Exploring Text Generation with Recurrent Neural Networks

Derek So ddso@ucsd.edu

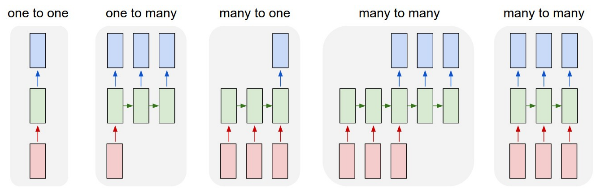
Abstract

In this paper, I use two rudimentary recurrent neural network algorithms to explore the efficacy of a text generation model on two large datasets of writing. The performance of these models, across various parameters, are nominally evaluated and improved upon by loss, but ultimately better judged by the understandability of the text generated. Despite the relative simplicity of both approaches to character-based recurrent neural networks, surprisingly sensible outputs were achieved given well-tuned parameters.

1. Introduction

Recurrent Neural Networks (RNN) are powerful and robust, utilizing internal memory to consider both current and previous inputs in its predictions. Thus, this makes it well suited for sequential data such as audio, finances, and in this paper, text. Each progressive character in text generation is reutilized in the model in order to inform the next character. Thus, while the model is only explicitly aware of individual characters, entire words and comprehensible full sentences can be generated.

There are several variations of combination weight mapping that allows RNNs to be so powerful. Given that inputs are vectorized in order to be learned, their outputs can be radically different depending on the training task. In a standard one to one task, inputs and outputs are fixed, like in our text generation. One to many tasks expands upon the output, such as image captioning, while many to one condenses, like sentiment analysis of a text (*Unreasonable*). And while there are more variations of sequence inputs and outputs, what these methods share are the hidden layers of neural nets that can vastly expand beyond the data constraints of the input.



2. Methods

A. Algorithms

The first approach method used pure NumPy and variations of two parameters to help in developing a feel for how RNNs operate. The methodology follows the basics of RNN, with matrices to transform the vector into the hidden layer, from hidden to hidden layers, and from hidden layer to output, along with a hidden and output bias. The parameters for this early exploration was the size of that hidden layer of neurons and the number of steps to unroll or the number of hidden layers.

This method was influenced heavily by Karpathy's min-char-rnn.py (Minimal) and is designed to run indefinitely until stopped by the user. I ran both the base parameters of 100 neurons and 25 unrolls, and combinations of parameters from [64,128,256,512] neurons and [3,10,25,50] layers, and loss and results were recorded every 1000th iteration. In the attached code, this RNN was only trained on the Shakespeare dataset, and the results were shortened for readability.

The second approach utilized the TensorFlow Keras library, informed by the core tutorial ("Text"). The functionality of the model is effectively the same but has more parameters that are much more easily modifiable. While more options were available, the parameters in this systematic approach of learning TensorFlow sequence modeling was limited to the size of the embedding dimension (input vector), the number of neurons, and the number of epochs to train for.

Due to extensive training time, 256 embedding dimensions, 1024 RNN units, and 10 epochs were used as the baseline, and variations of parameters were attempted one at a time. For each respectively, they were [64,128,256,512,1024] for both embed dimensions and RNN neurons, while epoch from [5,10,20,30,40,50] were explored.

As using TensorFlow was more streamlined and effective, text generators were created for both the complete works of Shakespeare and Sherlock.

B. Datasets

The complete works of Shakespeare and the collection of Sherlock Holmes are text files, 1,115,394 and 3,381,928 characters long respectively. The former has fewer unique characters, 65 in total, while the latter has 97 unique characters. Some notable descriptors of each dataset are the play/script format of the Shakespeare text, while the Sherlock text follows the more standard novel paragraphs and dialogue.

3. Experiments

For the 0th iteration of the base 100 neurons and 25 layers, the results were expectedly nonsensical, producing the following with a loss of 104.36.

```
vzNSHmEG3r;etQMqoceZVqvsIanUXGA
OX$M.jYgcUDCPvAzCdPX jQlMs- bavH!qHEAekR'PF
:mNwnDo?J
:tnzEA;ygosWrthn;SF?$vFrXM
g3HGnvazpNvNoj&z'dhdZUB!;QDzwKy,diriojFRm&;kp;sMcx
:erU:leVbk:3Hbh;;KF:E:DwT:rIeH$YKV
```

General word shapes were introduced by the 1000th iteration, along with a semblance of paragraph structure, though punctuation is eluded:

```
 lrd nve be ca,cMot; tesi: .nut ltolcces bo cor.: Waucus.
Whural fnadr ous to 'arus us t etwe, he the bnit,
CNSPtoAnnasir thas yid somlme dot sin hthi.! T TwA
pCHy thepar mekersorat toi yhiont. Mes
```

