Project 4 - Recurrent Neural Network (RNN)

*

Varsha Nagarajan
200261357
Department of CSE
North Carolina State University
vnagara2@ncsu.edu

Sameer Watve
200249738
Department of ECE
North Carolina State University
sswatve@ncsu.edu

Neel RakeshKumar Pandit
200255109
Department of CSE
North Carolina State University
nrpandit@ncsu.edu

***Abstract*—The following report details our design and implementation of Recurrent Neural Network using LSTM units to demonstrate the power of LSTMs in applications such as sequence counting and language modeling. For sequence counting, a single LSTM unit was manually configured with different parameters to count the digit 0 in a sequence under different criteria. In case of language modeling, both word-level and character-level models were trained on the same neural network architecture and the model perplexity was plotted against the number of epochs. The network architecture comprises of an embedding layer dimension of 300, two LSTM layers with 300 units each followed by a dense layer with softmax activation. The hyperparameters chosen and few other observations are discussed in detail and associated results have been visualized through various plots. The perplexity on the validation set after 50 epochs of training came out to be 182.6 for the word-level model and 2.98 for the character-level model.**

***Index Terms*—Recurrent Neural Network, LSTM, PTB, deep learning, sequence generation**

I. Introduction

Sequence Counting and Language Modeling have numerous applications in today's world and are widely researched topics. Both problems pose the challenge to **"remember"** previously seen inputs to draw meaningful conclusions. While counting sequences, let's say we wish to identify the number of 0's that occurred, or the number of even numbers or many different complex formulations. Only if the neural network is successful in remembering the previous inputs can we develop a counter to keep track of sequences based on different imposed criteria.

Now, lets look at Language Modeling. The notion of a language model is inherently probabilistic. A language model is a function that puts a probability measure over strings drawn from some vocabulary. Besides assigning a probability to each sequence of words, the language models also assigns a probability for the likelihood of a given word (or a sequence of words) to follow a sequence of words. For a neural network to efficiently do this, it must **"remember"** the context. This excerpt from a famous blog will make the problem even clearer: "Humans dont start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You dont throw everything away and start thinking from scratch again. Your thoughts have persistence." So essentially, we want the network to remember things it has seen in the past. As opposed to traditional neural networks that were incapable to model such problems, Recurrent Neural Networks(RNN) addressed this issue and hence became popular in the field of Natural Language Processing. That being said, one major drawback of RNN was that it could only remember recent information. This might not be very helpful in cases where dictating phrases, terms, tense or feature occurred further away from the current text. And it is entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. In order to address this issue, researchers added a memory cell to these RNN units which led to the introduction of LSTMs (Long Short-Term Memory). These networks are powerful and the most successful in the field of language modeling because they can not only remember contextual information and other relevant information located far away from the point where it is needed but they can also remember to forget. The idea of what to remember, how long to remember and when an information can be safely forgotten to make way for new information is beautifully modeled in this network with the help of memory cell, input, forget and output gates and thus makes it our choice of network for the problem we set out to solve.

II. Dataset

For Part 2 of this project, we will be working with the Penn Tree Bank dataset that can be downloaded from Tomas Mikolov's webpage *here.* This dataset comprises of a vocabulary size of 10000 words. All words that fall outside this identified vocabulary are marked as $< unk >$. For the character dataset that was made available, the original data was just rewritten as sequences of characters, with spaces rewritten as $'\_'$ in order to facilitate training character-level models. For the purposes of our project, we will be working with both word and character-level datasets.

III. Structure of LSTM

Long Short-Term Memory networks are a special kind of RNN, capable of learning long-term dependencies. The key feature of LSTM is a cell state that imparts memory to the unit. Adding or removing information from the cell state is carefully regulated by structures called gates. They can be viewed as a

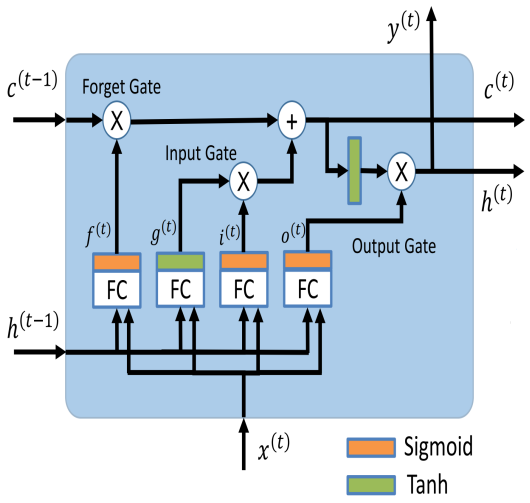

Fig. 1. A single LSTM unit

way to optimally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means let nothing through, while a value of one means let everything through.

