e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 07 Issue 05
May 2020

GENERATION OF LYRICS USING Recurrent Neural Network (RNN)

Jayendra Kumar[1],
Asst. Prof. , Department of Computer Science and Engineering,
Email: jayendrakumarcse@cvsr.ac.in
K Nikhil Raja[1], R Sai Pooja[1], V Sri Charan[1]
Department of Computer Science and Engineering,
Anurag Group of Institutions, Hyderabad

Abstract. Training RNN character-level language model on lyrics dataset of most popular and recent artists. Having a trained model, we will sample a couple of songs which will be a mixture of different styles of different artists. After that, updating model to become a conditional character-level RNN, making it possible for sample songs conditioned on artist. And finally, concluding it by training the model on lyrics data set. While solving all these tasks, we will explore some interesting concepts related to RNN training and inference like character-level RNN, conditional character-level RNN, sampling from RNN

Keywords: RNN, Character-level RNN, LSTM.

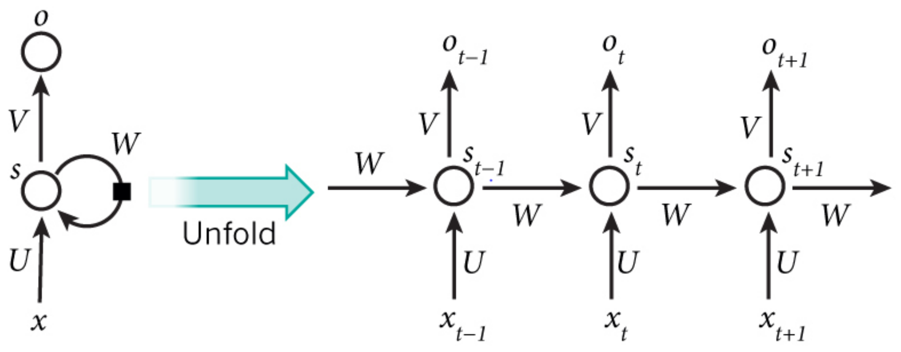
1 Introduction

Our Recurrent Neural Network model is defined on terms of character level. In this, We will be creating a dictionary which will have all the English characters and special symbols, like period, comma, and end of line symbol. So that each character will be represented as one hot encoded tensor. Sequences of characters can be formed from characters. With a fixed probability, we can generate sentences even now just by randomly sampling character after character. It is called as the most simple character level language model. So that, we can compute the probability of occurrence of each letter from our training corpus which means the number of times a letter occurs.

We can now form sequences of characters as we are having the characters. We can generate sentences even now just by randomly sampling character after character with a fixed probability. That's the most simple character level language model. We can compute the probability of occurance of each letter from our training corpus which means the number of times a letter occurs divided by the size of the given dataset and randomly sample letter using these probabilities. Hence, this model is better but the drawback is that it totally ignores the relative positional aspect of each letter. For instance, pay attention on how you read any word: So, initially, you start with the first letter which is usually hard to predict, but as you reach the end of a word you can sometimes guess the next letter. So, when you read any word you are implicitly using some rules which you learned by reading other texts.

2 Recurrent Neural Network:

So, the main idea behind RNN is to make use of sequential information. Unlike in traditional neural network like Feed Forward Neural Networks, we assume that all inputs and outputs are independent of each other. So for many tasks it cannot remember the information that’s a very bad idea. So in order to predict the next word in a sentence you should know which words came before it. RNN are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations that is the previous hidden layer. Another advantage of RNNs is that they have a “memory” which captures information about what has been calculated. In theory RNN can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps. Here is how a typical RNN looks:



RECURRENT NEURAL NETWORK

The above diagram depicts a RNN being unfolded(or unrolled) into a full network. By unfolding, it simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network should be unfolded into a 5 layer neural network that is one layer for each word. The formulas of RNN that govern the computation happening in it are as follows:

- x_t is the input at time step t. For instance, x_1 would be a one hot code vector corresponding to the second word of a sentence.

- s_t is the hidden state at time step t. It is the memory of the network. It is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is non linearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, which is initialized to all zeroes.

- o_t is the output at step t. For instance, if we want to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \mathrm{softmax}(Vs_t)$ that is Softmax function.

There are a few things to be noted in RNN:

- Here the hidden state s_t acts as the memory of the network. s_t captures and stores information about what happened in all the previous time steps. The output at step o_t is calculated solely based on the memory at time t. As briefly mentioned earlier, it is a bit more complicated in practice because s_t typically can’t capture information from too many time steps ago.

- Unlike a traditional deep neural network like Feed Forward Neural Networks, which uses different parameters at each layer like weights and biases should be given to each and every layer whereas a RNN shares the same parameters (U, V, W above) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. So this greatly reduces the total number of parameters we need to learn.

- The above given diagram has outputs at each time step, but while depending on the task this may not be necessary. For example, while predicting the sentiment of the sentence we will only care about the final output, not the sentiment after each word. Similarly, we may not need inputs at each time step. The main feature of an RNN is its hidden state, which captures some information about a sequence of inputs.

What can a RNN do?

RNN has shown a great success in many Natural Language Processing tasks. At this point we should mention that the most commonly used type of RNN is LSTM, which are much better at capturing long-term dependencies than vanilla RNN(Simple RNN) are. But don't worry, LSTM is essentially the same thing as the RNN we will develop in this, they just have a different way of computing the hidden state. Hence, using LSTM it is way more easier to computer the hidden states.

