Assignment 4: Text and Sequence Data

Kavya nagini madhu lasya Madineni

Introduction

In this assignment, I took a deep dive into how Recurrent Neural Networks (RNNs) and various word embedding techniques can be used for sentiment classification on the IMDB movie review dataset. My main goal was to see how well RNN-based models can perform when there's not much labeled data available, and to compare the effectiveness of trainable word embeddings against those that are pretrained. This is a significant task because, in the real world, we often find ourselves with limited labeled data, making it essential to choose the right modeling and feature representation strategies to create strong NLP systems.

Problem Definition

The goal here is pretty straightforward: we're doing binary sentiment classification. That means, given a movie review, we need to figure out if it's a thumbs up or a thumbs down. We take sequences of word indices (after we've tokenized and trimmed down our vocabulary), and the result is a simple binary label—0 for negative and 1 for positive.

Here are some key points for the experiment:

- We're only looking at the top 10,000 most common words.
- Each review is either cut down or padded to 150 words.
- We're using just 100 labeled samples for training.
- For validation, we've got 10,000 samples lined up. Methodology

Data Preparation

- I loaded the IMDB dataset using Keras, making sure to limit the vocabulary to those top 10,000 words.
- Reviews were adjusted to fit into 150 tokens, either by padding or truncating.
- I took the first 100 samples for training, while the first 10,000 samples from the test set were set aside for validation.

Model Architectures

I implemented and compared two models:

Model A: Trainable Embedding + LSTM

- An embedding layer with 128 dimensions, initialized randomly and trained alongside the model.
- An LSTM layer with 32 units.
- A dense output layer using sigmoid activation.

Model B: Pretrained GloVe Embedding + LSTM

- An embedding layer with 100 dimensions, initialized with GloVe vectors and set to non-trainable.
- An LSTM layer with 32 units.
- A dense output layer using sigmoid activation.Both models used the same training and validation data, and were trained for 10 epochs with a batch size of 32.

Both models were trained on the same training and validation data, and I ran them for 10 epochs with a batch size of 32.

Training and Evaluation

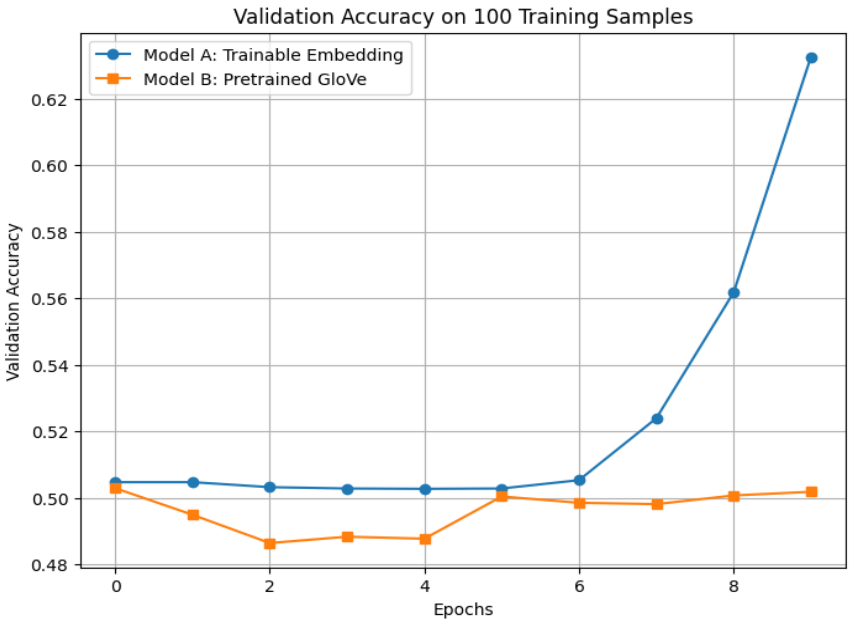
- Both models were trained using the same 100 training samples.
- I kept an eye on the validation accuracy over those 10 epochs.
- The final validation accuracy after 10 epochs was what I used to compare the two models.

