Abstract

Natural Language Processing (NLP) is one form of AI that has been receiving a lot of attention, as it directly engages with humans and must overcome barriers like speech recognition and language understanding to be successful. An adapted Reuters data set from Keras, with one topic per document, is used to determine the optimal structure of an NLP model, focusing on the number of layers and types of cells within each layer. A dense network serves as a baseline, while Recurrent Neural Networks (RNNs) with and without LSTM cells are used to test the effectiveness of different model architectures. Ultimately, this analysis serves as an introduction to NLP, with commentary on future enhancements and challenges in the form of a hypothetical chatbot implementation.

Introduction

NLP has far-reaching applications across various industries but is centered on its ability to help organizations gain efficiencies through the mining and understanding of this data. For instance, an organization that uses NLP can influence its customer service processes through services like a chatbot and better understand and respond to customer sentiment by mining reviews. Given its broad applicability, this assignment serves as an introduction to RNN models and explores different architectures to help inform future modeling endeavors. By using an adapted version of the Keras Retuers dataset, the following goals can be achieved:

1. Compare and contrast model performance of a dense network and a RNN.
2. Test the performance of different RNN architectures and summarize findings. There is emphasis on exploring the number of layers and types of cells within each layer as well as a flat versus pyramid structure.
3. Generalize findings to a hypothetical chatbot implementation.

Literature Review

The Chollet Deep Learning with Python textbook was the primary source for this assignment and provided code on word embeddings and basic RNN architectures. Moreover, this textbook helped to outline the importance of a RNN model over a dense model by stating: "A major characteristic of…densely connected networks…is that they have no memory. Each input shown to them is processed independently, with no state kept in between inputs. A recurrent neural network (RNN)….processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far" (Chollet, p. 196). This distinction is crucial when working with text data, as both past and recent information play role in contextual understanding. Therefore, these two different types of models are evaluated since the way they process information is extremely different.

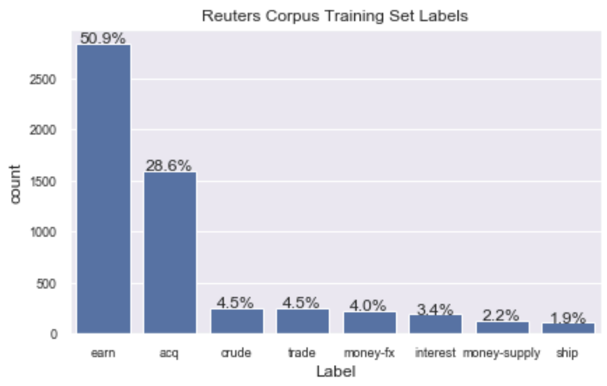
Furthermore, there are several different cells or layers that are tested in this assignment. While a vanilla RNN is considered, it is really only provided as a baseline. Through several sources, it was better understood that one of the limitations of training an RNN model on long sequences is that it may suffer from the vanishing/exploding gradients problem. Therefore, certain types of cells, such as LSTM, have been created and are primarily used when building RNN models. According to Jozefowicz et al. (2015): “As a result [of vanishing/exploding gradients], RNNs can easily learn the short-term but not the long-term dependencies. The LSTM addresses the vanishing gradient problem by reparameterizing the RNN. Thus, while the LSTM does not have a representational advantage, its gradient cannot vanish” (p. 2343). In short, LSTM cells are very powerful and help to ensure that long-term dependencies are detected in the data. Most of the models generated in this assignment use LSTM cells for this reason.

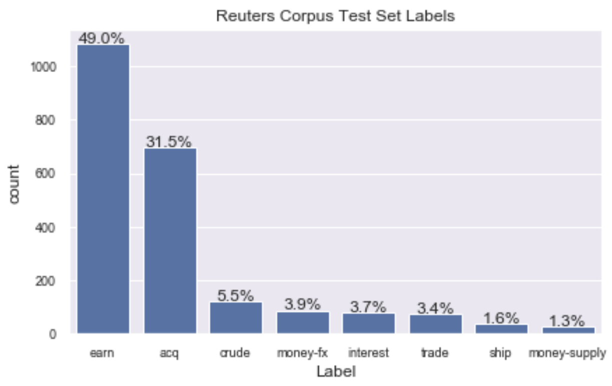
Finally, Bengio provided guidance on broader model architecture through his comments on the size of hidden layers. He states, “we found that using the same size for all layers worked generally better or the same as using a decreasing size (pyramid-like) or increasing size (upside down pyramid), but of course this may be data dependent” (p. 11). While this finding was tested in the previous assignment, it is considered here as well as the problem definition is entirely different.

