PRML ASSIGNMENT II

1 Problem Definition

"In this assignment, you are required to design a RNN-based neural network to simulate an adder."

Specifically, given two sequences of decimal digits, use RNN-based neural network to calculate the sum of two integers.

The given data is less than 10 digits. We then test data with thousands of digits and evaluate them with different models. These will be described later.

2 Experiment Settings

Part I

As Recurrent Neural Network (RNN) can process time series data of any length by using neurons with self-feedback, we choose a multi-layer RNN with $tanh$ non-linearity to the input sequences. Firstly we use word embedding method to convert each of the input sequences (i.e. input numbers) into a low-dimensional dense vector $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{w}}$, where w denotes the dimension of embedding vectors (default 32). Then we concatenate the two input vectors into a vector with the dimension of $\boldsymbol{2w}$, and go through several layers (default 2) of RNN. The update formula of a basic RNN at time $\boldsymbol{t}$ is:

$$\boldsymbol{h_t} = f(\boldsymbol{U h_{t-1}} + \boldsymbol{W x_t} + \boldsymbol{b}) \tag{1}$$

$$\boldsymbol{y_t} = \boldsymbol{V h_t} \tag{2}$$

where $\boldsymbol{h}$ is the hidden state, $f(\cdot)$ is a non-linearity function like $tanh$, and $\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{V}, \boldsymbol{b}$ are all parameters of the network.

Then we train our models by optimizing cross-entropy loss, and using Adam to facilitate computation of learning rates (default 0.001) for each parameter.

Since the model converges quickly, we set the default number of steps to 500. Then we achieve the accuracy = 1 with the basic model in Part I.

Part II

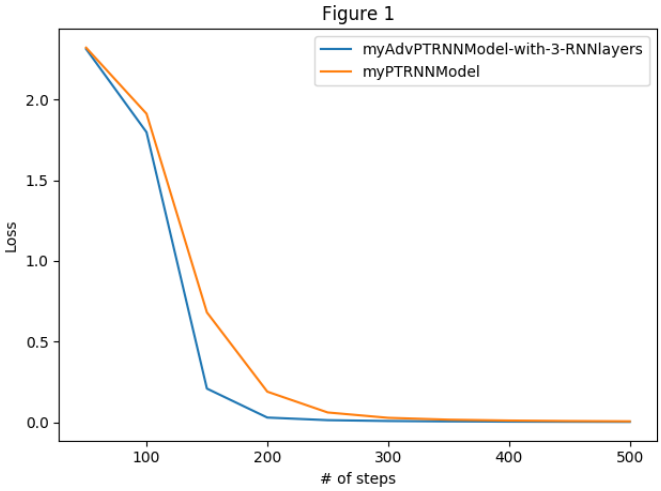
In the Part II, we set up a high-precision adder to generate input data with more digits (default 100 bits) , and make use of several RNN-based models, such as LSTM, Bi-LSTM and GRU, to better capture the performance of the different models.

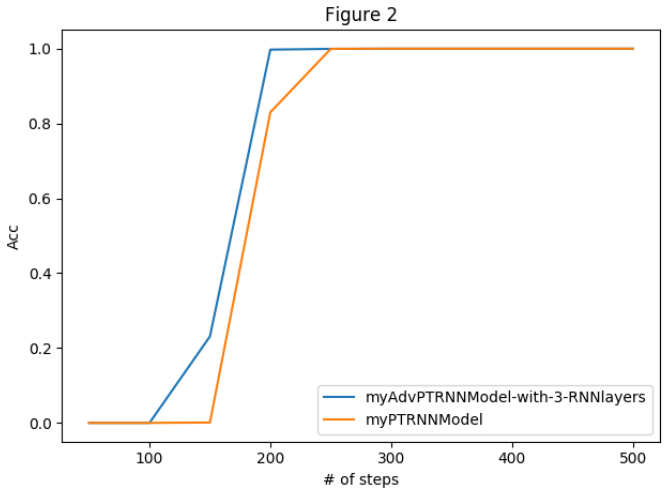
We also tune the parameters, such as the number of input layers, the dimension of hidden states, the dimension of input vectors, and the training and testing sequence lengths, to better understand the impact of these parameters on the models.

