

CSCI 5922 Fall 2022–Lab 3

Abulitibu Tugulu

October 11, 2022

1 Impact of RNN Architecture

Methods

In this lab, we used SMS Spam Collection Data Set ¹, which consists 5719 data points, of which 4827 labeled as **ham**, 747 as **spam**. The hardware configuration is While the 3 RNN architecture we constructed as the

Amazon SageMaker G4 instance						
Instance Size	vCPUs	Instance Memory (GiB)	GPUs-T4	Network Bandwidth (Gbps)	Instance Storage (GB)	EBS Bandwidth (Gbps)
ml.g4dn.xlarge	4	16	1	Up to 25	1 x 125 NVMe SSD	Up to 3.5

following:

Vanilla RNN

Model: "sequential_12"		
Layer (type)	Output Shape	Param #
simple_rnn_5 (SimpleRNN)	(None, 32)	1088
dense_5 (Dense)	(None, 1)	33
Total params: 1,121		
Trainable params: 1,121		
Non-trainable params: 0		

LSTM RNN

Model: "sequential_11"		
Layer (type)	Output Shape	Param #
lstm_12 (LSTM)	(None, 100, 32)	4352
lstm_13 (LSTM)	(None, 1)	136
activation_6 (Activation)	(None, 1)	0
Total params: 4,488		
Trainable params: 4,488		
Non-trainable params: 0		

GRU RNN

Model: "sequential_14"		
Layer (type)	Output Shape	Param #
gru_2 (GRU)	(None, 100, 32)	3264
gru_3 (GRU)	(None, 1)	102
activation_8 (Activation)	(None, 1)	0
Total params: 3,366		
Trainable params: 3,366		
Non-trainable params: 0		

The Hyperparameters through all three models are the set the same: batch size = 10, epochs = 10, and we picked **Adam** as our optimizer, while the compiling loss is set to *binary_crossentropy*.

¹<https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>

How to recreate the experiment?

1. Import the data: by splitting the text into 2 part, part 0 set is as 'label', part 1 as 'text'.
2. Tokenizing the text data: by using **Tokenizer**, and padding the textual data using **pad_sequences**, but encoding the categorical classification labels, we need to convert as numpy array for the next step.
3. Train/Test split: by using inbuilt **train_test_split**, setting 7/3 ratio.
4. Reshaping input data set: for modeling fitting ((3901, 100, 1), (1673, 100, 1), (3901,), (1673,)) are shapes of *X_train, X_test, y_train, y_test*
5. Build model functions: as seen in source code model/add/add/compiule return, and model summary in previous section 1.
6. Model fitting: by setting batch and epoch as 5.
7. Predict the X_test result using **model.predict** and extract the predicted class labels in order to get classification report.
8. Print Confusion matrix for all three model: using **sns.heatmap**.
9. **Part 2:** Split training data into 3 parts: by **sort_values** in pandas and feat each set into step 6: Model fitting.

Results

In our experiment, the result for LSTM RNN outperformed the Vanilla and GRU RNN, as shown in Table 1, while it is also the longest one to train. The precision and recall for label **ham(0)** and **spam(1)** are respectfully: 0.95/0.95, 0.68/0.66. One thing that need to pay extra attention to are the precision/recall for both Vanilla and GRU: **0!!!**, meaning the models failed at detection a single spam message. We will dive in to my understanding of the reasons in next section.

Analysis

We conclude this experiment by analyzing the reason why our model failed. Tokenization of input data does not provide a robust inside into our data. In our case, it consists of too many zeros as follows

```
array([[ 0,    0,    0, ...,    5,  272,  555],
...
[ 0,    0,    0, ..., 2057,   57,  129]], dtype=int32)
```

on top more **ham(0)** than **spam(1)** in our data, it is no doubt that Vanilla will over emphasize signal 0s, in the end: **ham(0)**. What amazed me is the learning ability of LASM model, for picking up even a wick signal 1s, due to long term memory mechanism, it definitely is the go-to algorithm for NLP, with the help of word embedding (as we will see in the next section), the precision for **spam(1)** will jump from 0.6 tp 0.8. Even without the help of pre-trained models, LSTM already out perform even on the **ham(0)** signal (0.95) as compared to the rest's 0.86. The recall did not tell us much of the model, due to the fact of other 2 models tilted to 0s and not 1s. Another reason, in my opinion, is the total parameters in all three models: Vanilla/LSTM?GRU—1121/4488/3366, the more parameter gives us better results as demonstrated by LSTM.