Correct punctuation is achieved by iteration 4000, along with capitalization in for the first letter of sentences and 'names':

```
 thal son lised, no es tics on then aso ines ou. Wo wet eat, Morr co fall ok ardt
till foby ad senod.
Pgaoidst, A'esh ale fleunln fiold,r. S'ep, the huspnos kew'll herid thii, vufime is!
San go thee B
```

The play format of 'NAME: Verse' is clear by iteration 400000, with some understandable words. Loss by this point has dropped to 45.73.

```
I you yeald age, in rection Jade the tratt, And and notess comke?
MENRIINTE: Thinung; Mis, make might on pie.
TONTET: Inis your with, Brong? Thou here Swold youip! I will combin
sontent you' mintu?
```

Unfortunately, there is no significant improvement from here on for this vanilla RNN model, as even by the 11,000,000th iteration the loss is still 43.66.

```
 u peroth hooster, and Caurth. het were thy holld fier up lost he in pad your face
thou shall ere the ray. Pesphing. We, hoped tengets him.
DUCHESS OF APAUS MIRIA: Noly pakest and neixt they did, reme
```

Experiments with higher hidden sizes resulted in more extensive training times, and text generation did not necessarily improve. For hidden size 512 and unroll 25, loss started at reached 59.61 by iteration 500000. The greater time cost did not produce better text by the same iteration.

```
e int. shat, my toll.
BUNETI: Thap at ut vomes: yor I hel them in me dow ald bren sellisden you,
starngt ef,' con! Wow Rot sthe duker wy, dovu mupnts'grases, spon: Pve: And
mand in Bithan: Sigome ous
```

Lower unrolls consistently meant lower starting losses, as seen in the table on the next page, and in fact, could achieve equal text generation results in the same number of iterations.

Loss of Model at Iteration

Iteration	100 neu, 25 unrolls	100 neu, 3 unrolls	512 neu, 25 unrolls	512 neu, 3 unrolls
0	104.36	12.52	104.36	12.52
250000	48.98	6.10	62.89	10.22
500000	45.73	6.05	59.61	10.06

Iteration 500000 of 100 neurons and 3 unrolls had an equivalent word and play structure to the 100 neurons 25 unroll model, at the cost of fewer comprehensible words.

no dare laved they sanderd fore par oly wi laildion I ran oul ma? I read fairen,
Lecermast,
I.
And Grus peacimes trince. We knoots.

KING OF YORD
The asing wath herswnrold wilcunat,
Frosorvenve'lors

With more neurons and fewer unrolls, however, text generation was even worse. For 512 neurons and 3 unrolls, text generation at 500000 iterations was about equal quality to the 1000th iteration with 100 neurons and 25 unrolls:

.S nw ,thiEsohnred, o-u tagtOmEo ihre ',aveeBh ahsds no'Wtrl oro ns
tfiehsPsaoduroo nhrWisoinl
othElw orrMivtrts dh rirrHd snwn't iocnesdt rh yldy g T dliR ode ,Sd korwt
Reohiia, e aooJo ebtret

Meanwhile, the second streamlined TensorFlow method was far more efficient and effective. Loss is recorded every epoch, and for the 256 embeds 1024 neurons 10 epochs base model, began at 3.27 at Epoch 1 and fell to 1.24 by Epoch 10.

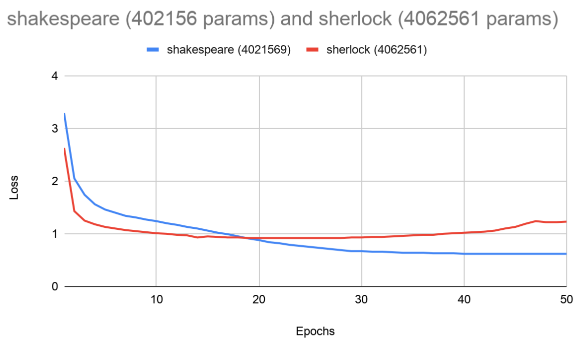
The base model took a far shorter time to achieve far better results. Given the string “ROMEO: “ to start, the model generated:

