ECE542 Project 05: Implementation of a Recurrent Neural Network

Eshaan V Kirpal
Electrical & Computer Engineering
North Carolina State University
Raleigh, USA
evkirpal@ncsu.edu

Laura Gonzalez
Electrical & Computer Engineering
North Carolina State University
Raleigh, USA
llgonzal@ncsu.edu

Bhargavram
Electrical & Computer Engineering
North Carolina State University
Raleigh, USA
bmysore@ncsu.edu

***Abstract*—The objective of this project was to implement various RNNs for multiple purposes. In part one we implement RNNs for the purpose of counting numbers in a sequence. In part 2, we implement two RNNs. The first for character prediction, and the second for word prediction.**

***Index Terms*—RNN, LSTMs, Language Modeling,**

I. Introduction

Recurrent neural networks (RNN) are neural networks with a feedback loop present in the structure. They contrast feed forward neural networks (FNN), whose acyclic structure treats all inputs and outputs as independent. In other words FFNs do not exploit the relationships of data within a sequence or time. The feedback loop of an RNN, which can be local to neurons or global to the entire structure allows the network to retain memory or a state space that makes it suitable to work with sequential data such as time series data. [1]. One way to better understand how an RNN interacts with a sequence of input is to unfold it. A simple RNN is given as an example below. The RNN takes the input x at time t. The hidden states under node s have a value of s_t at time t. s_t is calculated by $s_t = f(Ux_t + Ws_t - 1)$ where f is normally a nonlinear function. The first hidden state at $t = -1$ is normally initialized to zero. Each neuron gets an input from a previous neuron at the previous time step. The output sequence is determined by considering current and previous inputs [4] . A similar FNN to the unrolled RNN would need to learn a different set of parameters for each layer, the RNN below only needs to learn one set up parameters updated at each time step, which reduces the computational complexity of the network. Another consideration is that an FNN can learn certain temporal dependencies by feeding it inputs at different time lags, but this is limiting because that requires an understanding upfront of what those fixed lags should be. A RNN will learn on its own what the temporal relationship in the input is without needing a set of fixed input lags, and if this temporal dependence between inputs at different times changes under different circumstances. Although RNNs do well learning from previous inputs, they have trouble with long term dependencies[5] . One type of RNN, Long Short Term Memory (LSTM) reduces this issue. This type of RNN will be addressed further in the text.

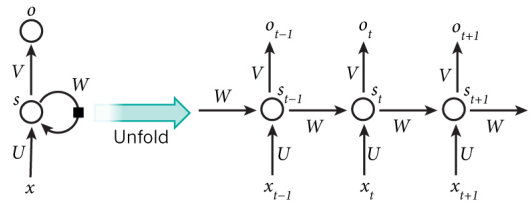

Fig. 1. A recurrent neural network and the unfolding in time of the computation involved in its forward computation. (source: Nature)

A. Long-Short term Memory RNN

Besides time series prediction, applications in which RNNs have been utilized include but are not limited to Robot control [6], acoustic novelty detection[7] , Speech recognition [8], Music composition [9], Grammar learning[10], Handwriting recognition[11], Human action recognition[12], Protein Homology Detection [13]. In this project we design two RNNs with the purpose of becoming familiar.

The purpose of the second part of the project is to come up with a pair of LSTMs that will predict text output. The first should predict the next letter, while the latter should predict the next word.

II. Part I

The purpose of the first RNN is to take an sequence of numbers, then count the number of times a zero has occurred under a set of conditions. The first is to count the total number of zeros to have occurred. The second is to count the number of zeros that have occurred after receiving the first number two in that sequence, and the third condition is to count the number of zeros that have occurred after receiving the first two in the sequence, then erase the count after the appearance of a three. The processes starts over again after another number two is encountered. In order to give a sequence of numbers to our input we had to use one hot encoding. One hot encoding produces a vector of zeros, and a single one. The location of the one in the vector is unique to the specific input integer.

A. LSTM Network Structure

Designed to be able to use long term dependencies without encountering the vanishing gradient problem, they have suc-

ceeded in predicting longer term sequences. Units made out of four hidden states and three gates. Information through time is passed through these units instead of the hidden layers. These gates are responsible for deciding what information needs to be kept, forgotten. As information flows through the top of the unit, called the cell state, its interactions with the output of the four gates decide on how the information should be modified for the next time stamp.

The first gate to interact with the cell state is the forget state, modeled by the equation below where t is the current time step, sigma represents a sigmoid function, W_f is the weighted matrix applied to the forget gate, h_{t-1} is an output obtained from the previous gate, and bf is some bias belonging to the forget gate. $f_t = (W_f[h_{t-1}, x_t] + b_f)$ The input to the forget gate, x_t and h_{t-1}, go through a sigmoid function, normally the linear identity function, that returns a scalar value between [0, 1]. That output is multiplied with the cell state as shown in figure 3. The closer the scalar value is to 1, the less information is retained from the original cell state.