I've spent way too much time on dividing the training set into 3 part, by counting the zeros in the input data and find percentile of 33,66 so to get same sized training sets of 3, turns out we can just rank the text input with string length and chop it into 3, sorry I did not have time to run all 9, instead, we will test it only on LSTM, the results can be seen in Table 2. That short and medium did not help with accuracy at all, yet only the long texts gave us an acceptable precision and recalls, the explanation is either only long text has contextual meaning for training or I code it wrong somewhere, which I need to revisit later.

Table 1: Three RNN architectures with no pre-train Embedding

Architecture	Time	Confusion matrix	Precision & Rcall																														
Vanilla RNN	5 min		<table><thead><tr><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.87</td><td>1.00</td><td>0.93</td><td>1451</td></tr><tr><td>1</td><td>0.00</td><td>0.00</td><td>0.00</td><td>222</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.87</td><td>1673</td></tr><tr><td>macro avg</td><td>0.43</td><td>0.50</td><td>0.46</td><td>1673</td></tr><tr><td>weighted avg</td><td></td><td>0.75</td><td>0.87</td><td>0.81</td><td>1673</td></tr></tbody></table>	precision	recall	f1-score	support	0	0.87	1.00	0.93	1451	1	0.00	0.00	0.00	222	accuracy			0.87	1673	macro avg	0.43	0.50	0.46	1673	weighted avg		0.75	0.87	0.81	1673
precision	recall	f1-score	support																														
0	0.87	1.00	0.93	1451																													
1	0.00	0.00	0.00	222																													
accuracy			0.87	1673																													
macro avg	0.43	0.50	0.46	1673																													
weighted avg		0.75	0.87	0.81	1673																												
LSTM RNN	15 min		<table><thead><tr><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.95</td><td>0.95</td><td>0.95</td><td>1451</td></tr><tr><td>1</td><td>0.68</td><td>0.66</td><td>0.67</td><td>222</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.91</td><td>1673</td></tr><tr><td>macro avg</td><td>0.81</td><td>0.81</td><td>0.81</td><td>1673</td></tr><tr><td>weighted avg</td><td></td><td>0.91</td><td>0.91</td><td>0.91</td><td>1673</td></tr></tbody></table>	precision	recall	f1-score	support	0	0.95	0.95	0.95	1451	1	0.68	0.66	0.67	222	accuracy			0.91	1673	macro avg	0.81	0.81	0.81	1673	weighted avg		0.91	0.91	0.91	1673
precision	recall	f1-score	support																														
0	0.95	0.95	0.95	1451																													
1	0.68	0.66	0.67	222																													
accuracy			0.91	1673																													
macro avg	0.81	0.81	0.81	1673																													
weighted avg		0.91	0.91	0.91	1673																												
GRU RNN	14 min		<table><thead><tr><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.87</td><td>1.00</td><td>0.93</td><td>1451</td></tr><tr><td>1</td><td>0.00</td><td>0.00</td><td>0.00</td><td>222</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.87</td><td>1673</td></tr><tr><td>macro avg</td><td>0.43</td><td>0.50</td><td>0.46</td><td>1673</td></tr><tr><td>weighted avg</td><td></td><td>0.75</td><td>0.87</td><td>0.81</td><td>1673</td></tr></tbody></table>	precision	recall	f1-score	support	0	0.87	1.00	0.93	1451	1	0.00	0.00	0.00	222	accuracy			0.87	1673	macro avg	0.43	0.50	0.46	1673	weighted avg		0.75	0.87	0.81	1673
precision	recall	f1-score	support																														
0	0.87	1.00	0.93	1451																													
1	0.00	0.00	0.00	222																													
accuracy			0.87	1673																													
macro avg	0.43	0.50	0.46	1673																													
weighted avg		0.75	0.87	0.81	1673																												

