2019SP_MSDS_458-DL_SEC56 Week7- Assignment3

Prabhat Thakur

Deep Learning - Natural Language Processing (NLP) Study

Abstract: Natural Language Processing is the technology used to aid computers to understand the human's natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable. There has been remarkable progress in NLP algorithms and various neural network architectures can be used for text processing like fully connected (dense) network, one-dimensional convolutional neural network (CNN), and recurrent neural networks such as RNN, LSTM, and GRU. One of the application of NLP is Sentiment analysis, such as classifying the sentiment of tweets or movie reviews as positive or negative.

Introduction: The goal of this assignment is work on a text classification (sentiment analysis) problem using different deep learning neural networks to evaluate their classification performance and time required for training. In this exercise we will be fitting a fully connected (dense) network, one-dimensional convolutional neural network (CNN), recurrent neural networks using RNN & LSTM (Long Short-Term Memory) algorithms. We will also compare classification performs using one-hot coded text and word embedding in a fully connected (dense) network model.

We'll be working with "IMDB dataset", a set of 50,000 highly-polarized reviews from the Internet Movie Database. They are split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting in 50% negative and 50% positive reviews. The IMDB dataset comes

packaged with Keras. It has already been preprocessed: the reviews (sequences of words) have been turned into sequences of integers, where each integer stands for a specific word in a dictionary.

Literature: Natural Language Processing (NLP) is a way of analyzing texts by computerized means. NLP involves gathering of knowledge on how human beings understand and use language. This is done in order to develop appropriate tools and techniques which could make computer systems understand and manipulate natural languages to perform various desired tasks.

"Apart from common word processor operations that treat text like a mere sequence of symbols, NLP considers the hierarchical structure of language: several words make a phrase, several phrases make a sentence and, ultimately, sentences convey ideas," John Rehling, an NLP expert at Meltwater Group, said in How Natural Language Processing Helps Uncover Social Media Sentiment. "By analyzing language for its meaning, NLP systems have long filled useful roles, such as correcting grammar, converting speech to text and automatically translating between languages."

NLP is used to analyze text, allowing machines to understand how human's speak. This human-computer interaction enables real-world applications like automatic text summarization, sentiment analysis, topic extraction, named entity recognition, parts-of-speech tagging, relationship extraction, stemming, and more. NLP is commonly used for text mining, machine translation, and automated question answering.

Methods: I used Keras-based deep neural networks due to their higher performance and ease of training on large datasets. When loading the IMDB movie review text, we only keep the top

10,000 most frequently occurring words in the training data. Rare words will be discarded. This allows us to work with vector data of manageable size. The variables train_data and test_data are lists of reviews, each review being a list of word indices (encoding a sequence of words). train_labels and test_labels are lists of 0s and 1s, where 0 stands for "negative" and 1 stands for "positive":

As part of this exercise, we will evaluating classification performance of 5 different models. I used one hot encoding of words and word embedding methods to compare fully connect dense network model performance, Model 1 and Model 2 respectively. Other three models, Model 3 - using SimpleRNN layer, Model 4 - using LSTM layer , and Model 5- using Conv1D convonutional layer are built on top of word embedding layer.

One-hot encoding is the most common, most basic way to turn a token into a vector. It consists of associating a unique integer index with every word and then turning this integer index i into a binary vector of size N (the size of the vocabulary); the vector is all zeros except for the i'th entry, which is 1.

Dense word vectors is a popular and powerful way to associate a vector with a word, also called word embeddings. Whereas the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros), and very high-dimensional (same dimensionality as the number of words in the vocabulary), word embeddings are low dimensional floating-point vectors (that is, dense vectors, as opposed to sparse vectors). Unlike word vectors obtained via one-hot encoding, word embeddings are learned from data.

Final layer of all 5 models uses sigmoid activation so as to output a probability (a score between 0 and 1, indicating how likely the sample is to have the target "1", i.e. how likely the review is to be positive). Also as we are facing a binary classification problem and the output of

our network is a probability is it best to use the binary_crossentropy loss function. It isn't the only viable choice: you could use, for instance, mean_squared_error. But crossentropy is usually the best choice when you are dealing with models that output probabilities. Rmsprop optimizer is used and accuracy is monitor during training on train and validation data.

Results: From the training results , the training loss decreases with every epoch and the training accuracy increases with every epoch. This is expect when running gradient descent optimization - the quantity we are trying to minimize should get lower with every iteration. But that isn't the case for the validation loss and accuracy: they seem to peak at the fourth to six epoch in all the models. This is an example of "overfitting" - a model that performs better on the training data isn't necessarily a model that will do better on data it has never seen before. In this case, to prevent overfitting, I stopped the training after the epochs model started overfitting.

It is important to understand the impact and function of the different layers in a deep learning application. While the addition of multiple layers makes that interpretation more difficult, it is still critical to understand the elements of the input data the model is responding to when making a prediction.

Below table shows a comparison of performance between the five different models. The accuracy for all five is fairly high due to the simplicity of the data but worth observing the relative differences.

