Continuous Assessment Project
Charles Dickens Style Text Generation

Siddharth Pandey , Pranshu Ranjan Singh,
Nyon Yan Zheng, Tan Kok Keng,
Zaira Hossain

Institute of Systems Science, National University of Singapore

Abstract

This paper discusses the text generation task using recurrent neural networks (RNN) based on a corpus of Charles Dickens works. Two types of networks are used in this paper; Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM). The corpus is trained at both the word level and character level where the latter produces a better quality of text generated. The character level model also has its advantages of requiring less computational power compared to the word level model. The texts generated from the models are evaluated using both the machine and human evaluation score.

1 Introduction

This project involves the training and evaluation of a language model for text generation using deep learning.

Predicting the next word given a context in the form of a sequence of preceding words is the goal of statistical language modelling. A language model (LM) learns the joint probability distribution of words in a corpus of word sequences. In text generation, the language model generates word sequences based on its internal representations of the learned probability distribution.

Apart from language models based on methods from classical text-mining using parsing and morphology, neural networks have also been proposed for this purpose. Bengio et al. used feedforward neural networks in their neural language model [1]. Mikolov et al. explored recurrent neural networks to overcome the shortcomings of using only fixed length sequences as the context in feedforward networks [2].

2 Text Corpus

The corpus is obtained from https://www.kaggle.com/fuzzyfroghunter/dickens. It consists of a collection of all the works of Charles Dickens that are available through Project Gutenberg. Charles Dickens was an English writer in the eighteenth century.

The dataset consists of 32 fiction books by the author. These 32 books are combined to become one large corpus of the author's work. In total, there are approximately 174,000 sentences with an average of 5,400 sentences per book. The average sentence length is about 30 words. The vocabulary size is about 60,600 words.

Charles Dickens has a very distinct writing style; he writes in a poetic way and uses a lot of satire and consequently humour [4]. These features subsequently became one of the evaluation criteria of text generated from the models.

3 Task

The task in this project is to generate sentences based on the Charles Dickens style of writing.

4 Models

In this project, two different types of deep language models were explored:

i. a word-based language model, and
ii. a character-based language model.

4.1 Word-based Language Model

In a word-based language model, the tokens are words, the complete set of which forms the vocabulary of the corpus.

4.1.1 Data Pre-processing

The opening and ending paragraphs of the corpus comprises administrative details that are not relevant to the body of the story of each book. These were manually removed before all the 32 books were combined into one single text file.

Some pre-processing of the text data by line using regular expressions was performed to remove the following:

- Titles e.g. 'THE MORTALS IN THE HOUSE.', 'Chapter I'
- Image placeholders e.g. '[Picture: The haunted house]'
- Section separator symbols e.g. '* * * * * * *'

The text corpus was first separated into sentences using a sentence tokenizer from the Natural Language Toolkit (NLTK). Secondly, the sentences were converted to lowercase and tokenized using a word tokenizer also from NLTK.

Sentences were then cropped to a maximum of 100 tokens to reduce the computational load during model training.

4.1.2 Training / Validation Split

For the initial runs using the data from 5 books, after tokenization, there were 8,975 sentences out of which 6,000 were randomly selected for the training split while the remaining 2,975 formed the validation split.

The vocabulary size on the training split was 16,326. In addition to the tokens found in the data, 4 special tokens were added to the vocabulary.

Token	Description	Index in Vocabulary
<pad>	Tokens added to a sentence to increase its length to a fixed maximum length	0
<unk>	Token which represents a word not present in the vocabulary	1
<s>	Token which represents the start of a sentence	2
</s>	Token which represents the end of a sentence	3

For the final runs using all the data from 32 books, after tokenization, there were 92,996 sentences out of which 60,000 were randomly selected for the training split while the remaining 32,996 formed the validation split. The vocabulary size on the training split was more than 57,000.

Vocabulary size on the final runs was reduced due to computational platform limitations. We have used the Google Colaboratory platform to train our models. The graphics processing unit (GPU) random access memory (RAM) on Google Colaboratory is limited to 12GB.

```
Gen RAM Free: 11.1 GB  | Proc size: 142.5 MB
# of CPU: 2
CPU type: uname_result(system='Linux',
node='9792ee334f00', release='4.14.79+', version='#1
SMP Wed Dec 19 21:19:13 PST 2018',
machine='x86_64', processor='x86_64')
GPU Type: Tesla K80
GPU RAM Free: 372MB | Used: 11069MB | Util  97% |
Total 11441MB
```

4.1.3 Word-level Language Model GRU Model Architecture

RNNs were selected for the word-level LM. The recurrent hidden unit used is the GRU given its computational efficiency over the more complex LSTM unit. The models were implemented using the PyTorch deep learning framework.

