Assignment-4

Text and Sequence data

Keerthi Tiyyagura
Rupesh Suragani

Introduction:

The purpose of this assignment is to is to apply RNNs, or Transformers, to text and sequence data. We'll demonstrate each approach on a well-known text classification benchmark which is the IMDB movie review sentiment-classification dataset. Here we must apply RNNs or transformers to text and sequence data. Different NLP architectures handle word order differently. Bag-of-words models ignore order, recurrent models process words sequentially, and Transformers are order-agnostic yet incorporate positional information. So, both RNNs and Transformers are categorized as sequence models.

Goal of the project:

The goal of the assignment is to accomplish:

1. To apply RNNs or Transformers to text and sequence data
2. To improve the performance of network, especially when dealing with limited data
3. To determine which approaches are more suitable for prediction improvement.

Methodology:

Using google colab for our assignment, we have imported IMDB data source as our dataset. The IMDB dataset contains 50,00 reviews in total but only the top 10,000 words are considered. We took different ranges of training samples varying 100, 5000, 1000 & 10,000 and validation is done on 10,000 samples.

Our procedure had undergone two different approaches for generating word embeddings for the IMDB dataset which are:

1. Custom trained embedding layer (with 100, 5000, 1500 and 10000 samples).
2. Pretrained word embedding layer using GloVe model (with 100, 5000, 1500 and 10000 samples).

GloVe model is a widely used pretrained word embedding model and it is trained on large amounts of textual data.

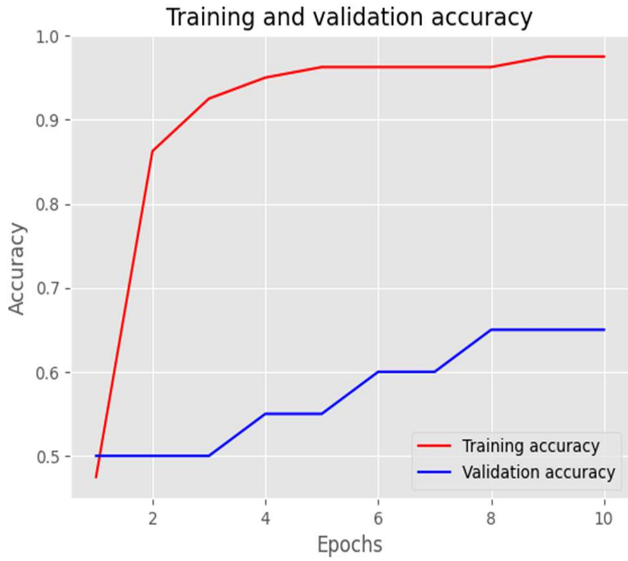
Here, I have compared the custom trained embedding layer with the pretrained word embedding using GloVe model to evaluate the effectiveness of those two different approaches. Comparison is done between the custom trained embedding layer on different training sample sizes of 100, 5000, 1500 and 10,000.

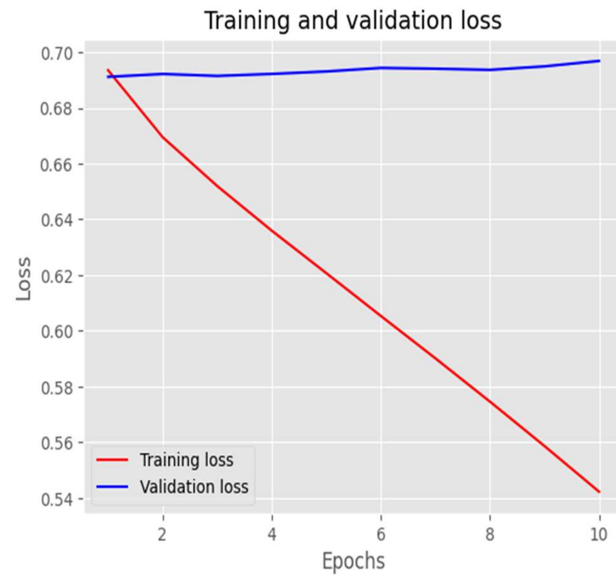
Customed Trained Embedding Layer:

This custom trained embedding layer is a NN model where the embedding vectors are trained from scratch and is specific to the task and the dataset we are going to work with. They might capture nuances and details better than pre-trained embeddings because they are tailored to our specific situation.

