National Tsing Hua University

Deep Learning Class

Homework 4: Report

Submitted to:
Prof. Lin, Che

Submitted by:
Name: Gaddisa Olani

ID Number: 106062867

PhD Student, Taiwan International Graduate Program
Academia Sinica, Institute of Information Science

June 6, 2019

Part 1: IMDB Dataset

1 A) Explain why do we need embedding.

- Loosely speaking, Embedding's are vector representations of a particular word that helps the machine to understand the semantic of human language. It's a distributed representations of text in an n dimensional space. These distributed representations are essential for solving most NLP problems and sequence prediction tasks. For example, you can calculate the similarity between words by vector operation. For instance, the similarity between "**Deep**" and "**Learning**" seems to be higher than the similarity between "**Deep**" and "**road**".

In general, here are some main reason to use embedding:

- It solves the main issue of using one hot encoding. As we know, the length of one hot encoded vector increases with the size of vocabulary. But in the case of embedding, even if the number of vocabulary increases, you do not increase the number of dimensions of each word
- Embedding can help us to transform discrete feature into a vector form (thus why it's very useful in NLP and speech recognition).
- It is computationally efficient, because smaller embedding requires less memory
- It regularizes your model, because the smaller number of parameters your model has, the better it is regularized.

1b) Explain the idea of gated models and the main differences between GRU and LSTM.

- Gated models are primarily designed to combat the vanishing gradient problem of Vanilla RNN through a gating mechanism. This means that we have dedicated mechanisms for when the hidden state should be updated and also when it should be reset.

- These mechanisms are learned and they address the concerns listed above. For instance, if the first symbol is of great importance we will learn not to update the hidden state after the first observation. Likewise, we will learn to skip irrelevant temporary observations. Lastly, we will learn to reset the latent state whenever needed. We discuss this in detail below.

Difference: GRU Vs LSTM

- ✓ In simple words, the GRU unit does not have to use a memory unit to control the flow of information like the LSTM unit. It can directly makes use of the all hidden states without any control. GRUs have fewer parameters and thus may train a bit faster or need less data to generalize. This can be clearly seen from the following equations:

GATED RECURRENT UNIT

$$\tilde{c}_t = \tanh(W_c[G_r * c_{t-1}, x_t] + b_c)$$
$$G_u = \sigma(W_u[c_{t-1}, x_t] + b_u)$$
$$G_r = \sigma(W_r[c_{t-1}, x_t] + b_r)$$
$$c_t = G_u * \tilde{c}_t + (1 - G_u) * c_{t-1}$$
$$a_t = c_t$$

LONG SHORT TERM MEMORY

$$\tilde{c}_t = \tanh(W_c[a_{t-1}, x_t] + b_c)$$
$$G_u = \sigma(W_u[a_{t-1}, x_t] + b_u)$$
$$G_f = \sigma(W_f[a_{t-1}, x_t] + b_f)$$
$$G_o = \sigma(W_o[a_{t-1}, x_t] + b_o)$$
$$c_t = G_u * \tilde{c}_t + G_f * c_{t-1}$$
$$a_t = G_o * c_t$$