With the embedding vectors of 128-dimension, 64-dimension in the hidden space, and the step number of 500, we were surprised to find that with the training of **10-bit addition**, and then through **2 layers of RNN**, we can calculate **1000-bit addition** very well, and achieve an accuracy of 1. Even through just one layer of RNN, we can achieve an accuracy of 0.9985.

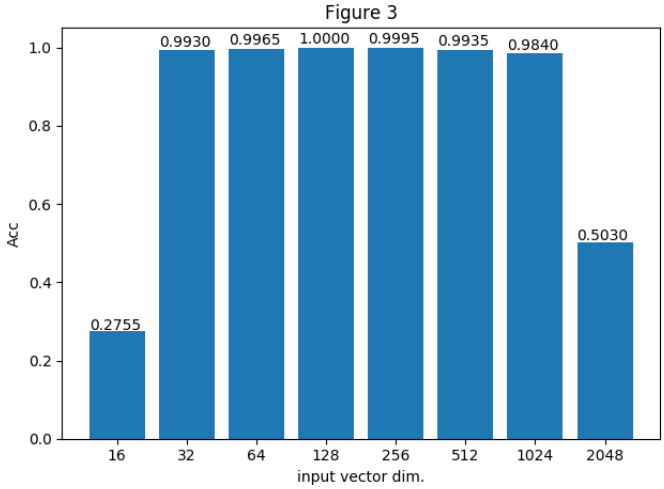
3 Evaluations

We first compared the loss and accuracy of our basic model and advance model with 3 RNN-layers during the iteration process. Here we set our training and testing data length both as 100, with the embedding vectors of 128-dimension, 64-dimension in the hidden space, and the step number of 500. The only difference between the two model here is the number of RNN layers. As the two model both reach the accuracy of 1, we can see that the model with **more RNN layers will converge faster**, with the loss function falling faster and accuracy reaching 1 earlier. However, using more RNN layers also means much more time consuming.

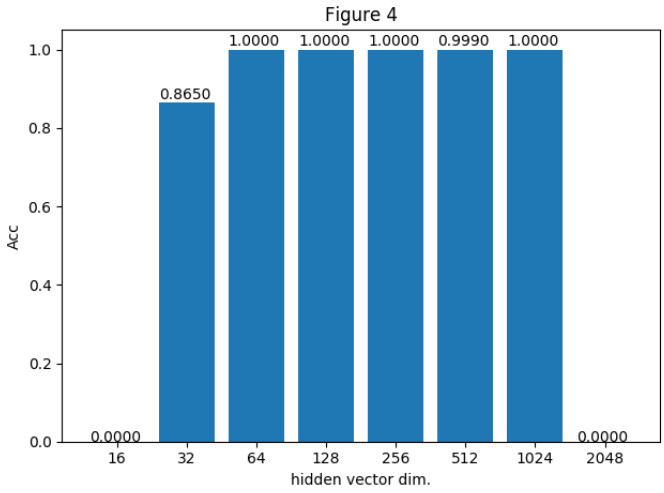




Then we change the input vector dimension to see the influence on model performance. In order to make the comparison more obvious, here we set our advance model with 2 RNN-layers, 500 steps, 4-bit training sequences and 100-bit testing sequences. As Figure 3 shows, when the input vector with 16-dimension or 2048-dimension is underfitting or overfitting, respectively, we achieve the best performance with **128-dimension input vectors**.



Similarly, we also change the vector dimension in the hidden space, with 128-dimension input vectors, other settings are the same as what we mentioned above. As Figure 4 shows, when the input vector with 16-dimension or 2048-dimension is underfitting or overfitting, respectively, we choose the **64-dimension in the hidden space** considering both performance and computational complexity.