Table 2: impact of text length on LSTM model with no pre-train embedding

Text length	Confusion matrix	Precision & Rcall									
Short texts	<p>Confusion matrix for Short texts:</p> <table border="1"> <thead> <tr> <th>Predicted \ True</th> <th>0</th> <th>1</th> </tr> </thead> <tbody> <tr> <th>0</th> <td>1451</td> <td>222</td> </tr> <tr> <th>1</th> <td>0</td> <td>0</td> </tr> </tbody> </table>	Predicted \ True	0	1	0	1451	222	1	0	0	<pre> precision recall f1-score support 0 0.87 1.00 0.93 1451 1 0.00 0.00 0.00 222 accuracy 0.87 1673 macro avg 0.43 0.50 0.46 1673 weighted avg 0.75 0.87 0.81 1673 </pre>
Predicted \ True	0	1									
0	1451	222									
1	0	0									
Medium texts	<p>Confusion matrix for Medium texts:</p> <table border="1"> <thead> <tr> <th>Predicted \ True</th> <th>0</th> <th>1</th> </tr> </thead> <tbody> <tr> <th>0</th> <td>1451</td> <td>222</td> </tr> <tr> <th>1</th> <td>0</td> <td>0</td> </tr> </tbody> </table>	Predicted \ True	0	1	0	1451	222	1	0	0	<pre> precision recall f1-score support 0 0.87 1.00 0.93 1451 1 0.00 0.00 0.00 222 accuracy 0.87 1673 macro avg 0.43 0.50 0.46 1673 weighted avg 0.75 0.87 0.81 1673 </pre>
Predicted \ True	0	1									
0	1451	222									
1	0	0									
Long texts	<p>Confusion matrix for Long texts:</p> <table border="1"> <thead> <tr> <th>Predicted \ True</th> <th>0</th> <th>1</th> </tr> </thead> <tbody> <tr> <th>0</th> <td>1413</td> <td>137</td> </tr> <tr> <th>1</th> <td>38</td> <td>85</td> </tr> </tbody> </table>	Predicted \ True	0	1	0	1413	137	1	38	85	<pre> precision recall f1-score support 0 0.91 0.97 0.94 1451 1 0.69 0.38 0.49 222 accuracy 0.90 1673 macro avg 0.80 0.68 0.72 1673 weighted avg 0.88 0.90 0.88 1673 </pre>
Predicted \ True	0	1									
0	1413	137									
1	38	85									

2 Impact of Pre-trained Word Embedding

Methods

In second of this lab, we used the same data from last section and split the data the same way (70/30), while adding a pre-trained embedding model GloVe 50d and 300 d for comparisons, the 3 RNN architectures with embedding layers constructed as the following:

Vanilla RNN with Embedding

Model: "sequential_9"

Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	(None, 100, 100)	1000000
simple_rnn_7 (SimpleRNN)	(None, 100, 32)	4256
flatten_4 (Flatten)	(None, 3200)	0
dense_8 (Dense)	(None, 1)	3201

Total params: 1,007,457
 Trainable params: 7,457
 Non-trainable params: 1,000,000

LSTM RNN with Embedding

Model: "sequential_10"

Layer (type)	Output Shape	Param #
embedding_9 (Embedding)	(None, 100, 100)	1000000
lstm_1 (LSTM)	(None, 32)	17024
dense_9 (Dense)	(None, 1)	33

Total params: 1,017,057
 Trainable params: 17,057
 Non-trainable params: 1,000,000

GRU RNN with Embedding

Model: "sequential_11"

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	(None, 100, 100)	1000000
gru_1 (GRU)	(None, 32)	12768
dense_10 (Dense)	(None, 1)	33

Total params: 1,012,801
 Trainable params: 12,801
 Non-trainable params: 1,000,000

The Hyperparameters through all three models are the set the same: batch size = 10, epochs = 5, and we picked **SDG** as our optimizer, while the compiling loss is set to *binary_crossentropy*.

