Sentiment analysis using LSTM

1. Provide brief details about the nature of your dataset. What is it about? What type of data are we encountering? How many entries and variables does the dataset comprise?

→

The dataset we have chosen is the "Twitter US Airline Sentiment" dataset, and it contains information related to tweets about various airlines. The primary goal is to analyze the sentiment expressed in these tweets.

Objective: The dataset aims to capture the sentiment of Twitter users towards different airlines. It includes information about the sentiment expressed in tweets, the confidence level of the sentiment, reasons for negative sentiment, and various other details.

Type of Data: The dataset is a structured dataset, it contains both numerical and categorical data types.

Entries and Variables: The dataset comprises 14,640 entries (tweets) and 15 variables (columns). The variables include tweet ID, sentiment labels, confidence scores, reasons for negative sentiment, airline details, user information, tweet content, and additional metadata.

Below are few statistics of the dataset -

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 15 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   tweet_id                      14640 non-null  int64  
 1   airline_sentiment             14640 non-null  object 
 2   airline_sentiment_confidence  14640 non-null  float64
 3   negativereason                9178 non-null   object 
 4   negativereason_confidence     10522 non-null  float64
 5   airline                       14640 non-null  object 
 6   airline_sentiment_gold        40 non-null     object 
 7   name                          14640 non-null  object 
 8   negativereason_gold           32 non-null     object 
 9   retweet_count                 14640 non-null  int64  
 10  text                          14640 non-null  object 
 11  tweet_coord                   1019 non-null   object 
 12  tweet_created                 14640 non-null  object 
 13  tweet_location                9907 non-null   object 
 14  user_timezone                 9820 non-null   object 
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB
```

2. Provide the details related to your LSTM and Improved LSTM architectures.
→
1 - Basic LSTM model

1a. Model Class: LSTMModel

- Takes the following parameters during initialization:
  - `input_dim`: Dimensionality of the input data.
  - `embedding_dim`: Dimensionality of the word embeddings.
  - `hidden_dim1`, `hidden_dim2`, `hidden_dim3`: Dimensions of the hidden states for the three LSTM layers.
  - `output_dim`: Dimensionality of the output.

- Defines the following layers:
  - `embedding`: Embedding layer to convert input indices into dense vectors of fixed size (`embedding_dim`).
  - `lstm1`, `lstm2`, `lstm3`: Three LSTM layers, each with batch normalization, taking the output of the previous layer as input. They have different hidden state dimensions.
  - `dropout`: Dropout layer with a dropout probability of 0.3, applied after the third LSTM layer.
  - `fc`: Fully connected (linear) layer to produce the final output.

1b. Forward Method:
- Takes input `x` and passes it through the layers in sequence.
- Embeds the input using the embedding layer.
- Feeds the embedded input through three LSTM layers (`lstm1`, `lstm2`, `lstm3`) successively.
- Applies dropout after the third LSTM layer.
- Extracts the last time-step output from the third LSTM layer (`lstm_out3`) and passes it through the fully connected layer (`fc`) to get the final output.

2 - Stacked LSTM model -

2a. Model Class: ImprovedLSTM

- Takes the following parameters during initialization:
  - `input_dim`: Dimensionality of the input data.
  - `embedding_dim`: Dimensionality of the word embeddings.
  - `hidden_dim`: Dimensions of the hidden states for each LSTM layer.
  - `output_dim`: Dimensionality of the output.
  - `num_layers` (default=3): Number of LSTM layers in the stack.
  - `bidirectional` (default=True): Whether to use bidirectional LSTMs.

- Defines the following layers:
  - `embedding`: Embedding layer to convert input indices into dense vectors of fixed size (`embedding_dim`).
