2018_dsai_hw3

Student ID: P76065013

Student Name: LEUNG Yin Chung 梁彥聰

I. Introduction

This assignment is to try to use a **sequence to sequence** learning model for training a string-based numerical operation. The input data is a **stringified numerical operation** like `"535-456"`, where the target result is the string based numerical equivalent, eg. `"79 "`. For the convenience of basic recurrent neural network implementation, I have used string padding with spaces at the end, where the length spans the range of possible input or output scenarios.

Personally I have tried implementation using Tensorflow based on a similar sequence to sequence model official tutorial of word prediction, but an unacceptable results were experienced. Therefore, the models I present here is mainly adjusted from the **sample codes** of the TA, and will also focus of analysis report later on.

The corpuses are saved in the **corpus** folder, where different datasets were used in different tutorials. The source code with the model building and data generation is saved in main.py. The visual presentation of the model and details of comparison is saved in the report.pdf report document.

II. Corpus Simulation

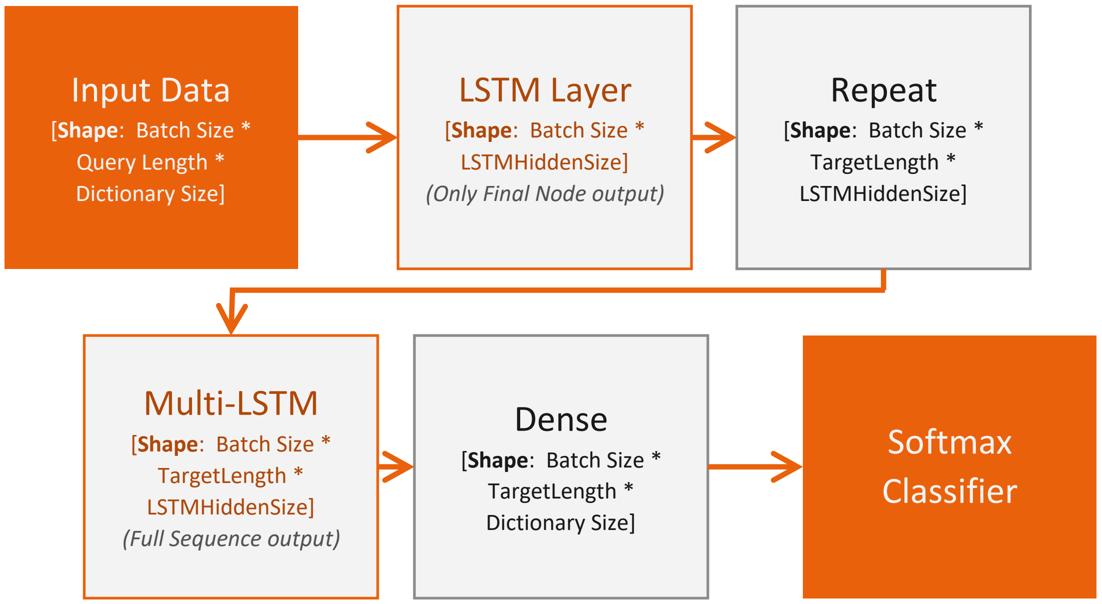
The corpus data algorithm is entirely **self-written**, where you may find the algorithm in the `prepareData()` function. The **basic substractor dataset** is saved in the 3 files ending with `Corpus - 3-minus.csv`. The data was generated once only, while different models were tested using retrieval of the saved dataset through passing `useOldData` parameter in this data preparation function. (NOTE: any old data should be placed in the same folder with `main.py`) As requeseted in the report section, there may be **different scenarios** that we may explore. Here I will have explored 3 to 6 digits of numerical operations; substraction, addition/substraction, and multiplication operations.

Firstly, I need to identify the **numerical operation**. Given the operation(s), the programme will **randomly** select 2 numbers in numerically-descending order and apply a **random** operation from the given list of operation(s). The query of operations on 2 numbers, as well as the numerical results, are then converted into strings and padded with a fixed length. For example, if it is a multiplication case of 4-digit numbers, the query string can be of length `4+1+4`(9), and the result string can be of length `4*2`(8).

