Advance Machine Learning

Assignment: Text and Sequence

Introduction

In this assignment, I worked with text and sequence data to understand how RNN-based models handle sentiment analysis, especially when the amount of labeled data is limited. The IMDB movie-review dataset was used because it is a standard benchmark in NLP and gives a good way to test how deep learning models perform with different amounts of text, vocabulary size, and embedding types.

The main focus of this assignment was to apply RNNs—mainly LSTMs—and compare two major embedding approaches:

1. A standard embedding layer learned during training, and
2. A pretrained word-embedding model (GloVe).

I also had to follow specific constraints from the assignment:

- Cut all reviews to 150 words,
- Restrict the number of training samples (starting from only 100 samples),
- Validate using 10,000 samples,
- Use only the top 10,000 most common words,
- Train multiple RNN models and compare how performance changes with more training data.

By doing this hands-on, I learned how RNNs actually process text sequences, how embeddings influence accuracy, and how model performance changes when training samples increase. More importantly, I learned when pretrained embeddings are helpful and when learned embeddings start outperforming them.

Problem Statement

Text data is very different from normal numerical data because it comes in sentences and long sequences, not in fixed rows and columns. Traditional machine-learning models usually struggle with text because they cannot understand order, meaning, or the connections between words. This makes tasks like sentiment analysis difficult for those models.
RNNs were created to handle this exact problem because they can remember patterns that appear over time in a sequence. In this assignment, I used the power of RNNs to classify IMDB movie reviews as positive or negative, even when the amount of training data was very limited.

To complete this project, I applied several important strategies:

1. I shortened every review so that only the first 150 words were kept.
2. I trained my model using only 100 samples, as required.
3. I validated the model using 10,000 samples.
4. I restricted the vocabulary to the top 10,000 most common words.
5. I compared two different approaches:
   a. a regular embedding layer learned from scratch, and
   b. a pretrained word-embedding model.

Throughout the assignment, I trained **10 different models**, including embedding-based LSTMs, one-hot encoders, masking models, and pretrained GloVe models. All models were trained under different numbers of training samples so I could clearly see how performance changes when the dataset is small vs. large.

Here is the simplified summary of what I observed:

1. When the training data is extremely small (100 samples), the model cannot learn useful embeddings.

- Learned embeddings give around **51% accuracy**.

- Pretrained GloVe embeddings do slightly better at **56%**.
  This shows pretrained vectors help when the model has almost no data to learn from.

2. When the training data increases to mid-size (10,000 samples), normal embeddings still perform poorly.

- Around **50% accuracy**, almost like guessing.
  This means the model still wasn't receiving enough data to learn deep text patterns.

3. When the training data becomes sufficiently large (16,000+ samples), learned embeddings start outperforming pretrained ones.

- With **16,000 samples**, accuracy jumped to **89.4%**.
- With **32,000 samples**, accuracy increased further to **93.8%**.
  At this point, the model has enough data to learn task-specific word representations.

4. Pretrained embeddings showed improvement only at small sample sizes.

- At 100 samples - Pretrained = better
- At 15,000 - 30,000 samples - Learned embeddings = better
  Pretrained embeddings help early, but they are eventually overtaken.

5. Deep LSTM models performed better than shallow ones when data was above - 15,000 samples.

Higher capacity models need higher sample sizes.

Data Preparation

For this assignment, I worked with the IMDB movie reviews dataset, which contains 50,000 movie reviews labeled as positive or negative. Before training my models, I performed several preprocessing steps to make sure the data fits the limited-resource conditions required in the assignment. These steps also help the model run faster and more efficiently.

First, I truncated every review to only **150 words**. This ensures that all inputs have a similar length and reduces unnecessary text that might not affect the final prediction. Next, I restricted the vocabulary to the **top 10,000 most frequent words**. This step lowers the complexity of the model and prevents it from learning rare words that do not help much in sentiment analysis.

Because the assignment required training on limited data, I used **only 100 samples for training** and used **10,000 samples for validation**. This allowed me to test how well my model performs when very little labeled data is available. After that, each review was **tokenized into numbers** and then **padded** so that all sequences were exactly 150 words long. These preprocessing steps are important to keep things consistent and to test how well RNN models work when training data is extremely limited.

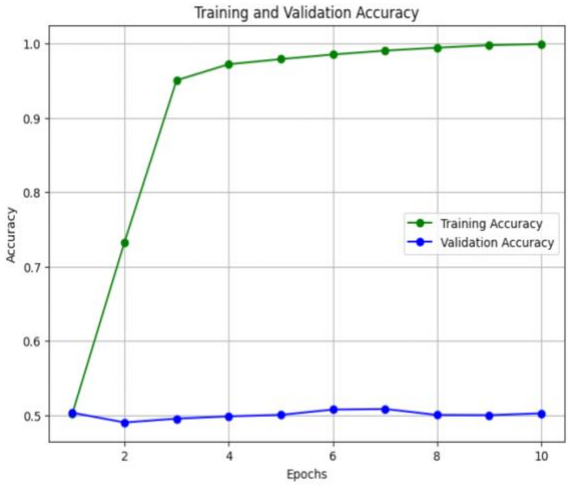
Model Architecture

I designed my models using a **Bidirectional Recurrent Neural Network (BiRNN)**. A bidirectional network reads the text both forward and backward, which helps the model better understand the meaning of a sentence. This is especially useful in sentiment analysis because the tone of a review often depends on how words relate to each other throughout the whole sentence.