The input to each model is a vector made up of a word-tokenized sentence less the last word, indexed on the vocabulary. The labels against which the input is compared is a vector of the same input word-tokenized sentence less the first word, similarly indexed on the vocabulary. The loss function used is cross-entropy. The output is a vector of logits for each word in the vocabulary.

The first layer in the model is an embedding layer of dimension 256. The next layers are GRU layers using hyperbolic tangent (tanh) as the activation function in the hidden nodes. No dropout was implemented within the GRU units.

A single dropout layer was implemented between the recurrent and the linear layers. Only a single linear layer of 64 hidden nodes was explored in one of the model due to GPU memory constraints. The activation function used was a rectified linear unit (ReLU).

The output layer is a linear layer. No activation function was applied to the output. The learning algorithm used to train the model is Adam.

The following models were trained:

Model	No. of Books	Vocabulary Size	GRU Layers	Hidden Linear Layers	Linear dropout	Learning Rate	Minimum Loss	Batch size	Perplexity
1	5	5,000	2x256	-	0.5	0.030	4.715	1,199	133
2	5	10,000	2x256	-	0.5	0.001	4.656	671	122
3	5	All (16k)	2x256	-	0.5	0.001	5.339	839	254
4	5	All	3x256	-	0.5	0.001	5.641	671	332
5	5	All	2x512	-	0.6	0.001	5.140	301	206
6	**32**	**40,000**	**2x256**	**-**	**0.5**	**0.001**	**4.562**	**3,727**	**119**
7	32	40,000	2x256	1x64	0.5	0.001	4.589	5,175	123

Temperature	Sample of Generated Sentences
1.00	<s> how silent it is you <unk> or only <unk> <unk> stir to me <unk> <unk> send you free of ‘em <unk> <unk> clennam — hush <unk> you shall whistle once <unk> can you write in feature to be a shallow sort of truth <unk> to no less as much enjoyment behind that <unk> though your fact is over ? </s>
0.75	<s> it was not likely that he was with the person of the circumlocution feelings <unk> <unk> had a more confident positive occupation than he had forwarded there <unk> they had been brought in at the best time by the lapse of a few years <unk> <unk> in the morning . </s>
0.50	<s> it was the matter <unk> by the fact of a moment <unk> <unk> that he had been in a state of an officer in the town of a moment <unk> <unk> <unk> with a man which had been in the same way in his way to his own <unk> <unk> the old man <unk> <unk> his own <unk> <unk> a bold man <unk> <unk> a great deal of man <unk> s <unk> <unk> we were always the great <unk> of the circumlocution house <unk> <unk> to make the most <unk> as there was a human sort of deceased <unk> but
0.25	<s> i am not a man of a man in the world <unk> <unk> i have no doubt <unk> <unk> i have not a man of course that i am not to be a man of course <unk> <unk> i am not a man <unk> <unk> i am not <unk> <unk> i am not a man <unk> <unk> i am not in my mind <unk> <unk> i am not so fond of my own <unk> <unk> i am not a man of course <unk> <unk> i am not obliged to you <unk> <unk> i am not a man of course <unk>

4.1.4 Text Generation

To generate sentences after training, the trained model is first presented with a sentence start token ‘<s>’.

At each time step, the output of the model is a sequence of vectors the last of which is the vector for the next word in the input sequence predicted by the model. Each vector comprises the logits for each word in the vocabulary. This vector is exponentially scaled by a temperature parameter before it is used to initialize a multinomial distribution. The index in the vocabulary of the predicted next word is sampled from the multinomial distribution. The temperature parameter 'squashes’ the distribution of the logits. Higher temperature makes the logits less different which produces a more random model output.

The predicted word is added to the input sequence and presented to the model at the next timestep. This loop repeats until the generated sentence reaches the maximum sentence length or a sentence end token ‘</s>’ is generated.

4.1.5 Evaluation

A sample of the generated sentences from the best model (lowest perplexity), model 6, is shown above.

A large number of ‘<unk>’ tokens seen in the generated sentences is due to the less than complete vocabulary used during model training.