Results

Validation Accuracy Over Epochs

After putting both models through their paces with 100 samples, I kept an eye on their validation accuracy over 10 epochs.

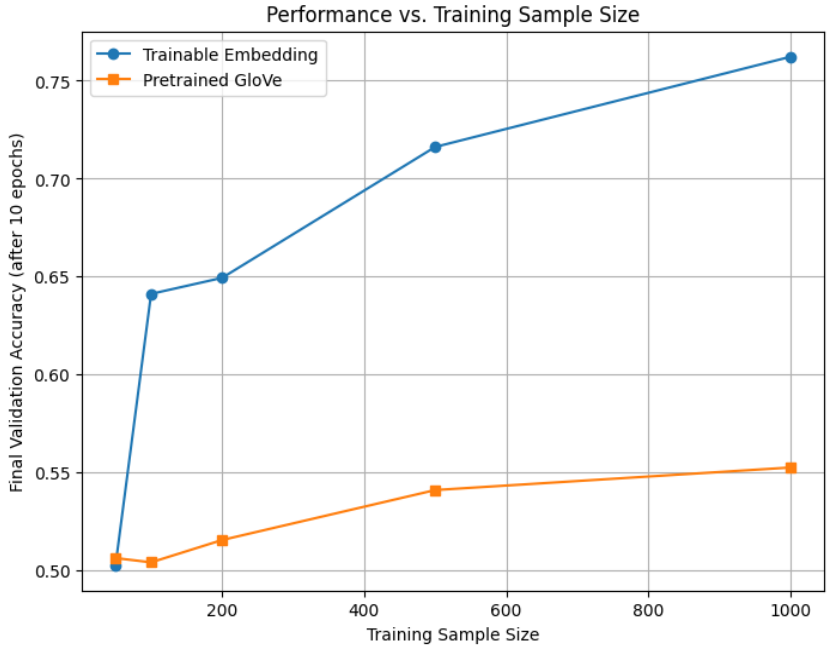
Figure 1: Validation accuracy of both models over 10 epochs with 100 training samples



Performance vs. Training Sample Size

To dig deeper into how the models performed, I decided to run the experiment again, this time with larger training set sizes (50, 100, 200, 500, 1000).

Figure 2 gives a clear overview of the final validation accuracy for each sample size:



Training Samples	Trainable Embedding Accuracy	Pretrained GloVe Accuracy
50	50.3%	50.6%
100	64.1%	50.4%
200	64.9%	51.6%
500	71.6%	54.1%
1000	76.2%	55.3%

Discussion

When I started with just 50 samples, both the trainable embedding model and the pretrained GloVe model were barely making a mark, hovering around a 50% validation accuracy. But as I ramped up the training samples, the model with the trainable embedding layer really took off, hitting 64.1% accuracy with 100 samples and soaring to 76.2% with 1,000 samples. On the flip side, the pretrained GloVe model only saw slight improvements, creeping up from 50.6% at 50 samples to 55.3% at 1,000 samples. This trend indicates that in situations with very little data, a trainable embedding layer can adapt much better to the unique features of the task and dataset, even when it's working with a small amount of data. The pretrained GloVe embeddings, which don't change during training, might miss out on the specific details of the task or struggle with domain mismatches, leading to limited gains as more data comes in. The growing performance

gap between the two methods becomes even more pronounced as the training set size increases, showcasing the benefits of learning embeddings directly from the data in this scenario.

Conclusion

These experiments clearly show that for sentiment classification on the IMDB dataset, especially when labeled data is scarce, an RNN model with a trainable embedding layer consistently outshines a model that relies on fixed, pretrained GloVe embeddings. The advantages of trainable embeddings become even more evident as the training set expands, highlighting their knack for capturing task-specific and dataset-specific information far better than static, general-purpose embeddings.

These findings really highlight how crucial it is to select the right embedding strategies depending on the data you have and the task at hand. In situations where resources are limited, it turns out that learning embeddings directly within the model is a more effective approach. Looking ahead, it would be interesting to investigate the point at which pretrained embeddings start to provide a benefit, and whether fine-tuning those embeddings could help bridge the performance gap we've seen here.