Models	**Structure**	**Training Time/Epoch**	**Test Accuracy**
Model1	Fully-connected (Dense) layers with relu activations. 2 Layers of Dense(16, activation='relu') , Output Layer Dense(1, activation='sigmoid') One-Hot encoding	6s	0.8694

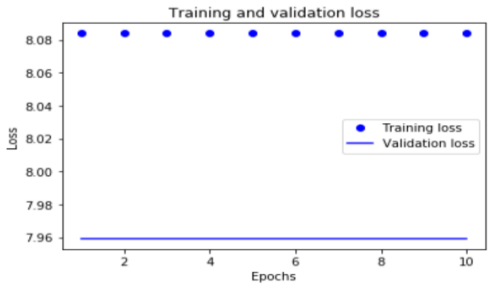
Model2	Fully-connected (Dense) layers with relu activations. 1 Layers of Dense(32, activation='relu') and 2nd layer with 16 nodes, Output Layer Dense(1, activation='sigmoid') Word Embedding	9s	0.8556
Model3	Embedding(10000, 32,500) and two SimpleRNN(32) layers Output Layer Dense(1, activation='sigmoid')	62s	0.7793
Model4	Embedding(10000, 32,500) and LSTM(32) layer Output Layer Dense(1, activation='sigmoid')	126s	0.8449
Model5	Embedding(10000, 32,500) -> Conv1D(32, 7, activation='relu')-> MaxPooling1D(5) -> Conv1D(32, 7, activation='relu') -> GlobalMaxPooling1D() -> Dense(1)	160s 155s	0.5 0.8262

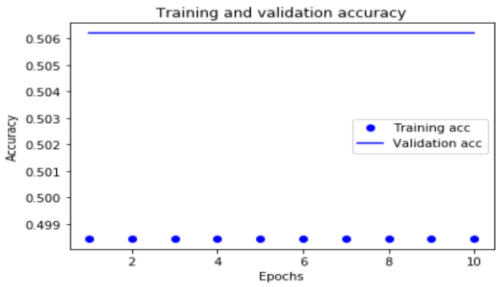
Plots of Loss and Accuracy between train and validation data set for all 5 models are geneated in the notebook html file.

Model 5 results were not consistent. Below are two results:

Output 1, 10 epochs:

```
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
```



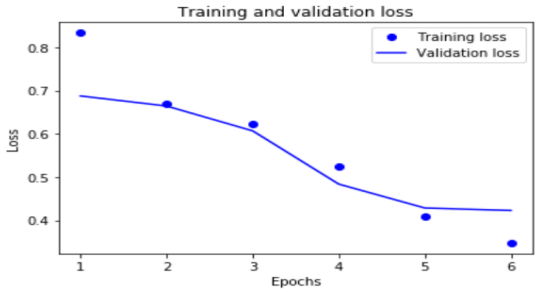


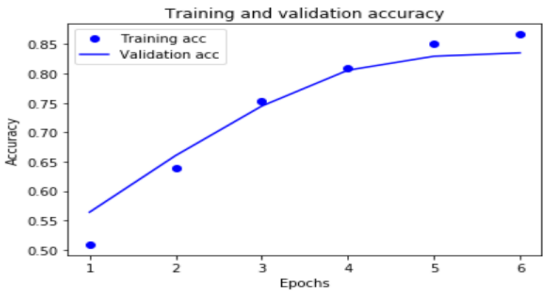

```
25000/25000 [==============================] - 78s 3ms/step

Test loss: 8.0590
Test accuracy: 0.5000
********************************************************************************
```

Output 2, 6 epochs:

```
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
_________________________________________________________________
```






```
25000/25000 [==============================] - 86s 3ms/step

Test loss: 0.4286
Test accuracy: 0.8262
********************************************************************************
```

Conclusion: For simple problems like this, it seems Model 1 :Fully-connected (Dense) layers with one-hot word encoding the is best choice as it less time expansive and give a decent prediction accuracy. After fitting a series of neural networks to the IMDB data, it is clear that incorporating convolutional layers adds computation time but also significantly increases accuracy. Too few nodes and layers lead to higher error, where too many nodes and layers tend to increase the time it takes to train a network. In addition, too many nodes and layers lead to a risk of overfitting the training data and obtaining a model that doesn't generalize well. For complex problem, because RNNs are extremely expensive for processing very long sequences, but 1D convnets are cheap, it can be a good idea to use a 1D convnet as a preprocessing step before an RNN, shortening the sequence and extracting useful representations for the RNN to process.

References

[1] Chollet, F. (2018). Deep learning with Python. Shelter Island, NY: Manning Publications.

[2] Joseph, Sethunya & Sedimo, Kutlwano & Kaniwa, Freeson & Hlomani, Hlomani & Letsholo, Keletso. (2016). Natural Language Processing: A Review. Natural Language Processing: A Review.

[3] Young, T., Hazarika, D., Poria, S., & Cambria, E. (2017). Recent trends in deep learning based natural language processing.https://arxiv.org/pdf/1708.02709.pdfGoogle Scholar

[4] Jumpstart Code: Dr. Miller and Dr. Maren

[5] https://blog.algorithmia.com/introduction-natural-language-processing-nlp/