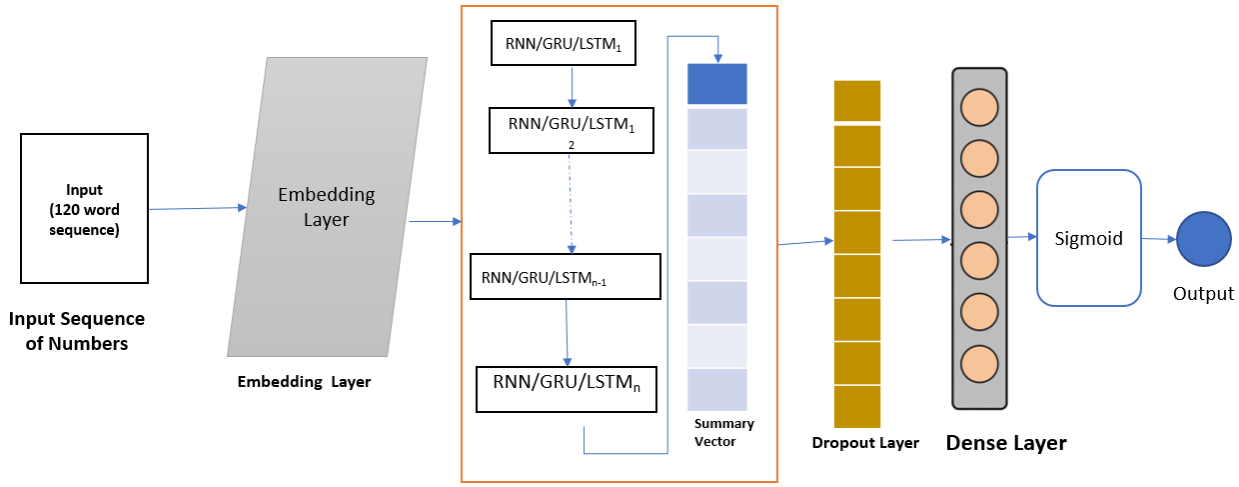
- ✓ As can be seen from the above equations, the LSTM does so via **input**, **forget**, and **output** gates; where the input gate regulates how much of the new cell state to keep, the forget gate regulates how much of the existing memory to forget, and the output gate regulates how much of the cell state should be exposed to the next layers of the network.
- ✓ Whereas, the GRU operates using a **reset** gate and an **update** gate only. The reset gate sits between the previous activation and the next candidate activation to forget previous state, and the update gate decides how much of the candidate activation to use in updating the cell state. The GRU unit controls the flow of information like the LSTM unit, but without having to use a memory unit. It just exposes the full hidden content without any control.

- ❖ Due to the aforementioned facts, GRU is relatively simple, computationally more efficient (less complex structure as pointed out) with a comparable performance with LSTM.

1c) Train a vanilla RNN (simple RNN) first to do the binary classification task.

The structure of the network: Embedding size=300, hidden state=128, dropout=0.7, Regularization=L2, batch_size=256,e pochs=100, learning_rate=0.0001,Optimizer=Adam,

Parameters of the Network: Embedding size=300, hidden state=128, dropout=0.7, Regularization=L2, batch_size=256, epochs=100, learning rate=0.0001, Optimizer=Adam,



Overall Model Structure

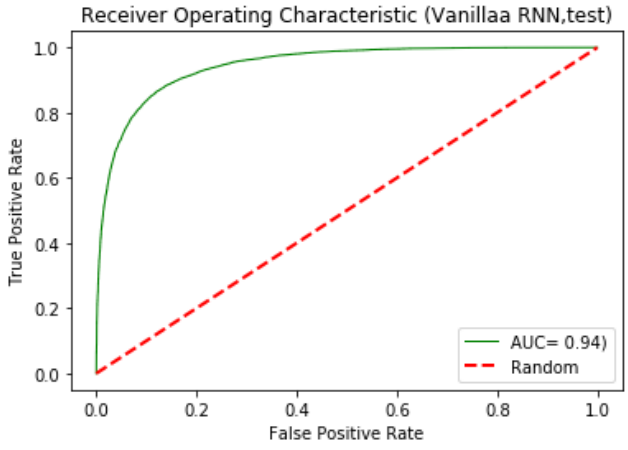


Figure 1: ROC Curve

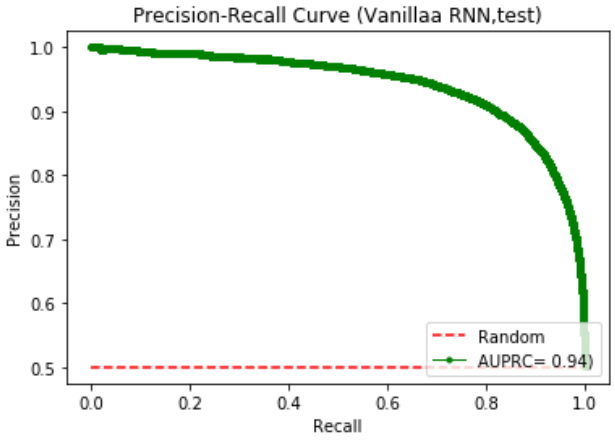


Figure 2: Precesion-Recall Curve

The validation accuracy on the test set using Vanilla RNN at the end of the iteration is: 0.80 (80%)