  - `lstm_layers`: ModuleList containing multiple LSTM layers. The number of layers is determined by the `num_layers` parameter.
  - `fc`: Fully connected (linear) layer to produce the final output.

2a. LSTM Layers:

- The model uses a loop to dynamically create and add multiple LSTM layers to the `lstm_layers` ModuleList. The input size for each layer is determined based on whether it's the first layer or a subsequent layer and whether the LSTMs are bidirectional.
- Each LSTM layer has `batch_first=True` to expect input of shape `(batch_size, sequence_length, input_size)`.

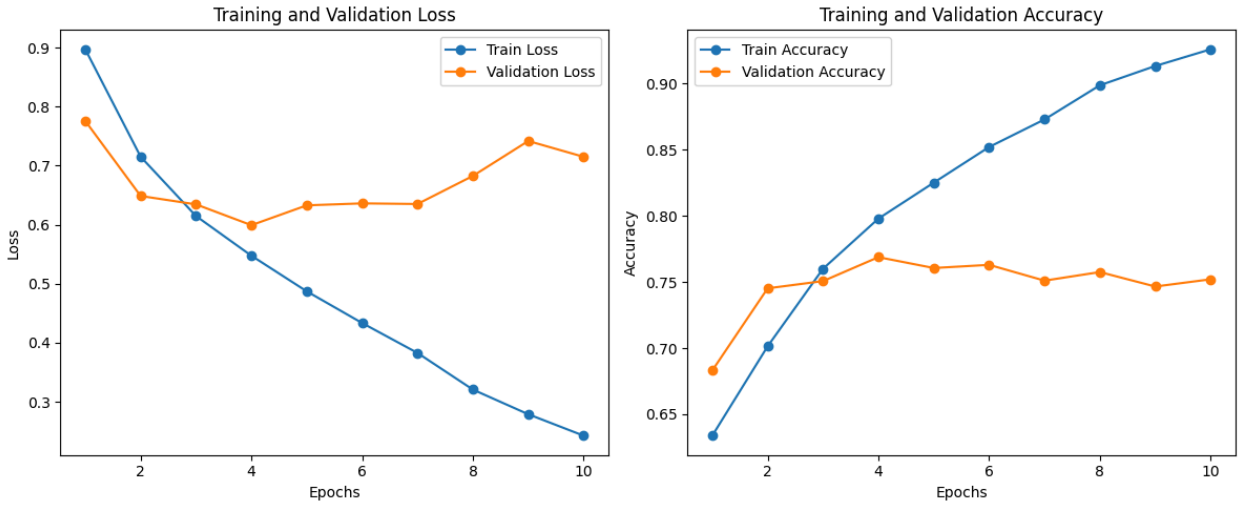
2c. Forward Method:

- Takes input `x` and passes it through the layers in sequence.
- Embeds the input using the embedding layer.
- Iterates through the LSTM layers in the `lstm_layers` ModuleList, passing the embedded input through each layer sequentially.
- Extracts the last time-step output from the last LSTM layer and passes it through the fully connected layer (`fc`) to get the final output.

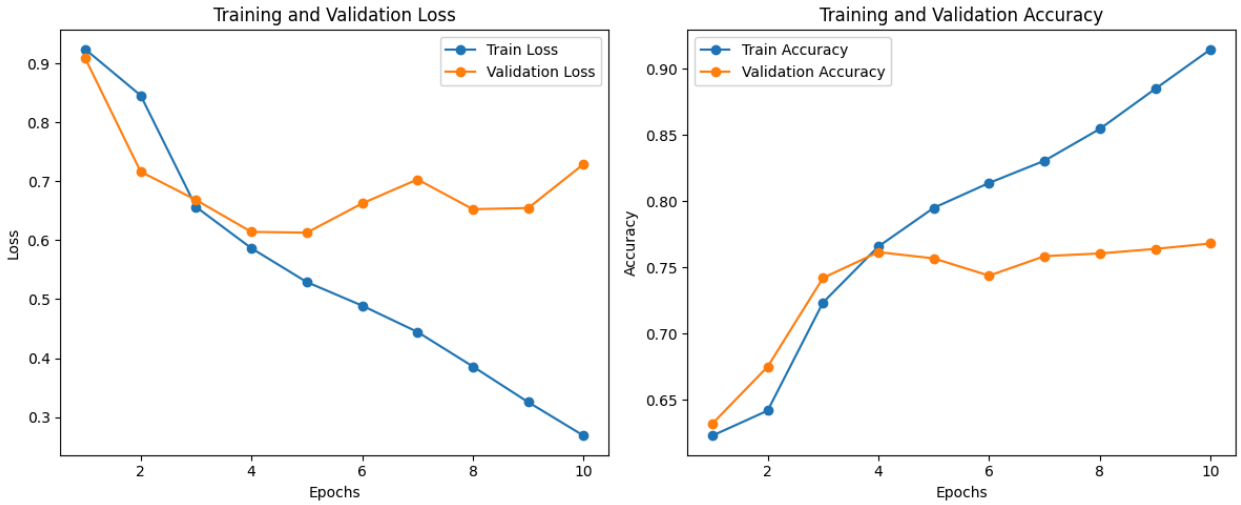
3. Discuss the results and provide the graphs for both models. Compare the performance of both Improved LSTM and LSTM models applied to the same dataset.

→

Basic LSTM model performance -



Stacked (improve) LSTM model performance -



The test accuracy for both the models is as follows -

Basic LSTM Test Set Accuracy: 0.7503

Stacked LSTM Test Set Accuracy: 0.7609

The accuracy for improved model was expected to be better than basic LSTM model increased but by only 1%, this might be due to following reasons-

1. Data Complexity:
Stacking more LSTM layers may not always lead to better performance, especially if the task is not complex enough to benefit from the added capacity. In some cases, a single-layer LSTM might be sufficient.

2. Gradient Vanishing/Exploding:
As the number of LSTM layers increases, the model might suffer from the gradient vanishing or exploding problem. Proper initialization and normalization techniques should be applied to address these issues.

3. Hyperparameter Tuning:
The performance of stacked LSTMs depends on hyperparameters such as the number of layers, hidden state dimensions, and bidirectionality. If these hyperparameters are not tuned properly, the model might not achieve optimal performance.

4. Discuss the strengths and limitations of using recurrent neural models for sentiment analysis.

→

Strengths of RNNs for Sentiment Analysis

1. Capturing Context: RNNs excel at understanding the flow of information in a sentence. Unlike simpler models that analyze words in isolation, RNNs consider the order and relationships between words. This is crucial in sentiment analysis, where sarcasm or negation can drastically change the meaning.

2. Learning Long-Term Dependencies: Traditional RNNs can struggle with long sentences because they might forget the sentiment of words at the beginning of the sentence. However, Long Short-Term Memory (LSTM) networks, a specific type of RNN, are designed to address this. LSTMs can learn dependencies between words even if they are far apart in the sequence, allowing them to grasp sentiment in longer texts.

3. Adapting to Different Text Lengths: Unlike fixed-size models, RNNs can analyze sentences of varying lengths. This makes them suitable for handling diverse data sources like tweets, reviews, and social media posts.

Limitations of RNNs for Sentiment Analysis

1. Vanishing and Exploding Gradients: Traditional RNNs can suffer from vanishing or exploding gradients during training. This makes it difficult for the network to learn long-term dependencies effectively. LSTMs mitigate this issue, but they can still be computationally expensive.

2. Data Sensitivity: RNNs are data-driven models. Their performance heavily relies on the quality and size of the training data. Biases or inconsistencies in the data can lead to biased sentiment analysis.

3. Interpretability: Understanding how RNNs arrive at their decisions can be challenging. This lack of interpretability makes it difficult to debug errors or identify specific aspects of text influencing the sentiment classification.