SIMPLE SENTIMENT CLASSIFICATION ON THE IMDB DATASET

AI6127 Deep Natural Language Processing

Riemer van der Vliet
G2304212K

2024 - March - 12

1 Introduction

This report investigates the performance of various deep learning models for sentiment classification on the iMDB dataset. We begin by reimplementing a provided RNN example, then experiment with different optimizers (SGD, Adam, Adagrad), epoch variations, and the use of pre-trained Word2Vec embeddings. Additionally, we compare the efficacy of different neural network structures, including feedforward networks, CNNs, and LSTMs, in sentiment analysis. Our analysis focuses on how these variables affect model accuracy and performance, aiming to offer concise insights into optimizing deep learning approaches for sentiment classification.

1.1 loss and accuracy

Our loss function is defined as a BCEWithLogitsLoss. As can be seen in equation 1, as you can see it combines a Sigmoid activation function with a binary cross entropy loss. Within the the codebase we have used the pytorch function 'nn.BCEWithLogitsLoss()'. Regarding the accuracy of our tests we have defined a binary accuracy function as seen in equation 2.

$$\text{loss}(x, y) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \cdot \log(\sigma(x_i)) + (1 - y_i) \cdot \log(1 - \sigma(x_i))\right] \tag{1}$$

$$\text{binary\_accuracy}(\text{preds}, y) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\left[\text{round}\left(\sigma(\text{preds}_i)\right) = y_i\right] \tag{2}$$

1.2 different optimizers

Here we conduct some experiments with a few different optimizers. 'stochastic gradient descent' (SGD), 'adaptive moment estimation' (Adam) and 'adaptive gradient descent' (Adagrad). Where the difference between these optimizes is how we change the model depending on the loss function. As can be seen in table 1 we look at the test loss and test accuracy.

Optimizer	**Test Loss**	**Test Accuracy** (%)
Adam	0.733	58.83
SGD	1.116	64.85
Adagrad	1.161	65.03

Table 1: Optimizer Performance of 3 different optimisers. Adam has the lowest tesst loss and the lowest test accuracy.

There is actually minimal variation in test accuracy among the three optimizer. This suggests that for this specific task the optimizer does not actually influence in terms of accurancy that much. Except for Adams, that shows the lowest loss and lowest test accuracy. This implies that Adams is most effective in minimizing the loss function, but possibly not best in generalizing the data as much as SGD or Adagrad are.

1.3 different epochs

Here we conduct some experiments with a few different number of epochs. Using the previously tested Adams optimizer. The results can be seen in table 2.

Epochs	**Test Loss**	**Test Accuracy** (%)
5	0.652	62.22
10	0.609	68.24
20	0.629	66.21
50	0.672	59.14

Table 2: Epoch Performance. The best performing accuracy is at 10 epochs.

As can be seen in the data, 5 epochs are too little to generalize the data and 50 epochs is a bit on the long side. Which could even suggest overfitting. However, as can be seen in the notebooks, we only save the best performing model, and therefore, decreasing performance over epochs can be somewhat reduced.

2 word2vec

In this section we used the adam optimizer, 50 epochs and we download and use the pretrained Word2Vec embeddings as can been seen within the github repository in the jupyternotebook file. Where we use Gensin python package. We use the pretrained Word2Vec embedding by only taking a subsample of the vector database for our vocabulary. Words present in our vocabulary, but not in the pretrained embedding have been assigned a random vector of corresponding input dimentions. This allows the model to get a headstart on learning. The results can be seen in table 3.

Configuration	Test Loss	Test Accuracy (%)
No Word2Vec *	0.672	59.14
Word2Vec	0.504	78.37

Table 3: Comparison of model performance with and without Word2Vec embeddings over 50 epochs using Adam optimizer. * This data is seen before in table 2.

Notably in the results the model using the pretrained embeddings is outperforming the other model substantially. Increasing the test accuracy from approximately 60% to almost 80%. Such Word2Vec models learn from large and diverse corpora to capture semantic similarity within a vector space. The benefit of this is to provide a foundation of linguistic knowledge rather than starting form scratch.

3 different models

In this section we return to using randomly initialized embedding to run the experiments with a 'feed forward network' (FFN), 'convolutional neural network' (CNN) and some variants of 'long short-term memory' (LSTM), such as 'bi-LSTM'. The model performances can be seen in table 4, where notably the CNN performs very well. It can also be seen that the CNN is the only model that actually find a stable minimum as can be seen in the losses figure 1. Finally, we see that the model parameters for these architectures remain approximately $2.7 * 10^6$ as can be seen in table 5.