When the value of the temperature parameter is small, more uniformity (less randomness) is observed in the generated sentences as the logits output by the model contributes more to the result.

4.2 Character-Level Language Model

A major challenge we faced with word level language models is exploding vocabulary size.

Decreasing the number of sentences to restrict vocabulary has a negative effect on sentence samples per word, making it difficult for neural network generalize.

An alternative approach to handle the large amount of text is using character level models. As the number of characters is mostly fixed (26 letters, 10 digits and punctuation symbols), the vocabulary size is small and easily manageable.

4.2.1 Pre-Processing

Preprocessing for character level language models is much easier because word and sentence tokenization are not required. But the presence of quite a few accented characters like (â, è, î, ô, ü) increases the vocabulary size unnecessarily. Thus, some preprocessing is still required to remove such infrequent characters.

```
import unicodedata
def strip_accents(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s)
                   if unicodedata.category(c) != 'Mn')

with open("../data/all_story.txt", encoding="utf-8-sig") as f:
    all_story = f.read().lower()
    all_story = all_story.replace("\n", " ")
    all_story = all_story.replace("  ", " ")
    all_story = strip_accents(all_story)
    chars_to_remove = ["{", "}", "£","%","æ","œ", "(",")","[", "]", "_","/","—","*",":"]
    for c in chars_to_remove:
        all_story = all_story.replace(c, " ").replace("  ", " ")
```

By using Python's inbuilt Unicode library and checking whether the character belongs to a certain category, accents were removed. Additionally, symbols like '{', '}' and other infrequent were also remove. Double spaces and new lines were replaced with single space, effectively combining all books into a single string object.

4.2.2 Data Preparation for Language Modeling

Once the data is cleaned, it is formatted and restructured before it is fed as input to the neural networks. Two processing steps are followed to reshape the data:

1. Tokenization of the characters.
2. Generating X, y pairs, where X is input sequence and y is the target character.

Tokenization of Text

Tokenization is performed by assigning all the unique characters an integer number ID. Then, this map of character to integer is applied on the complete text. Detokenization is performed by applying a reverse mapping. The total count of unique characters after preprocessing is 50.

Figure 1 Generating input and target pair

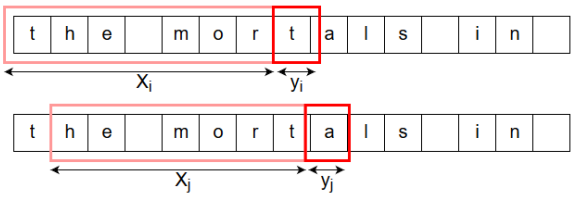
```
"char_to_int":{
  " ":0,
  "!":1,
  "\"":2,
  "&":3,
  "'":4,
  ",":5,
  "-":6,
  ".":7,
  "0":8,
  "1":9,
  "2":10,
  "3":11,
  "4":12,
  "5":13,
```

```
"int_to_char":{
  "0":" ",
  "1":"!",
  "2":"\"",
  "3":"&",
  "4":"'",
  "5":",",
  "6":"-",
  "7":".",
  "8":"0",
  "9":"1",
  "10":"2",
  "11":"3",
  "12":"4",
  "13":"5",
```

Generating Input and Target Pairs

After tokenization, the data is reshaped before given as input to the model. The task for training is defined as "Given a string of text of fixed length, the model must predict the next character that will follow". The tokenized text is one hot encoded to generate pairs of input and target. A sliding window is used to generate a large amount of such pairs for training.

Assuming batch size, *batch_size*,
Maximum length of the seed text be *max_len*,
Vocabulary size be *vocab_size*
The pairs of input and target were generated to be of shape:
X = (batch_size, max_len, vocab_size)
Y = (batch_size, vocab_size)



4.2.3 Data Generator and Train/Val Split

For character level models, the Keras instead of the PyTorch framework is used. Keras is a high-level API wrapper over other popular deep learning frameworks like TensorFlow, CNTK.

