CSCM38: Adv Topic - Artificial Intelligence and Cyber Security - Coursework 2: Comparing an RNNs LSTM and GRU Cells

Andy Gray
445348

18/12/2020

1 Introduction

We have previously stated in a prior report that we will be conducting an experiment that will be comparing two Recurrent Neural Network (RNN) cells. The two cells we proposed to compare are the LSTM (Long Short Term Memory) and the GRU (Gated Recurrent Unit) cell. LSTMs got introduced in 1997 [5], and they aimed to solve the RNNs long term memory problem as they were only able to have a short term memory along, along with solving additional issues the RNNs had. GRUs got first introduced in 2014 [2], and it is like an LSTM but is a simpler cell. However, just like an LSTM, GRUs are good at polyphonic music modelling, speech signal modelling and natural language processing [4], but GRUs have displayed better results with smaller and less frequent datasets [3].

With the rise of Natural Language Processing (NLP) and the vast amounts of data available, RNNs have become a great way to process this information and gain incites. However, with RNNs only having a short term memory this generated issues when dealing with larger amounts of text. That was until the LSTM vell got created and this was able to solve the short term memory issue as well as the vanishing gradient, well almost. However, the development of GRUs, which is a variation of the LSTM cell, has also brought about significant advancements. Therefore, overall improving the effectiveness of the RNNs. With these cells being more straightforward in design, they will tend to trainer quicker than an LSTM cell and have almost just as good accuracy as the LSTM.

With so many people using the internet [12], websites and web applications will get a lot of traffic and visits to them. However, these users expect fast responses from the apps, as 1 in 4 people who visit a website will leave the site if it takes longer than 4 seconds to load. User satisfaction drops by an extra 16% for every 1-second delay [10]. So the responsive time to apps is critical for user satisfaction. While Twitter, a popular social media platform, has 330 million active users monthly, which is 145 million a day [7]. Twitter also has 500 million tweets posted a day [7]. Twitter is a great way for people to pass on information as well as to contact services.

With the literature saying that there is not much between the LSTM and GRU cell, we want to see what RNN would be better to solve a real-world problem in terms of understanding people's tweets and potentially get used for deployment. As the nature of RNNs, they are good for solving NLP challenges. Based on the criteria of the RNN being accurate, fast to train and deploy.

We will be using a dataset that contains Twitter tweets that are about disasters [6]. However, some tweets are not about a disaster. We aim to compare variations of an LSTM and GRU RNNs to see what model is the best at performing this task of labelling and predicting the tweets to see if they are about disasters or not. Four different RNN variations, two LSTMs and two GRUs got compared against each other. The model had the same parameters, and

the same dataset got used, but the number of cells was the variable that was changed. 32, 64 and 128 cells got used in comparison with each model, with each model getting tested five times, with the average score then used for the comparison. The dataset got resplit for each of the five runs. However, the same dataset was used for all iterations of the RNNs to get tested upon on each run. With this challenging being an NLP classification problem, this will be an excellent medium to be able to compare the performance of the two RNN cells.

The results did show that the performance between the cells was indeed very close with the deeper LSTM and GRU variations taking longer to train as expected than the not as deep iterations of the RNNs. However, the more cells and layers the RNNs had, the quicker the GRUs got in regards to training compared to the LSTM versions. Still, these improvements didn't seem to be valid enough, taking into account the small gains they had but with a much more significant amount of training time required. The LSTM with the fewer amount layers and 32 neurons did not have the highest accuracy and metrics but was the quickest to train, with 33.608 seconds and a 0.770 prediction accuracy. Still, in comparison, its overall results were near the top in contrast to the other RNN variations. Which we believe could be down to the nature of the dataset and their not being a large amount of text to analyse. However, all the models did tend to provide a false negative (FN). So, therefore, before this could get used to predicting potential disasters, this bias would have to be rectified, if possible.

We will be exploring the proposed solution and the libraries, dataset, preprocessing, algorithms, metrics that will get used for comparison and the NN parameters that we used. We will then be analysing the results, and discussing what insights they provide, ending with a conclusion.

2 Proposed Method

We will be creating an experiment that will compare the two different types of RNN cells, the LSTM and the GRU. We used several parameters to train and test the different RNNs as well as several other metrics to be able to compare the performance of the cells. The RNNs aim to be able to accurately predict if a Tweet posted on Twitter is about a real disaster or not. This experiment used many different Python 3 libraries.

