Supervised Text Generation via Character-wise RNN

Robert Zhang

March 17, 2023

1 Abstract

This report presents an implementation of a character-wise Recurrent Neural Network (RNN) from scratch. The goal of this work is to provide a comprehensive overview of the key principles and techniques involved in building a Char RNN for text generation.

2 Introduction

Large pre-trained language models like GPT, ELMO, BERT, and RoBERTa can certainly solve a wide range of NLP tasks, while having a different approach, RNN has also being widely used in speech recognition, machine translation, and language modeling for a long time. Since Artificial Neural Networks were developed since 1980's, it became clear that feedforward neural networks are not good enough to capture dependencies that change over time. RNN was inspired by biological neural networks which have recurring connections. In 1989, Time Delay Neural Network (TDNN) was the first attempt to add memory module to neural networks, which is clearly disadvantaged since the temporal dependencies are limited to a chosen time window. The vanishing gradient problem, in which contributions of information decayed geometrically over time, remains a key issue for the following variants of RNNs like Elman Networks and Jordan Networks in the early 90s. Long Short-term Memory (LSTM) was the first architecture to address this problem to keep the state variables using gates so that arbitrary time intervals can be represented.

The main feature of RNN is temporal dependencies, in other words, the current output does not only depend on the current input, but also the past inputs. Take machine translation as an example, the correct word to use definitely depends on all the other words in the sentence.

Original text	Interpretation of the italic word alone
Bruce Wayne saw a lot of *bats* flying in the cave.	(def1) a flying mammal (def2) a piece of sports equipment used for hitting a ball

Table 1: Example of word ambiguity without context

In terms of text generation, for instance we have a Shakespeare line "To be or not to be", the network will learn one character at a time and then generate new text one character at a time.

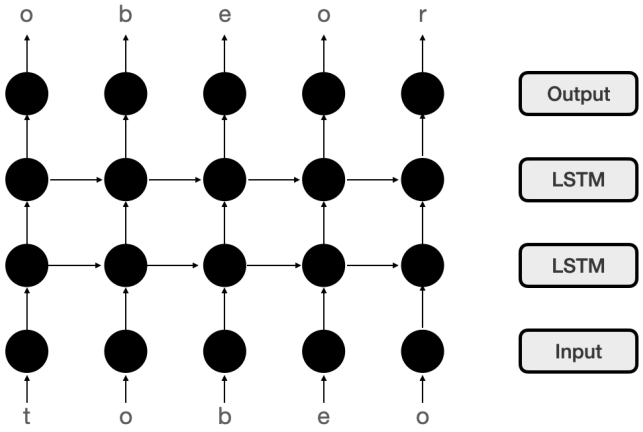

Figure 1: Char RNN Model Architecture

Figure 1 above shows the data flow of the Char RNN. We pass in the characters as one-hot encoded vectors into the input layer, which then go to the two hidden layers built with LSTM cells. Using softmax activation at the output layer, we calculate the probability of each character. The target sequence will be the input sequence but shifted by one. During the training process, cross-entropy loss is usually used with gradient descent. In practice, we will expect the trained model to get a probability distribution for the most likely next character.

3 Methodology

In this project, I based my design on Andrew Karpathy's implementation in torch [1] and post on RNNs [2]. In addition, R2RT's blog [3] and sherjilozair's repo [4] are also extremely helpful and provides great insight on using Tensorflow specific features and functions to have greater control over the model.

4 Experimentation

I'll follow a typical working pipeline in recurrent neural network training: read the data, pre-process the data, build the model, train the network, and finally sample it (i.e., generate new text).

4.1 Model Architecture

My model definition is similar to Figure 1, the main feature is that it has two LSTM layers. Chris Olah's [5] and Edwin Chen's [6] blogs helped me further understood the importance of LSTM cells and why I should use it in lieu of traditional vanilla RNN architecture. In short, traditioanl recurrent neural network suffers from the problem of vanishing gradient, during back propagation, the gradients could become very small through many time steps since the weights are being multiplied again and again.

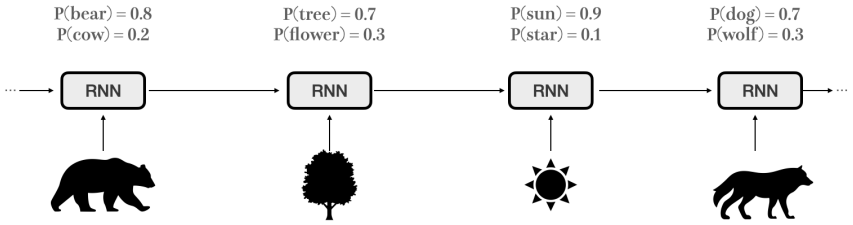

Figure 2: RNN Pipeline Demonstration

Figure 2 shows a vanilla RNN trying to interpret a story happening in the forest. The first image is predicted as a bear and should provide some context to the interpretation of the fourth image which is a wolf. However, the second and the third images are less irrelevant to the context of the forest, and since vanilla RNN has no "long term memory", it is not really helping the model correctly predict the output.