The second gate is called the input gate, given by i_t. The input gate is tasked with determining how much of any previously existing information should be replaced. Similarly in the forget gate, the amount of information kept is given a set of scalars between [0,1] outputted by the sigmoid activation function. $i_t = (W_f[h_{t-1}, x_t] + b_f)$ The output of the second gate is multiplied with the output of the third gate before being aggregated to the cell state. The third gate is responsible for providing possible information that may or may not be added to the cell state. Together the second and third gate establish what information previously stored should be updated and to what degree. $i_t tanh(W_c[h_{t-1}, x_t] + b_C)$

The last gate takes the cell state after having recently being updated by the first three gates, puts it through a tan function then multiplies that with the input, and previous cell state sigmoid output to produce the output of the unit h_t. Instead of outputting the entire cell state, this ensures only the currently relevant part of the cell state is being output. The rest will continue to be stored in the cell state for later use. The function in 3 shows the output h_t is computed using the cell state, C_t. For this tasks 1-3, our output h_t a 2 dimensional vector that counts the occurrence of a zero in one dimension, and the occurrence of a 2 in the other dimension. We ignore the second dimension since we only care about

B. Parameters for Task 2 and 3 and Results

The parameters in this LSTM were set manually so that we could learn how and why each gate operates the way it does. We also plotted values of input node, internal state, input gate, forget gate and output gate as functions of time step for the example sequence [1, 1, 0, 4, 3, 4, 0, 2, 0, 2, 0, 4, 3, 0, 2, 4, 5, 0, 9, 0, 4]. For task 2, as stated previously, we use a single LSTM cell which outputs a 2 dimensional vector. The first dimension is used to count the total number of zeros, while the second dimension is used to count 2. To explain what each gate is doing during our procedure. Gate G has an output of [1 0] when the input sequence has a 0 and an output of [0 1]

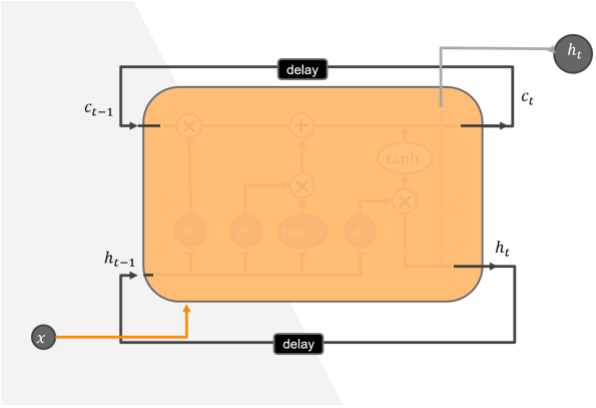

Fig. 2. LSTM inputs and outputs

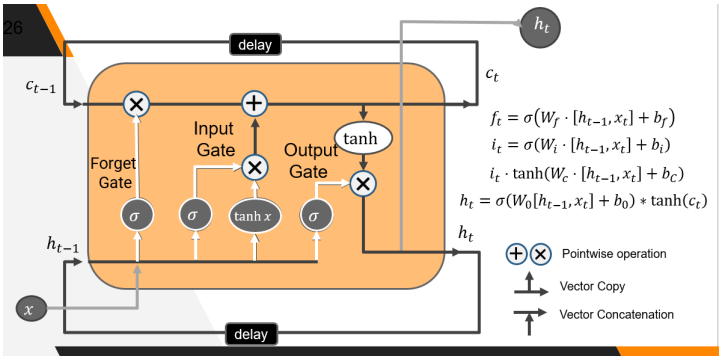

Fig. 3. Inside an LSTM

when the input is 2. This is because we set the parameters of gate G to the following:

$$W_{gx} = \begin{bmatrix} 100 & 0 \\ 0 & 0 \\ 0 & 100 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, W_{gh} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},$$

$$b_g = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Hence, the first dimension of the output of g saturates and becomes 1 when input is 0, the second dimension of the output of g becomes 1 when we have 2 as input. Task 2 requires us to start counting the 0's only after we encounter a 2. To achieve this we set the weights of the input gates such that the first dimension of input gate i is always kept 0 until we encounter a 2. We can observe this in the figure 6, where the input gate is 0 until it encounters a 2 at input time step 7 and then on-wards

it becomes 1. The input gate parameters which are required to make this happen are given below:

$$W_{ix} = \begin{bmatrix} -100 & -100 \\ -100 & -100 \\ -100 & +100 \\ -100 & -100 \\ -100 & -100 \\ -100 & -100 \\ -100 & -100 \\ -100 & -100 \\ -100 & -100 \\ -100 & -100 \end{bmatrix}, W_{ih} = \begin{bmatrix} 0 & 0 \\ 150 & 150 \end{bmatrix},$$

$$b_i = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

The high negative values in the matrix Wix ensures that the output of the input gate is always driven to 0 unless the input is 2 in which case the input gate is driven to 1 due to high value in the second dimension. The high values in the Wih matrix ensures that the input remains saturated once a 2 has been encountered in the input sequence. For task2 the output gate and the forget gate is always 1 as can be seen from the figure 5 and figure 4 respectively. Figure 7 shows the input sequence. Task 3 is similar to task 2, only difference being that we need to reset the count value to 0 as soon as we encounter a 3 in the input sequence. This is accomplished by driving the value of forget gate to 0 once 3 is encountered which causes the count to be reset to 0. This can be observed in the figure 9 where the forget gate value becomes 0 at time step 0 and 12. We observe in figure 13 that the count value is set to 0 at time step 12 and the counting starts again at time step 14 when it encounters a 2.