These gates are named as input gate, forget gate and output gate. They essentially do the same thing as evident from their names. Input gate decides which values we will update into the cell state. The *tanh* layer creates a vector of new candidate values that could be added to the cell state. So, by combining the outputs of the *tanh* layer and input gate, we create an update to the cell state. The forget gate, on the other hand, decides which information from the cell state needs to be thrown away or forgotten. Finally, the output gate decides what the output will be. The *sigmoid* layer first decides which parts of the cell state needs to be output and then the cell state is put through a *tanh* function and then multiplied with the output of the output gate. This way, we only output the parts we decided to.

After understanding this working, we tried to fiddle around with the parameters in order to force the input gate to remain open or update the cell state or force the cell to forget certain information in order to formulate a parameter set to compute the counts of 0s in provided sequences. We present our configurations on conditioned sequences below.

IV. Part 1: Design LSTM cells for counting digits in a sequence

A. Count number of all the digit 0 in the sequence

We provide an explanation of the code already provided in the assignment just to give more context of how the problem is solved. There are several suitable parameter sets to achieve this task. Since we just need to keep track of one input value, which is the count of the digit 0, a one dimensional internal state will suffice. One possible parameter set to solve the problem is as below:

The input, output and forget gate are always open. This is achieved by providing a high value to their bias parameters bi, bf and bo. The input node gets the value 1 when it sees a 0, and value 0 when receives other digits. Thus, the internal state will be incremented by 1 every time the input node sees a 0.

Parameters:
Dimensions are: 10 x 1 (we have 10 digit values and 1 dimensional internal state hence 1 dimensional output)
Input Node:
wgx =[[100], [0], [0], [0], [0], [0], [0], [0], [0], [0]]
wgh = [0]
bg=[0]
For the parameters associated with the input gate, output gate and forget gate, all weights are zero and biases = [100]. This high bias ensures that the gates are always open.

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

Here, we need to keep track of two values, the number of digit 0 and digit 2. Only after we have seen the digit 2 once can we proceed with counting the number of digit 0s encountered in the sequence. Hence, the internal state is two-dimensional and so is the output dimension of every gate and input node.

The input node's output for the first dimension is 1 i.e., ON when it sees a 0 and the second dimension is 1 i.e., ON when it sees a 2. This is because both these information are useful and we need them and so we allow it. Now, the input gate will be ON only when it sees a 2 and shall remain ON thereafter. To formulate this, we set both dimensions to high values of 100 when the input gate sees a 2 in order to move the gate to open or ON state. For any other value it sees, the output is set to -100. After it has seen a value of 2, it shall remain on thereafter. The input node value combined with the hidden state of input gate help in dictating which information needs to be updated. Now, the hidden state associated with the input gate i.e., "wih" ensures that after the input gate is ON, all the later 0s encountered in the sequence will result in the internal cell state counter (first dimension) to be updated to record the number of occurrences of 0. Both the states of Forget gate and Output gate are always ON.

Parameters are as follows:
Dimensions are: 10 x 2 (we have 10 digit values and 2 dimensional internal state hence 2 dimensional output)
Input node:
wgx = [[100, 0],[0,0],[0,100],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]
wgh = [[0,0],[0,0]]
bg = [0,0]
Input gate:
wix = [[-100, -100],[-100,-100],[100,100],[-100, -100],[-100, -100],[-100, -100],[-100, -100],[-100, -100],[-100, -100],[-100, -100]

wih = [[0,0], [200,200]]
bi = [0,0]
For the output and forget gates, all the weights are zero and biases = [100,100]. This high bias ensures that the gates are always open.

C. 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

In this case, we need to keep track of three things: the number of 0s, whether a 2 was encountered and whether or not a 3 was encountered. For this problem, a two-dimensional internal state is required. The first part of the problem is similar to what we explained in **Part 2** with a small addition to it. We need to erase the counts after receiving a 3, which means we need to use the forget gate to clear the cell state of any previous count information (number of 0s) that it holds. So we need to push the forget gate to a LOW state i.e, turn it OFF. This will ensure that the previous cell state will be erased. In order to formulate this, whenever a digit 3 is seen in the input sequence, we set weights to a low value of -100 for both dimensions which force the *sigmoid* output to 0. Thus, we achieve the "forget" part for our problem. The output gate is always ON in this case.