How to recreate the experiment?

1. Import the data: by splitting the text into 2 part, part 0 set is as 'label', part 1 as 'text'.
2. Tokenizing the text data: by using **Tokenizer**, and padding the textual data using **pad_sequences**, but encoding the categorical classification labels, we need to convert as numpy array for the next step.
3. Train/Test split: by using inbuilt **train_test_split**, setting 7/3 ratio.
4. ² Importing the GloVe word embeddings (after download the source txt files): Pick '**glove.6B.50d.txt**' and '**glove.6B.300d.txt**', and create **embedding_matrix**
5. Reshaping input data set: for modeling fitting ((3901, 100, 1), (1673, 100, 1), (3901,), (1673,)) are shapes of *X_train, X_test, y_train, y_test*
6. Build model functions: as seen in source code model/add/add/compiule return, and model summary in previous section 2.

²Embedding layer added, one new feature than part 1

7. Model fitting: by setting batch and epoch as 5.
8. Predict the X_test result using **model.predict** and extract the predicted class labels in order to get classification report.
9. Print Confusion matrix for all three model: using **sns.heatmap**.

We add one more hardware, just to speed up the running of multiple models at once, along with the machine we used for part 1.

Standard Instances	vCPU	Memory
ml.m5.4xlarge	16	64 GiB

Results

In this experiment, as shown in both Table 3 Table 4, the the results (e.g.: accuracies) improved significantly as compared to Part 1, Both Vanilla and GRU can both detect **spam(1)**, as confusion matrix shown, for both embedding 50d and 300d, the true positive are 196/209/147, with respect to 204/203/144, that is a big jump for Vanilla. I would still pick LSTM as the best performed model due to its high accuracy and less training time, between 50d and 300d GLOVE embedding, 300d is better. Next, we will explain how is it the case.

Analysis

We conducted this lab by studying the effect of 3 different architectures and 2 different embedding techniques effects. The baseline model for tokenized representation presented in part one has less information for our models to learn, since there are too many zeros in the input. This is a clear indication of Pre-trained embedding layers over others, the input format is no longer many zeros, instead, a non-space matrix with more vector representation of each word learned already from input, architecture plays less of a role in this session due to the fact that full embedding matrix has enough info for W in the model. The precision and recalls are not much of difference due to the, again, emedding. Vanilla/LSTM/GRU's precision for **ham(0)** and **spam(1)** are 0.98/0.98/0.94, 0.92/0.90/0.86, if we have to decide which model is best at predicting, it is Vanilla with embedding. Same goes for recall, detection mechanism, shows GRU performed the worst. One explanation I have is 'over-training', since the embedding already learned about the input, another complicated architecture did not improve the model but speed.

Second, Glove 50d vs 300d is . matrix of size (max_words, embedding_dim) which indicate dimesion 50 vs 300, the more the merrier, I can only imagine the sparse TF-IDF doing less of a job due sparcity of the input matrix. Hence I believe for NLP problems, word embedding count half the success.

One last thing to mention is the advantage of GPU machine over CPUs on training these tensorflow models, as shown in time difference, that GPU cuts the training time in half.