2.1 Libraries & Frameworks

We used a collection of different Python 3 libraries to conduct this experiment. We used Tensorflow 2 [1] for creating the RNN model, LSTM and GRU cells. It got also used for preprocessing the text and sequence with 'Tokenizer' and 'Pad Sequence'. We used Sci-Kit Learn [11] for splitting the dataset and for creating the confusion matrix. Additionally, Pandas [9] got used for handling the dataset and Numpy to allow the other libraries to be able to do their scientific calculations. We also used NLTK [8] for NLP's stop words as well as some of Python's extra libraries os, time, re and string.

2.2 Dataset and Preprocessing

We used a dataset from Kaggle called "Natural Language Processing with Disaster Tweets" [6]. While a training and CSV file is available, we decided to use the training set due to the test dataset not having any labels. Therefore if we used this dataset, it would be hard to compare how well each cell performed on with an unseen dataset.

The dataset had a shape of 7613, 5. These include the features 'id', 'keyword', 'location', 'text' and 'target'. Due to the 'id' feature not having any relevance and the 'keyword' and 'location' containing null values, these features got dropped from the dataset. The dataset's targets were either a 0 for a non-disaster tweet or 1 for a disaster. There were 4342 non-disaster and 3271 disaster observations. We then removed the characters **[List the characters]** from the dataset's text. We have done this to make sure that the RNN focuses on the contents of the text and not have to focus on the punctuation as this could impact on the model's performance if we had left them in. We also removed all the stop words from the text by using the NLTK library. This action got done to make sure that these stop words also don't impact on the model's understanding of the text. The stop words included: [list stop words]. Along with removing the stop words, we removed any URLs that were in the tweets as these have no relevance on if the tweet is about a disaster or not. When the stop words, punctuation and URLs got removed from the text, we ended up with 17,971 different words contained within all the tweets. The most common words appearing were 'like' (345), 'I'm' (299), 'amp' (298), 'fire' (250) and finally 'get' (229).

Once the dataset was all preprocessed, we split the data set into 2/3 training 1/3 testing. We also then split the training set into an 80/20 split of training and validation data. We did this to see if the dataset was getting overfitted within its training. Additionally, we did this to add an extra method of comparison between the cells. We then tokenised the unique words to create a word index to give the text a number representation for feeding through the RNNs. We then padded the sentence to 20 sequences. We did this to ensure that the length of text within the text would all match up, as the tweets have varying sizes anyway and with the previous clean up actions being down, potentially additional text has also been removed.

An example disaster text is "malaysia airlines flight 370 disappeared 17 months ago debris found south indian ocean" and a non-disaster is "walk plank sinking ship".

2.3 Algorithms

For this experiment, we will use the LSTM and the GRU cells. These are both modifications of the RNN. The vanilla RNN, an unaltered RNN, is a robust network. However, it suffers from some issues. These issues are that it only has a short term memory, a suffers from a vanishing gradient point and an exploding gradient too.

2.3.1 LSTM

LSTMs have the form of a chain of repeating modules of NN that all feed into each other. However, instead of having a single NN layer, LSTM has four, therefore interacting in a very different way to a vanilla RNN [20].

The cell state runs straight down the entire chain, with only some minor linear interactions. They are therefore allowing for the information to flow through the cells, unchanged if needed. However, LSTM does have the ability to remove or add information to the cell state.

2.3.2 GRU

GRUs are a simpler cell compared to an LSTM but perform just as well as the LSTM [12]. GRUs are an RNN that have a gated method to collect dependencies while being practical and adaptive as possible [5].

GRUs have an update gate, and a reset gate, which is responsible for selecting what information is to get taken forward and the rest gate is in between the two successive recurrent units. This gate decides on how much data needs to get forgotten. Another striking aspect of GRUs is that they do not store cell state in any way. Therefore, there is no output gate. The full state vector gets outputted at every time step [10]. Therefore they are unable to control how much memory the next unit will get given [9].

2.4 Metrics for Comparison

To be able to compare the model's performances, we used several metrics. These metrics include the time it took to train the model, the model's accuracy and loss, along with a validation loss and accuracy. We used the time to see how long it would take to train the model. We only timed how long the models took to train from the moment we ran the fit method and not with the initial set up the parameters. This action was to allow us to be able to see how long exactly the model took to complete the number of epochs set and finishing training. We used the accuracy, loss and validation metrics as additional information to see if the model was overfitting.

We also used a confusion matrix to be able to see how well the different models performed on an unseen dataset. Using a confusion matrix allowed us to see if the model was biased in a particular direction or if it was predicting well. Additionally, this allowed another form of comparison to see how accurate the models were on the unseen testing data.

