted to start from 1 instead of 0.

The same preprocessing steps were applied to the test dataset to maintain consistency with the training and validation sets, preparing the data for model training and evaluation.

Input Features, Tokenisation and One-Hot Encoding

Displaying few training datapoints...

	text	label
0	The earth is run by psychopathsKids could be e...	0
1	@StarfmZimbabwe News from the US The statues o...	0
2	Real committed to watch Netflix all day Wednesday	2
3	yo I'm excited for Royal Rumble might book tha...	2
4	They're showing the Motown 25th on PBS. Michae...	2

For text tokenisation and preprocessing, I used a library to tokenise input text and replace out-of-vocabulary (*OOV*) words with a special token, *UNK*, to handle unknown words effectively. Tokens were generated by converting words to lowercase and verifying their presence in the vocabulary. To handle variable text lengths, I applied padding or truncation to ensure all sequences had a fixed length, determined by the maximum length of the training data. Padding involved appending the special token *PAD* to shorter sequences. For the preprocessing pipeline, I implemented a function to process the training, validation, and test datasets. The training data was tokenised, and word

frequencies were counted to create a vocabulary of words appearing at least five times. Special tokens *UNK* and *PAD* were added to this vocabulary.

Afterward, tokenisation and padding were applied consistently to all datasets. The average token length of the training data was used to determine uniform padding. The padded tokens were added to the respective *DataFrames*, and the vocabulary was converted into a dictionary mapping each word to a unique index. The processed datasets and vocabulary were then returned. Finally, I used a dataset class to create *DataLoader* instances for each dataset. The training *DataLoader* was shuffled for randomness, while the validation and test *DataLoaders* were kept fixed for consistent evaluation.

```
Training data...
                                                text  label  \
0  The earth is run by psychopathsKids could be e...      0
1  @StarfmZimbabwe News from the US The statues o...      0
2  Real committed to watch Netflix all day Wednesday      2
3  yo I'm excited for Royal Rumble might book tha...      2
4  They're showing the Motown 25th on PBS. Michae...      2

                                              tokens  \
0  [the, earth, is, run, by, UNK, could, be, expo...
1  [UNK, news, from, the, us, the, UNK, of, liber...
2  [real, committed, to, watch, netflix, all, day...
3  [yo, i, 'm, excited, for, royal, rumble, might...
4  [they, 're, showing, the, UNK, 25th, on, UNK, ...

                                       padded_tokens
0  [the, earth, is, run, by, UNK, could, be, expo...
1  [UNK, news, from, the, us, the, UNK, of, liber...
2  [real, committed, to, watch, netflix, all, day...
3  [yo, i, 'm, excited, for, royal, rumble, might...
4  [they, 're, showing, the, UNK, 25th, on, UNK, ...
```

LSTM for Multi-Class Classification

(slight modification)

I slightly tweaked the LSTM class to suit for multi-class classification. In the binary classification model, I used a *Sigmoid* activation function at the output layer, which outputs a single probability value between 0 and 1, indicating whether the input belongs to the positive class or the negative class. This is suitable for binary classification tasks where the model predicts one of two possible outcomes.

Now, for multiclass classification, I replaced the *Sigmoid* function with a *Softmax* activation function. The *Softmax* function outputs a probability distribution over all possible classes, where each class receives a probability between 0 and 1, and the sum of the probabilities across all classes is 1. This change allows the model to classify an input

into one of multiple classes of three, instead of just two. Rest of the model architecture is unchanged.

