A.3 Second Research/Programming Assignment

By: Robin Singh

Abstract:

For this assignment, I am working with the AG_News_Subset data from TensorFlow, which consists of Reuters data of news articles and which categories they belong in. In particular, I am working with the descriptions of the articles and the category they are assigned. I compared various NLP algorithms to determine the best way to properly analyze and categorize the news articles based on their description after mapping the descriptions to various sizes of dictionaries to be vectorized, trying to find the best setup and hyperparameters for a Simple RNN and with LTSM layers (both unidirectional and bidirectional), and work with a 1-dimensional CNN to determine the best algorithm for this set of data. The management question to be answered this time is which method would be optimal to create a chatbot that could assist customer support representatives in this case, and whether this would be a substantial development project.

Introduction:

This research is conducted to analyze various parameters and features of a recurrent neural network and compare it against a convolutional neural network to determine what they mean and how they help make up a successful neural network for predicting objects in the AG_News_Subset dataset. We need to determine how to preprocess the data, split the data into train, validation, and test sets, and build the neural networks capable to accepting the data in the processed format and successfully classify them into one of 4 categories in the end. The accuracy will be judged by how well it is predicting the set of vectorized descriptions into objects after being trained by the training set, validated by the validation set, and finally tested on the test set that had not yet been seen by the model, which would be used to determine the

essential features of a chatbot system that would require a neural network to conversate and assist customers with their problems in the future. This way, in a real-world scenario, had this occurred, we would know what type of neural network to use and what it takes for that neural network to successfully analyze the precise details and features of the text entered by comparing it with similar text entries that it had been trained on, what sort of layers the neural network should have, how many hidden layers the neural network should have, how the results should be categorized, and what we need to be weary of when feeding the data in (such as similar numbers, human error, etc.).

Literature Review:

Methods:

For the first experiment, I will run the default RNN setup consisting of the LTSM layer presented in the GitHub code. I will compare the performance against different vocabulary sizes (1000, 2000, and 3000 words).

Afterwards, I will conduct a similar RNN setup but replacing the LTSM layer with a Simple RNN layer instead with varying nodes within the hidden layers.

Next, I will compare various RNN setups with LTSM layers with varying node and hidden layer amounts.

Finally, we will compare the performance of those RNNs with a 1D CNN to compare which would be the best neural network to use in a scenario similar to mine.

For ease of reading and comparing the various performances of the different neural networks, a table with the various algorithms' accuracies is presented under the results section.

The setups have a default batch size of 64, buffer size of 10000, and run for 200 epochs with an early stop callback with a patience of 2 epochs.

Methods:

Neural Network	Initial Validation Accuracy	Validation Accuracy	Test Accuracy
Default GitHub RNN with LTSM layer (Vocab Size of 1000)	0.8263	0.8585	0.8497
Default GitHub RNN with LTSM layer (Vocab Size of 2000)	0.8600	0.8805	0.8739
Default GitHub RNN with LTSM layer (Vocab Size of 3000)	0.8515	0.8820	0.8788
Default GitHub RNN but with Simple RNN layer (Vocab Size of 1000)	0.8257	0.8545	0.8479
Default GitHub RNN but with 2 Simple RNN layer with 128 and 64 nodes respectively (Vocab Size of 1000)	0.8135	0.8224	0.8293
Default GitHub RNN but with Simple RNN layer (64 nodes) and additional Dense layer w/ 64 nodes (Vocab Size of 1000)	0.8412	0.8550	0.8447
Alternative RNN with GRU layer (256 nodes) and Simple RNN layer (128 nodes) and additional Dense layer (64 nodes)	0.6695	0.8413	0.8317
Default GitHub RNN but with additional LTSM layer of 32 nodes after 64 node original (Vocab Size of 1000)	0.8367	0.8448	0.8442

Default GitHub RNN but with additional LTSM layer of 128 nodes and LTSM layer of 64 nodes after original (Vocab Size of 1000)	0.8599	0.8628	0.8528
Default GitHub RNN but with unidirectional LTSM layer instead of original Bidirectional as before (Vocab Size of 1000)	0.8272	0.8605	0.8545
Alternative Conv1D setup with 128 node Conv1d layer and 0.5 dropout layer and “regularizers” (Vocab Size of 1000)	0.8017	0.8553	0.8497
Alternative Conv1D setup with 128 node Conv1d layer, 0.5 dropout layer, and 64 node dense layer and “regularizers” (Vocab Size of 1000)	0.8302	0.8680	0.8572