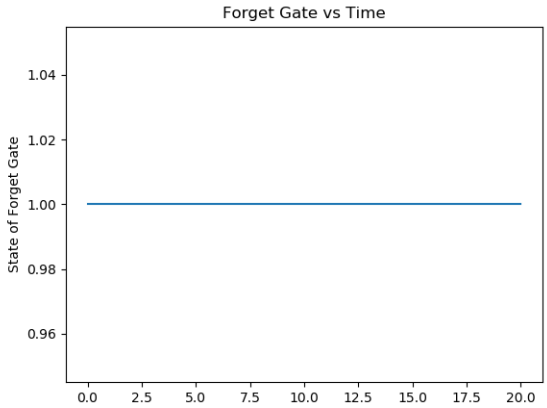

Fig. 4. Task2 Forget gate values vs Input time step

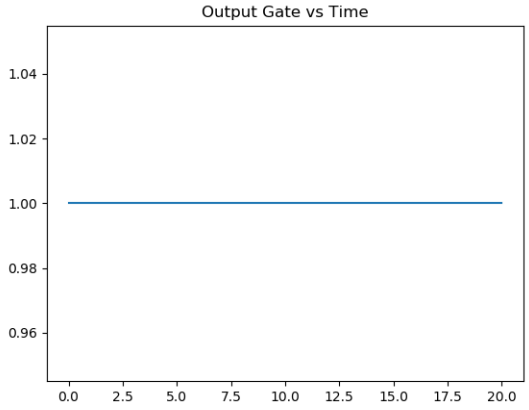

Fig. 5. Task2 Output gate values vs Input time step

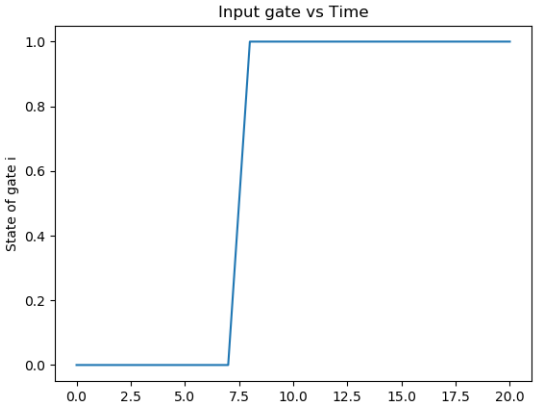

Fig. 6. Task2 Input gate values vs Input time step

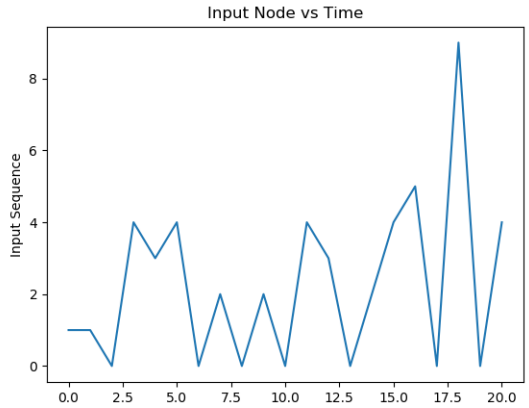

Fig. 7. Task2 Input Sequence values vs Input time step

III. Part II

A. Dataset

We used the Penn tree bank dataset which contains tokenized text files for both characters and words.

Before we were able to feed the data as input, we had to perform one hot encoding on the targets. One hot encoding labels each input and output to a numerical category. For