Training, Validation and Testing

I initialised the *LSTM* model with parameters like vocabulary size, embedding dimension, hidden dimension, and output dimension, and moved it to the device. *Binary Cross-Entropy Loss (BCELoss)* was used as the criterion, and the *Adam optimiser* was selected for parameter updates. I tracked metrics like accuracy, precision, recall, *F1-score*, and loss for both training and validation in a *METRICS* dictionary, setting the best validation *F1-score* to 0 for model selection.

```
Training...
Training completed...for 15 epochs!

Training Metrics...
Training Loss = 0.7093
Training Accuracy = 84.23 %
Training Precision = 0.8496
Training Recall = 0.8117
Training F1-Score = 0.8222
```

The model was trained for 15 epochs, where predictions were generated, loss was calculated, and weights were updated using backpropagation. Metrics were calculated for each batch and averaged for the epoch. Validation followed a similar process without weight updates. The best-performing model was saved based on the validation *F1-score*.

```
Validation Metrics...
Validation Loss = 0.9302
Validation Accuracy = 61.42 %
Validation Precision = 0.5995
Validation Recall = 0.5647
Validation F1-Score = 0.567
```

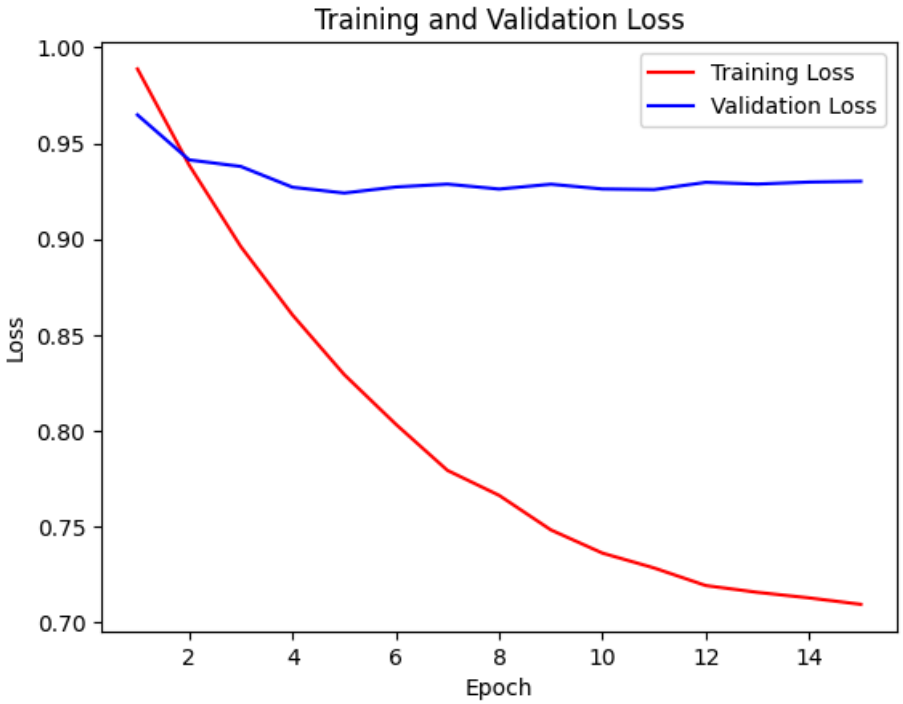
After training, I evaluated the model on the test dataset using the same metrics and printed the final accuracy, precision, recall, *F1-score*, and loss to assess performance on unseen data.

```
Testing...
Testing Metrics...!
Testing Accuracy = 79.61 %
Testing Precision = 0.8182
Testing Recall = 0.7491
Testing F1-Score = 0.7596
```