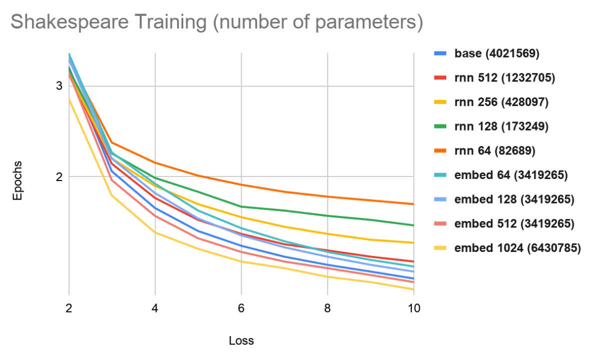
ROMEO: he is like a saith, And le in that doth she's a little spirit with shroud Weeping wind Romeo; you are so, bring his charms.
GLOUCESTER: Why, closely-tongues. No more of my tongue? Dost look your whose soveries in the trumpets born, To find some finning and sovereigns, who should reliss our petnicon Romeo all hours in good honourable from a famous! By not a seit doth same heir For henter this mouther there's than direction of it:

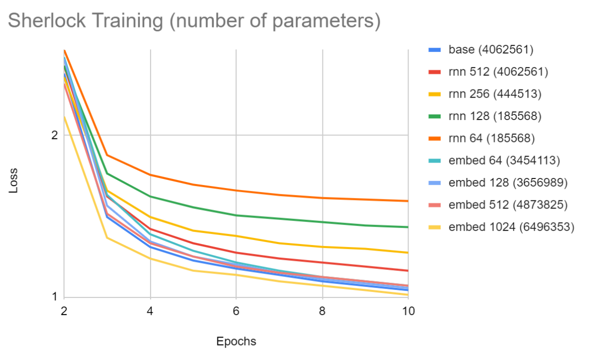
Words were almost all correct, and those that were not followed a pronounceable consonant/vowel logic. And while grammar was choppy, it was clearly comprehensible.

In decreasing the amount of neurons, only word creation was significantly affected, degrading more and more with fewer neurons, while punctuation and structure remained sensible. For example, with 64 neurons, the following was produced after 10 epochs reached a loss of 1.75: “Whence if you rearn, and name a not, them dean, in the Luptibles wo$lidaf the say, cont, fort' sust:” The logical structure of the sentence is there, though strange things such as $ appear in a word. With this method, as all loss values are within the same range, a lower loss value is a reasonable indicator that the text generator is more understandable.

With the Sherlock dataset, the same results were found, with one major difference (generated texts may be found in the code, and they follow the same overall learned punctuation and structure). Training the text generator for longer than ~22 epochs led to a gradual rise in the loss.







4. Conclusion

As seen by the graphs, there is a clear reduction in loss when training with more parameters and when training for more epochs, regardless of the dataset, besides the aforementioned exception. These results are evident in the effectiveness of the text generator itself, notably in the character accuracy in individual words. However, overall punctuation, capitalization, and even sentence structure are picked up fairly quickly in the algorithm and effectively generated, and even words that are legitimate follow a consonant and vowel structure that can be easily verbalized.

That this sequence generation model can achieve so much through direct awareness of only individual characters is astounding. The applicability of RNN on text is not merely limited to alphabetical languages, either, as there have been similar efforts with another of the world's most common languages, Chinese (Zhang et al.). The applicability of sequence structure to the strokes of an individual pictograph is novel, and that perspective makes me curious about the many other applications RNN has on language alone.

In the future, I hope to explore more of what Machine Learning has to offer with text interpretation and generation. Whether it be in other loss methods or long short-term memory applications, or new learning models altogether, my greatest passion is language and I can't wait to see what artificial intelligence has in store.

Acknowledgments

Thank you to Professor Zhuowen Tu and the TA staff who taught this course, and their efforts in accommodating the COVID-19 crisis. I very much appreciated the professor reaching out to see my progress when I arrived early in class.

References

Karpathy, Andrej. Minimal Character-Level Language Model with a Vanilla Recurrent Neural Network, in Python/Numpy, Gist, gist.github.com/karpathy/d4dee566867f8291f086

Karpathy, Andrej. *The Unreasonable Effectiveness of Recurrent Neural Networks*, Github, 21 May 2015, karpathy.github.io/2015/05/21/rnn-effectiveness/.

.

"Text Generation with an RNN: TensorFlow Core." *TensorFlow*, www.tensorflow.org/tutorials/text/text_generation.

X. Zhang, F. Yin, Y. Zhang, C. Liu and Y. Bengio, "Drawing and Recognizing Chinese Characters with Recurrent Neural Network," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, pp. 849-862, 1 April 2018, doi: 10.1109/TPAMI.2017.2695539.