Text Classification using various Neural Network Architectures in Keras

Wazeer Shah
Northwestern University

Abstract

Neural network models have demonstrated to be capable of achieving optimal performance in text classification. Keras is designed to be a high-level neural network API that enables the creation of multiple neural network architectures to achieve an optimal text classification score. Since Keras provides numerous layers that can be used to achieve the optimal score, it is important to understand the impact of each layer. In this paper, we take a close look at what role each unique keras layer may have in a neural network, and what model architecture contributes to the highest scoring predictor in determining if an Amazon Review's text sentiment is either positive or negative. The experimental results show that the LSTM model has the strongest performance over other various architectures such as Basic Embedding, Bi-directional LSTM, GloVe, and BERT.

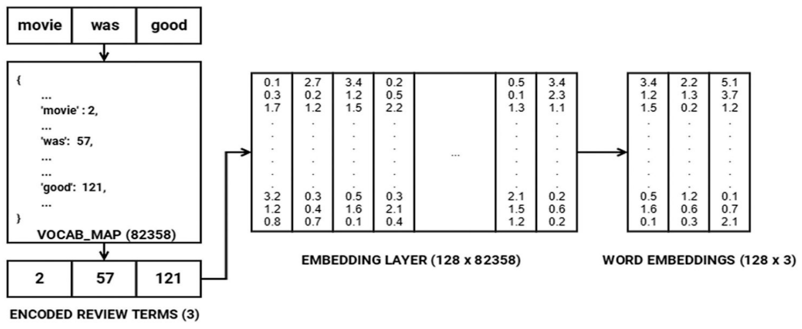
Introduction

The vast amount of reviews available on Amazon contain customer insights regarding specific products that can help product creators improve the overall customer experience by analyzing the negative reviews.The process of labeling sentiment for an Amazon Review is a very manual task, but with the ability of Keras this process can be automated with a strong performing model. However, the development of deep learning, pre-trained word embeddings, and transformers have led to breakthroughs in such problems by leveraging text classification. Layers are the basic building blocks of neural networks in Keras. In order to achieve an optimal accuracy for a particular review, it is important to find out the impact of each Keras layer. Therefore, in this paper, we evaluate the impact of adding a Keras layer or changing the properties of a neural network architecture on the classification accuracy. In this work, we compare the performance of models such as LSTM, Bi-Directional LSTM, GloVe, and BERT in order to reveal the most effective way to solve the sentiment prediction problem as of today.

The dataset used for this study consists of around 10 million Amazon customer reviews (input text) and star ratings (output labels) for learning how to train for sentiment analysis. Due to computing constraints, 280K rows have been randomly selected from the dataset to be used for model training and testing ("Amazon Reviews For Sentiment Analysis", 2020). The **Keras Tokenizer** is used to vectorize a text dictionary, by turning each text sentence into a sequence of integers. After fitting the training data, we turn each sentence within the review into sequences using the keras function which converts a sequence of text to a sequence of words. For our neural network, each sequence needs to possess the same length, therefore the keras pad_sequence feature allows us to take our sequences and determine a maximum sequence length for our primary input, which was set as 250. This is a number that can be tweaked based on the type of analysis but will be the baseline throughout the model training and testing phase. This data will effectively be the input layer that feeds into our various neural networks.

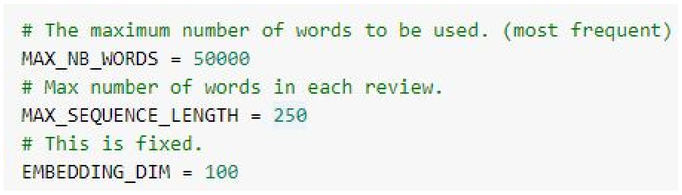
Word Embeddings

Word embedding is an NLP technique for representing words and documents using a dense vector representation compared to the bag of word techniques, which uses a large sparse vector representation. Embeddings are a class of NLP methods that aim to project the semantic meaning of words into a geometric space (Ghotra, Singh, Dua, 2017).



