ECE 542 Project 5

1st Ali Rahmati
ECE Department
NC State University
Raleigh, NC
arahmat@ncsu.edu

2nd Nathan Starliper
ECE Department
NC State University
Raleigh, NC
nstarli@ncsu.edu

3rd Shaunak Chokshi
ECE Department
NC State University
Raleigh, NC
schoksh@ncsu.edu

***Abstract*—This report summarizes the results of Project 5 for ECE 542 Fall 2018. In this project, we worked on LSTM networks to get familiar with its application to design LSTM cells for counting digits in a sequence and language modeling using RNN with LSTM units.**

I. Part 1: Counting with LSTM

In this part, given a digit sequence, we need to choose three different sets of parameters for a LSTM cell for the following three tasks. The structure of the LSTM network is shown in Fig. 1.

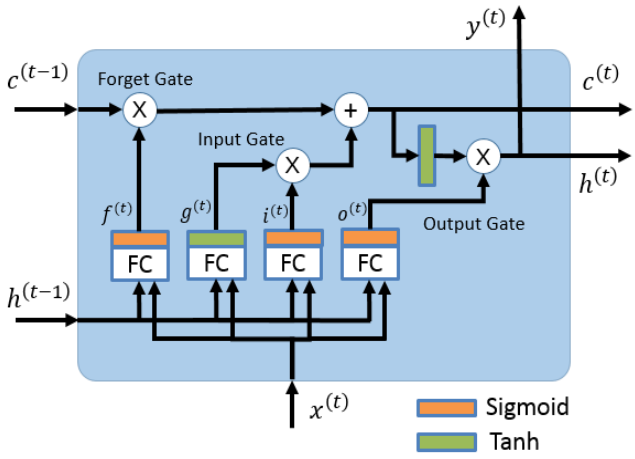

Fig. 1. Structure of the LSTM unit.

A. Task 1: Count number of all the digit 0 in the sequence

This task is already done in the code.

B. Task 2: Count number of 0 after receiving the first 2 in the sequence

For this task, the parameters for this task is given in Fig. 2.

C. Task 3: Count number of 0 after receiving the 2 in the sequence, but erase the counts after receiving 3, then continue to count from 0 after receiving another 2.

For this task, the parameters for this task is given in Fig. 3.

Input node						Input gate					
Wgx		Wgh		bg		Wix		Wih		bi	
100	0	0	0	0	0	-100	-100	0	0	0	0
0	0	0	0			-100	-100	200	200		
0	100					-100	100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
Forget gate						Output gate					
Wfx		Wfh		bf		Wox		Woh		bo	
0	0	0	0	100	100	0	0	0	0	100	100
0	0	0	0			0	0	0	0		
0	0					0	0				
0	0					0	0				
0	0					0	0				
0	0					0	0				
0	0					0	0				
0	0					0	0				
0	0					0	0				
0	0					0	0				

Fig. 2. LSTM parameters for task 2.

Input node						Input gate					
Wgx		Wgh		bg		Wix		Wih		bi	
100	0	0	0	0	0	-100	-100	0	0	0	0
0	0	0	0			-100	-100	200	200		
0	100					-100	100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
0	0					-100	-100				
Forget gate						Output gate					
Wfx		Wfh		bf		Wox		Woh		bo	
100	100	0	0	0	0	0	0	0	0	100	100
100	100	0	0			0	0	0	0		
100	100					0	0				
-100	-100					0	0				
100	100					0	0				
100	100					0	0				
100	100					0	0				
100	100					0	0				
100	100					0	0				
100	100					0	0				

Fig. 3. LSTM parameters for task 3.

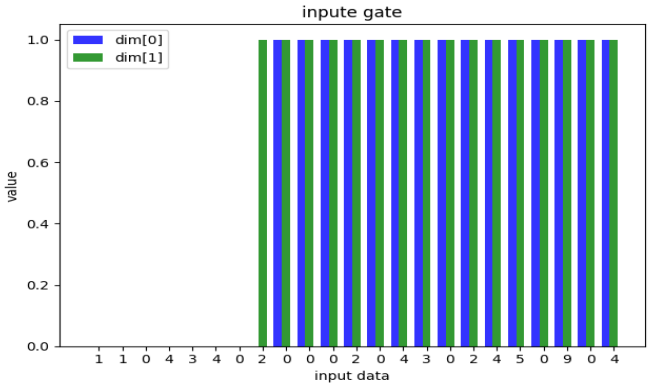

Fig. 4. Input gate for task 2.