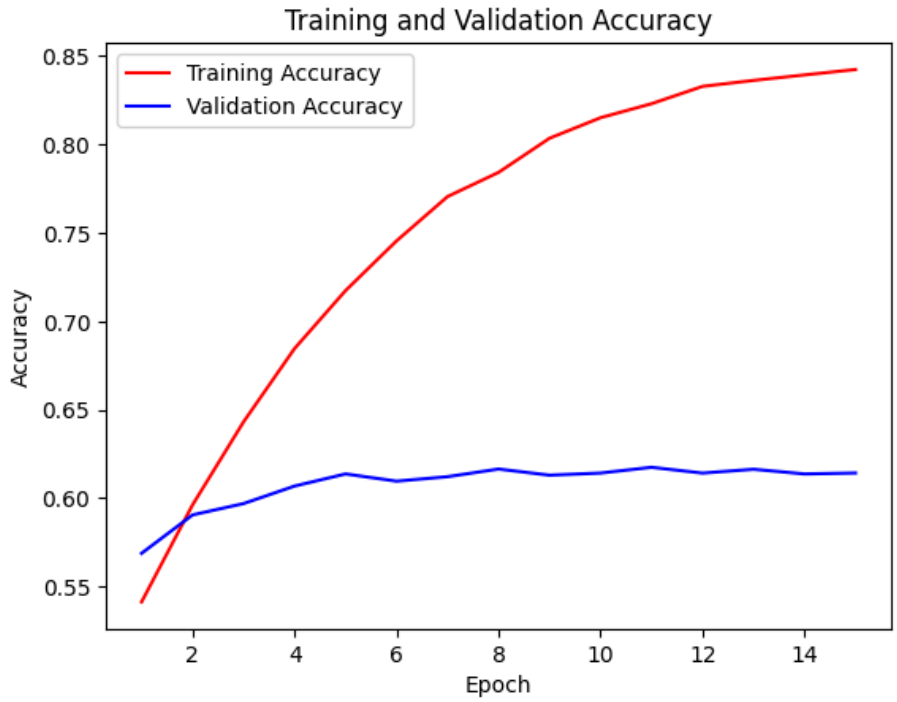
Plots and Heatmaps for Attention Weights

Once the model was trained, I visualised key performance metrics over the epochs. This helps to track how well the model is learning during training and how it performs on the validation set.

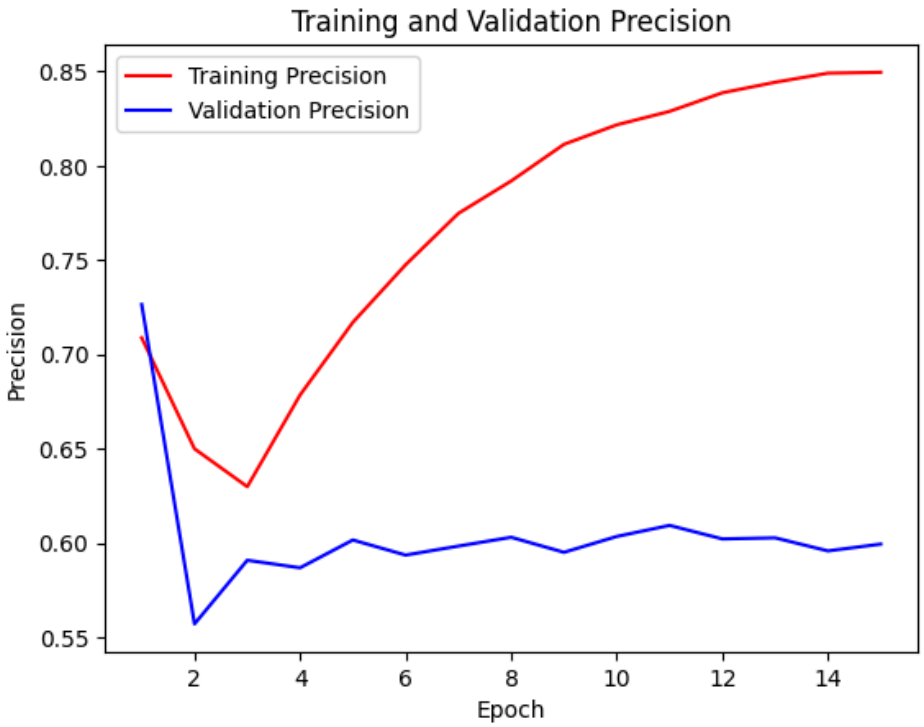
- **Loss** plot:



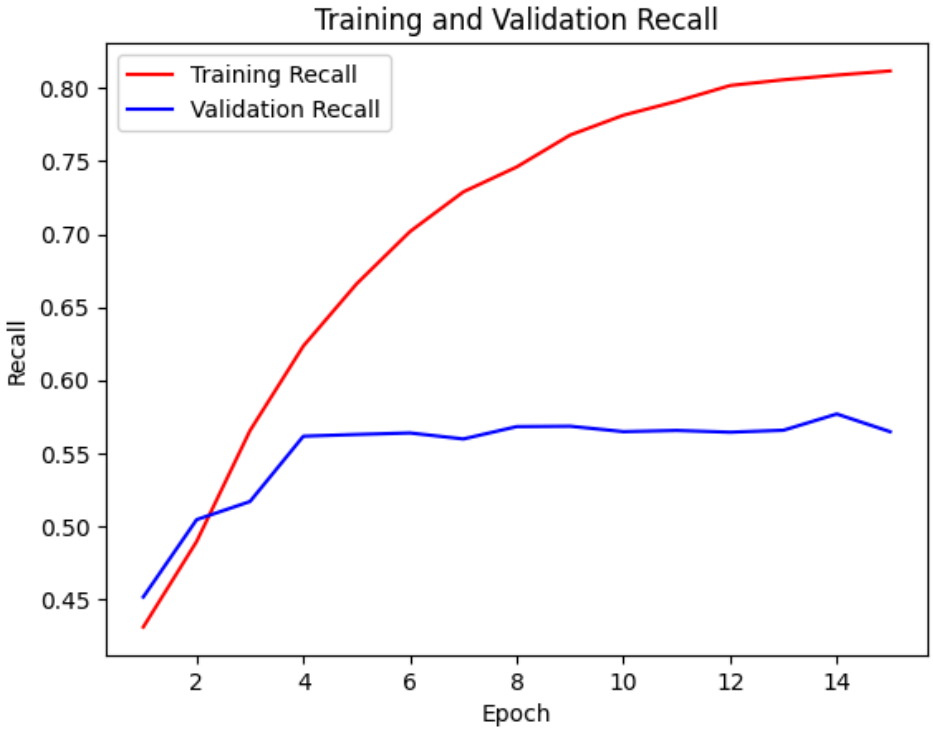
- **Accuracy** plot:



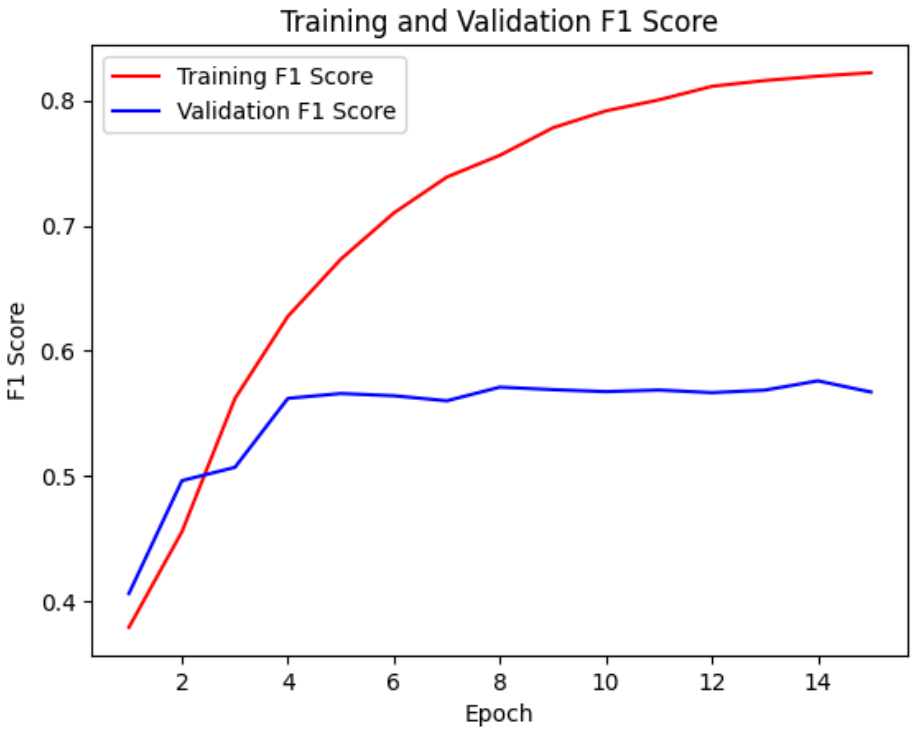
- **Precision** plot:



- **Recall** plot:



- **F1-Score** plot:



In the final section, I created a function to visualise the attention heatmap for a given review sentence, allowing me to understand how the model is focusing on different words while making predictions.

Visualising Attention Heatmap

My function *visualising_attention_heatmap* takes the following inputs: the trained model, tokeniser, review sentence, vocabulary, maximum length of the input sequence, and the device (CPU/GPU). Here's what I did step-by-step:

1. *Tokenisation and Padding/Trimming:*
    - I first tokenised the review sentence and adjusted the token length to *max_length*. If the sentence was shorter, I padded it with "PAD" tokens. If it was longer, I truncated it.
    - The tokens were then converted into indices based on the vocabulary, replacing unknown words with the index of the 'UNK' token.
2. *Model Prediction:*
    - I passed the tokenised input through the model in evaluation mode (*model.eval()*), ensuring no gradients are calculated during inference using *torch.no_grad()*.
    - The model's output is the predicted sentiment and attention weights. I mapped the predicted output to the sentiment label (Negative, Neutral, or Positive) based on the highest predicted class.

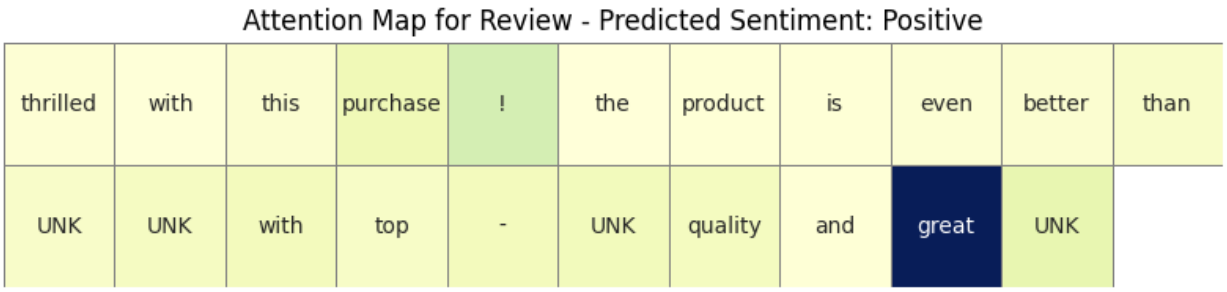
3. *Attention Weights Visualisation:*
    - The attention weights were extracted and reshaped for visualisation. To make it easier to read, I divided the tokens and attention weights into two chunks and padded them to ensure the chunks had equal lengths.
    - I then created a heatmap using *seaborn.heatmap* to display the attention distribution across the tokens. The tokens are annotated in the heatmap, and the plot is titled with the predicted sentiment.

I tested the function using three different types of review sentences: positive, neutral, and negative. For each review, I visualised how the model paid attention to different parts of the sentence while making its sentiment prediction.