Results:

As you can see from above, most of the algorithms resulted in very similar results, with the highest accuracy coming from the default RNN with a 3000-word vocabulary. That means that the more words the algorithm must work with, the better it performed as it had a diverse corpus to choose and vectorize words from. However, it was most insightful to recognize how the results were achieved and the differences in the learning process of the algorithms. There were a few things that stuck out to me. The LTSM layer was by far the most successful and efficient neural network to work with in this case. It learned the data quicker, as show with a

high initial validation set accuracy when compared with the other neural networks. This means it also runs for fewer epochs as it learns the trends and features sooner. The LTSM algorithms are also more confident in its predictions when compared to other neural networks, as it confusion matrices show a significantly higher percentage on its prediction of which category the news article should fall under. For example, you can see the default LTSM compared to a Simple RNN and a 1D CNN here.

LTSM:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
World	0.15%	99.97%	0.03%	2.18%	99.85%	0.27%	1.49%	6.07%	99.00%	0.73%	99.99%	0.10%	0.51%	89.31%	99.97%
Sports	99.72%	0.00%	0.00%	97.37%	0.00%	0.02%	98.18%	93.68%	0.01%	0.03%	0.00%	0.00%	0.03%	0.83%	0.00%
Business	0.04%	0.01%	5.68%	0.40%	0.06%	6.90%	0.08%	0.03%	0.25%	98.77%	0.01%	0.36%	98.91%	2.08%	0.01%
Sci/Tech	0.09%	0.02%	94.29%	0.05%	0.09%	92.81%	0.25%	0.22%	0.74%	0.47%	0.01%	99.54%	0.55%	7.78%	0.02%

Simple RNN:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
World	0.16%	99.94%	0.14%	0.52%	99.61%	0.58%	0.69%	18.71%	91.67%	1.60%	99.95%	0.19%	0.64%	91.15%	99.53%
Sports	99.78%	0.00%	0.02%	99.34%	0.02%	0.21%	98.97%	78.18%	0.40%	0.10%	0.00%	1.40%	0.05%	0.98%	0.00%
Business	0.03%	0.03%	8.83%	0.08%	0.13%	14.18%	0.08%	0.22%	0.81%	78.87%	0.02%	1.10%	98.05%	1.81%	0.19%
Sci/Tech	0.02%	0.03%	91.01%	0.06%	0.24%	85.03%	0.25%	2.89%	7.13%	19.43%	0.02%	97.31%	1.27%	6.06%	0.28%

1D CNN:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
World	28.02%	98.61%	10.77%	31.99%	95.33%	16.25%	44.45%	81.39%	90.95%	22.61%	99.47%	7.30%	35.58%	92.51%	99.24%
Sports	99.38%	16.03%	4.01%	97.56%	17.49%	5.10%	99.20%	89.06%	17.25%	7.54%	5.08%	28.01%	4.48%	15.33%	0.68%
Business	9.75%	20.48%	52.39%	16.51%	20.50%	81.47%	3.68%	4.96%	27.84%	89.54%	17.27%	28.69%	93.65%	43.66%	45.84%
Sci/Tech	5.12%	12.82%	95.42%	12.22%	31.36%	84.18%	10.07%	32.89%	43.44%	61.21%	15.40%	92.00%	38.64%	21.42%	26.03%

The LTSM algorithms were also the quickest to learn in general. The Simple RNN was by far the slowest in learning the trends, taking the longest time to run each epoch even with the assistance of a GPU. The 1D CNN had a quicker time per epoch, but ran more epochs due to slightly more unstable results.

LTSM Time:

```
Epoch 1/200
1782/1782 [==============================] - 46s 21ms/step - loss: 0.6858 - accuracy: 0.7437 - val_loss: 0.4366 - val_accuracy: 0.8515
Epoch 2/200
1782/1782 [==============================] - 36s 20ms/step - loss: 0.3773 - accuracy: 0.8712 - val_loss: 0.3504 - val_accuracy: 0.8788
Epoch 3/200
1782/1782 [==============================] - 35s 20ms/step - loss: 0.3278 - accuracy: 0.8872 - val_loss: 0.3302 - val_accuracy: 0.8867
Epoch 4/200
1782/1782 [==============================] - 35s 20ms/step - loss: 0.3027 - accuracy: 0.8947 - val_loss: 0.3213 - val_accuracy: 0.8883
Epoch 5/200
1782/1782 [==============================] - 35s 20ms/step - loss: 0.2877 - accuracy: 0.8989 - val_loss: 0.3197 - val_accuracy: 0.8870
Epoch 6/200
1782/1782 [==============================] - 35s 19ms/step - loss: 0.2785 - accuracy: 0.9019 - val_loss: 0.3238 - val_accuracy: 0.8820
```