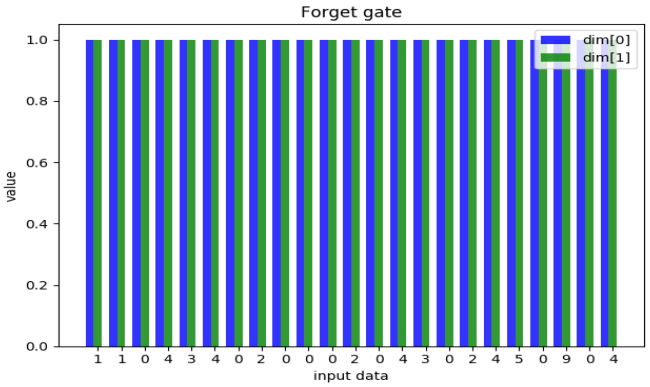

Fig. 7. Forget gate for task 2.

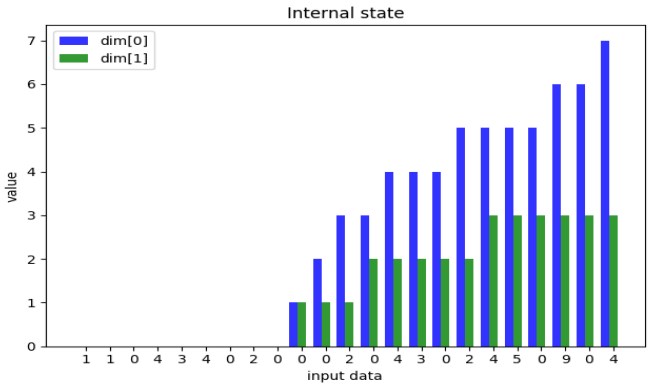

Fig. 5. internal state for task 2.

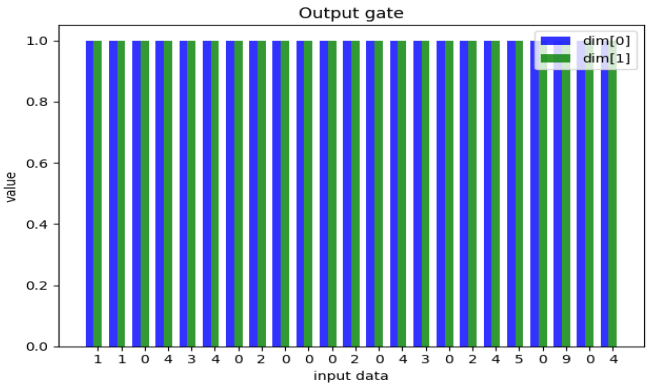

Fig. 8. Output gate for task 2.

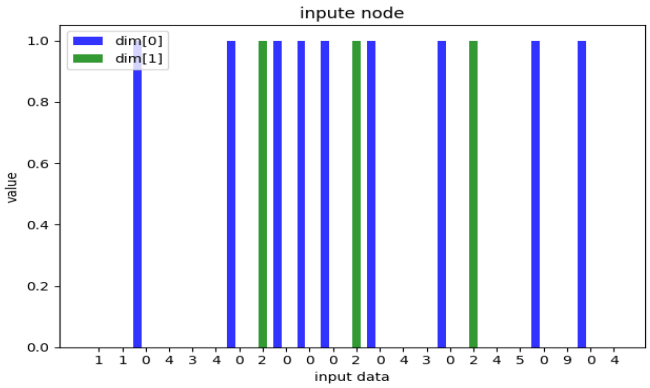

Fig. 6. Input gate for task 2.