Parameters are as follows:
Dimensions are: 10 x 2 (we have 10 digit values and 2 dimensional internal state hence 2 dimensional output)
Input node:
wgx = [[100, 0],[0,0],[0,100],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]
wgh = [[0,0],[0,0]]
bg = [0,0]
Input gate:
wix = [[-100,-100],[-100,-100],[100,100],[-100,-100],[-100, -100],[-100,-100],[-100,-100],[-100,-100],[-100,-100],[-100, -100]
wih = [[0,0], [200,200]]
bi = [0,0]
Forget Gate:
wfx = [[100,100],[100,100],[100,100],[-100,-100],[100, 100],[100, 100],[100,100],[100,100],[100,100],[100,100]
wfh = [[0,0],[0,0]]
bf = [0,0]
For the output gate, weights are all set to zero and bias = [100,100]. This high bias ensures that the gates are always open.

D. Plots

The plots shown in Figure 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 details the the internal cell states and states of the input, output and forget gates as observed while processing the sequence [1, 1, 0, 4, 3, 4, 0, 2, 0, 2, 0, 4, 3, 0, 2, 4, 5, 0, 9, 0, 4] for Task 1, Task 2 and Task 3.

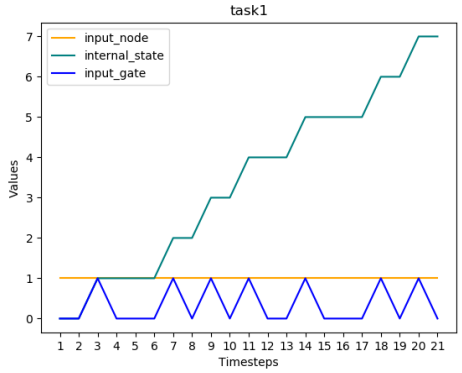

Fig. 2. Task 1 (Input node, Internal state and Input gate)

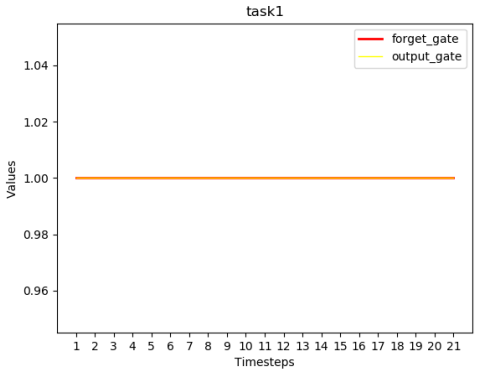

Fig. 3. Task 1 (Forget Gate and Output Gate)

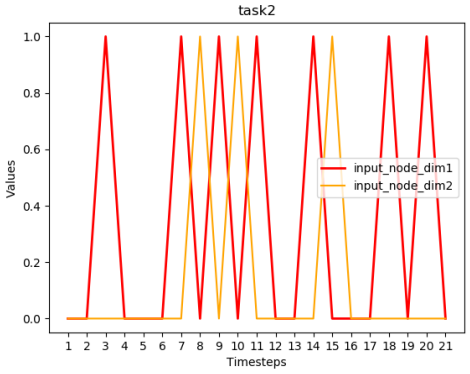

Fig. 4. Task 2 (Input node)

V. Part 2: Language Modeling using RNN with LSTM units

A. Data Preprocessing

The PTB dataset is a relatively clean dataset and did not require much of preprocessing to be done. However, there are a few things that we tried to experiment with in order to understand how it impacts the final results.