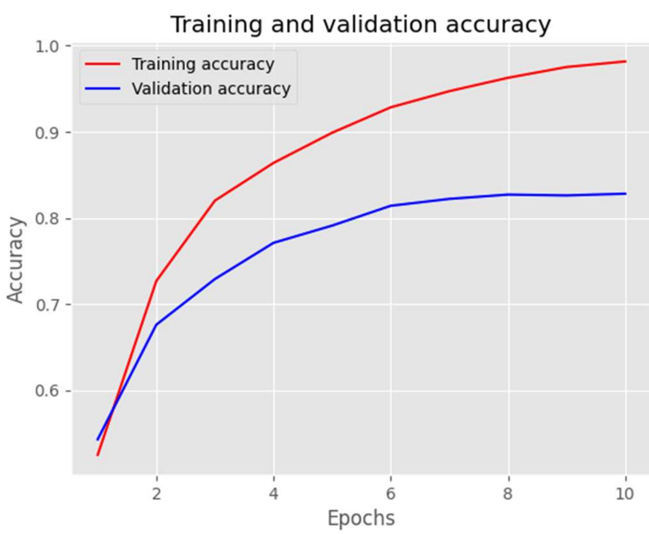
Here, we started creating a custom trained embedding layer by training each model on different sizes of training samples and measured its accuracy by evaluating the model with the test dataset.

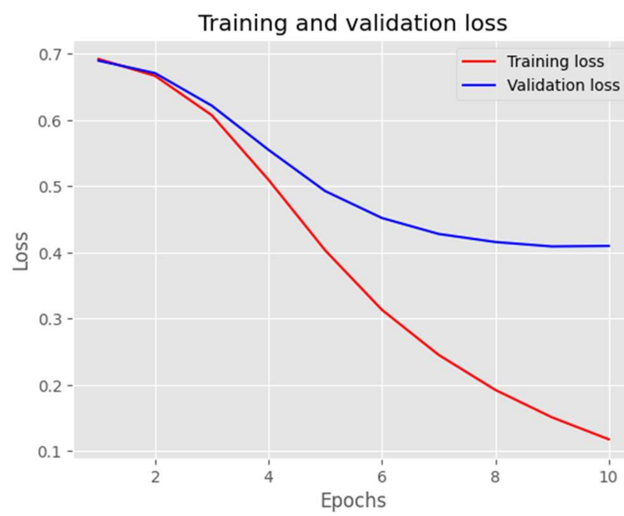
1. Custom trained embedding layer with 100 training samples



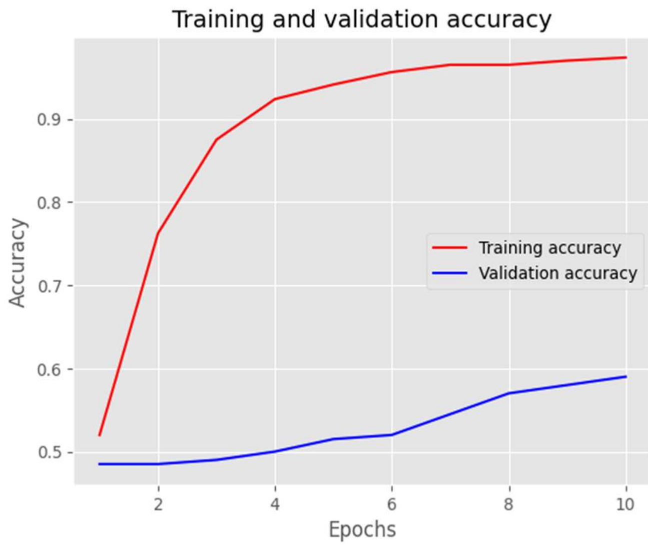


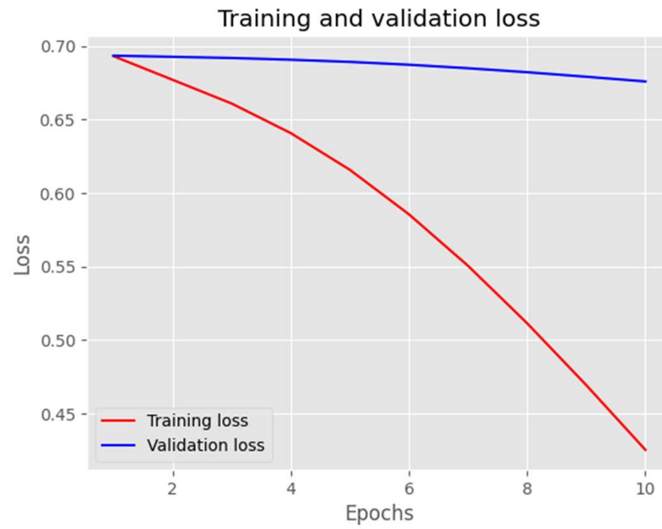
2. Custom trained embedding layer with 5000 training samples