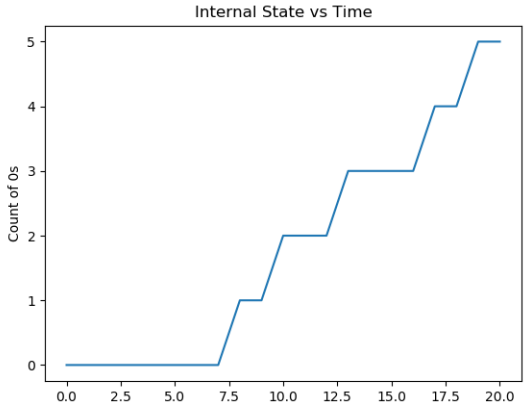

Fig. 8. Task2 Internal state values vs Input time step

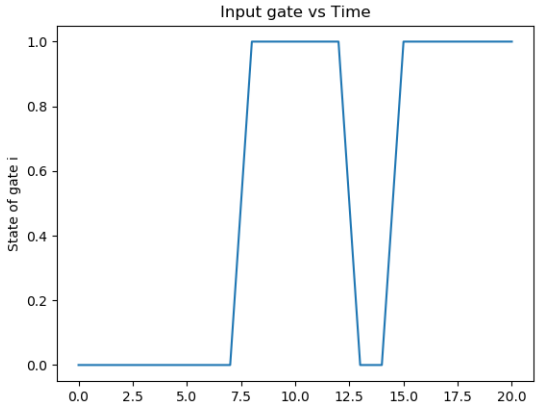

Fig. 11. Task3 Input gate values vs Input time step

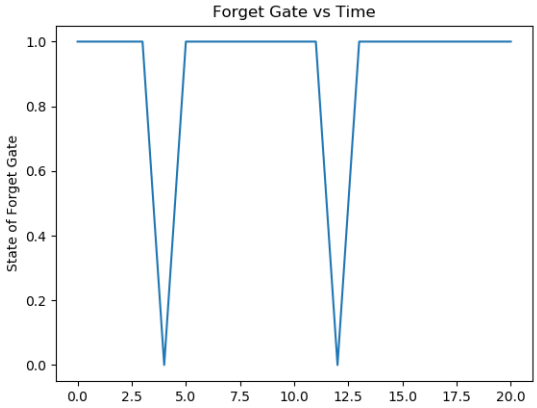

Fig. 9. Task3 Forget gate values vs Input time step

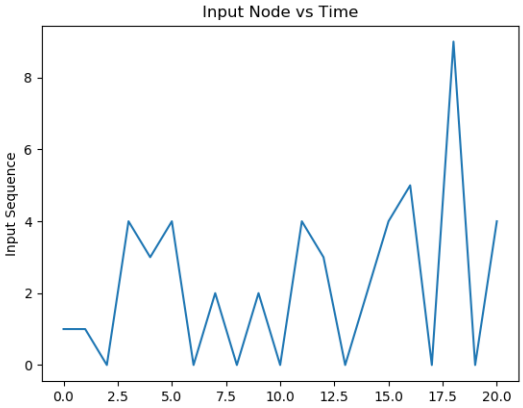

Fig. 12. Task3 Input sequence values vs Input time step

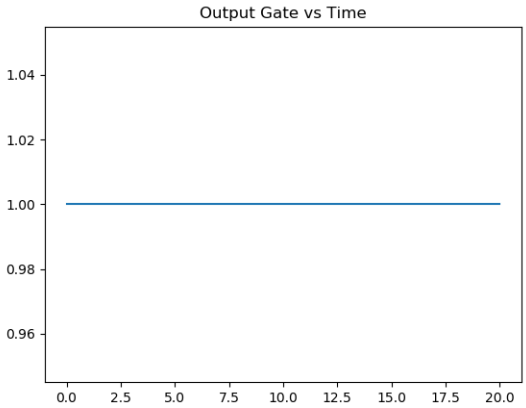

Fig. 10. Task3 Output gate values vs Input time step

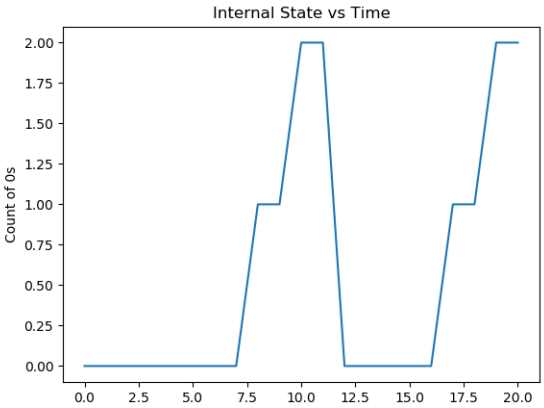

Fig. 13. Task3 Internal state values vs Input time step

the first RNN, each letter is represented by a number, and each number is represented by a vector of all zeros but a single value of one. The placement of the value of one in that vector is unique to each letter. This input is what is practiced when feeding categorical data into a neural network. A similar method is done to the second RNN. Each word in the data set

is one hot encoded.

B. Network Structure

Both Word and Character level RNN models were written using the Keras deep learning framework and the generated models were Sequential models, i.e. the network has exactly one input and exactly one output and it consists of a linear stack of layers. We used the adam optimizer for the both and because our targets were one-hot-encoded, we used the categorical crossentropy as the loss to train the model.

1) Word RNN Network Architecture: For the word level RNN, we first split the training data into separate words, then each unique word is mapped to a unique integer and finally the original text file is converted into a list of unique integers, where each word is substituted with its new integer identifier. We then create n-gram sequences of maxlen 30 using this integer based file and call this 2-dim array of size (no-of-sequences, maxlen) as the predictors array. Simultaneously we prepare an array of labels containing the corresponding targets: one-hot-encoded characters that come after each extracted sequence.

We pass our predictors and labels into an embedding layer with 500 hidden units. The rest of the network is a double LSTM layer with 500 hidden units followed by a dropout layer and finally a Dense softmax classifier over all possible characters.

Below is a summary of the model used for word level RNN.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 30, 500)	5000000
lstm_1 (LSTM)	(None, 30, 500)	2002000
lstm_2 (LSTM)	(None, 30, 500)	2002000
dropout_1 (Dropout)	(None, 30, 500)	0
time_distributed_1 (TimeDist	(None, 30, 10000)	5010000
activation_1 (Activation)	(None, 30, 10000)	0

Total params: 14,014,000
Trainable params: 14,014,000
Non-trainable params: 0

Fig. 14. Network Architecture for Word Level RNN

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 250)	299000
dense_1 (Dense)	(None, 48)	12048

Total params: 311,048
Trainable params: 311,048
Non-trainable params: 0

Train on 186652 samples, validate on 79994 samples

Fig. 15. The network structure the character level RNN

C. Tensorboard

In order to track all results, we used Tensorboard. Tensorboad is a suite of visualization tools to plot neural network parameters, training, validation and testing output from Tensorflow. It can also generate a graph of the network. If the information is written correctly, this makes visualizing the architecture very easy. Figure shows the tensorboard graph for our (BLANK) architecture. Every parameter varied was written to tensorboard with the accuracy and run time attached. Tensorboard allowed us to quickly visualize how different parameters impacted accuracy. It also allowed us to plot the histogram and distributions of our weights, biases and activation functions to see how they varied over time [2]