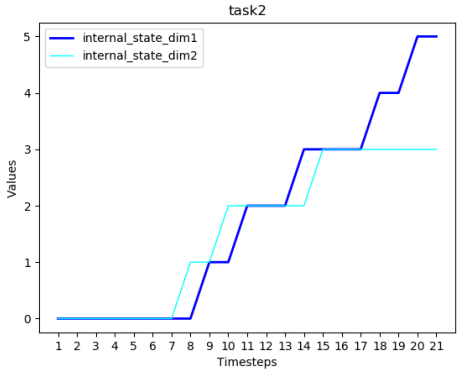

Fig. 5. Task 2 (Internal state)

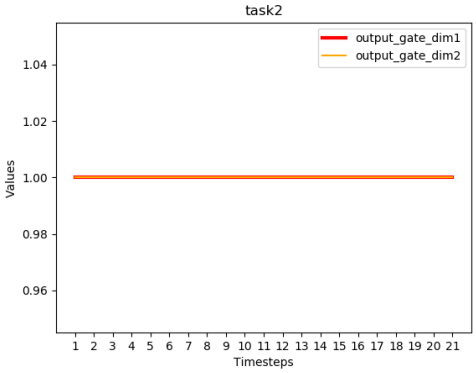

Fig. 8. Task 2 (Output Gate)

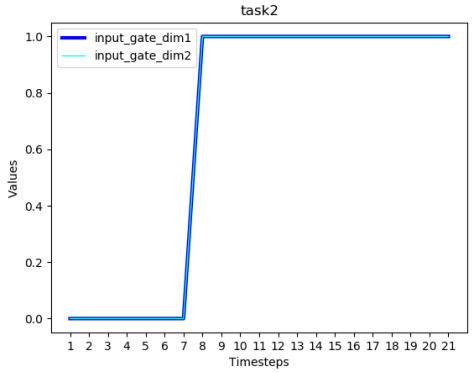

Fig. 6. Task 2 (Input Gate)

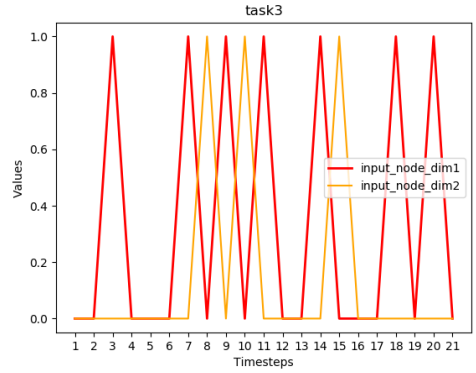

Fig. 9. Task 3 (Input node)

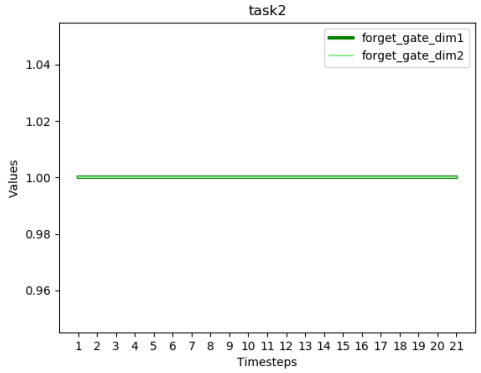

Fig. 7. Task 2 (Forget Gate)

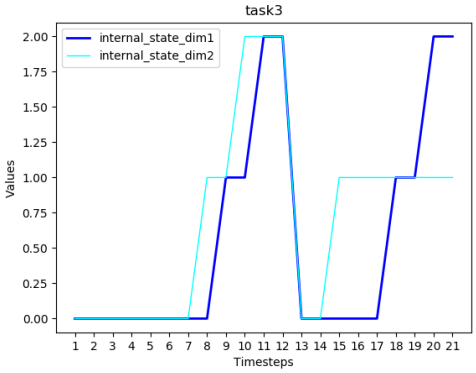

Fig. 10. Task 3 (Internal state)

1) Word-Level Model: As an initial preprocessing step, all the newline characters were replaced with $< eos >$ tags. As already mentioned, the dataset comprises of a vocabulary of size 10000 and all words that fall outside this vocabulary are marked unknown with the tag $< unk >$. So out of curiosity, we tried to train two models, one with the unknown tags intact (most researchers follow this methodology) and the other with the unknown tags stripped off from the data. There were no significant differences noted except that withtout the $< unk >$ tag, the generated text made more sense. We have presented our results in a later section.