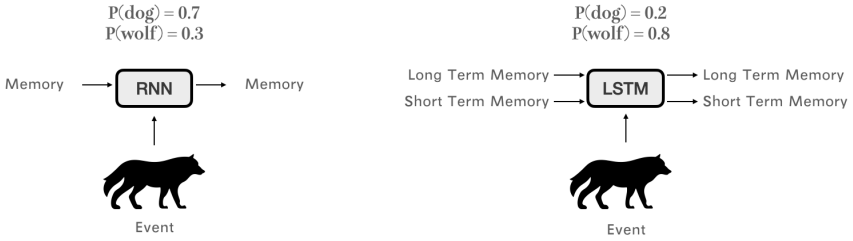

Figure 3: RNN vs. LSTM

For RNN, the memory comes in and merges with the current event, together they predict an output and serves as the input for next iteration; for LSTM, it differentiates "long-term memory" and "short-term memory", in which it forms two streamlines to protect "old memory" more in some sense.

Another focus of this project is to experiment the use of embedding layer in such a natural language processing task. Generally if we train the model in English where the meaning of the word is not as closely tied to its spelling or morphology as language like Chinese or Japanese, character-level embedding is not as good as word-level embedding in terms of helping the model learn the semantic information in the context. Still, character-level embedding does a decant job when it comes to handling uncommon words, linguistic nuance, character dialogue, punctuation and formatting, especially we will be training our model on text like Sherlock Holmes. Modeling wise, it's just another fully connected layer; when we are doing one-hot encoding for each character and matrix multiplication, we are wasting computational resources since there are lots of zeros in the resulting value. Instead of doing matrix multiplication, we directly grab the values from the weight matrix. We will compare and contrast our model with and without the embedding layer. We expect the addition of the embedding layer will greatly increase the efficiency of our network.

4.2 Data Preparation

Our Char RNN model is trained on The Complete Sherlock Holmes. After we load the text file, we one-hot encode the characters by converting them into integers.

For the modified network with an extra embedding layer, we encode the characters as integers and do embedding lookup in the embedding matrix instead. Since it's a fairly large dataset so I believe it's a good practice to pre-process our input text to mini-batches for training. By processing multiple input sequences in parallel rather than one at a time, we take advantage of parallel computing to speed up our training and make our network more efficient.

The initial step involves discarding incomplete text to obtain complete batches, where each batch consists of $N \times M$ characters, N being the batch size and M the number of steps. The total number of batches K can be determined by dividing the length of the array by the number of characters per batch. The total number of characters to keep from the array can then be calculated as $N * M * K$. After we reshape the array to size $N \times (M * K)$, we can iterate through the array to obtain batches, where each batch represents a $N \times M$ window of the $N \times (M * K)$ array.

4.3 Training

The training of a character-wise RNN is supervised. The loss is calculated by comparing the predicted output with the true output, which is the next character in the sequence. To calculate the softmax cross-entropy loss using logits and targets, we first convert the targets into a one-hot encoded format. Subsequently, we reshape the one-hot encoded targets into a two-dimensional tensor with a size of $(M * N) \times C$, where C represents the number of classes or characters in the data. Given that the LSTM outputs are reshaped and passed through a fully connected layer with C units, the resulting logits will also have a size of $(M * N) \times C$. I also implemented a helper function to clip the gradient above some threshold to prevent vanishing/exploding gradient, then using an AdamOptimizer for the learning step. Some other hyperparameters I tuned for the network include batch size (number of sequences running through the network in one pass), number of steps (number of characters in the sequence the network is trained on), LSTM size (the number of units in the hidden layers), number of layers (number of hidden LSTM layers to use), learning rate, and the keep probability of dropout layer. I trained the model as per the advice in Andrew Karpathy's repo. I settled on having batch size set to 100, number of steps set to 100, LSTM size set to 512, number of layers set to 2, learning rate set to 0.001, and keep probability set to 0.5. For the modified network with an extra embedding layer, the embedding size is set to 256. Both networks are trained for 20 epochs; while more epochs might produce better results, I believe it is sufficient for a proof of concept given the limited computational resources on Datahub.