1d) Based on the definition of ROC, explain why random guess forms the red dotted line in

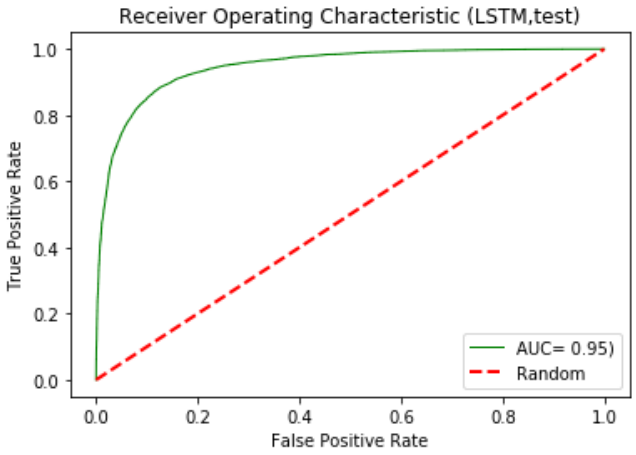
This is because, if you classify a fraction k of your cases as positive then, because of the randomness, the same fraction k of cases which should be positive will be classified positive (true positives), and the same fraction k of cases which should be negative will be classified positive (false positives). So the true positive rate and the false positive rate are the same. Due to this the ROC curve of a random guess forms a red line x=y (this is the reason why the ROC curve of random ((by 50% 50% chance) assignment is a diagonal line).

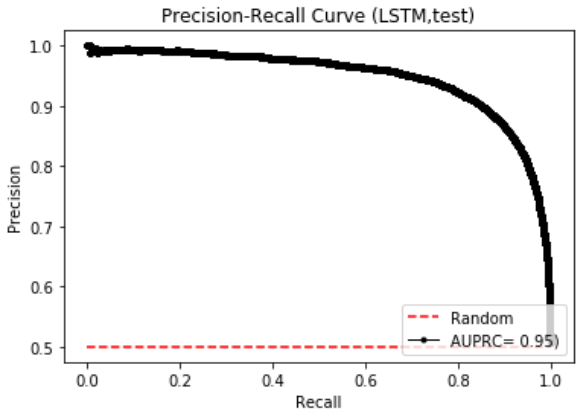
1e) What curve would random guess form in the PRC case?

0.5 for the classifier with the random performance level (horizontal line) y=0.5
The "baseline curve" in a PR curve plot is a horizontal line with height equal to the number of positive examples P over the total number of training data N. Thus as in our dataset the proportion of those data's are equal, the baseline curve will become a horizontal line with the equation y=0.5 (please refer to Figure 2).

1f) Repeat (c) with GRU and LSTM.