Model	Test Loss	Test Accuracy (%)
FFN1	0.335	86.75
FFN2	0.345	85.94
FFN3	0.349	85.96
CNN	0.280	88.43
LSTM	0.343	85.95
BiLSTM	0.443	80.31

Table 4: Model Performance Comparison

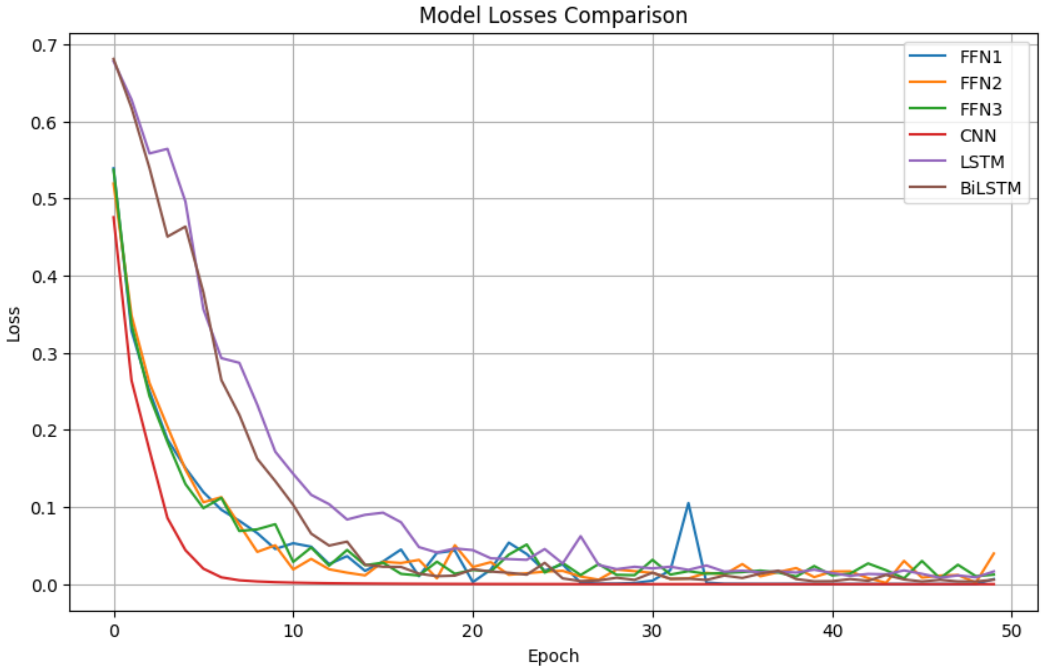

Figure 1: Model losses comparison between the previously seen models as described in table 4.

4 Final model

As we see both the CNN and RNN with Word2Vec increase the task performance substantionally, in this section I have trained a CNN with a Word2Vec embedding layer. The results however do not indicate much of an improvement, as the Test Loss: 0.275 and Test Acc: 88.54% do not indicate substantial improvement over the randomly initialized CNN nor the RNN with randomVec.

5 discussion

The model using Word2Vec embeddings performs very well, this is due the foundation of linguistic knowledge which gives it a substantial head start of the the other models. One could argue the actual comparison is not entirely fair as such pre-trained embeddings have seen massive corpora, opposed to the models we are comparing it with, which do not.

Furthermore regarding the different models, my concern was with the number of trainable

Model	Number of Parameters
FF1	2,551,201
FF2	2,701,301
FFN3	2,761,401
CNN	2,803,201
LSTM	2,982,901
BiLSTM	3,465,301

Table 5: Number of Parameters for Different Models.

parameters but as I noted in the table 5, there is not much difference between the models. So this comparison is fair. It appears the CNN is able to classify higher dimensional features which is why it might perform better.

To further optimize this model I have combined the CNN with the pretrained vector embeddings. However, no substantial improvement was seen. It could be that this is the threshold on this dataset. However I would still advice to play around with some other architectures but primarily try some hyper parameter optimization.

The code for this report can be found in the github repo: `https://github.com/Riemer1818/AI6127-dNLP-Ass1`

6 Bibliography

References

no external sources have been used that need to be cited.