III. Model Building

The core model is based on the **sample code** from TAs, and it's built based on Keras. **Variants** were tested, like changing the LSTM layer counts. Another model was tried, but the performance is significantly worse than the original model. The table below shows the final accuracy of the training, validation and test datasets. [The raw output can be found in `finalResults.csv`]

Figure A: Core Model (Model #0)


Input Data
[Shape: Batch Size * Query Length * Dictionary Size]
LSTM Layer
[Shape: Batch Size * LSTMHiddenSize]
(Only Final Node output)
Repeat
[Shape: Batch Size * TargetLength * LSTMHiddenSize]
Multi-LSTM
[Shape: Batch Size * TargetLength * LSTMHiddenSize]
(Full Sequence output)
Dense
[Shape: Batch Size * TargetLength * Dictionary Size]
Softmax Classifier

To deal with different scenarios, the programme was written in a parametric fashion, and the model above (Figure A) was trained and tested independently by feeding with parameters. The above model is compared the basic subtractor model ($-$) with:

- **other numerical operators** ([$+$, $-$],[$\times$]) [Trials #1, #2, #3];
- **different digit sizes** (4-digit, 5-digit, 6-digit numbers) [Trials #4, #5, #6];
- **different LSTM hidden sizes** (64, 32) [Trials #7, #8];
- **different LSTM layer counts** (2, 3) in the Multi-LSTM units [Trials #9, #10];
- **different training sizes** (80%, 60% of the original) [Trials #11, #12];
- **more LSTM layers for other operators** [Trials #13, #14, #15, #16];
- **newly-designed model** [Trial #17]; and
- **larger epoch size** (500) [Trial #18]

The results as shown below, which can be found in `finalResults.csv` :

Table 2A: Loss & Validation Accuracy

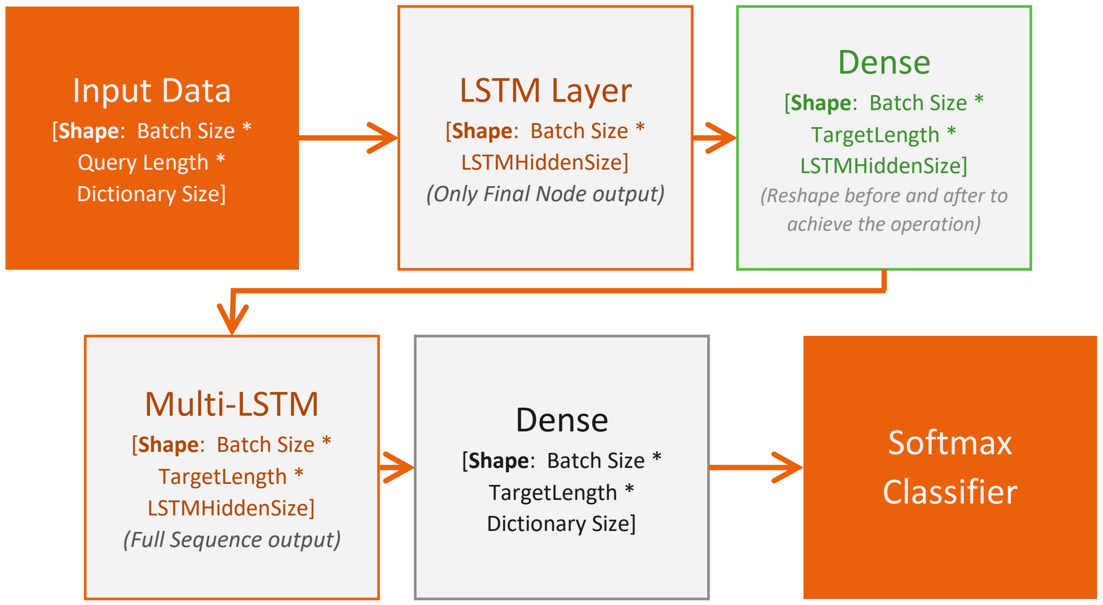
Trial ID	Final Training Loss	Final Training Accuracy	Final Validation Accuracy	Final Test Accuracy
1	0.022	0.997	0.981	0.935

2	0.122	0.971	0.914	0.730
3	0.872	0.683	0.576	0.018
4	0.855	0.687	0.613	0.053
5	1.489	0.444	0.398	0.001
6	1.688	0.360	0.319	0.000
7	0.267	0.926	0.905	0.755
8	1.115	0.585	0.565	0.072
9	0.019	0.996	0.990	0.965
10	0.078	0.972	0.964	0.910
11	0.093	0.973	0.912	0.770
12	0.023	0.996	0.979	0.932
13	0.125	0.961	0.941	0.813
14	0.796	0.709	0.582	0.019
15	0.305	0.882	0.838	0.478
16	0.766	0.710	0.591	0.024
17	1.208	0.536	0.437	0.026
18	0.001	1.000	0.987	0.955