Methods

Data Preparation

Several different data preparation methods were used to prepare the data for modeling. First, after the data was read in and sorted into training and test sets, the labels of the data set were inspected. Below is the distribution of labels for the training and test sets.





From the visuals above, right away, it is evident that both the training and test sets have imbalanced data sets. The majority of the labels are for ‘earn,’ followed by ‘acq.’ This observation is extremely important when splitting and building models as class weights will be used during training to appropriately weight misclassifications, giving higher weight to the minority classes.

In working with NLP problems, one of the main data preparation tasks is to tokenize the data. For this dataset, word embeddings are used over one-hot word vectors because "word embeddings pack more information into far fewer dimension" (Chollet, p.184). The process of tokenizing the data involves splitting the text of each document into words and encoding only the unique words as integer values. For this assignment, only the top 10,000 words in the data set are used, and each document is cut off after 300 words, as the 25% percentile of the original training set was 241. If a document has a length under 300, then it will be padded with zeroes. During this tokenization process, the training data set is further split into training and validation sets, using a 60/40 split. Lastly, the labels are transformed into a one-hot-encoding scheme, as this format is required for modeling. This process is difficult to visualize, and as such, sample outputs are omitted.

Modeling Approach

Overall, twelve different models are built. With all models, the number of epochs is held constant at 10 to help speed up model building time. When dropout is used, then the number of epochs increases to 20 since convergence is slower.

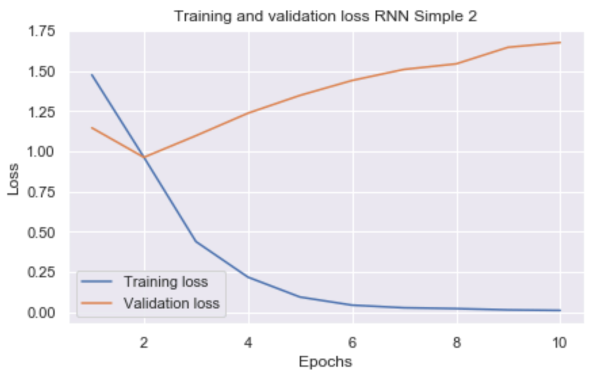
Several different types of models are considered, including: dense, vanilla RNN (uses simpleRNN cells), LSTM RNN, and No LSTM (a model with just an embedding layer). This last model, No LSTM, is added because it is "a model that treats each word in the input sequence separately without considering inter-word relationships and sentence structure" (Chollet, p. 187). Therefore, both the dense and No LSTM models provide a good comparison to RNN models, which look holistically at the data. Models with a single layer are considered, although greater emphasis is given toward models with two to three hidden layers.