2.5 RNN Parameters

For both the LSTM and GRU RNNs, we used a set value for the parameters. We made sure that these stayed the same when training both models to see

Metric	LSTM	GRU	Deep LSTM	Deep GRU
Loss	0.021	0.020	0.026	0.021
Accuracy	0.987	0.987	0.987	0.988
Validation Loss	1.564	1.279	1.406	1.494
Validation Accuracy	0.761	0.763	0.763	0.767
Prediction Accuracy	0.770	0.767	0.763	0.776
Time	30.2 secs	33.608 secs	81.772 secs	84.288 secs

Table 1: 32 cells training metric results.

which one performed better given each situation.

The loss function used was the binary cross-entropy, and the optimiser was adam with a learning rate of 0.001. These we both implemented using TensorFlows methods. We decided to keep everything within TensorFlow to make sure that everything was consistent and that having these implemented by SKLearn could have been implemented differently, resulting in changes to the model's outputs that are not as a result of the models themself. Therefore, as a result, creating a potential variation that would take focus away from what we were trying to compare.

The metric selected in the model's initiation was the accuracy metric, and we used 20 epochs for training. However, we did use a total of three different cells for each model. These were 32, 64 and 128 cells. We have done this to see how these affect the models training speed and performance in predicting.

Each LSTM and GRU both had an embedding layer of the number of unique words, by 32 and input length of the maximum size. Additionally, all variations had a dense output layer with an activation function of sigmoid and one neuron. The variations to the RNNs came in the hidden layers. One version had only two hidden layers while another had four layers. Each hidden layer had a dropout rate of 0.1. For the RNNs with two hidden layers, the return sequence was false, and for the four hidden layers, this was true. While both RNN variations are both a deep version, we will refer to the RNN with more layers as the deep network and the lesser one as just the network.

3 Results

When we compare the direct metrics that were produced by the model's while they were training (see tables: 1, 2 & 3), we can see that the LSTM with 32 cells

Metric	LSTM	GRU	Deep LSTM	Deep GRU
Loss	0.024	0.020	0.024	0.022
Accuracy	0.987	0.988	0.987	0.988
Validation Loss	1.558	1.584	1.579	1.541
Validation Accuracy	0.755	0.766	0.765	0.762
Prediction Accuracy	0.766	0.766	0.770	0.766
Time	35.876 secs	36.504 secs	96.626 secs	103.72 secs

Table 2: 64 cells training metric results.

Metric	LSTM	GRU	Deep LSTM	Deep GRU
Loss	0.028	0.022	0.029	0.021
Accuracy	0.986	0.987	0.986	0.987
Validation Loss	1.450	1.445	1.398	1.539
Validation Accuracy	0.759	0.764	0.770	0.758
Prediction Accuracy	0.766	0.771	0.767	0.763
Time	44.892 secs	42.416 secs	147.456 secs	108.162 secs

Table 3: 128 cells training metric results.

trained the fastest with an average time of 30.2 seconds while the most prolonged time training was the deep LSTM with 147.456 seconds. A fascinating insight is that the GRU became quicker when we added more layers and cells and then beat the LSTM when we had 128 cells. The GRU, with 128 cells beat the LSTM by 2.476 seconds, and the deep GRU beat the Deep LSTM by 39.294 seconds.

LSTM	1199	330	**1175**	314
	247	737	**271**	753
Deep LSTM	1191	339	**1236**	327
	255	728	**210**	714

Table 4: 32 cells confusion matrix results.

When we look at the model's metric results for 32 cells (see table 1, we can see that deep GRU had the best scores for all of the metrics apart from the loss and validation loss. However, when we compare the prediction accuracy to the LSTM's scores, we can see that the LSTM had only 0.006% less but took 63.27 seconds less to train and all its other metrics were very close to the deep GRU. Therefore, showing that there is not much difference in the predicted output

LSTM	1167	310	**1174**	315
	279	757	**272**	752
Deep LSTM	1208	340	**1182**	324
	238	727	**264**	743

Table 5: 64 cells confusion matrix results.

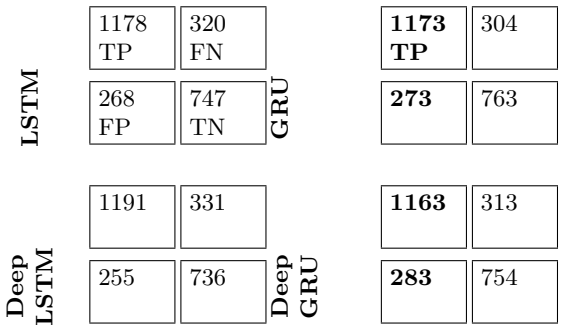

Table 6: 128 cells confusion matrix results.

compared to the overall training time.