Table 2B: Model Parameter Setup

Trial ID	Digit Size	Numerical Operators	LSTM Layer Counts	Training Datasize	LSTM Hidden Units	Epoch Size	Model Number
1	3	−	1	18,000	128	100	0
2	3	+, −	1	18,000	128	100	0
3	3	×	1	18,000	128	100	0
4	4	−	1	18,000	128	100	0
5	5	−	1	18,000	128	100	0
6	6	−	1	18,000	128	100	0
7	3	−	1	18,000	64	100	0
8	3	−	1	18,000	32	100	0
9	3	−	2	18,000	128	100	0
10	3	−	3	18,000	128	100	0
11	3	−	1	14,400	128	100	0
12	3	−	1	10,800	128	100	0
13	3	+, −	2	18,000	128	100	0
14	3	×	2	18,000	128	100	0
15	3	+, −	3	18,000	128	100	0
16	3	×	3	18,000	128	100	0
17	3	−	1	18,000	128	100	1
18	3	+, −	3	18,000	128	500	0

Figure B: Customized Model (Model #1)



Input Data
[Shape: Batch Size * Query Length * Dictionary Size]
LSTM Layer
[Shape: Batch Size * LSTMHiddenSize]
(Only Final Node output)
Dense
[Shape: Batch Size * TargetLength * LSTMHiddenSize]
(Reshape before and after to achieve the operation)
Multi-LSTM
[Shape: Batch Size * TargetLength * LSTMHiddenSize]
(Full Sequence output)
Dense
[Shape: Batch Size * TargetLength * Dictionary Size]
Softmax Classifier

IV. Results Discussion

1) Key Questions

a) *Model Building*

The original model was based on Tensorflow word prediction sequence-to-sequence learning model. The model was amended to have a direct LSTM layers on the original query on a shortened length of target sequence. Unfortunately, the results were **unacceptably done**. However, when switching to using the model from TAs, the results are very nice. It is interesting that the default model hyperparameters from Keras may be different from those I set on Tensorflow.

Also, I have built another model that **does not perform well** (Model #1, Trial #17). From these observations, I suppose that the criteria of setting up a pre-LSTM layer from input data may have a key success on the model building. The repeat action may just act as **looking the sequence once** again manually, instead of auto-learning from a dense layer or consecutive LSTM layers. From my point of view, this action may just like in manual calculations, the digits are somehow added or subtracted are doing at different positions to be affecting the previous result digit.

b) *Combining Adder & Subtractor*

Models #2, #13, #15, #18 were testing with the required scenarios. The original model was pretty **awesome** with 73% final test accuracy. It is known that with original model setup, it may be complicated to handle a completely different operations to attain the same results. Therefore, it is found that **a more complex model** may let the model learns how to deal with different cases. The final test of 3-layer Multi-LSTM model with a larger epoch-size (500) would bring the final test accuracy up to 95%. However, the time cost to train the model takes significantly longer, yet no formal timer was recorded.

2) Results Analysis

a) Different Number of Digits

According to Trials #4, #5, #6, the results **goes worse** as the number of digits becomes larger. This is reasonable because the model needs to learn a longer sequence learning. It is suggested to use a larger training dataset, more complex model and larger epochSize.

b) Different Training Sizes

Smaller training data were tried to understand how the models can learn up to the training data limit in Trials #11, #12. It is interesting to see the smallest trial #12 (used with the originally 60% samples) have a **comparable performance** (93%) to the original, while the 80% sample only have 77%. Due to limited time, no further investigation was taken. From my point of view, this have different results may be due to the availability of the data that those 20% may not have included complicated cases that may affect the performance.

c) LSTM Layer Adjustments

Less hidden units were tried with **lower performance** results. It is noted that on halving the hidden size to 64 may still have acceptable performance at final test accuracy of 75%, but on halving once more to 32 will have unacceptable performance at 7% only. Stacking **more LSTM layers** may have **performance boost** according to Trials #9 vs 1, #13 vs 2. However, when stacking to 3 layers, the results is not so acceptable (Trials #10, #15). Because of a complex model, **more epochs** of training would be needed. Therefore, in Trial #18 can bring the performance back to a nice value on 500 epoch sizes.

d) Multiplication Model

From the experiment results, the multiplication operation **does not perform well** in Trials #3, #14, #16. It is observed that more complex models have brought insignificant improvements. Personally, I believe LSTM models can train such a result. Despite the fact that the results are sparse (some string combination may not be a result because a prime number is not a multiplication of 2 numbers), there are rules in compromising the product of 2 numbers. However, I think a larger dataset should be used with a longer epoch size. A pretraining of lower-digit multiplication may also be useful.