2) Character-Level Model: As an initial preprocessing step, all the newline characters were replaced with a "." (a dot). The rationale behind this idea was that since the model is being trained on characters rather than words, a newline character could be learned as something that belongs to a word rather than an end-of-line marker. This might interfere with the text generated later on. So in order to aid the network to distinguish

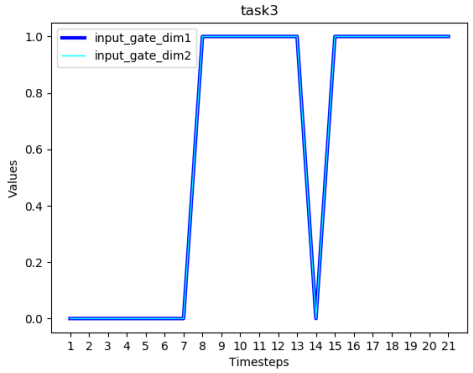

Fig. 11. Task 3 (Input Gate)

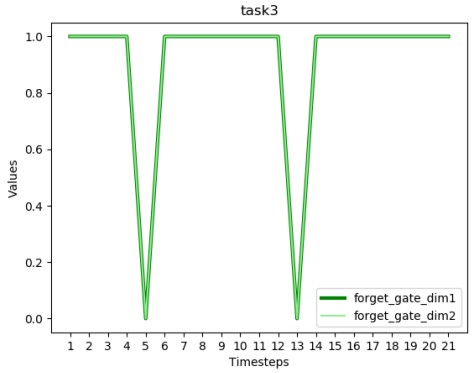

Fig. 12. Task 3 (Forget Gate)

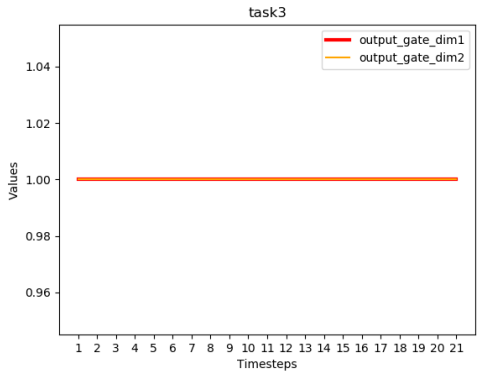

Fig. 13. Task 3 (Output Gate)

between an end-of-line marker and a possible character that could be a part of a potential word, we replaced all newline characters with dots. Additionally, all the spaces between characters were removed as they weren't essentially spaces but just a character delimiter. The underscores represent the space between words and so we retained the underscores as replacing them with spaces will have the same effect because what we finally feed to the model is just a vector representation of character indexes. All the other special characters were also retained because we believe that, in the context of natural language, they all make sense. Since, it is a sequence generation problem, all those special characters add meaning to the text as opposed to text classification problem where we would usually want to filter out such characters.

B. Recurrent Neural Network (LSTM) Architecture

Our proposed neural network architecture comprises of an embedding layer with an embedded vector-space dimension of 300, two LSTM layers (*tanh* activation) with 300 units each, followed by a time-distributed dense layer (with units = size of the vocabulary) with *softmax* activation. A dropout layer with a dropout rate of 0.5 was added after each LSTM layer in order to ensure that the model generalizes well instead of overfitting. We compute the categorical cross entropy loss and use this to evaluate the perplexity of the model. The same model architecture was used for Word-Level and Character-level models. Detailed model summaries are shown in Figure 14 and 15.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 32, 300)	3000000
lstm_1 (LSTM)	(None, 32, 300)	721200
dropout_1 (Dropout)	(None, 32, 300)	0
lstm_2 (LSTM)	(None, 32, 300)	721200
dropout_2 (Dropout)	(None, 32, 300)	0
time_distributed_1 (TimeDist	(None, 32, 10000)	3010000
activation_1 (Activation)	(None, 32, 10000)	0

Total params: 7,452,400
Trainable params: 7,452,400
Non-trainable params: 0

Fig. 14. RNN Architecture for Word-Level Model

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 100, 300)	14700
lstm_1 (LSTM)	(None, 100, 300)	721200
dropout_1 (Dropout)	(None, 100, 300)	0
lstm_2 (LSTM)	(None, 100, 300)	721200
dropout_2 (Dropout)	(None, 100, 300)	0
time_distributed_1 (TimeDist	(None, 100, 49)	14749
activation_1 (Activation)	(None, 100, 49)	0

Total params: 1,471,849
Trainable params: 1,471,849
Non-trainable params: 0

Fig. 15. RNN Architecture for Character-Level Model

C. Hyperparameters

Due to limitations of computational resource and time, we were not able to do a whole lot of hyperparameter tuning.

In most cases, we followed the best parameters reported in various popular researches in the field.