Following the previously discussed procedure of generating pairs, this will result in generating a large number of such possible pairs. When we tried generating the pairs iteratively, it was not

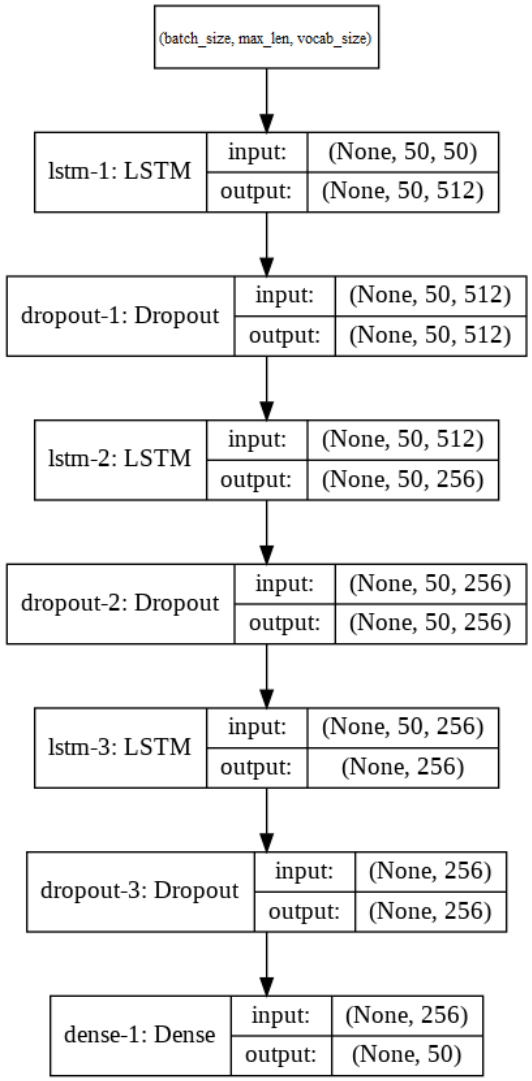
only taking a lot of time, but the computer went out of memory. Thus, we decided not to prepare all possible pair beforehand, and use random sampling to generate batches of such pair on the fly during the training. A similar procedure was used to generate the validation data. As the number of possible pairs is large, the validation and train data will not overlap in most cases.

4.2.4 Language Model Architecture

LSTM is used as recurrent units; between each LSTM layer, a dropout layer is added to provide regularization effect. The last layer is a fully connected layer with SoftMax activation. The loss function used is the Categorical Cross-Entropy and Adam is used as the optimizer.

Layer (type)	Output Shape	Param #
lstm-1 (LSTM)	(None, 50, 512)	1153024
dropout-1 (Dropout)	(None, 50, 512)	0
lstm-2 (LSTM)	(None, 50, 256)	787456
dropout-2 (Dropout)	(None, 50, 256)	0
lstm-3 (LSTM)	(None, 256)	525312
dropout-3 (Dropout)	(None, 256)	0
dense-1 (Dense)	(None, 50)	12850

Total params: 2,478,642
Trainable params: 2,478,642
Non-trainable params: 0



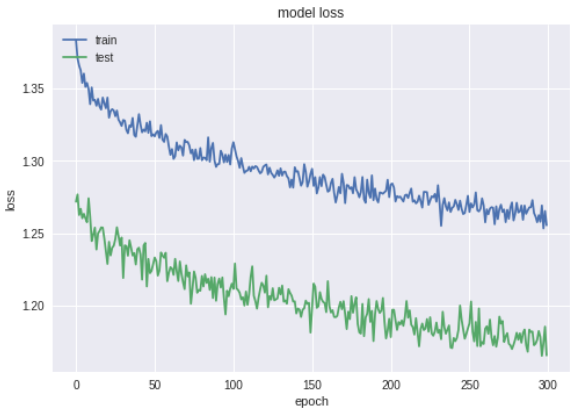
Training LSTM Models

While keeping the pre-processing procedure same, we experimented with various hyperparameters to successfully train a language model capable of producing text similar to Charles Dicken's style text.

During the training, **exploding gradients** emerged as one of the major challenges. The initial learning rate of 0.01 help to increase accuracy to around 50%, but afterwards, it frequently caused an increase in loss value. To counter that, we saved checkpoints of the best model (based on loss function value) and monitored the training process. Once the loss started increasing, the saved model weights were loaded and optimizers learning rate was decreased.

Another advantage of decreasing the learning rate was that it helped in a loss being minimized further. For instance, for most of the experiments, the loss would not decrease after certain value and it keeps oscillating around it.

However, by decreasing the learning rate, we were able to push loss even lower. We decided not to use decay factor, because of we were uncertain about its appropriate value, and were concerned about decreasing learning rate to early, which would slow down training. Thus, we manually monitored the losses to determine the apt time to decrease the learning rate.