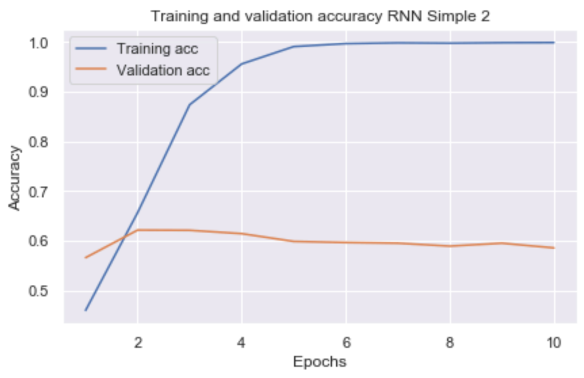
Results

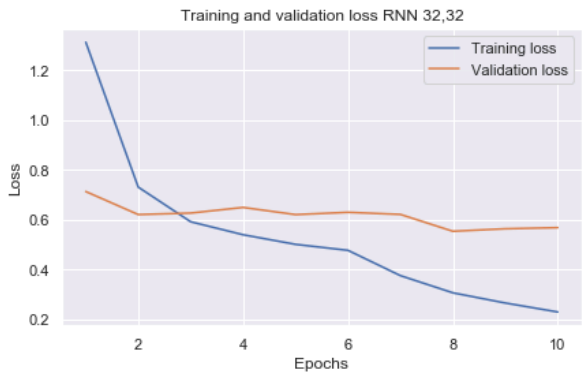
The table below captures accuracy and processing time results for all models. While the dense networks have the lowest training times, they have the worst accuracy results amongst all models. Surprisingly, the No LSTM and simple models perform fairly well on the training data; however, they are overfitting due to their wide separation between training and validation set results. The LSTM models seem to have the best performance, as the training and validation accuracies are very similar. The pyramid models seem to have marginally better performance, but since the flat structures have lower processing times, they are slightly preferred from a model-building perspective. Lastly, by adding both dropout and recurrent dropout, both the training and validation accuracies are improved. It is noted, though, that the processing times for these models are significantly longer than their non-dropout counterparts.

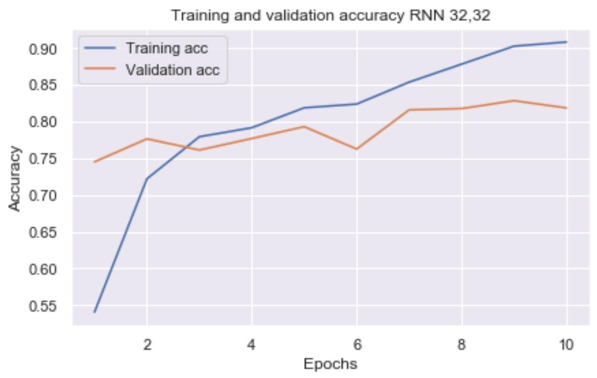
```
+--------------------------------------------------------------------------------------------+
|                                      Training Results                                      |
+------------------+--------------+----------------+--------------+-----------------+
|      Model       | Layers/Nodes | Train Accuracy | Val Accuracy | Processing Time |
+------------------+--------------+----------------+--------------+-----------------+
|    Dense Base    |      32      |     46.2%      |    53.2%     |       3.3       |
|  Dense Multiple  |    16,32     |     37.6%      |    44.1%     |       3.7       |
|  RNN Base LSTM   |      32      |     73.6%      |    75.3%     |      561.6      |
| RNN Base No LSTM |      32      |     93.1%      |    89.2%     |       13.9      |
| RNN Flat Simple  |      32      |     87.9%      |    72.6%     |      120.8      |
| RNN Flat Simple  |   32,32,32   |     89.3%      |    59.8%     |      2694.3     |
|  RNN Flat LSTM   |    32,32     |     80.2%      |    79.0%     |      857.0      |
| RNN Pyramid LSTM |    32,64     |     83.4%      |    82.4%     |      879.0      |
|  RNN Flat LSTM   |   32,32,32   |     78.4%      |    79.1%     |      1342.4     |
| RNN Pyramid LSTM |  32,64,128   |     83.9%      |    80.5%     |      1447.0     |
+---------------+---------------------+----------------+--------------+-----------------+
| RNN Flat LSTM |  32,32 w/ dropout   |     83.4%      |    81.5%     |      1830.5     |
| RNN Flat LSTM | 32,32,32 w/ dropout |     81.7%      |    79.4%     |      2779.9     |
+---------------+---------------------+----------------+--------------+-----------------+
```