1) **LSTM Layers:** 2 LSTM layers with 300 hidden units each.
2) **Dropout Layer:** In our model, a Dropout layer follows each LSTM layer. We have 2 dropout layers in total with dropout rate of 0.5.
3) **Activation Function:** For the LSTM layers, we have used default *tanh* activations which have been reported to work really well in these cases but we hope to experiment with ReLU as well in the future. For the final Dense layer, we use softmax activation.
4) **Optimizer:** Adam optimizer with learning rate of 0.001 and decay of 0.0.
5) **Epochs:** 50
6) **Batch Size:** For word-level model, we used a batch size of 32 and for character-level model we used a batch size of 64. For character-level model, we increased the batch size as the model was taking a lot of time to train.
7) **Input Length:** In case of word-level model, our input sequence length is 32 words. For character-level models, we use an input sequence length of 100 characters. Both of these seem to work well in our case.
8) **Embedding Layer:** We define the Embedding layer with a vocabulary of 10000 for word-level model and 49 for character-level model, a vector space of 300 dimensions in which words/characters will be embedded, and input sequences that have 32 words/ 100 characters each.

D. Evaluation

Perplexity is a popular evaluation metric used for Language modeling and hence we used the same metric to evaluate our results. It is the average branching factor in predicting the next word. In other words, perplexity is the exponentiation of the cross entropy loss. Less entropy (or less uncertainty) is favorable over more entropy as predictable results are preferred over randomness. Hence, low perplexity is considered better than high perplexity. The perplexity of a model q is defined as:

$$b^{-\frac{1}{N}\sum_{i=1}^{N} log_b q(x_i)} \tag{1}$$

or more generally,

$$b^{loss} \tag{2}$$

where b is the base of the logarithm used to compute the loss. In our case, b is e as we use the natural logarithm for computing the categorical cross entropy loss.

E. Plots and Results

In this section we present the plots of the obtained perplexity vs the epochs. In case of word-level model, we note that beyond some 15 epochs, the evaluated perplexity on the validation set becomes consistent and increases slightly starting from 30 epochs. Though it doesn't increase by a very large amount, we need to look into our model and tune it further, maybe try different architectures and compare the performance achieved. For the word-level model we experimented with, after removing the $< unk >$ tag from the input data, we see that the curve becomes a bit bumpy after some 25 epochs. This was just an experiment done out of curiosity. Our studies show that researchers generally replace all words falling out of the vocabulary with an unknown tag such as $< unk >$ and train the model with it. We presented this result just to show the result of our experimentation.

For the character-level model, we achieved a decent perplexity value of 2.98 and as we can see from the Figure 18, the perplexity gradually decreases with increase in epoch which is something we would want and hence this model is performing as expected. Due to time and resource constraints, we could not experiment further but we believe that when trained with say 100-200 epochs might give a better result in both word-level and character-level models.

The perplexity on the validation set after 50 epochs of training came out to be 182.6 for the word-level model and 2.98 for the character-level model.

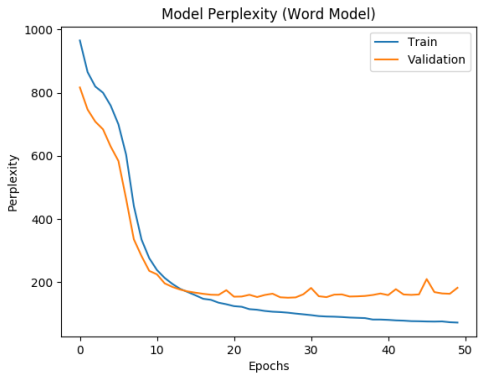

Fig. 16. Word-level Model (Perplexity vs Epochs)

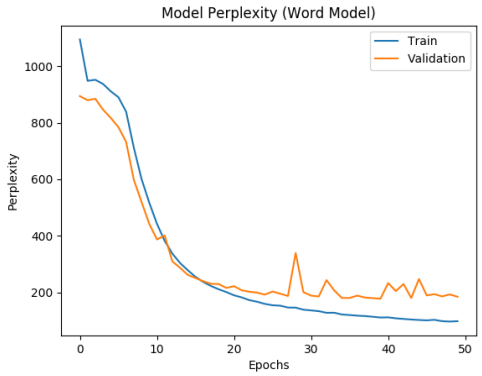

Fig. 17. Word-level Model (Perplexity vs Epochs) (This was obtained after filtering out $< unk >$ unknown words from the train data)