Table 3: Three RNN architectures with GLOVE 50d Embedding

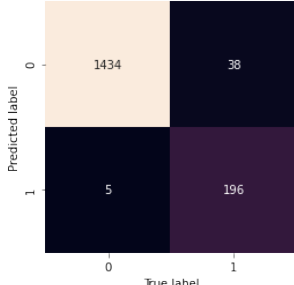
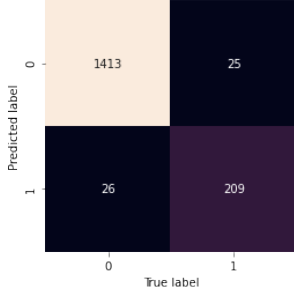
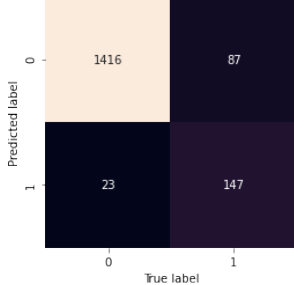
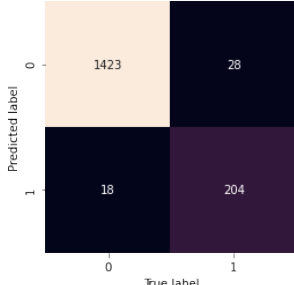
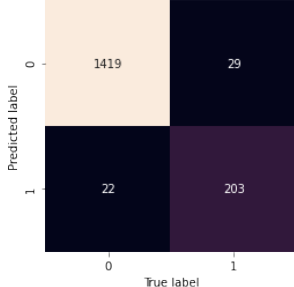
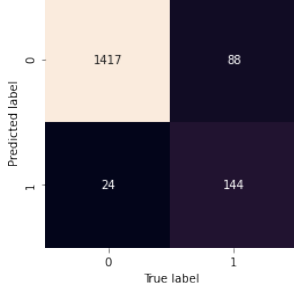
Architecture	Time	Confusion matrix	Precision & Recall
Vanilla RNN + Embedding	3.4 min	 <pre> 0 1 0 1434 38 1 5 196 0 1 True label </pre>	<pre> precision recall f1-score support 0 0.97 1.00 0.99 1439 1 0.98 0.84 0.90 234 accuracy 0.97 0.92 0.94 1673 macro avg 0.97 0.92 0.94 1673 weighted avg 0.97 0.92 0.94 1673 </pre>
LTSTM RNN + Embedding	14 min	 <pre> 0 1 0 1413 25 1 26 209 0 1 True label </pre>	<pre> precision recall f1-score support 0 0.98 0.98 0.98 1439 1 0.89 0.89 0.89 234 accuracy 0.97 0.94 0.94 1673 macro avg 0.94 0.94 0.94 1673 weighted avg 0.97 0.94 0.94 1673 </pre>
GRU RNN + Embedding	8 min	 <pre> 0 1 0 1416 87 1 23 147 0 1 True label </pre>	<pre> precision recall f1-score support 0 0.98 0.98 0.98 1439 1 0.89 0.89 0.89 234 accuracy 0.97 0.94 0.94 1673 macro avg 0.94 0.94 0.94 1673 weighted avg 0.97 0.94 0.94 1673 </pre>

Table 4: Three RNN architectures with GLOVE 300d Embedding

Architecture	Time	Confusion matrix	Precision & Recall
Vanilla RNN + Embedding	1 min	 <pre> 0 1 0 1423 28 1 18 204 True label </pre>	<pre> precision recall f1-score support 0 0.98 0.99 0.98 1441 1 0.92 0.88 0.90 232 accuracy 0.97 1673 macro avg 0.95 0.93 0.94 1673 weighted avg 0.97 0.97 0.97 1673 </pre>
LTSTM RNN + Embedding	4 min	 <pre> 0 1 0 1419 29 1 22 203 True label </pre>	<pre> precision recall f1-score support 0 0.98 0.98 0.98 1441 1 0.90 0.88 0.89 232 accuracy 0.97 1673 macro avg 0.94 0.93 0.94 1673 weighted avg 0.97 0.97 0.97 1673 </pre>
GRU RNN + Embedding	3.5 min	 <pre> 0 1 0 1417 88 1 24 144 True label </pre>	<pre> precision recall f1-score support 0 0.94 0.98 0.96 1441 1 0.86 0.62 0.72 232 accuracy 0.93 1673 macro avg 0.90 0.80 0.84 1673 weighted avg 0.93 0.93 0.93 1673 </pre>