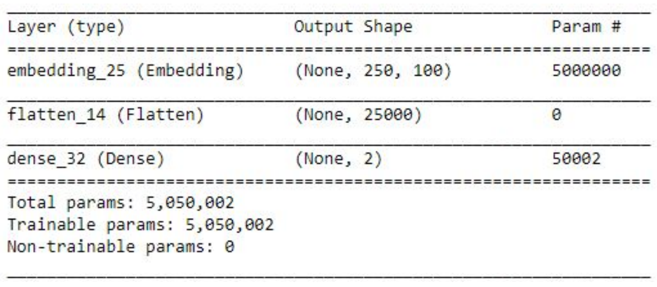
Neural Network Architecture

Using the Keras functional API, the **embedding** layer is always the second layer in the network after the input layer. The embedding layer uses parameters to properly feed in information from the input layer. These parameters include : the overall corpus vocabulary size, the size of the vector space we're embedding those words into, and the sequence length (Bernico, 2018). By specifying these parameters we're able to apply standardization in the presence of information in the embedding layer so that each array has a sufficient format appropriate for training purposes.



```
# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 50000
# Max number of words in each review.
MAX_SEQUENCE_LENGTH = 250
# This is fixed.
EMBEDDING_DIM = 100
```

The **Flatten** layer flattens the input, followed by the "sigmoid" optimizer. The sigmoid function's output is bounded between 0 and 1 and can be interpreted stochastically as the probability of the neuron activating (Vasilev et. al, 2019). The last layer in our neural network is the **dense** layer, and every node in one layer is connected to every other node in the adjacent layer, so it's classified as a fully interconnected or dense neural network (Warr, 2019).

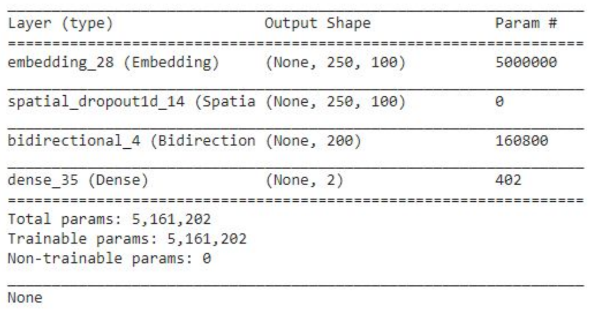


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_25 (Embedding)     (None, 250, 100)          5000000
_________________________________________________________________
flatten_14 (Flatten)         (None, 25000)             0
_________________________________________________________________
dense_32 (Dense)             (None, 2)                 50002
=================================================================
Total params: 5,050,002
Trainable params: 5,050,002
Non-trainable params: 0
_________________________________________________________________
```

The problem with this basic architecture is that it is missing a dropout layer, ensuring that the model does not place too much emphasis on certain weights (Loy, 2019). We decide to randomly drop with the dropout probability some of the values propagated inside our internal dense network of hidden layers. We do this by adding a keras **SpatialDropout1D,** this version performs the same function as Dropout, however, it drops entire 1D feature maps instead of individual elements. The addition of this layer can help improve the overall accuracy when evaluating the results.

LSTM

Additionally, another new and interesting approach to supervised deep learning is the use of long **short-term memory networks** (LSTMs). A benefit of the LSTM model is that they "try to overcome the shortcomings of RNN models, especially with regard to handling long-term dependencies and problems that occur when the weight matrix associated with the units (neurons) become too small (leading to vanishing gradient) or too large (leading to exploding gradient) (Sarkar, 2019). Dr. Abhishek Pal mentions in his Practical Time Series Analysis book that, "LSTM has been long preferred as the first choice for language models, which is evident from their extensive use in language translation, text generation, and sentiment classification" (Pal, Avishek, and P. K. S. Prakash, 2017). A **bidirectional LSTM**, as shown below, not only uses the sequence of integers provided as input but also makes use of its reverse order as additional input. There could be situations where this approach may help to achieve further model classification performance improvements by capturing useful patterns in the data that may not have been captured by the original LSTM network (Rai, 2019).



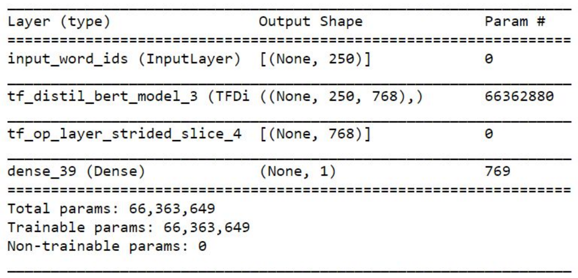
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_28 (Embedding)     (None, 250, 100)          5000000
_________________________________________________________________
spatial_dropout1d_14 (Spatia (None, 250, 100)          0
_________________________________________________________________
bidirectional_4 (Bidirection (None, 200)               160800
_________________________________________________________________
dense_35 (Dense)             (None, 2)                 402
=================================================================
Total params: 5,161,202
Trainable params: 5,161,202
Non-trainable params: 0
_________________________________________________________________
None
```