D. Hyperparameters

1) Hyperparameters for Character RNN: We used our initial grid search to identify promising hyperparameter settings that could be put under more rigorous validation. In the interest of time, we initially set the epochs to 5 while we searched for some of the best parameters. Once the top 3 parameters were determined, we ran those for 20 epochs each. Tensorboard allowed us to easily visualize and compare how different hyperparameter settings affected accuracy. Figures show output from tensorboard for learning rate, and architecture, respectively. The architecture was varied by number of hidden units per LSTM, and total of LSTMs. We can see that while performance for different architectures were highly variable, a learning rate of 1e-1 and 1e-2 to a less extent, provided very consistent optimal results. We also noticed a pattern. The most hidden units we had, the smaller the optimal learning parameter became. It should be noted that for our preliminary run in figure 16, we did not implement any cross validation, .For our final run, we implemented used randomized cross validation, where 80 percent of our data was used for testing, and the other 20 percent was used for validation. This made our final runs score lower than our pre-runs. In the interest of time, we ran the top three parameters for the character prediction LSTM, and reused them for the word level LSTM. They were able to demonstrate similar results.

Our preliminary runs taught us that a single LSTM works better than two LSTMs. For the final parameter tuning, we decided to only run our model for a single LSTM. We also found that the higher number of hidden units gave us better results, with the best being about 250 hidden units with a learning parameter of .001.

2) Hyperparameters for Word RNN: In the interest of time, hyperparameter tuning for word RNN was performed manually in comparison to using Tensorboard. We again used two LSTMs as they performed better than one with 500 hidden units. One dropout layer was used with 0.5 regularization. Best accuracy was obtained with a learning rate of 0.01 for the adam optimizer and when the sequences were processed in batches of 128.

E. Results

1) Results for RNN Character Predictions: Surprisingly we learned that we could achieve results higher than expected for very simplistic structures. For example, our character LSTM structure using two hidden units was able to achieve between

Name	Smoothed	Value	Step	Time	Relative
hl50_lr0.01_pl0	0.7094	0.7094	4.000	Mon Nov 5, 22:29:32	7m 51s
hl50_lr0.001_pl1	0.6957	0.6957	4.000	Tue Nov 6, 00:02:57	16m 42s
hl50_lr0.001_pl0	0.6831	0.6831	4.000	Mon Nov 5, 22:39:54	8m 11s
hl10_lr0.01_pl0	0.6689	0.6689	4.000	Mon Nov 5, 18:16:54	13m 6s
hl10_lr0.001_pl0	0.6458	0.6458	4.000	Mon Nov 5, 18:33:16	12m 50s
hl10_lr0.1_pl0	0.6410	0.6410	4.000	Mon Nov 5, 18:00:20	12m 48s
hl50_lr0.0001_pl0	0.6022	0.6022	4.000	Tue Nov 6, 00:14:32	8m 15s
hl2_lr0.001_pl0	0.5848	0.5848	4.000	Mon Nov 5, 19:40:10	8m 24s
hl2_lr0.001_pl1	0.5848	0.5848	4.000	Mon Nov 5, 20:49:33	13m 34s
hl2_lr0.01_pl0	0.5848	0.5848	4.000	Mon Nov 5, 19:29:28	8m 29s
hl2_lr0.01_pl1	0.5848	0.5848	4.000	Mon Nov 5, 20:32:16	13m 52s
hl2_lr0.1_pl0	0.5848	0.5848	4.000	Mon Nov 5, 19:18:40	8m 34s
hl2_lr0.1_pl1	0.5848	0.5848	4.000	Mon Nov 5, 20:14:37	13m 34s
hl2_lr1_pl0	0.5848	0.5848	4.000	Mon Nov 5, 19:07:45	8m 39s
hl50_lr0.01_pl1	0.5848	0.5848	4.000	Mon Nov 5, 23:41:44	16m 18s
hl10_lr1_pl0	0.5806	0.5806	4.000	Mon Nov 5, 17:44:08	12m 37s
hl25_lr1_pl0	0.5486	0.5486	4.000	Mon Nov 5, 21:33:49	8m 46s
hl2_lr1_pl1	0.5000	0.5000	4.000	Mon Nov 5, 19:57:17	13m 34s
hl100_lr0.01_pl0	0.4940	0.4940	2.000	Tue Nov 6, 00:35:09	3m 15s
final_hl100_lr0.001_pl0	0.4594	0.4594	4.000	Tue Nov 6, 01:46:13	8m 49s
hl100_lr0.001_pl0	0.4147	0.4147	2.000	Tue Nov 6, 00:30:07	3m 13s
final_hl100_lr0.0001_pl0	0.3346	0.3346	4.000	Tue Nov 6, 01:02:07	8m 46s
hl250_lr0.0001_pl0	0.2784	0.2784	1.000	Tue Nov 6, 00:44:48	4m 43s
hl100_lr0.0001_pl0	0.1663	0.1663	0.000	Tue Nov 6, 00:47:39	25m 47s
hl50_lr1_pl0	0.08484	0.08484	4.000	Mon Nov 5, 22:09:31	8m 11s
hl50_lr1_pl1	0.03255	0.03255	4.000	Mon Nov 5, 23:00:40	16m 16s
hl50_lr0.1_pl0	4.2022e-3	4.2022e-3	4.000	Mon Nov 5, 22:19:33	7m 51s
hl50_lr0.1_pl1	4.2022e-3	4.2022e-3	4.000	Mon Nov 5, 23:21:09	16m 3s

Fig. 16. List of preliminary results for the character level LSTM. hl refers to the number of hidden units. lr refers to the learning rate, and pl refers to number of LSTMs in series. The graph above demonstrates that the pairing between hidden units and learning rate has a dramatic impact on the results.