Hyperparameter	Values	Remarks
Sequence Length	150, 100, 70, 50	Larger sequence length was difficult to fit, and loss and accuracy did not improve after some epochs. Finally, after some experiment, we realized sequence length 50 is easier to train. Our only concern with smaller sequence lengths is too little context. We realized for language level model, this might be the case, but for character level model, letters in near vicinity matter more. Smaller sequences also produced lesser spelling mistakes.
Number of LSTM Layers	2, 3	Both the configuration produced similar results, once they were trained to a certain extent.
Number of Hidden Units	128, 256, 512	After a few runs, we realized 128 is too small to capture the complexity of the text. Also having 2 layers with 512 hidden units was too slow to train. Thus, final architecture relies on 512 as hidden units for the first layer, followed by a smaller number of hidden units in further layers.
Dropout Rate	0.4	For most experiments, drop out value was fixed.
Learning Rate	0.01, 0.001, 0.0001	We realized that a high learning rate is not helping in decreasing loss after few hundred epochs. Smaller learning rate help minimize loss further.

The figure below shows how decreasing the learning rate from 0.001 to 0.0001 helps improve the accuracy further than 58% to nearly 61%.

Sample Text Generation

This section provides a few sample texts generated for different seed text, selected randomly. Randomness of the generated text can be varied using the temperature parameter while sampling. Temperature is the scaling parameter applied to probability before sampling. Higher values generate more diverse text, while lower values generate text that will appear closer to actual text corpora.

2 Layer LSTM

Temperature: 0.5

Seed:
uld be the boys of wished of the way before his ve
Generated text:
uld be the boys of wished of the way before his very life is the street and the ship, and i was likely to the old lady to a moment the prisoner of the sound of the day and politeness, and a red of the best and confidence, and so shame and all the mor

Temperature: 0.7

Seed:
people as poor as we are, until we see our way qu
Generated text:
uld be the boys of wished of the way before his very life is the street and the ship, and i was likely to the old lady to a moment the prisoner of the sound of the day and politeness, and a red of the best and confidence, and so shame and all the mor

3 Layer LSTM

Temperature: 0.6

Seed:
was reasonable, the and an opportunity of the tem
Generated text:
was reasonable, the and an opportunity of the temple of the present and great respects of a portion of the stairs of mind mr. snagsby, without a boot, brought to him in the time to the story of the part of a gentleman as she was trembled by which yo

Temperature: 0.5

Seed:
art of a gentleman as she was trembled by which yo
Generated text:
art of a gentleman as she was trembled by which you have come along with the window and the new police beating for the carriage and being otherwise to the track of the side. the party of the most coming at her with a countenance down in the door and

More samples of the generated text were given separately in the submission folder.