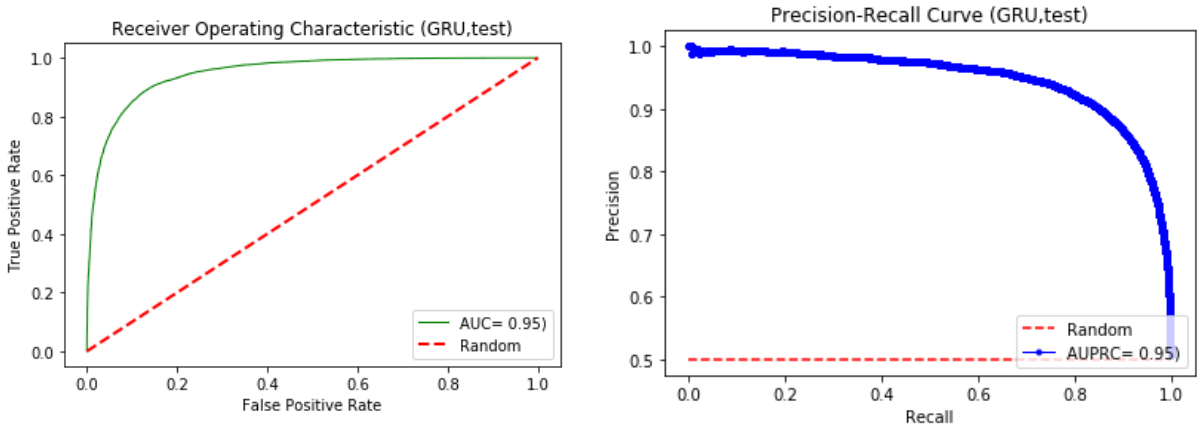
Part1: Long Short Term Memory





The validation accuracy on the test set using LSTM at the end of the iteration is test_acc: 0.8816

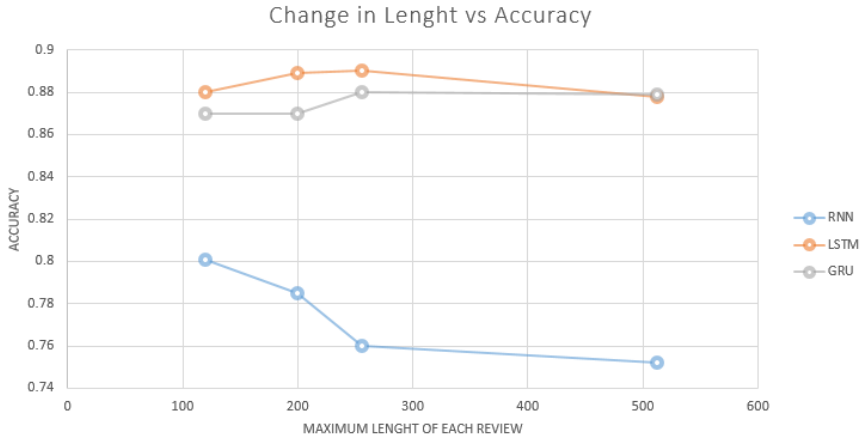
Part 2: Gated Recurrent Unit



The validation accuracy on the test set using GRU at the end of the iteration is test_acc: 0.8760

1g) Change the maximum length of each review from 120 words to 256 words in (c) and (f). State your observations and comments.

To further understand the influence of review length on the accuracy of the model, I repeat the experiment by changing the sequence length to 120, 200,256, 500. The result of the experiment is shown in the figure below.



As it can be clearly seen from the figure the accuracy of the model drops as we increase the length of the sequence in the case of RNN from 0.80 to 0.752. This result is quite acceptable that the RNN always try to remember all past observation. Thus it leads to a vanishing gradient problem, and thus impacts the accuracy of the system. As we learned in the class there are different techniques to overcome this longer

temporal dependency such as by stacking more number of recurrent layers or using truncated back propagation through time.

Overall it seems that various word lengths have no major impact on the model performance of the LSTM and GRU. This is because they have a gate to control which information to keep and reject at time t, to avoid long dependency.

Main findings,

☞Since performance does not vary with different maximum lengths (LSTM & GRU), it may suggest that users do not actually need to finish the entire review before deciding whether or not the review is helpful

☞Further increase of the review length does not bring any improvement.

