Deep Learning 4 NLP @ lab 1

Dorin Doncenco

November 2023

1 Introduction

Sentiment classification is a basic task in NLP, which involves predicting the sentiment expressed in a text. The IMDB movie reviews dataset is such a collection of positive and negative reviews. In this project, we will be doing sentiment classification, where our goal is to predict the type of the review (positive or negative) given the review itself (a string). The goal is to compare our ability to learn the embeddings using a linear classifier, a classifier with a convolutional layer, and the usage of a pretrained word2vec model's embeddings.

2 Dataset

The dataset consists of 600000 reviews, evenly split into positive and negative. We take 5000 reviews of each type, and split it into a train/dev/test set of 60%, 20%, 20%, ensuring that the classes are balanced in each group.

3 Neural Networks

We will be training 4 different types of neural networks. On one hand, we will compare our performance of learning an embedding on our own (and superficially observe if these embeddings make sense) compared to using a pretrained word2vec vectorizer, and on the other hand we want to see the performance difference between a linear classifier and a CNN classifier.

3.1 Linear classifier

In the case of a linear classifier, the principle is simple: Feed a datapoint into a linear layer and get the output! However, we notice that this is not an easy task. Each sentence has each word (token) embedded as a tensor of vectors. Because each sentence has different amount of tokens, the resulting tensor will be of a varied length, preventing it from being fed into a static linear layer. To overcome this, we will be taking the average of the embeddings of the sentence's tokens to consider its meaning.

3.2 CNN classifier

The limitation of a linear classifier as described above, is that it does not take into account for the order of words: "this is not worth a watch, the movie was bad!" is different than "this is worth a watch, the movie was not bad!" - yet the representation will be the same. To allow for order to be considered, we will be first feeding the embeddings into a convolutional layer to extract features, followed by a (max) pooling layer to reduce the dimensionality of the features extracted. This dimensionality reduction, again, allows us to feed these results into a linear layer for classification purposes.

The convolutional layer makes use of the convolution operation, which takes into account the order of the input data (e.g. embeddings of words) by concatenating sequences of embeddings together, and then passing a convolutional kernel over it (lecture 1, page 43). In terms of implementation, thankfully, this is taken care of by the torch library.

3.3 Learning embeddings vs word2vec

Each of the model architectures above have been trained with

1. learning the embeddings during training
2. using pretrained embeddings from word2vec

To learn the embeddings, we make use of a dictionary of word-to-index, to convert sentences into tensors, which will later on be converted to a latent space. The embedding space is of dimension 100, due to it being computationally expensive.

4 Results

First of all, it is clearly observed that learning the embeddings ourselves is very computationally expensive, compared to using a pretrained word vectorizer. We also see that the distance between words does not make sense; synonyms and antonyms are at similar distances, and when we average the embeddings of sentences, they get much closer to each other than their meanings truly are, as seen in Table 1. Note: The distances are arbitrary (values such as 16 or 4.8 are meaningless alone), and only serve as a point of reference when compared to others from the same system.

The more important aspect to take into consideration is model performance. The accuracy of these models is displayed in the Table 2. In the case of learnt embeddings, both models quickly overfit and peak in performance at around $\tilde{0}.79$ accuracy on the dev set. On the other hand, using a pretrained word vectorizer, the model training is much more balanced and the train accuracy is closer to the dev accuracy. In this case, we see that the CNN slightly outperforms the linear model by 0.01 accuracy (which is a significant amount; we could consider this

sentence 1	sentence 2	distance
good	bad	16
bad	terrible	15.5
good	amazing	14.6
this movie makes me hate	this movie is literally the best in the world	4.8

Table 1: Distances between various sentences (sent1, sent2)

model	train accuracy	dev accuracy
linear learnt embeddings	0.99	0.78
CNN learnt embeddings 0.99	0.78	
linear pretrained embeddings	0.87	0.8
CNN learnt embeddings	0.91	0.81

Table 2: Accuracies of various models on train and dev dataset.

a 5% increase in performance when considering the difference from an accuracy of 1).

5 Conclusion

Our assumption about CNNs making use of word ordering does not hold up when we learn the embeddings ourselves; the embedding space is easily adapted by the network, leading to quick overfitting and stagnation in performance. The difference between CNNs and linear classifiers with averaged embeddings can be seen when the latent space is constant, and it has been trained on a larger dataset to be able to accurately convey meaning.