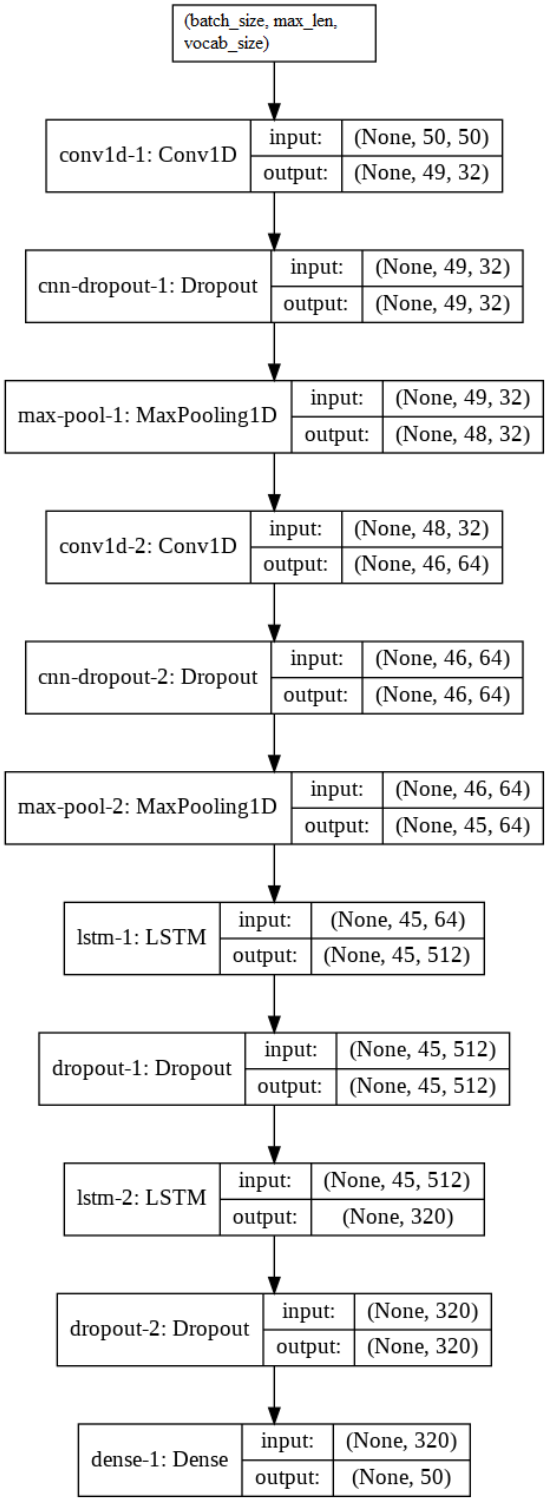
4.3. Temporal CNN to Enhance LSTM

In the article [5], authors demonstrate the effectiveness of temporal convolution net for text processing task. Inspired by this article, we decided to implement a simple CNN networks to enhance the LSTM predictions. A major advantage of Conv1D and MaxPool layers is their ability to decrease the length of the temporal dimension. This can allow larger sequence length to be used as an input also. The preprocessing steps for text and sample generation procedure is identical to LSTM model discussed above.

The basic architecture of recurrent layer remains the same. But on top of the top recurrent layer, two triples of (Conv1D, Dropout, MaxPool1D) are placed.

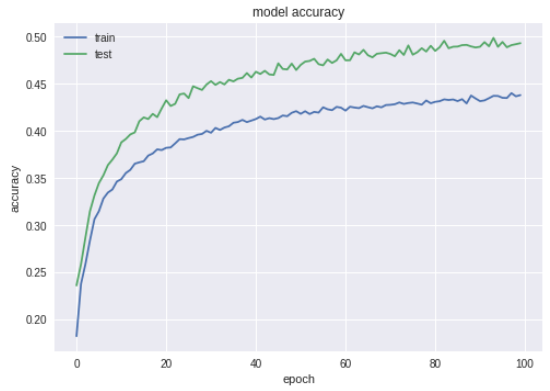
Layer Name	Hyperparameter	Value
Conv1D-1	Kernel Size	2
	Stride	1
	Channels	32
Dropout-1	Rate	0.3
MaxPool1D-1	Pool Size	2
	Stride	1
Conv1D-2	Kernel Size	3
	Stride	1
	Channels	64
Dropout-2	Rate	0.3
MaxPool1D-2	Pool Size	2
	Stride	1