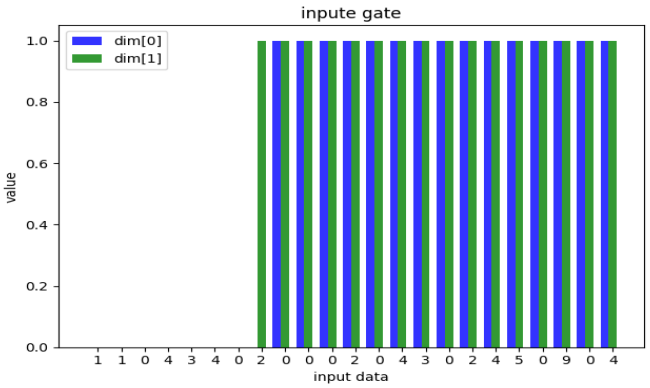

Fig. 9. Input gate for task 3.

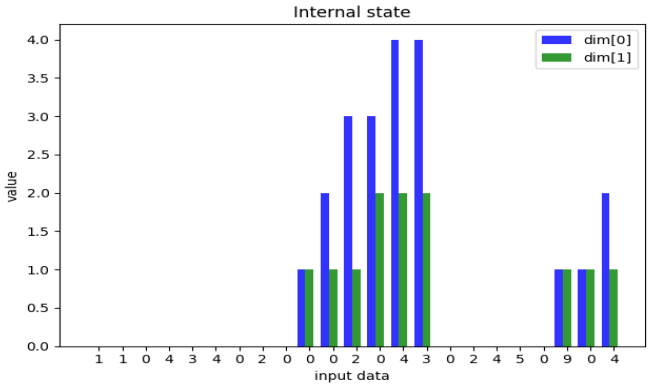

Fig. 10. internal state for task 3.

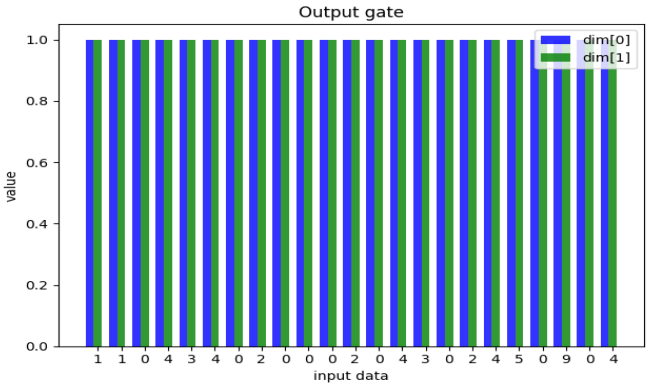

Fig. 13. Output gate for task 3.

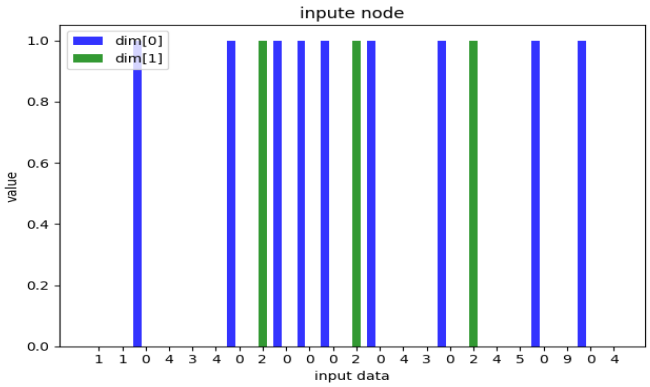

Fig. 11. Input gate for task 3.

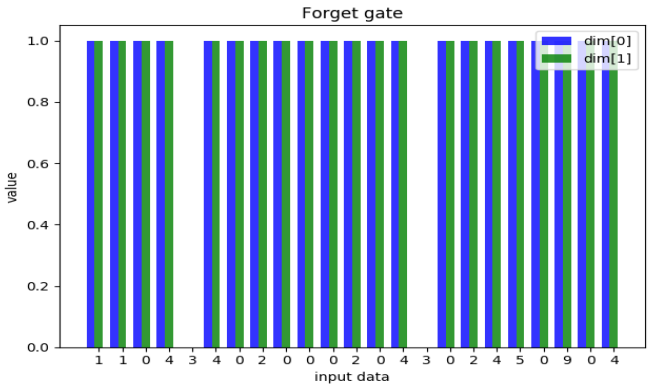

Fig. 12. Forget gate for task 3.