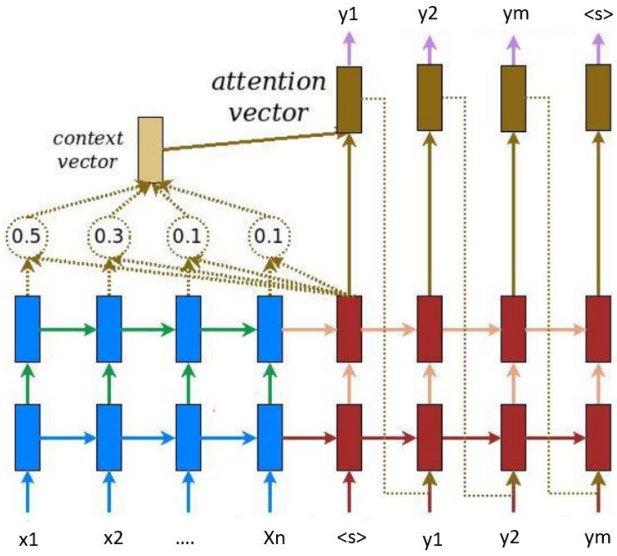
☞In the case of Vanilla RNN, Performance of the model decreases

////////////////////////// End of Problem 1//

Part 2: (English to French Translation using Sequence to sequence)

Proposed Architecture: Encoder decoder with Attention mechanism

The overall architecture of my implementation for question 2 is shown below:



I adopt this architecture from https://www.tensorflow.org/~/nmt_with_attention

As it can be seen in the above figure the model four main components: Embedding, Encoder, Attention and Decoder Layer in addition to the output layer. The details of each component will be discussed as follows

a) Data preprocessing:

The following data preprocessing task has been applied to the dataset before feeding to the mode:

- Tokenizing text by white space.
- Turn each sentence into a sequence of words ids (unique) and use transition table to maintain the mapping information
- convert all text to lowercase letter.
- Removing punctuation from each word.
- Padding: Make sure all the English sequences have the same length and all the French sequences have the same length by adding padding to the end of each sequence

b) Embedding Layer

This layer converts the numeric input to a distributed representation where each word is mapped to a fixed-sized vector of continuous values. The benefit of this approach is that different words with similar meaning will have a similar representation. Accordingly, I used an embedding size of 300 defines the length of the vectors used to represent words.

c) Encoder

This layer is responsible for stepping through the input time steps and encoding the entire sequence into a fixed length vector called a context vector. For this purpose, I used one directional LSTM (I tried to use two dimensional LSTM but I was unable to run it on my CPU based computer).
In my architecture, the encoder takes a series of inputs X1, X2…. Xn. At every time step the encoder updates the hidden state. The encoder's final hidden state C will become the initial hidden state of the decoder.

d) Attention:

As we learned in the class, there are some problems with a naive Encoder-Decoder model that the encoder maps the input to a fixed-length internal representation from which the decoder must produce the entire output sequence. The attention mechanism is an improvement to the model that allows the decoder to "pay attention" to different words in the input sequence as it outputs each word in the output sequence.

e) Decoder

The decoder is responsible for stepping through the output time steps while reading from the context vector. In my case, the decoder generates y1, y2, yn, one at each time step. At each such time step, the decoder reads the previous hidden state (ht-1), the initial hidden state (C) and the previous output (yt-1).

Some Hyper parameters of the System are summarized in the following table:

Hyper parameter	Value
Embedding Dim	300
RNN Type	Dynamic LSTM
Encoder Depth	2
Decoder depth	2
Batch_size	256
Learning rate	0.001
LSTM size	256
Attention Dimension	300
Regularization	L2
Epochs	100
Optimizer	Adam
Dropout	0.7

What techniques do you use to enhance the performance?

At the beginning I implement the model without attention mechanism, and later on I found that the performance of the system gets improved after adding attention layer to the system. In addition, I used regularization to avoid over fitting in addition to the dropout technique utilized.

How do I allow variable Length input?

Answer: I used dynamic LSTM models which is provided in tf. Dynamic LSTM allow for variable sequence lengths. You might have an input shape `(batch_size, max_sequence_length)`, but this will allow you to run the model for the correct number of time steps on those sequences that are shorter than `max_sequence_length`.

Since we do not know the target sequence lengths in advance, we use maximum_iterations to limit the translation lengths. One heuristic is to decode up to two times the source sentence lengths as follows: maximum_iterations = tf.round(tf.reduce_max(source_sequence_length) * 2)

How the evaluation is done?