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d-1 (Conv1D)            (None, 49, 32)            3232
_________________________________________________________________
cnn-dropout-1 (Dropout)      (None, 49, 32)            0
_________________________________________________________________
max-pool-1 (MaxPooling1D)    (None, 48, 32)            0
_________________________________________________________________
conv1d-2 (Conv1D)            (None, 46, 64)            6208
_________________________________________________________________
cnn-dropout-2 (Dropout)      (None, 46, 64)            0
_________________________________________________________________
max-pool-2 (MaxPooling1D)    (None, 45, 64)            0
_________________________________________________________________
lstm-1 (LSTM)                (None, 45, 512)           1181696
_________________________________________________________________
dropout-1 (Dropout)          (None, 45, 512)           0
_________________________________________________________________
lstm-2 (LSTM)                (None, 320)               1066240
_________________________________________________________________
dropout-2 (Dropout)          (None, 320)               0
_________________________________________________________________
dense-1 (Dense)              (None, 50)                16050
=================================================================
Total params: 2,273,426
Trainable params: 2,273,426
Non-trainable params: 0
_________________________________________________________________
```



A major issue while training the model, was that the model's progress becomes stagnant after a hundred epochs. Decreasing the learning rate only helps it reached around 48% in terms of accuracy, before the learning decays again. We tried with different values of hyperparameters discussed above and with different sequence length, but the

performance of this architecture does not reach close to the 3-Layer vanilla LSTM architecture discussed in Section 4.2.4 Though the generated sentences are better than word level model, it contains more misspelt words as compared to LSTM model.



Pair Embedding to Reduce Spelling Mistake

Continuing the exploration of temporal convolution-based model, we tried another very similar to LSTM model discussed in section 4.2.4 Instead of using only 1 character as minimum temporal unit, by using 1D convolution of kernel size 2 and stride 1, two character can be used as minimum temporal unit. In simple terms, two characters will be used as input to the LSTM.

The simple architecture is described in next figure. Another advantage of using the similar architecture of model in Section 4.2.4, we were able to load the saved weights and continue training the model. In terms of minimization of loss and accuracy, the model is able to reach similar values to that of the three-layer LSTM.

Layer (type)	Output Shape	Param #
conv1d-1 (Conv1D)	(None, 49, 50)	5050
lstm-1 (LSTM)	(None, 49, 512)	1153024
dropout-1 (Dropout)	(None, 49, 512)	0
lstm-2 (LSTM)	(None, 49, 256)	787456
dropout-2 (Dropout)	(None, 49, 256)	0
lstm-3 (LSTM)	(None, 256)	525312
dropout-3 (Dropout)	(None, 256)	0
dense-1 (Dense)	(None, 50)	12850

Total params: 2,483,692
Trainable params: 2,483,692
Non-trainable params: 0

Sample Generated Text

Two samples of text generated from the 2-layer LSTM model with Char-CNN are provided below.

Temperature: 0.5

Seed:
such the same and light with the munker of a chee
Generated text:
such the same and light with the munker of a cheerer so most spire for the cussale, and it is the same man and the possion of the most interdection of the taper of the consident and penerally, and the with mr. winkle, the sind of the ratter, and a f

Temperature: 0.6

Seed:
g of this old world and heart of the cite. and it
Generated text:
g of this old world and heart of the cite. and it the treet of the other servance and of his good without his agay, and poor of her looked of a bispast was her booms was any alment to an intikation and more of the looks about the back to the same of

Two samples of text generated from the 3-layer LSTM model with Pair Embeddings are provided below.

Temperature: 0.7

Seed:
of those children, born and bred in neglect and vi
Generated text:
of those children, born and bred in neglect and visit for himself in pickwick how are some more of my face on the grace in the man was shone distracted for the little more, in his landlength steps present of his sawot. â€˜if you never have stood and th

Temperature: 0.6

Seed:
the princes of ralph of the manner at a were thro
Generated text:
the princes of ralph of the manner at a were through the stairs; the exclaiming of it and unhandled to be three swall side that we should have got to the happened to the hearts of which he was so cheerfully pounded and lingered to any one of the few

More samples of the generated text were given separately in the submission folder.

5 Models Evaluation

Two methods of evaluation of the text generated by the models have been performed; (1) machine evaluation and (2) human evaluation.

5.1. Machine Evaluation

Machine evaluation for word level model is done by comparing the perplexity value. The model with lower perplexity score is considered better than other. From section 4.1.3, we see that Model 6 performs the best among all the word-level model in terms of perplexity. Model 6 is a 2-layer word-level GRU model. This model has been selected to be compared with the rest of the character level models.

As a measure of model performance for character level models, the following three metrics have been used to deduce which one performs the best: (1) Loss Minimization (2) Error Count. As perplexity of character level cannot be calculated directly based on loss value, we decided not to use it for comparison between character level models.

Model Name	**Accuracy**	**Loss Value**
Character level 2-Layer LSTM	62.00%	1.20
Character level 3-Layer LSTM	**62.80%**	**1.18**