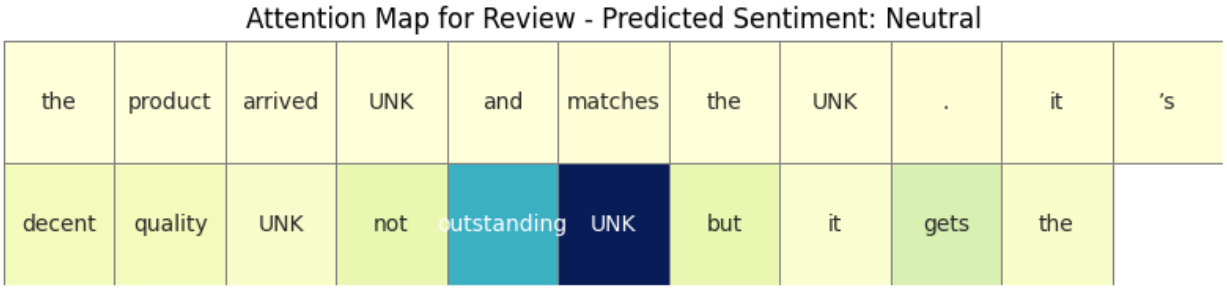
Custom Sentences and Attention Visualisation

I ran the following custom sentences through the *visualising_attention_heatmap* function:

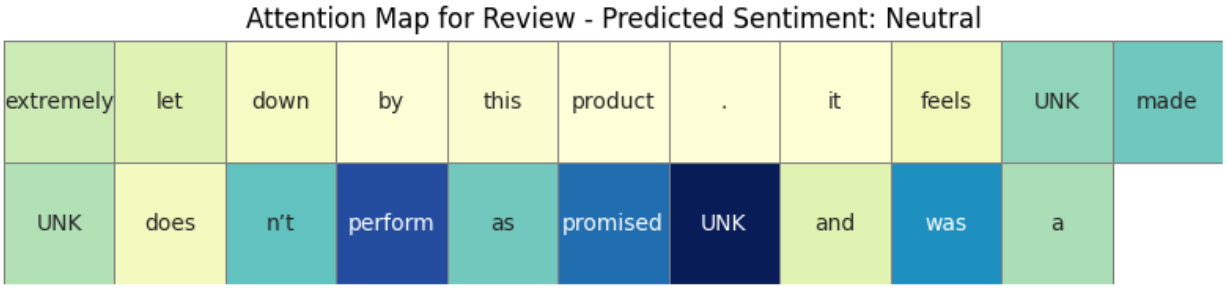
1. ***Positive Review***: *"Thrilled with this purchase! The product is even better than described, with top-notch quality and great functionality. Highly recommend it to anyone looking for something reliable and worth every penny!"* Here, my LSTM model predicted it correctly as *positive*.



2. ***Neutral Review***: *"The product arrived promptly and matches the description. It's decent quality, not outstanding, but it gets the job done and meets my needs for now."* Here, my LSTM model predicted it correctly as *neutral*.



3. ***Negative Review***: *"Extremely let down by this product. It feels cheaply made, doesn't perform as promised, and was a complete waste of money. I wouldn't recommend it to anyone."* Here, my LSTM model predicted the incorrect sentiment of *neutral* instead of *negative*; due to its moderate testing accuracy of 79.61 %



Each custom sentence produced a heatmap showing the tokens and their corresponding attention weights. These heatmaps provided insights into which parts of the sentence the model was focusing on when determining the sentiment.

Comparison between the Models for Multi-Class Classification

Phase	Metric	FFNN	RNN	LSTM
Training	***Loss***	0.07	1.04	0.71
	Accuracy (%)	97.93	45.75	84.23
	Precision	0.98	1.00	0.85
	Recall	0.98	0.56	0.81
	F1-Score	0.98	0.72	0.82
Validation	***Loss***	2.47	1.03	0.93
	Accuracy (%)	57.21	45.58	61.42
	Precision	0.53	1.00	0.60
	Recall	0.54	0.56	0.56
	F1-Score	0.54	0.71	0.57
Testing	***Accuracy (%)***	63.12	44.17	79.61
	Precision	0.64	1.00	0.82
	Recall	0.58	0.55	0.75
	F1-Score	0.61	0.71	0.76

🟢→ best of the three models

🟡→ moderate of the three models

🔴→ worst of the three models