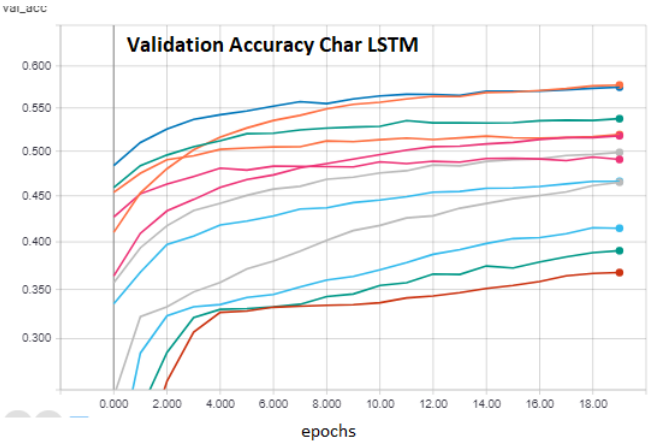

Fig. 17.

.56 - .59 accuracy for learning rates .01 and .001 with one or two LSTM layers as shown in 16. In the figure, the name refers to the hyperparameter configuration where hl refers to the number of hidden units, lr refers to the learning rate, and pl refers to the total number of LSTMs in series. The smoothed and value numbers are effectively the same, and represent the error rate. Time refers to when the particular configuration was run, and Relative refers to the total time it took for the configuration to run 4 epochs.

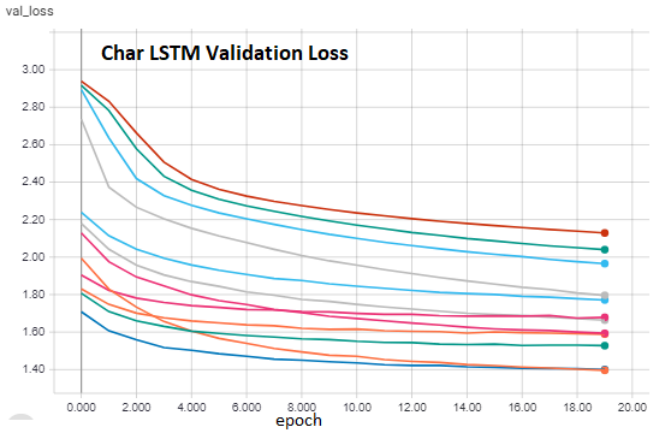

Fig. 18.

Name	Smoothed	Value	Step	Time	Relative
final_hl250_lr0.001_pl0	0.5772	0.5772	19.00	Tue Nov 6, 06:44:30	1h 40m 45s
final_hl250_lr0.01_pl0	0.5746	0.5746	19.00	Tue Nov 6, 08:48:16	1h 58m 0s
final_hl100_lr0.01_pl0	0.5375	0.5375	19.00	Tue Nov 6, 03:08:04	44m 50s
final_hl75_lr0.01_pl0	0.5191	0.5191	19.00	Tue Nov 6, 12:45:00	37m 36s
final_hl100_lr0.001_pl0	0.5177	0.5177	19.00	Tue Nov 6, 02:20:42	43m 17s
final_hl75_lr0.001_pl0	0.4987	0.4987	19.00	Tue Nov 6, 12:05:17	37m 10s
final_hl50_lr0.01_pl0	0.4906	0.4906	19.00	Tue Nov 6, 10:46:08	36m 24s
final_hl50_lr0.001_pl0	0.4660	0.4660	19.00	Tue Nov 6, 10:07:41	37m 18s
final_hl250_lr0.0001_pl0	0.4651	0.4651	19.00	Tue Nov 6, 04:58:16	1h 44m 27s
final_hl100_lr0.0001_pl0	0.4147	0.4147	19.00	Tue Nov 6, 01:35:03	41m 42s
final_hl75_lr0.0001_pl0	0.3906	0.3906	19.00	Tue Nov 6, 11:26:00	37m 38s
final_hl50_lr0.0001_pl0	0.3676	0.3676	19.00	Tue Nov 6, 09:28:13	37m 50s