Simple RNN time:

```
Epoch 1/200
1782/1782 [==============================] - 172s 96ms/step - loss: 0.7624 - accuracy: 0.6924 - val_loss: 0.5232 - val_accuracy: 0.8140
Epoch 2/200
1782/1782 [==============================] - 168s 94ms/step - loss: 0.4796 - accuracy: 0.8285 - val_loss: 0.4687 - val_accuracy: 0.8332
Epoch 3/200
1782/1782 [==============================] - 170s 95ms/step - loss: 0.4570 - accuracy: 0.8360 - val_loss: 0.4734 - val_accuracy: 0.8320
Epoch 4/200
1782/1782 [==============================] - 170s 95ms/step - loss: 0.4469 - accuracy: 0.8403 - val_loss: 0.4576 - val_accuracy: 0.8378
Epoch 5/200
1782/1782 [==============================] - 170s 95ms/step - loss: 0.4394 - accuracy: 0.8418 - val_loss: 0.4462 - val_accuracy: 0.8423
Epoch 6/200
1782/1782 [==============================] - 170s 95ms/step - loss: 0.4334 - accuracy: 0.8440 - val_loss: 0.4464 - val_accuracy: 0.8375
Epoch 7/200
1782/1782 [==============================] - 169s 95ms/step - loss: 0.4264 - accuracy: 0.8467 - val_loss: 0.4363 - val_accuracy: 0.8422
```

1D CNN time:

```
Epoch 1/200
1782/1782 [==============================] - 26s 13ms/step - loss: 1.1315 - accuracy: 0.6096 - val_loss: 0.6833 - val_accuracy: 0.8017
Epoch 2/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.6185 - accuracy: 0.8142 - val_loss: 0.5504 - val_accuracy: 0.8338
Epoch 3/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.5586 - accuracy: 0.8310 - val_loss: 0.5206 - val_accuracy: 0.8420
Epoch 4/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.5362 - accuracy: 0.8372 - val_loss: 0.5058 - val_accuracy: 0.8482
Epoch 5/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.5239 - accuracy: 0.8401 - val_loss: 0.4958 - val_accuracy: 0.8483
Epoch 6/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.5117 - accuracy: 0.8427 - val_loss: 0.4888 - val_accuracy: 0.8507
Epoch 7/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.5043 - accuracy: 0.8456 - val_loss: 0.4835 - val_accuracy: 0.8527
Epoch 8/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.4973 - accuracy: 0.8463 - val_loss: 0.4788 - val_accuracy: 0.8533
Epoch 9/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.4920 - accuracy: 0.8476 - val_loss: 0.4759 - val_accuracy: 0.8553
Epoch 10/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.4881 - accuracy: 0.8489 - val_loss: 0.4728 - val_accuracy: 0.8538
Epoch 11/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.4846 - accuracy: 0.8504 - val_loss: 0.4706 - val_accuracy: 0.8562
Epoch 12/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.4813 - accuracy: 0.8510 - val_loss: 0.4682 - val_accuracy: 0.8560
Epoch 13/200
1782/1782 [==============================] - 15s 8ms/step - loss: 0.4784 - accuracy: 0.8523 - val_loss: 0.4665 - val_accuracy: 0.8553
```

Conclusion:

From these experiments, it is shown that the hyperparameters when comparing the algorithms between themselves (LTSM to LTSM, Simple RNN to Simple RNN, or 1D CNN to 1D CNN). What matters was the differentiating algorithm itself. The LTSM was by far the best in terms of efficiency, accuracy, and resource allocation. The most important hyperparameter to adjust was the vocabulary size and increasing it to account for more words to differentiate the news

descriptions from each other. Hence, when it comes to training a neural network to use for a chatbot to assist customer representatives, the best way is to use an RNN with and LTSM for quick training and use a bigger vocabulary/corpus size for better accuracy. However, this may not be too much of a substantial development project because we need to look out for key words or phrases to reroute the algorithm to various responses depending on which category the issue lies within. Once the categories are differentiated, we need to ask simple questions to lead them into specific categories and get the specializing customer representative to help them out.