F. Generated Text

We now present the text generated by our models. A starting seed value was randomly picked from the provided test

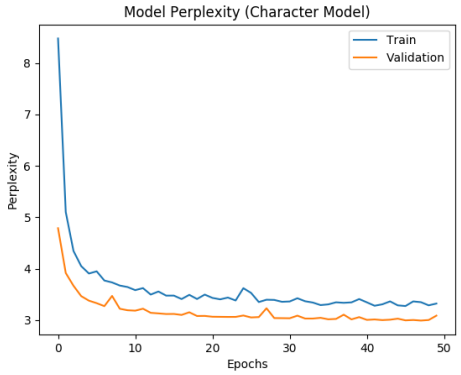

Fig. 18. Word-level Model (Perplexity vs Epochs) (This was obtained after filtering out $< unk >$ unknown words from the train data)

data. As mentioned before, the text generated after removing $< unk >$ (Figure 20) makes a tad bit more sense as compared to one with it. We are not drawing any conclusion here as it was just an experiment we tried out of curiosity. From Figure 19, we see that our generated text contains a lot of $< unk >$ tags which we believe will improve with more training. Many models that we surveyed had been trained for some 200 epochs to start with. Hence, we believe that with 50 epochs, our word-model has done a decent job but then we can surely obtain better results with more parameter tuning and training over some 200-300 epochs.

The generated text from character-model as seen in Figure 22 and 22, show that our model is performing fairly well. Though it is not able to generate text that is grammatically correct, many of the words have correct spelling which we believe is a decent performance given we trained it for only 50 epochs. We tried to fiddle around with the Temperature values of 1.0 and 0.8. The temperature defines how conservative or "creative" the model's guesses are for the next character in a sequence. Lower values of temperature (e.g., 0.2) will generate "safe" guesses whereas values of temperature above 1.0 will start to generate "riskier" guesses. The results seem to be pretty decent but more training will definitely yield better results.

Generating with seed:

"the prominent purchase is just the sort of deal that should be avoided for the time being <eos> in particular they <unk> the timing coming as it does on the heels of "

Generated text:

the prominent purchase is just the sort of deal that should be avoided for the time being <eos> in particular they <unk> the timing coming as it does on the heels of <unk> of the <unk> <unk> <unk> <unk> <unk> <unk> <unk> to the <unk> of the <unk> <unk> <unk> <unk> <unk> the <unk> and <unk> <eos> the <unk> <unk> <unk> <unk> <eos> the company said it expects to report a loss <unk> <unk> <unk> and <unk> <unk> and <unk> <unk> <eos> <unk> <unk> <unk> <eos> the <unk> <unk> <unk> <unk> <unk> time to be <unk> by the company 's <unk> <unk> flow for the <unk> of the <unk> of the <unk> bidder is n't likely to be <unk> <eos> the company <unk> of the company 's <unk> and <unk> <unk> <eos> share <eos> the company said it expects to report a company said it expects to report a loss of $ had been <unk> to <unk> <eos> the <unk> <unk> <unk> said the company 's <unk> <unk> <unk>

Fig. 19. Word-level Model: Generated Text

Generating with seed:

"at least n million <eos> separately the company said it expects to hold a special meeting for shareholders in early n to vote on its proposed reorganization <eos> many of the nation "

Generated text:

at least n million <eos> separately the company said it expects to hold a special meeting for shareholders in early n to vote on its proposed reorganization <eos> many of the nation of of n the than federal n't stock in state of n the this that of of union and the n the than that of of of lady the of have its cars n the this by on my a than n the this by on my a insurance in in able to the his that electric said the york of of officials n the this by on my a 's not firm n the this by on my a any to of of of are certain from of a the than said have the states plans a markets in a its keep a markets in able to the this for said have committee a its in current plans n n the than said the because so the this that n the this that order a its in state this the than said have in current plans n the than n the this by on my a insurance in pay 's the than n the billionaire by the accountants said in state health-care and the might exchange n the this some of of to the share n the share his this that support do n the this by on my

Fig. 20. Word-level Model: Generated Text (replaced all occurences of $< unk >$ with the word with next highest probability)

Generating with seed:

"n million to build <eos> the hotel will be on gorky street and initially will cater mostly to business travelers <eos> it will have a russian an english a and japanese and "