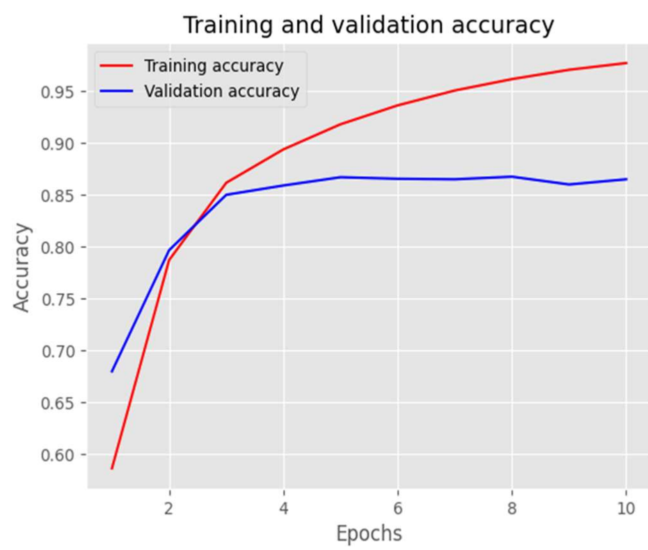


3. Custom trained embedding layer with 1000 training samples





4. Custom-trained embedding layer with 10000 training samples
   Images



Results from Custom Trained Embedding Layers:

Accuracy

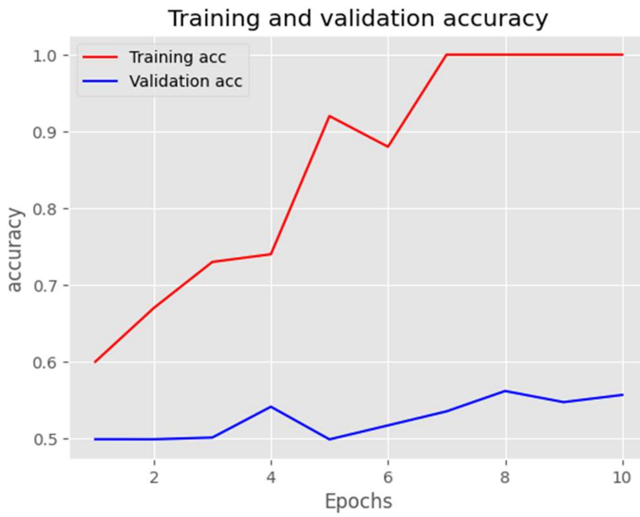
Training Sample size	Training	Validation	Test
100	97.5%	65%	50.3%
5000	98.5%	82.8%	82.85%
1000	97.37%	59%	55.6%
10000	97.72%	86.5%	85.7%

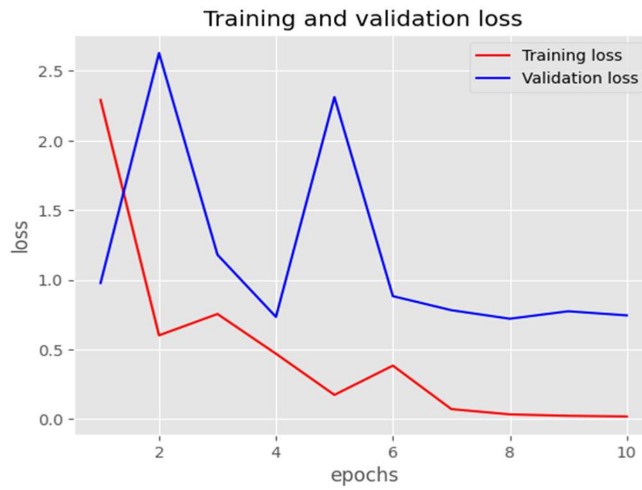
Pretrained Word Embedding Layer using GloVe model:

There are various precomputed databases of word embeddings that we can download and use in a Keras Embedding layer. Among them GloVe (Global Vectors of word representation is the most preferred one to use as a pretrained database. This is developed by Stanford researchers in 2014. This embedding technique is based on factorizing a matrix of word co-occurrence statistics.