II. Part 2

First the words/characters are converted into meaningful word vector embeddings. This vector is encoded in such a way as to capture some aspect of the meaning of the word. For the word-level training we used a 500-length vector. For the character-level training, we utilized a 150-length vector as there are much less character representations than words.

A. *Structure of the Model*

We use a sequential model consisting of a number of layers. We first feed the vector embedded inputs to two stacked LSTM layers. Those LSTM layers then feed to an output time distributed softmax layer for calculating output probabilities. The architecture is shown in figure 14. Next, we will be inputting the vector representations into our unrolled LSTM network. For each input row we will be inputting 35 of these word vectors (i.e. 35 time sequential steps of words) in order to perform truncated back propagation. So, the input for each row will be a tensor of size (35, 500) for word level and (35, 150) for character level. Finally, with TensorFlow, we can process batches of data via multi-dimensional tensors. We use a batch size of 100 for character training and 50 for word training, so our training input data will be of size (50, 35, 500) and (100, 35, 150) for word and character models respectively. The input and output of each stacked LSTM layer will have the same dimensions. The final output of the time distributed softmax will be dimension (num_steps, vocabulary_size), which for the word model is (35, 10000) and for the character model (35, 50). Thus the output provides us with probabilities of each of the time outputs being any of the items in our vocabulary/dictionary. We can then do a table lookup on the vocabulary key to find the corresponding word/character.

Our model utilizes dropout. We use the Adam optimizer along with categorical the cross-entropy loss function. The hyperparameters of our model are shown in table I

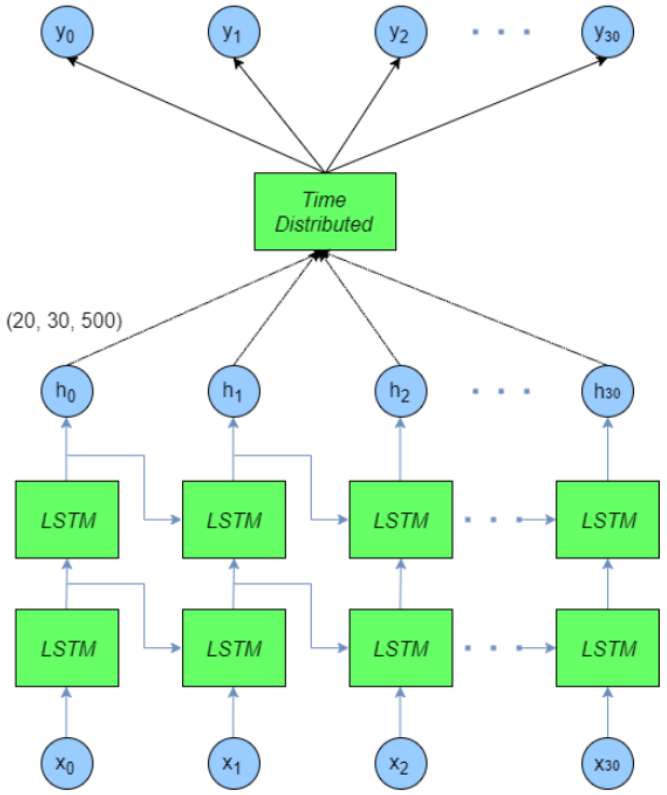

Fig. 14. Network Architecture

	Word Model	Character Model
Initial Learning Rate	1	1
Batch Size	50	100
Dropout Probability	0.5	0.5
Hidden Size	500	150
Number Time Steps	35	35
Number LSTM Layers	2	2

TABLE I
HYPER-PARAMETERS OF LSTM MODELS

B. Results

The results of the text generated by the two models can be found in the submission folder, they are named "results_char.txt" and "results_word.txt" respectively. These two text files show randomly selected sections of text predicted by the model. Above the predicted text, we also show the true text for comparison purposes. Please see the text files.

Perplexity curves were generated for the training and validation sets for each model. Training was run over 10 epochs. The training and validation perplexities curves per epoch for the character-level prediction are shown in figures 15 and 16 respectively.

The perplexity curves for the word-level models are shown in figures 17 and 18.