Generated Text:

n million to build <eos> the hotel will be on gorky street and initially will cater mostly to business travelers <eos> it will have a russian an english a and japanese and nine months the company said it expects to report a are n't going to be a good of the fed the senate judiciary committee 's chief executive officer <eos> the <eos> the house is a major of the of the of the company <eos> the company said it will be <eos> the of the u.s. 's n n of the <eos> the company said it will be a n million be reached for comment <eos> the company said it will <eos> the u.s. is a major role in the u.s. own stock <eos> the company said it will sell n result of the company 's financial condition <eos> the company the u.s. is the same thing the of the of the market 's value <eos> the dollar was quoted at at the <eos> the company said it expects to report to be a n n stake in the company <eos> been a director of the company <eos> the company said the end of the past decade <eos> the company said expects to report a n million loss for the third n <eos> the of the house and the senate judiciary year <eos> the company said it expects to report a the company <eos> the company said it will be a first of the of the u.s. chamber of commerce <eos> <eos> the company 's decision to be a major company <eos> the of the u.s. and the u.s. is the n n n bonds due nov. n n priced at <eos> the company said it was n't aware of the have a lot of people <eos> the of the u.s. u.s. <eos> the u.s. is the first time of the be a major role in the company 's decision to to n million from n million <eos> the company said

Fig. 21. Word-level Model: Generated Text after removing $< unk >$ tags

Generating with seed:

become_more_important_to_cabletv_channels.<unk>_'s_<unk>_habits_have_been_<unk>_by_its_format.its_s

Generated Text: (Temperature: 1.0)

become_more_important_to_cabletv_channels.<unk>_'s_<unk>_habits_have_been_<unk>_by_its_format.its_subscribers_played_was_willing_to_mr._layer_'s_because_nod_stabilized_much_<unk>_charges_and_paubil_crouth_for_businesses_on_the_awarded.since_small_senators.after_the_first_very_are_able.the_actually_mives_in_amounts_the_$_N_million_of_parts_at_six_months_N_N_N_within_N_million_shares_through_control_and_approval_of_<unk>_pharmaceuticals_issues_during_the_slowdown_had_n't_across_the_tryy.he_became_air_the_cantinate_july_in_his_heart_project.the_chief_executive_officer_<unk>.the_conservative_<unk>_package_or_are_been_the_current_helm_of_three_months.the_time_registered_funding_release_wo_n't_long_elected_down_over_<unk>_at_the_sausination_due_N_on_N_million_shares.but_however_is_heading_to_strong_evidence_of_visit_with_dejones_well_were_the_whester_options_before_their_wars_met_<unk>_it_may_be_that_private.approval_they_'re_seeming_a_end_in_property_dispatched_the_debling_onet_much_of_net_income.poland_for_commodity_opponents_are_quickly_at_least_damages_in_the_national_<unk>_shued.chan

Fig. 22. Character-level Model: Generated Text with Temperature 1.0

Generating with seed:

on.the_company_said_the_improvement_is_related_to_additional_<unk>_facilities_that_have_been_put_int

Generated Text: (Temperature: 0.8)

on.the_company_said_the_improvement_is_related_to_additional_<unk>_facilities_that_have_been_put_into_franklin_who_would_prevent_a_reports_that_small_board_'s_annual_gain_of_$_N_million_from_N_to_N_N_and_N_N_of_the_board_be_so_for_losses_by_the_earthquake_adding_that_the_company_said_it_was_<unk>_for_britain_'s_u.s._had_been_a_number_of_california_is_<unk>_not_that_corporate_paris_did_n't_extend_the_gain_on_the_new_york_stock_exchange_weeks_of_which_also_had_a_foreign_company_'s_executive_provision_a_security_company_is_received_with_delsis_was_united_was_the_state_a_N_N_restructuring_of_results_would_is_the_headquarters_of_huntary_executives_and_control_and_plans_to_resigned_being_gold_is_in_<unk>_of_international_debt_school_was_merrill_lynch.a_year_ago.mr._<unk>_are_likely_to_teums_the_development_of_partnership_the_company_can_go_for_lower_a_merrill_lynch_as_it_will_get_more_all_the_serviced_revenue_of_N_billion_shares.given_the_top_rates_increase_in_$_N_billion_as_a_representative_offering_trading_at_the_possible_third-quarter_net_income_are_the_government_that_it_will_were_brit

Fig. 23. Character-level Model: Generated Text with Temperature 0.8