In terms of the influence of the length of training sequence, we find that even training data with quite short length, our model can perform well on testing data with long length. When the training data is **4-bit**, we still achieve a pretty good performance on 1000-bit testing data, with an accuracy of 0.996 as the following tables shows.

testing data length (bit)	Acc (training data: 4-bit)
10	1.0
20	1.0
50	0.997
100	0.9995
200	0.9925
500	1.0
1000	0.996

However, when the training data length is less than 4, the performance of our model is no longer exellent.

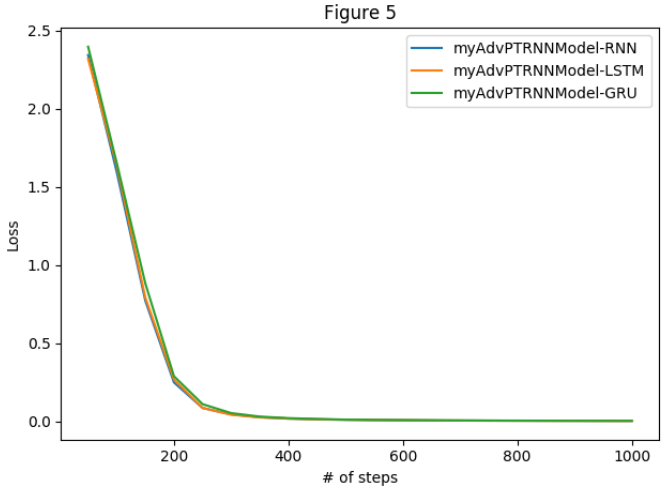
traing data length (bit)	Acc (testing data:100-bit)
1	0
2	0
3	0.297
4	1.0
5	1.0
10	1.0
50	1.0
100	1.0

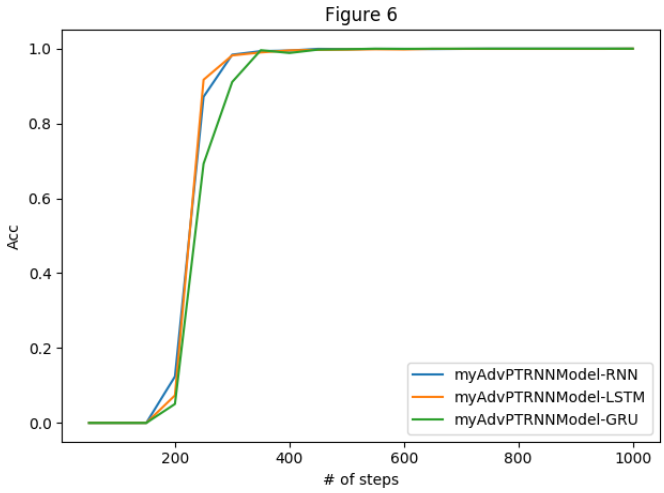
Also, there are many facors to affect our model's performance, such as step numbers, if you run more iterations within a certain range, the model will perform better. However, this is a **trade-off** between **time efficiency and accuracy**, so here we just **set the step number to 500**, which has done our task pretty well.

Finally, we change the RNN models to check whether the Gated RNN perform better. Here, we make comparison among three models, i.e. **RNN, LSTM, GRU**, with other settings the same as what we mentioned above. As shown in Figure 5 and Figure 6, the difference in performance of these three models is not very significant, maybe RNN will converge to the optimal solution earlier.

An explainable reason is that the long-term dependency problem has little effect on the RNN-adder task, so the gated RNN does not significantly improve the model performance.

Figure 5





4 Program Running

For the default settings, just open the terminal in the root directory of the file and enter the following statement:

```
python source.py
```

If you'd like to choose models like LSTM, you can enter the following statement:

```
python sourcy.py --algo=lstm
```

For more details please check the code.

5 References

"Neural Networks and Deep Learning" by Xipeng Qiu

"Pattern Recognition and Machine Learning" by Bishop