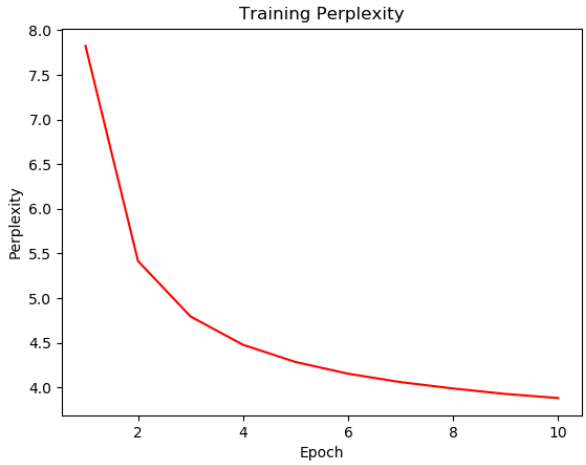

Fig. 15. Character Model Training Perplexity

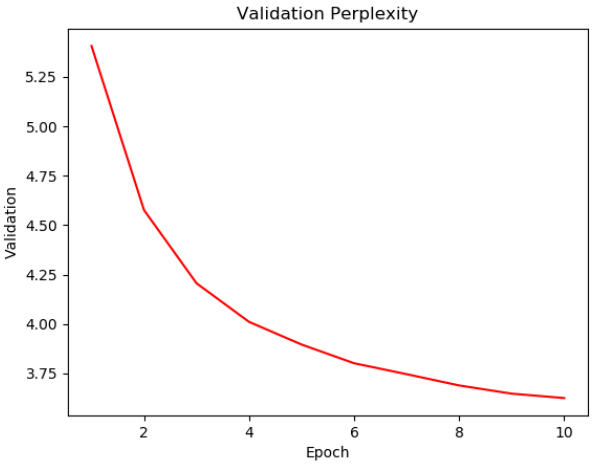

Fig. 16. Character Model Validation Perplexity

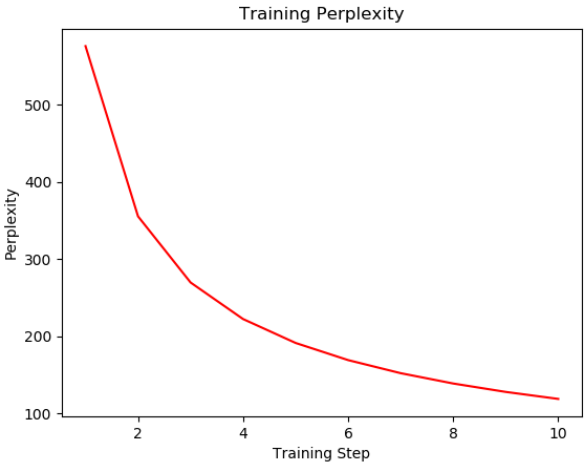

Fig. 17. Word Model Training Perplexity

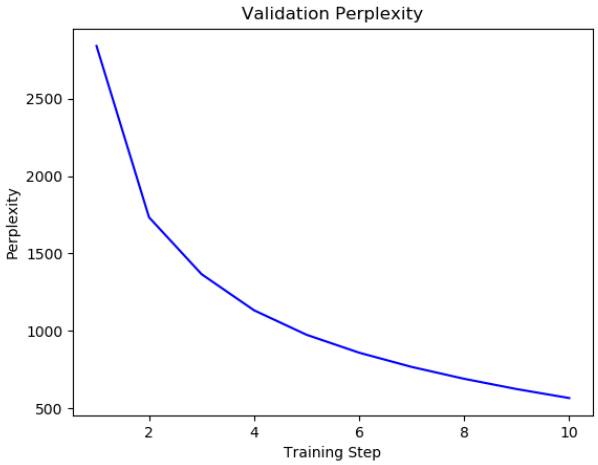

Fig. 18. Word Model Validation Perplexity

C. Discussion

It is clear to see that the character-level model performed much better than the world level model. We hypothesize this is due to the fact that the character level model has a much smaller vocabulary, and thus learns much quicker. In order to achieve better results with the word model, we would need a very large training set and would need to run for many more epochs (probably in the hundreds).