Name	Smoothed	Value	Step	Time	Relative
final_hl50_lr0.0001_pl0	2.129	2.129	19.00	Tue Nov 6, 09:28:13	37m 50s
final_hl75_lr0.0001_pl0	2.040	2.040	19.00	Tue Nov 6, 11:26:00	37m 38s
final_hl100_lr0.0001_pl0	1.965	1.965	19.00	Tue Nov 6, 01:35:03	41m 42s
final_hl250_lr0.0001_pl0	1.796	1.796	19.00	Tue Nov 6, 04:58:16	1h 44m 27s
final_hl50_lr0.001_pl0	1.771	1.771	19.00	Tue Nov 6, 10:07:41	37m 18s
final_hl50_lr0.01_pl0	1.679	1.679	19.00	Tue Nov 6, 10:46:08	36m 24s
final_hl75_lr0.001_pl0	1.664	1.664	19.00	Tue Nov 6, 12:05:17	37m 10s
final_hl100_lr0.001_pl0	1.593	1.593	19.00	Tue Nov 6, 02:20:42	43m 17s
final_hl75_lr0.01_pl0	1.589	1.589	19.00	Tue Nov 6, 12:45:00	37m 36s
final_hl100_lr0.01_pl0	1.527	1.527	19.00	Tue Nov 6, 03:08:04	44m 50s
final_hl250_lr0.01_pl0	1.400	1.400	19.00	Tue Nov 6, 08:48:16	1h 58m 0s
final_hl250_lr0.001_pl0	1.396	1.396	19.00	Tue Nov 6, 06:44:30	1h 40m 45s

Fig. 19. List of final results for the character level LSTM. hl refers to the number of hidden units. lr refers to teh learning rate, and pl refers to number of LSTMs in series. The graph above demonstrates that the pairing between hidden units and learning rate has a dramatic impact on the results.The top shows accuracy, while the bottom shows loss

2) Results for RNN Word Prediction: At around 10 epochs, we received a validation accuracy of 23 percent, which is quite low as compared to what we got for char-level RNNs. Even the generated text as seen in 23 is not a good reflection of the actual text. The model favours heavily ¡unk¿ and ¡eos¿ tokens over the other words. This could be partly because the frequency of ¡unk¿ and ¡eos¿ tokens were relatively quite high. Also we had poor accuracy because of the fact that we had 14,014,000 total number of trainable parameters for RNN model while the number of sequences we trained our model with were only 5,917,360.

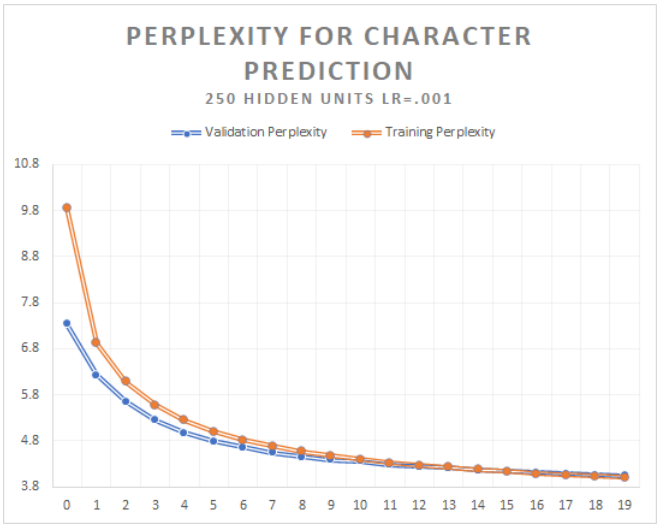

Fig. 20. Perplexity for the LSTM that gave the best results. Using 1 LSTM with 250 hidden units and a learning rate of .001

```
---Expected output:  today will unveil a new <unk> computer that will
compete with more expensive machines from companies such as sun
microsystems inc. and digital equipment corp
the closely held sunnyvale calif. company also will announce an
agreement to supply computers to control data corp. which will sell
mips machines under its own label
the new mips machine called the <unk> will cost $ n for a basic
system
the c


---Generating with seed: " is deputy editor of the journal
mips computer systems inc. "
 is deputy editor of the journal
mips computer systems inc. company 's <unk> and the <unk> and a state
in the <unk> of the company 's <unk> shares of the <unk> in the <unk>
and the market such as the bank and the <unk> and <unk> and the <unk>
and the state of the company 's stock in the <unk> and stock shares
of the president of the department and the <unk> of the president the
company 's the state of the <unk> and the <unk> of the company 's
<unk> and thee of the <unk> and the <unk> of the company 's <unk> and
the market that seek for the possible of the president and president
first and <unk> the past in the <unk> which as the added the company
's policial reserved and program trading in made of the out the new
york on the bush a shares the company 's company for the market of
the basks will be a <unk> the first the company 's department to
close the first <unk> and it would be company 's performation of  the
first <unk> and it would be company 's performation of the about $ n
million from $ n million results that it dicbsionally protects she
with a protection shipse deal sensically unchange dose and neations
to cutch income declinatics chairmanal co. orders and medicalt
mr. high-real-high hed imeric commerce
<unk> owner of be individual passibuted accustions to <unk>
discribitication of produce an <unk> federal sook
it has a chuirbing plant in the
```

Fig. 21. An example of the output of the character level RNN

We could expect to have improved text generation, given more training data.