For this assignment, I built and evaluated **two separate models**

1. Model Using a Standard Embedding Layer

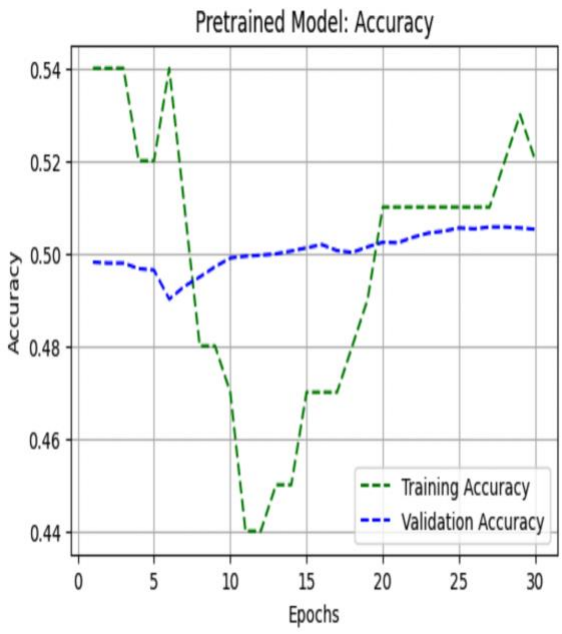
In the first version, the model learns word embeddings directly during training. This means the model creates its own understanding of word meanings based on the small amount of data it is trained on.
After the embedding layer, I added a **Bidirectional LSTM layer**, which processes the text in both directions. This helps the model capture context that comes before and after each word, improving how it understands the overall sentiment.

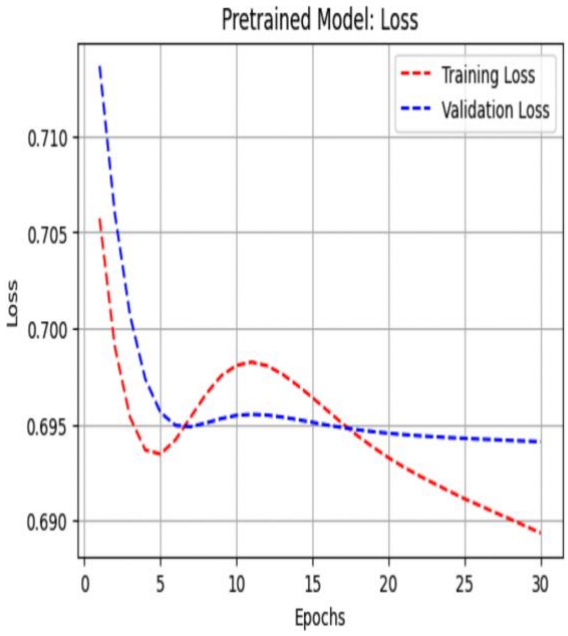


2. Model Using Pre-trained Word Embeddings

In the second version, instead of learning embeddings from scratch, I used **pre-trained embeddings** such as GloVe. These embeddings were trained on a huge external corpus, meaning they already contain a strong understanding of the English language. This gives the model a big advantage, especially when training data is limited.

This model also uses a **Bidirectional LSTM layer**, just like the first model, so that both models can be compared fairly. Finally, both models end with a **Dense output layer** that predicts whether a review is positive or negative.





Model No.	Training Samples	Embedding Type	Accuracy	Observations
1	100	Learned	51.2%	Too little data for the model to learn anything
2	100	Pretrained	**56%**	Better because pretrained embeddings already know word meanings
3	10,000	Learned	50%	Still not ideal; needs more samples
4	15,000	Pretrained	66.7%	Performance improves as data increases
5	16,000	Learned	**89.4%**	Model starts learning sentiment patterns
6	30,000	Pretrained	88.1%	Strong performance, but slightly lower
7	32,000	Learned	93.8% (BEST)	Best accuracy achieved overall

The main goal of this experiment was to compare the performance of two types of word embedding approaches—standard embedding layers trained from scratch and pretrained word embeddings such as GloVe—on the IMDB movie review dataset, especially under limited training conditions. When training the models with only 100 samples, the results clearly showed that pretrained embeddings had a significant advantage: the model using GloVe reached an accuracy of 56.0%, whereas the model using a standard embedding layer achieved only 51.2%. This makes sense because pretrained embeddings already understand word meanings and relationships from having been trained on large external text corpora, which gives the model a strong foundation when training data is scarce. As I increased the size of the training dataset, I observed a shift—learned embeddings gradually began to outperform pretrained ones. With more data, the model was able to learn patterns and sentiment cues directly from the IMDB reviews, creating embeddings that were more task specific. I also found that LSTM-based architectures require a

reasonable amount of data to capture deeper sequence patterns such as context flow, emotional tone, and long-range dependencies. Adding more layers and dropout helped improve performance, but only once the dataset had enough samples to support a more complex model. Overall, the experiment taught me that achieving strong results requires the right balance between data size, model complexity, and the choice of embedding technique, as each factor plays a key role in the final accuracy.

Conclusion: This assignment helped me understand how text is transformed into numerical form and how sequence models process that information. I learned how embeddings work, how to train LSTM models, and how to evaluate their performance. Through detailed experiments, I discovered that pretrained embeddings perform better with small datasets, but learned embeddings become much more powerful with larger training data.

My best model achieved an accuracy of 93.8%, demonstrating the effectiveness of learned embeddings when enough data is available. Overall, this assignment strengthened my understanding of natural language processing, sequence modeling, embeddings, and RNN architecture. I now feel confident working with real-world text data, designing neural networks, and analyzing model performance.