4.4 Result

Final Result (w/o embedding)	Final Result (w/ embedding)
... Farlan," he cried. "It's an information that he has set one of these, and the conclusions that I had no doubt. I have not brought at the stript all where he were that it was not to start in and should show her to–within a station which has been sent in that dark boy, who had suspicions which has been a seroratie, and there he was of his house. I would have a confluence of money as to the side of the maid that, went in the house of an instructive ...	... Farndame and I had seen the possession of them, sir. I thought how was it, I was that I could not have had me thrown outside. I have not the signs of a child. I can hear that the first time that you have already caused all the time, and that in the way there is one thing about this. If I waited they can take the case in the station in the hall and all which I can speak to-day, and I can gather yourself that I would not trust me a little that we may still see the criminal ...

200 iter. (w/o embedding)	600 iter. (w/o embedding)	1200 iter. (w/o embedding)
... Fareserad tothit oee ote tir hire hhae oot ate hed hretat thar teon. aod in ot ee aon. I waed ha oe ane tine whe at aon tot in atee andthe the ...	... Farryould of heverent to sally has bould, were, as waress toly out of out to as is a merasion the marke on the remat wolled to to to ...	... Farrancy; a fout of the matter. That his care. "I am no saight all some him. How, you an insent that I sam at the solise of that there of the ...

200 iter. (w/ embedding)	600 iter. (w/ embedding)	1200 iter. (w/ embedding)
... Fard to that the sard and. The comined who whele he stroud how her had had no though a sent which way to asterding if tather is. "There ...	... Farrren of his pulicam into my horse. I've the time of attaced. He his arting of the matter. As was in his last attation. A corning what ...	... Farryer. "It is this price that if you have say as you will be." "Well, then?" He comes. "It would have how that you ask to any sound. ...

Table 2: Example Results

Table 2 shows some results of the model. Note that the evaluation of a text generation task is qualitative and highly subjective as there is no definitive metric to assess the quality of the generated text. In this case, we evaluate the networks based on visual inspection to see if it produces some meaningful results and perplexity which is the cross-entropy loss.

The first groups shows the final result sampling from both networks, with and without the embedding layer. Both networks seem to produce grammatically correct results at the end with similar training loss.

The second group shows the checkpoints of the network without the embedding layer at 200 iterations, 600 iterations, and 1200 iterations, respectively. At 200 iterations, the model is simply uttering some letters which does not make any sense; at 600 iterations, it starts to produce some words rather than random letters; at 1200 iterations, the model seems to improve less compared to the first 600 iterations but we can see some structure in the text regardless of some made-up words. The third group

shows the checkpoints of the network with the embedding layer at 200 iterations, 600 iterations, and 1200 iterations, respectively. At 200 iterations, the model already has some structure and starts to produce some words, but the model demonstrates less improvement ever since.

It is evident that the network with the embedding layer outperformed the network without the embedding layer around 200 iterations, did slightly better in the next two checkpoints, and the two networks were pretty even in terms of the final result around 6760 iterations.

5 Conclusion

In this project, attempts were made to implement a character-wise RNN in the context of a text generation task. I compared and contrasted the performance of two char RNN architectures (with and without the embedding layer). We may safely conclude that both models are able to produce some meaningful results by the end of training; the model with the embedding layer converges faster than the model without the embedding layer, given that it provides a more efficient representation of the input data compared to one-hot encoding, but does not perform significantly better than the latter one at the very end of training. In early stages of training, the network is still learning the patterns and dependencies in the underlying data so a character-level embedding can accelerate this process, nonetheless, this advantage starts to diminish after thousands of iterations since both networks should have learned some useful features in the data and it is more about fine tuning the hyperparameters. Compared to a word-level RNN, a char RNN tends to produce less fluent and coherent texts, but has the greater potential to capture more fine-grained patterns and dependencies in the text. Still, that level of sophistication requires more training data and computational resources. More detailed examination and experimentation of the hyperparamters can also be the key to boost the performance of the char RNN, which ultimately can be generalized to various tasks.

References

[1] Andrej Karpathy. *Karpathy/Char-RNN: Multi-layer recurrent neural networks (LSTM, gru, RNN) for character-level language models in torch.* URL: https://github.com/karpathy/char-rnn.

[2] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks.* May 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[3] *Recurrent neural networks in Tensorflow II - R2RT.* July 2016. URL: https://r2rt.com/recurrent-neural-networks-in-tensorflow-ii.html.

[4] Sherjilozair. *Sherjilozair/Char-RNN-tensorflow: Multi-layer recurrent neural networks (LSTM, RNN) for character-level language models in python using tensorflow.* URL: https://github.com/sherjilozair/char-rnn-tensorflow.

[5] Chris Olah. *Understanding LSTM networks.* Aug. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[6] Edwin Chen. *Exploring lstms.* May 2017. URL: http://blog.echen.me/2017/05/30/exploring-lstms/.