To evaluate the translation, I perform two tasks (1) first generating a translated output sequence, and (2) then repeating this process for many input examples and summarizing the skill of the model across multiple cases.

Ideally, I use a separate validation dataset to help with model selection during training instead of the test set. Starting with inference, the model can predict the entire output sequence in a one-shot manner. This will be a sequence of integers that we can enumerate and lookup in the tokenizer to map back to words.

I perform this mapping for each integer in the translation and return the result as a string of words. Next, I repeat this for each source phrase in a dataset and compare the predicted result to the expected target phrase in French. Technically, I implement it as follows:

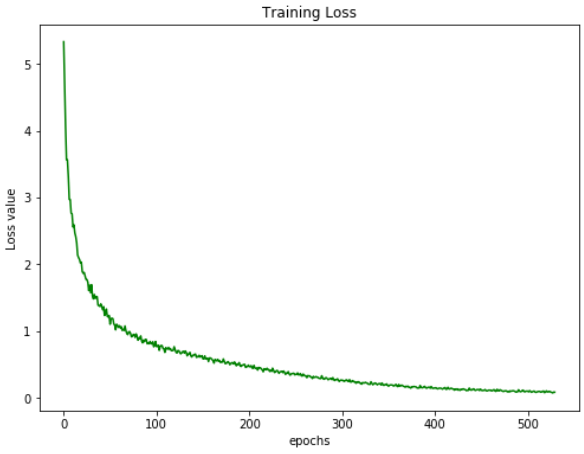
```
def get_accuracy(target, logits):
    max_seq = max(target.shape[1], logits.shape[1])
    if max_seq - target.shape[1]:
        target = np.pad(
            target,
            [(0,0),(0,max_seq - target.shape[1])],
            'constant')
    if max_seq - logits.shape[1]:
        logits = np.pad(logits,[(0,0),(0,max_seq - logits.shape[1])],'constant')
    return np.mean(np.equal(target, logits))
```

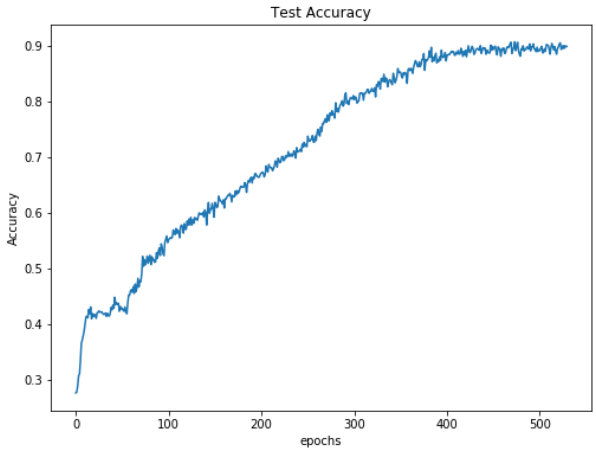
The summary of training accuracy, validation accuracy and test accuracy after 500 iterations is summarized in the table below:

#summary	#percentage
Training Accuracy	92.75%
Test Accuracy	91.86%

In general, both the training and testing phase almost took 1:30hr on Core i7 processor.

Plot the learning curve





Show some translation result

```
Source Language (English)
  Indices of words:      [14, 21, 226, 162, 36, 205, 113, 189, 183, 86, 226, 48, 164, 130, 221]
  Input words: ['new', 'jersey', 'is', 'sometimes', 'quiet', 'during', 'autumn', ',', 'and', 'it', 'is', 'snowy', 'in', 'april', '.']

Target Language(French)
  Indices of words:      [6, 189, 80, 138, 320, 285, 350, 16, 219, 212, 171, 297, 299, 80, 330, 37, 158, 182, 1]
  French words: new jersey est parfois calme au cours de l' automne , et il est neigeux en avril . <EOS>
```

//////////End of the report: Thank You //