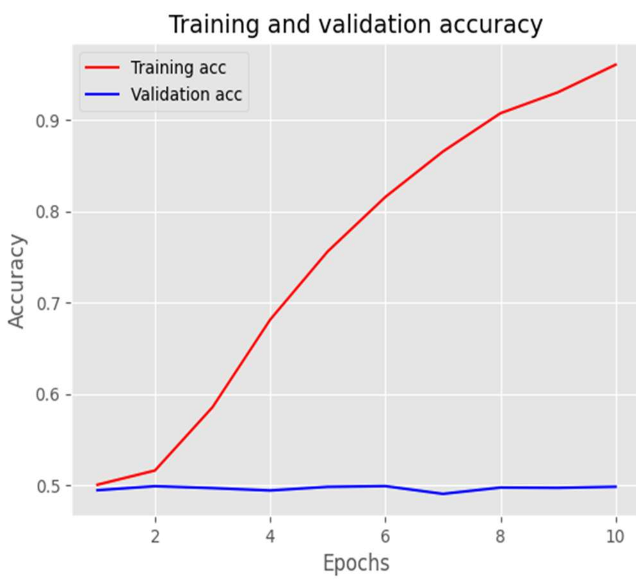
- We have downloaded the GloVe word embeddings on 2014 English Wikipedia dataset. It is an 822 MB zip file containing 100-dimensional embedding vectors for 400,000 words (or non-word tokens)
- For building an index that maps words (as strings) to their vector representation, we parsed the unzipped file (a txt file).
- Next, we have built an embedding matrix that you can load into an Embedding layer.
- Finally, we have used a constant initializer to load the pretrained embeddings in an Embedding layer. So as not to disrupt the pretrained representations during training, we have frozen the layer (trainable = false).
- Now, we're ready to train for a new model.

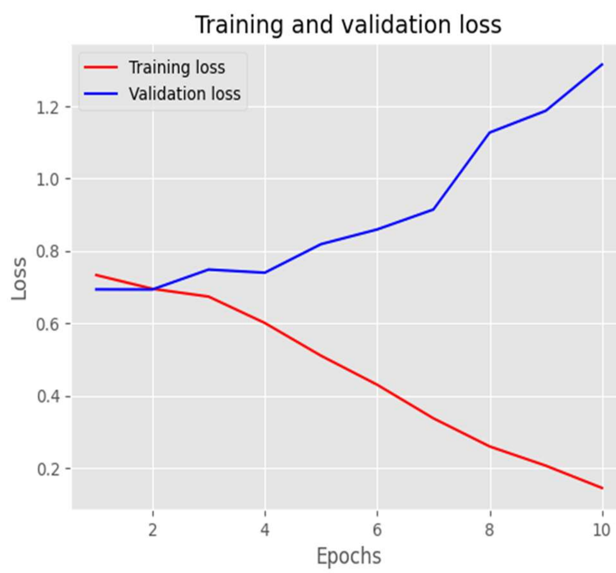
1. Pretrained word Embedding layer with 100 training sample