GloVe

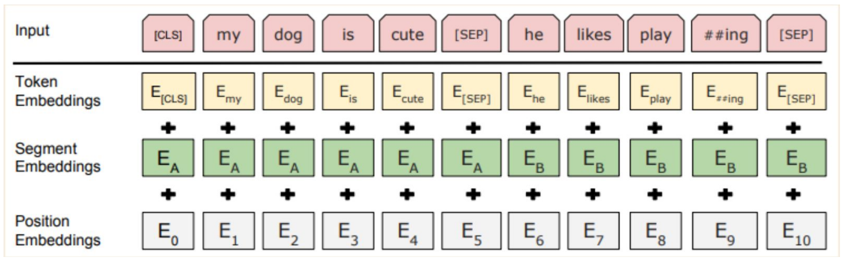
Released in 2014, GloVe is a pre-trained word embeddings vectors library curated by Jeff Pennington's team from Stanford University (Pennington et al., 2014). GloVe uses global word-to-word co-occurrence statistics from a corpus to train the model and derive the GloVe vectors. By creating the coefficient matrix for the training data, we are able to map the embedding layer weights to the GloVe embeddings, and can proceed to create a fully connected layer. Unfortunately, the GloVe method yielded the least successful results and may have benefited from an incorporation of an LSTM layer.

BERT

BERT introduces a number of new concepts to NLP and refines some old ones as well, as we will see. Likewise, the NLP techniques it introduces are now considered state of the art and will be leveraged in this analysis. Released in 2018, "BERT works by applying the concept of bidirectional encoding/decoding to an attention model called a **Transformer**. The Transformer learns contextual relations in words without the use of RNN or CNN but with an attention masking technique. Instead of reading a sequence of text like we saw before, BERT consumes the entire text all at once. It uses a masking technique called Masked LM (MLM) that randomly masks words in the input and then tries to predict them (Lanham, 2020). The transformer model eliminates the need for an LSTM and can produce better results depending on the text that is being dealt with. As shown in the figure below the BERT transformer layer is a representation of a combination of the token embeddings, segmentation embeddings, and the position embeddings (Vasilev et. al, 2019).