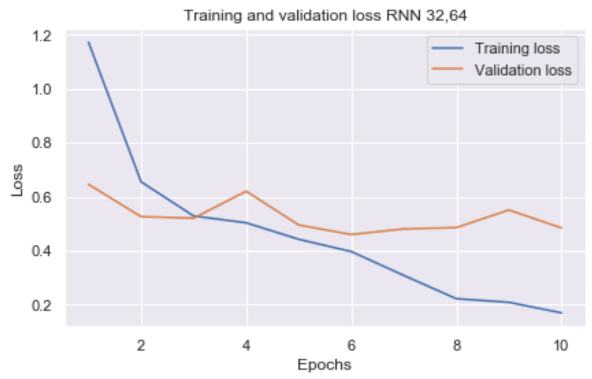
To expand upon the results summarized above, loss and accuracy curves are shown for select models. Specifically, the RNN simple model with layers 32,32,32 is included to show how inferior it is compared to models with LSTM cells once the layers increase. The flat and pyramid RNN models are also included because while the pyramid model had slightly better performance, it is has worse overfitting as the epochs increase. With the dropout model, the curves are flatter, but there overfitting is still occurring. While regularization was not a focus in this assignment, different values for dropout and other techniques to combat overfitting should be explored.

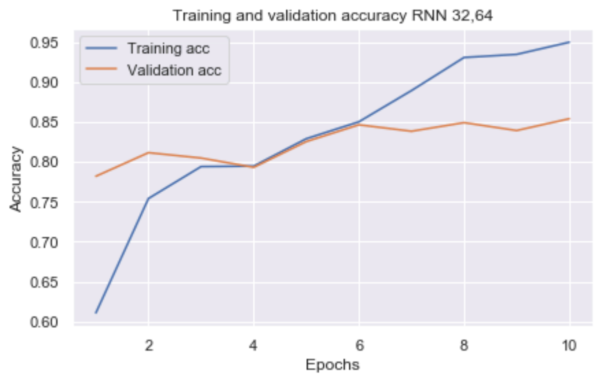


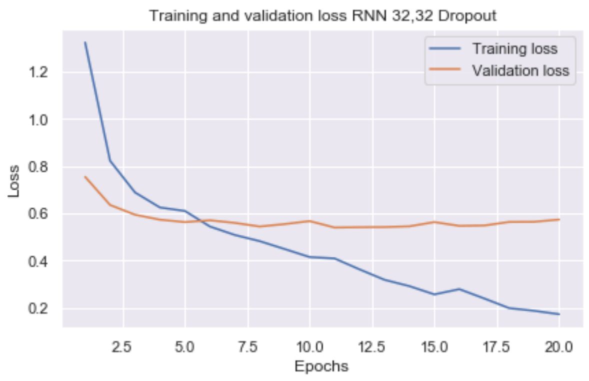


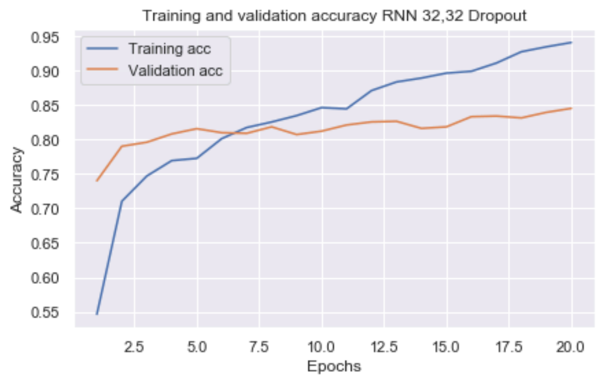












As a final step in the model building process, all models were evaluated on the test set. The table below summarizes these results. It is worth mentioning that the test data had to be tokenized using the same tokenizer from the training data sets. Otherwise, the test data set would have been encoded using a complete separate pattern.