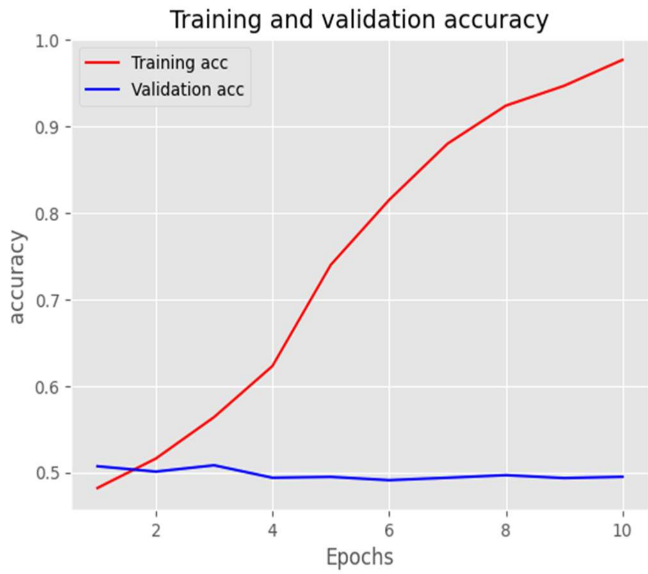


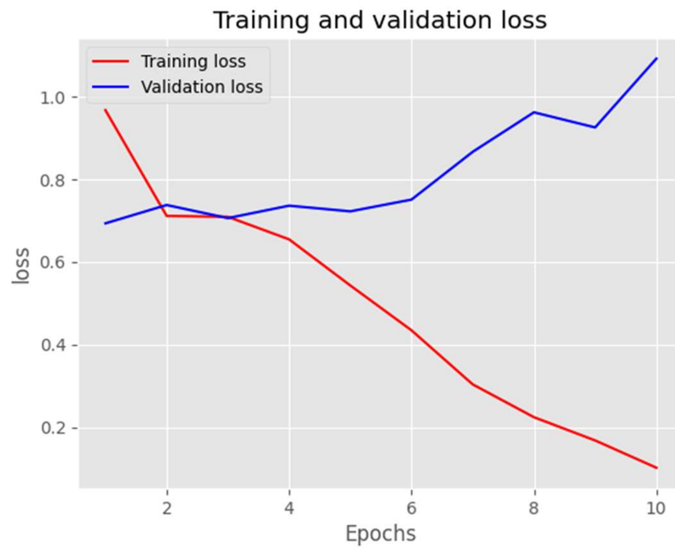
2. Pretrained word Embedding layer with 5000 training samples.



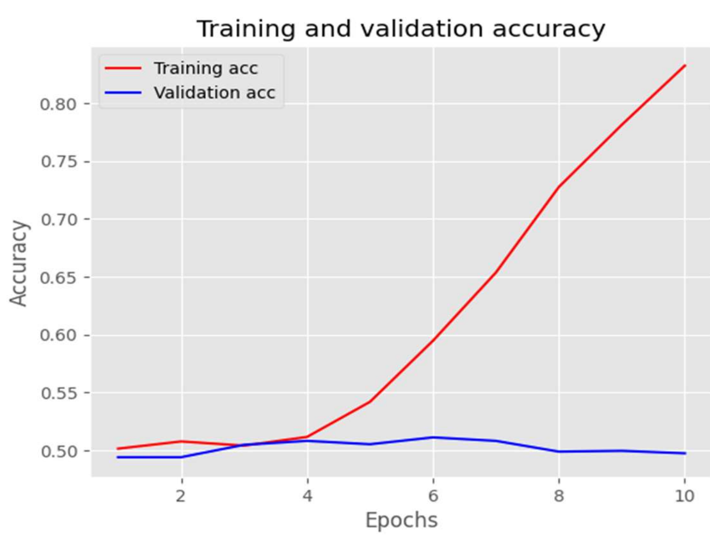


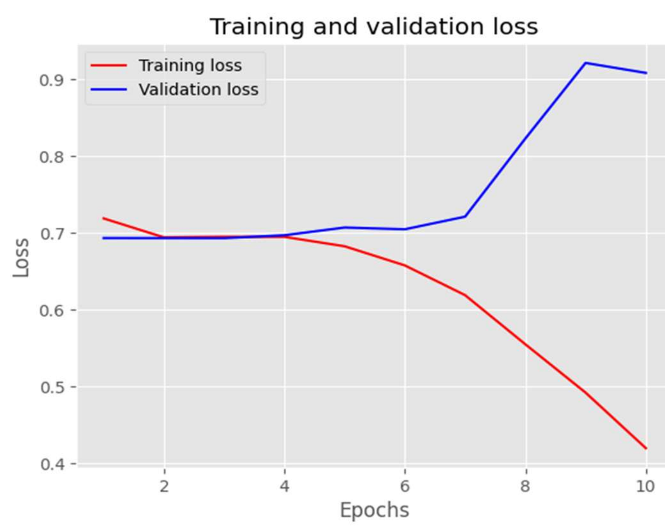
3. Pretrained word Embedding layer with 1000 training samples.





4. Pretrained word Embedding layer with 10000 training samples.





Results for the pretrained embedding layers:

Accuracy

Training Sample size	Training	Validation	Test
100	100%	55.68%	50.1%
5000	96.06%	49.82%	49.96%
1000	97.7%	49.5%	50.1%
10000	83.21%	49.76%	49%

Conclusion:

From our work, we got to know that the custom-trained embedding layer consistently outperformed the pretrained word embedding layer, especially when training with larger sample sizes. However, it's worth noting that the pretrained word embedding layer could still be considered a "better choice" in certain scenarios, particularly when computational resources are limited and a small training sample size is required, despite the risk of overfitting.