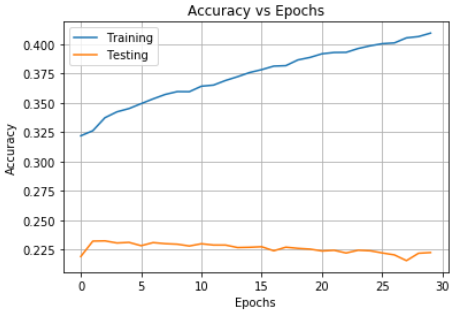

Fig. 22. Accuracy vs Epochs for Word RNN

```
Actual words: director of this british industrial conglomerate <eos> a form of
asbestos once used to make kent cigarette filters has caused a high percentage of
cancer deaths among a group of workers exposed to it more than N years ago
researchers reported <eos> the asbestos fiber <unk> is unusually <unk> once it
enters the <unk> with even brief exposures to it causing symptoms that show up
decades later researchers said <eos> <unk> inc. the unit of new york-based <unk>
corp. that makes kent cigarettes stopped using <unk> in its <unk> cigarette
filters in N <eos> although preliminary findings were reported more than a year
ago the latest results appear in today 's new england journal of medicine a forum
likely to bring new attention to the problem <eos> a <unk> <unk> said this is an
old story <eos> we 're talking about years ago before anyone heard of asbestos
having any questionable properties <eos> there is no asbestos in our products now
<eos> neither <unk> nor the researchers who studied the workers were aware of any
research on smokers of the kent cigarettes <eos> we have no useful information on
whether users are at risk said james a. <unk> of boston

Predicted words: director of the british bank conglomerate <eos> the spokesman of
<unk> <unk> <unk> to make the <unk> <unk> will been the <unk> <unk> of <unk>
<unk> <eos> other <unk> of <unk> <eos> to the <eos> than N N <eos> <eos> said
<eos> the <unk> <unk> <unk> is a <unk> and the would the <unk> <unk> <unk> more
<unk> to the <eos> widespread <eos> <unk> the the <eos> <unk> said <eos> the
<unk> a <unk> said <unk> york <unk> <unk> and <unk> <unk> <unk> in <unk> <unk>
<unk> the <unk> <unk> <unk> <eos> the <eos> the the <unk> were filed to than N
year ago the company <unk> include to the 's rally debt <unk> of <unk> a <unk> to
to be the <unk> to the marketplace <eos> the spokesman <unk> of the is the <unk>
<unk> <eos> the 're seeing about <unk> ago <eos> the <unk> the the with <unk>
<unk> <unk> <eos> the are no <unk> in the <unk> and <eos> the of nor the <unk>
who had the <unk> are <unk> of the <unk> to the of <unk> <unk> <unk> <eos> the
're a <unk> <unk> on the the are <unk> the <eos> <unk> <unk> <unk> a <unk>
```

Fig. 23. An example of the generated words of our LSTM

F. Perplexity v/s Epochs

In general, perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models. A low perplexity indicates the probability distribution is good at predicting the sample. The perplexity values for each epoch with decaying learning rates is computed and the performance is observed during training and validation. The following figures represent the results obtained while training both the models.

1) Perplexity v/s Epochs for Character-Level Model: The perplexity vs epochs is computed for 20 epochs for the Character-Level Model. The training and validation perplexity is plotted against number of epochs. The decreasing trend of perplexity is observed when the epoch size is increased in figure 20

2) Perplexity v/s Epochs for Word-Level Model: The perplexity vs epochs is computed for 30 epochs for the Word Level Model. The training and validation perplexity is plotted against number of epochs. The decreasing trend of perplexity is observed when the epoch size is increased in figure 24

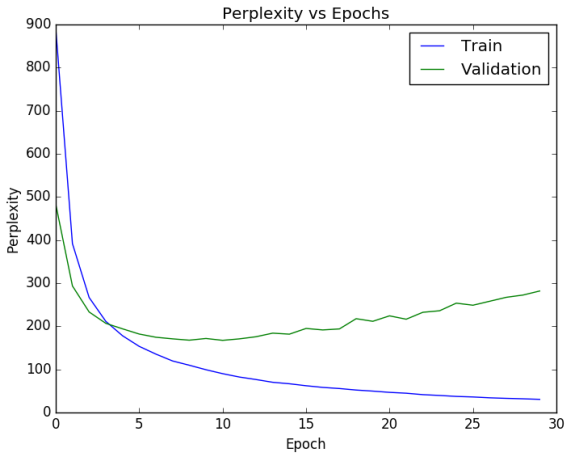

Fig. 24. Perplexity curve for Word Level RNN

IV. Discussion

The goal of this RNN is to predict which word will come after the current word that has been input. This ultimately

means that there are more categories of possible predictions. This changes the nature of the dataset. Introducing more categories allows our model more freedom for an erroneous prediction. We also notice that despite the dataset being the same, the fact that the letters are grouped into words, means that there are less overall time steps than there was for the character level RNN. Another way to explain this is to consider that the character prediction RNN models the temporal relationships between characters, while the word level RNN models the temporal relationships between words. Since there are more letters than words, the character level RNN has a relatively larger dataset to work with, therefore giving it an advantage over the word LSTM. This ultimately shortens the unrolled RNN structure mentioned briefly in the introduction. We hypothesize the reasons mentioned are responsible for the higher prediction error in the word level LSTM than for the character level LSTM. It is possible that this is why we notice word completion on mobile phones seems more accurate than sentence completion.

V. Conclusion

One area in which we could improve our analysis would be to run each configuration multiple times to be able to plot the standard deviation. Without doing this, we cannot be certain that our results are always consistent. It would also be very interesting to plot the accuracy of our character and word LSTM against a dataset size. There are several mobile text messaging apps which use word predictors. Some even store and allow you to download your own data. It would be especially interesting to use our personal data instead of our given dataset.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, November 1998.

[2] Tensorboard: Visual Learning Tensorflow, https://www.tensorflow.org/guide/summaries_and_tensorboard, November 11, 2018.

[3] X. Glorot, and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256).= March, 2010.