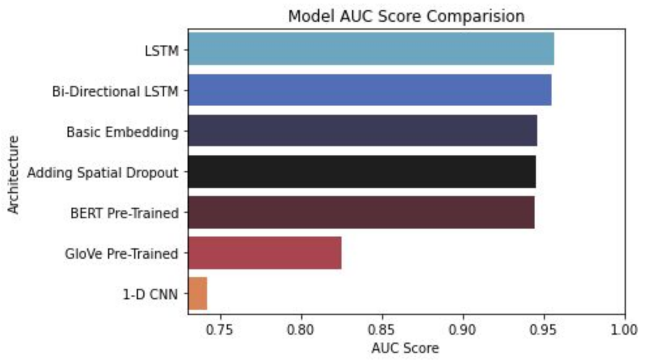
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_word_ids (InputLayer)  [(None, 250)]             0
_________________________________________________________________
tf_distil_bert_model_3 (TFDi ((None, 250, 768),)       66362880
_________________________________________________________________
tf_op_layer_strided_slice_4  [(None, 768)]             0
_________________________________________________________________
dense_39 (Dense)             (None, 1)                 769
=================================================================
Total params: 66,363,649
Trainable params: 66,363,649
Non-trainable params: 0
_________________________________________________________________
```


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Results

When evaluating the results, I decided to use the ROC curve as opposed to accuracy as the deciding metric for the strongest performing model. In his Machine Learning book Aurélien Géron explains, “The area under the ROC curve is one of the most used metrics for the evaluation of binary classification problems. The ROC curve is able to take into account the false positive rate (specificity) and the true positive rate (sensitivity). In a binary classification problem there is a trade-off between specificity and sensitivity. **AUC** is the area under the ROC curve and is in the range of [0,1], with the greater value representing a stronger model. A great utilization of AUC is to visually compare different algorithms. If the AUC score is rather low such as 0.5, this reflects that the model requires more features and/or more training data” (Garon, 2019).

Based on the results the LSTM model has the highest AUC score of **95.6**. Although this was the best performer, there are many things that were out of the scope of this analysis that could’ve improved the overall AUC scores. The LSTM unit size was 100 for this analysis, but it would have been interesting to see what the impact of the unit size would have on the performance of the model. Another method that could have been explored would be to “stack” LSTM models by adding another LSTM layer after the initial one and setting the return_sequences property to True. Overfitting also can be a big factor in determining the strength of each model, and although these properties were explored, tweaking recurrent dropout rate, learning rate, and activation functions could render different AUC scores. In this analysis the number of epochs run for every model was 5, and this number can be reconsidered to identify the ideal number of epochs. The basic embedding models reveal significant overfitting when approaching 5 epochs so it was decided to keep every model’s epoch count no more than 5.



Conclusions

Neural networks provide an effective method in solving text classification problems. However, there exist various architectures such as Basic Embedding, LSTM, Bi-Directional LSTM, GloVE, and BERT that can be used for accurate text classification. In this paper we look at various architectures to see which architectures produce the most optimal results. Our results show that using an LSTM model produces the highest score in determining the sentiment for Amazon reviews based on the Amazon Review dataset.

There are multiple deep learning methods that are worth experimenting to expand this analysis such as incorporation of LSTM with the GloVe pre-trained word embeddings. Additionally, other neural networks & transformers such as fasttext & GRUs could also be tested. Although the Amazon Reviews dataset was used for this study, alternative text classification datasets such as twitter or movie reviews could also be used to examine if a particular architecture works better with certain types of texts. It would also be worth comparing traditional machine learning techniques such as Tf-Idf & Random Forests, to see the impact that a change in the dataset size can have for the deep neural networks over traditional Machine Learning methods. Overall, this analysis does show that text classification problems are getting easier to solve with newer neural network architectures and pre-trained word embeddings and it will be interesting to see what the next breakthrough in deep learning will bring to the world of NLP.

References

Lai, Siwei, Liheng Xu, Kang Liu, and Jun Zhao. "Recurrent convolutional neural networks for text classification." In Twenty-ninth AAAI conference on artificial intelligence. 2015.

Liu, Bing. "Sentiment analysis and opinion mining." Synthesis lectures on human language technologies 5, no. 1 (2012): 1-167.

Trstenjak, Bruno, Sasa Mikac, and Dzenana Donko. "KNN with TF-IDF based framework for text categorization." Procedia Engineering 69 (2014): 1356-1364.

Ghotra, Manpreet Singh, and Rajdeep Dua. Keras Deep Learning Cookbook Packt Publishing Ltd, 2017.

Sarkar, Dipanjan. Text analytics with Python: a practitioner's guide to natural language processing. Apress, 2019.

"Tf.Keras.Preprocessing.Text.Tokenizer | Tensorflow Core V2.3.0". 2020. Tensorflow. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer.

"Amazon Reviews For Sentiment Analysis". 2020. Kaggle. https://www.kaggle.com/bittlingmayer/amazonreviews.

Bernico, Michael. Deep learning quick reference: useful hacks for training and optimizing deep neural networks with TensorFlow and Keras. Packt Publishing Ltd, 2018.

Vasilev, Ivan, Daniel Slater, Gianmario Spacagna, Peter Roelants, and Valentino Zocca. Python Deep

Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow. Packt Publishing Ltd, 2019.

Loy, James. Neural Network Projects with Python: The ultimate guide to using Python to explore the true power of neural networks through six projects. Packt Publishing Ltd, 2019.

Pal, Avishek, and P. K. S. Prakash. Practical Time Series Analysis: Master Time Series Data Processing, Visualization, and Modeling using Python. Packt Publishing Ltd, 2017.

Rai, Bharatendra. "Advanced Deep Learning with R." Experimenting with a bidirectional LSTM layer (2019). Packt Publishing Ltd, 2019.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. In Proc. of EMNLP, 2014.

Lanham. 2020. "Practical AI on the Google Cloud Platform". Understanding Language in the Cloud. O'Reilly Publishing, 2020.

Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, 2019.