When we look at the model's metric results for 64 cells (see table 2), We can see that the first GRU has had better results in all the metrics apart from the validation loss, which was the deep GRU by 0.043. However, overall the results were very close. With all variations of the RNN having a prediction accuracy of 0.766. Apart from the deep LSTM which had a prediction accuracy of 0.770.

When we look at the results of the 128 cells (see table 3), the model's metrics results start to show that the GRU is beginning to establish dominance over the other model types. It had the best prediction accuracy with 0.771, the closest accuracy to it was 0.04 lower and the deep LSTM. However, in contrast, the deep GRU had the worst scores but yet had the quickest training time.

The results show that the 64 neurons for both the LSTM and deep LSTM RNN, the better it had done compared to the GRU. However, the 64 cells deep LSTM took 2.7 times longer than the LSTM and only had 0.004 higher prediction accuracy. However, the more cells the GRU had but fewer layers, the better the GRU performed but would take longer to train than the LSTM counterpart. Still, in contrast, the more layers and cells the deep GRU had, the quicker it would train compared to the deep LSTM version but had lesser accuracy, even compared to the standard GRU.

When we look at the 32 cell confusion matrix average (see table 4), All the models tended towards a false negative. Therefore meaning the odds of a prediction, when it is wrong, to be predicting a negative for the text of the tweet but was positive was more likely. However, when we look at all the confusion matrixes (see tables 5, 6), we can see that all the models tended a false negative, which means that all the models are more likely to give a false negative rather than a false positive.

When the Confusion matrixes had a less FN, they would tend to have more false positives (FP), but the overall FN would still be more. The model variation that had the most FN was the deep LSTM with 64 cells, but the 32 celled deep LSTM was only one prediction away. The variation with the most FP was the deep GRU with 128 cells.

The GRU variations seem to have more of the tendency to have higher FP scores compared to the LSTM versions. However, these were still less than the

FN scores.

4 Discussion

While the results do show that the overall results are very close, in regards to the task that is a need in predicting the output, we believe the LSTM with 32 cells would be the better RNN to use. We have decided upon this since the results were not the highest but were very close to the top and it trained the quest, in 30.2 seconds. We believe further work is required to see if going any less would improve the accuracy and reduce the training time. However, there is a higher ration of FN predictions with these models, and this would cause some concern. Due to if the models were being used to predict live tweets coming in and seeing if the services need to respond, then there is a greater chance that the emergency services would not go and react as it is not a disaster, but in fact, it is. We believe a better result would be if the FP prediction were higher than the FN, as this would be a better scenario rather then a disaster being less likely to be attended. We believe that more research into changing other parameters is required. This studying is to ensure we can see what would improve the models' accuracy and improve the FN tendency.

An interesting insight we found was that the GRU only started to perform better than the LSTMs when they began to have more neurons in regards to the training time. However, the literature states that usually, the GRU would converge quicker than the LSTM, due to its simpler architecture. It was only until the RNNs had 128 cells did the GRU converge faster than the LSTM, but the deeper variation did have a more significant improvement over the training time compared to the deep LSTM.

5 Conclusion

While the literature states that GRUs should train quicker, due to their stripped-down architecture, we found that this was not always the case. When the GRU had 128 neurons, that is when the training time started to beat the LSTMs. However, the GRU did outperform the LSTM in training time the deep the architecture got, with significant time improvements. But, like the literature states, the performance between the two overall was very close, and the deeper layers did not provide any meaningful improvements to the prediction accuracy. Therefore, we believe that a simpler LSTM, with only 32 neurons is better for solving this problem of predicting the tweets. Its metrics were not all the best, but they were near the top, and the training time was overall the quickest with 30.2 seconds.

We believe more research is needed to see if changing any other the other parameters would make much of a difference to the overall models. Additionally, we believe that more work needs to carrying out to see what metrics are having the most significant impact on the models' FN predictions. As if this was to be used to aid in responding to disasters, this false prediction could lead to actual disasters happening, not getting attention.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[3] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

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

[5] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[6] Kaggle. Real or not? nlp with disaster tweets, 2020.

[7] Lin, Y. 10 twitter statistics every marketer should know in 2021, 2020.

[8] Loper, E., and Bird, S. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics* (2002).

[9] McKinney, W. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (2010), S. van der Walt and J. Millman, Eds., pp. 51 – 56.

[10] Monaghan, M. Website load time statistics: Why speed matters in 2020, 2020.

[11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[12] Visual Oak. Internet facts, history, and resources for 2020, 2020.