From the table below, it is clearly evident that test set performance is far and below the training and validation results. Techniques such as k-fold cross validation may aid in this endeavor. Regardless, the model with the best test set performance is the Flat LSTM model with two layers. LSTM models outperformed dense and simple RNN models, although the No LSTM model had comparable performance to other LSTM models. The dropout models had slightly inferior test set performance, which was anticipated. In summary, there appears to be minimal difference between flat and pyramid models and LSTM models have enhanced performance over non-LSTM models.

```
+-----------------------------------------------------+
|                  Test Set Results                   |
+------------------+--------------------+----------+
|      Model       |    Layers/Nodes    | Accuracy |
+------------------+--------------------+----------+
|    Dense Base    |         32         |  51.3%   |
|  Dense Multiple  |       16,32        |  48.7%   |
|  RNN Base LSTM   |         32         |  61.6%   |
| RNN Base No LSTM |         32         |  62.6%   |
| RNN Flat Simple  |         32         |  59.6%   |
| RNN Flat Simple  |      32,32,32      |  50.1%   |
|  RNN Flat LSTM   |       32,32        |  63.6%   |
| RNN Pyramid LSTM |       32,64        |  62.5%   |
|  RNN Flat LSTM   |      32,32,32      |  62.9%   |
| RNN Pyramid LSTM |     32,64,128      |  62.9%   |
|  RNN Flat LSTM   |   32,32 w/ dropout |  61.3%   |
|  RNN Flat LSTM   | 32,32,32 w/ dropout|  61.2%   |
+------------------+--------------------+----------+
```

If more time allowed, then additional model architectures would have been tested. Some ideas for future learning, include: comparing model performance using pre-trained word embeddings, assessing if 1D-convolutional neural networks or bidirectional RNNs offer a lift in performance, and varying the size of each document as well as the vocabulary size. In addition, more time could be spent on hyperparameter tuning. These additional tests would provide other perspectives on a more optimal model architecture for this data set.

Conclusions

While this assignment explored different NLP models for an adapted version of the Keras Reuters dataset, it helped to show that building—let alone implementing—a NLP model is no small endeavor. If management wanted to move forward with a chatbot to assist customer support representatives, it would first be important to understand where this chatbot fits within the customer lifecycle/journey. Understanding the business goals of this chatbot and what kind of tasks it will and will not solve is vitally important. Also, it is a worthwhile exercise to spend ample time up front making sure that this chatbot is going to meet business needs and that other less cost-intensive solutions, such as A/B testing or UI design, are not viable options. More importantly, it is necessary to ensure that a measurement framework is in place to validate that the chatbot is providing efficiency gains and meeting business goals.

The design of any chatbot is crucial. Because a chatbot involves the processing of natural language where both short- and long-term dependences are important, a RNN model is needed. However, the setup of this model will vary greatly depending on the business goals and complexity required. For instance, the model could be a supervised model that assigns a topic or similarity measure between user-input data and training data, which could in turn lead to a very specific action based on the data (Pandey, 2018). In contrast, the model could also be structured as a generative model, which creates more flexibility and adds a more 'human touch' in how the chatbot handles a response (Pandey, 2018). Moreover, multiple models could be stacked together to help classify the problem and then identify the solution.

Regardless of the approach taken, there are several considerations that are needed to make sure the implementation is successful. First, it is important to determine if a custom solution or a third-party solution can best meet business needs. Researching available options is paramount to ensure that whatever solution is used covers all the bases. Second, ample discussions are needed to ensure that the organization has the appropriate infrastructure to support the real-time nature of a chatbot, as it carries huge weight on the customer experience. Third, the design phase will be lengthy and needs to cover topics such as conversation structure, stop words, actions that the chatbot should take, and modeling frameworks. Last, it would be advantageous to roll out the chatbot on an iterative basis. By starting small and making incremental improvements, there will be minimal negative impacts to the customer experience.

References

Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. In G. Montavon, G. B. Orr, and K-R. Muller, eds., *Neural Network: Tricks of the Trade* (second edition). New York: Springer.

Chollet, F. (2018). Deep Learning with Python. Shelter Island: Manning.

Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever. Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:2342-2350, 2015.

Pandey, P. (2018, September 17). Building a Simple Chatbot from Scratch in Python (using NLTK). Retrieved May 21, 2019, from https://medium.com/analytics-vidhya/building-a-simple-chatbot-in-python-using-nltk-7c8c8215ac6e

Real Python. (2018, October 23). Practical Text Classification With Python and Keras – Real Python. Retrieved May 21, 2019, from https://realpython.com/python-keras-text-classification/#keras-embedding-layer