Character level 2-Layer LSTM with Char-CNN	44.80%	1.50
Character level 3-Layer LSTM with Pair Embeddings	58.55%	1.31

From the table above, we see that the 3-Layer LSTM has better loss value and accuracy score, followed closely by 2 Layer architecture. Pair embeddings model also performs decently, having scores close to LSTM architectures.

Spelling Error Count

25 samples of generated text have been taken for each of the character level models and then analyzed for spelling mistakes. As a metric for comparison between models, the average count of per sample spelling (sample size is fixed at 200 characters) errors has been computed to yield the following results:

Model	**Average Spelling Errors**
Character level 2-Layer LSTM	**2.32**
Character level 3-Layer LSTM	2.96
Character level 2-Layer LSTM with Char-CNN	9.64
Character level 3-Layer LSTM with Pair Embeddings	3.28

From the above results and the workings in the attached *spellEvaluation.xlsm,* the following notable observations can be made:

- Many errors are due to occurrence of unicode characters
- The 2 Layer LSTM yields the best performance with close competition from 3 Layer LSTM.
- The 2 Layer LSTM with Char-CNN yields the most inferior results

5.2. Human Evaluation

20 samples are generated from each of the five selected models and are human-evaluated based on 3 criteria: (1) adequacy (2) fluency, and (3) elements of Charles Dickens style. The first criteria, adequacy, measures if the generated text turns out to be a story or it a complete nonsense. The second criteria, fluency, measures the smoothness and flow of the text generated. The last and third criteria, intends to detect if the generated text contains any elements of Charles Dickens writing, which usually contain long sentences that are poetic and humorous in some way. The samples are randomly distributed between a few human evaluators and each criterion is given a score between 1 and 5 based on the description below.

Criteria 1: Adequacy

Score	**Description**
1	complete nonsense
2	very little meaning can be made from text
3	some meaning can be made from text
4	text is almost like a story
5	text is perfect as a story

Criteria 2: Fluency

Score	Description
1	incomprehensible
2	disfluent
3	acceptable
4	good
5	flawless

Criteria 3: Elements of Charles Dickens Style

Score	Description
1	undetectable poetic and/or humour elements
2	contain very little poetic and/or humour elements
3	contain some poetic and/or humour elements
4	contain significant poetic and/or and humour elements
5	completely poetic and/or humorous

Samples of text are generated from the 5 selected models below shown in the following table.

Model	Description
A	Word level 2-Layer GRU
B	Character level 2-Layer LSTM
C	Character level 3-Layer LSTM
D	Character level 2-Layer LSTM with Char-CNN
E	Character level 3-Layer LSTM with Pair Embeddings

The average between the 3 criteria scores is then calculated as the overall score for the model. The summary of the scores is given in the table below.

Model	Average Score			
	Adequacy	Fluency	Charles Dickens Elements	Overall
A	2.550	2.500	2.150	2.400
B	2.640	2.960	2.640	2.747
C	**2.920**	**2.880**	**2.720**	**2.840**
D	2.000	1.920	1.920	1.947
E	2.360	2.320	2.640	2.440

From the table above, the best model based on human evaluation is the 3-layer character level LSTM model.

A Note on Evaluation

A cursory internet search of the definition of Dickensian prose provided several insights [3].

Dickens was known for the unique and strong characterization in his stories. In his writings, personification was also a key marker. He associated inanimate objects, even situations and conditions with human-like properties. He was also known for using repetition to emphasize his narratives. His narratives tended to be based on concrete terms as opposed to abstract concepts.

As the project involved generating single sentences, it was not possible to evaluate on characterization. Personification, repetition and concrete terms can be observed in an individual sentence, but it is unlikely that they will be observed in all sentences.

A human expert, a teacher in literature in an educational institution at the secondary school level, concurred that evaluating single sentences for elements of Dickensian prose would not be informative. Many sentences in a Dickensian novel do not possess any unique elements of Dickensian prose.

6 Conclusion and Future Work

The study focused on text generation based on the Charles Dickens style of writing. Both word level and character level language models were used for experiments. Based on the human evaluation metrics, Character level 3-layer LSTM model performed the best among the five models used for experiments. To improve the character level LSTM model, 1D convolutions and a pair character embedding was implemented. The idea is to increase the length of the input sequence length and reduce the spelling mistakes. Although, theoretically convincing, the results from the added functionalities performed better than Word level GRU model but didn't improve the human evaluation metrics when compared with character level LSTM model. The task to improve the above character LSTM model using the mentioned ideas is under consideration and part of our future work.

References

[1] Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb (2003): 1137-1155.

[2] Mikolov, Tomáš, et al. "Recurrent neural network based language model." Eleventh annual conference of the international speech communication association. 2010.

[3] Dalfonzo, Gina. “How to Tell Charles Dickens's Prose From Every Other Dead White Guy's.” The Atlantic, Atlantic Media Company, 24 Apr. 2013, www.theatlantic.com/entertainment/archive/2013/04/how-to-tell-charles-dickenss-prose-from-every-other-dead-white-guys/275264/.

[4] “The Writing Style of Charles Dickens.” PC Dreams, 14 Oct. 2016, pcdreams.com.sg/the-writing-style-of-charles-dickens/.

[5] Zhang, Xiang, and Yann LeCun. "Text understanding from scratch." *arXiv preprint arXiv:1502.01710* (2015).