Language Modeling and Generating Text

For suppose, given a sequence of words we want to predict the probability of each word given in the previous words. Language Models allows us to measure how likely a sentence is, which is an important input for Machine Translation (since high-probability sentences are typically correct). A side-effect of being able to predict the next word is that we get a generative model, which allows us to generate new text by sampling from the output probabilities. And depending on what our training data is we can generate all kinds of stuff. In Language Modeling our input is typically a sequence of words (encoded as one-hot vectors for example), and our output is the sequence of predicted words. When training the network we set $o_t = x_{t+1}$ since we want the output at step t to be the actual next word.

Training RNNs

Training a RNN is similar to training a traditional Neural Network. We also use the backpropagation algorithm, but with a little twist. Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps. For example, in order to calculate the gradient at $t = 4$ we would need to backpropagate 3 steps and sum up the gradients. This is called Backpropagation Through Time (BPTT). If this doesn't make a whole lot of sense yet, don't worry, we'll have a whole post on the gory details. For now, just be aware of the fact that vanilla RNNs trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem. There exists some machinery to deal with these problems, and certain types of RNNs (like LSTMs) were specifically designed to get around them.

RNN Extensions

Over the years researchers have developed more sophisticated types of RNNs to deal with some of the shortcomings of the vanilla RNN model.

3 Long Short Term Memory Network(LSTM)

As mentioned above, one critical flaw of RNNs is that they have difficulty learning long range dependencies across time steps. They essentially have a short term memory: RNNs can't access information from a long time ago. Enter the Vanishing Gradient Problem — the gradient of the loss function (values used to update weights) shrinks exponentially during back-propagation and gradually decays to zero. A gradient that becomes too small (and eventually zero) doesn't contribute much to learning. Earlier layers in the neural network can't learn due to very minuscule adjustment of the weights by incredibly small gradients.

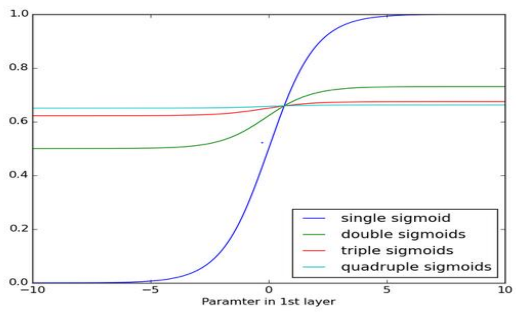


Figure 3: Vanishing Gradient Problem

To solve this problem, we can use LSTMs. An LSTM is a specialized type of RNN that is capable of learning long-term dependencies. LSTMs have the ability to preserve the error that can be back-propagated through time and layers. By holding a more constant error value, they enhance RNNs by allowing them to continue learning over many time steps.

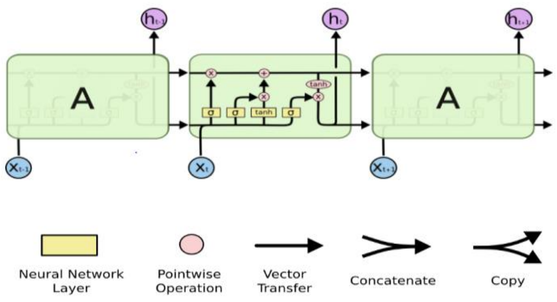


Figure 4: LSTM Network Diagram

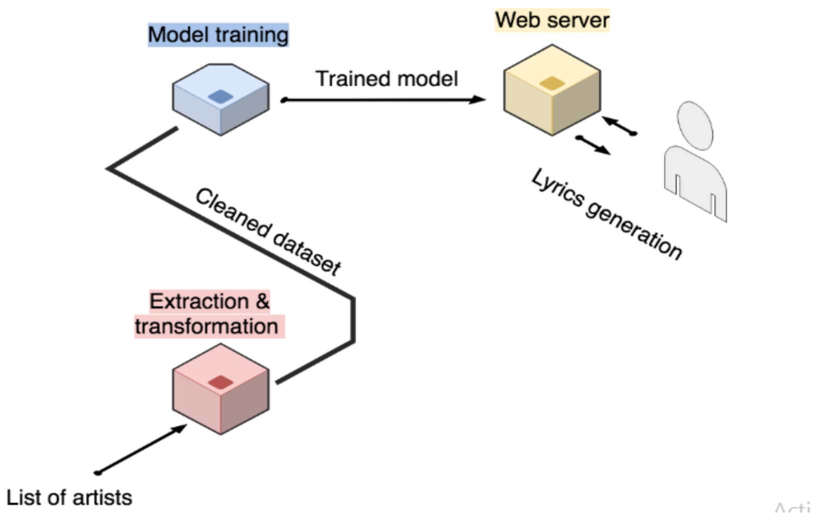
LSTMs are able to carry this out by using gates — tensor operations that can learn what information to add or remove from the hidden state. Each unit in the LSTM network has an input gate, a "forget" gate, and an output gate. The input gate has the ability to assess the value of given information. The "forget gate" has the ability to decide if information should be deleted or remembered for future use. The output gate essentially decides whether information is useful at a specific step. During each step, the gates take inputs as a sigmoid function parameter. The sigmoid returns a value between 0 (let nothing through the gate) and 1 (let everything through the gate). This concept gets applied during back-propagation (updating layer values).

By utilizing these gates, LSTMs are able to selectively choose what information they want to carry forward and what information they want to drop. These gates help control the flow of information within the network and let the error value persist through the network, offering a stark advantage over RNNs.

Proposed System:

Unlike every other generated model, in this process we generated a model using RNN and LSTM networks where the grammatical errors, Spelling mistakes and special symbols were greatly reduced. The time complexity of the generated model is much less when compared to that of the existing model. Another main advantage of having an RNN is its memory.

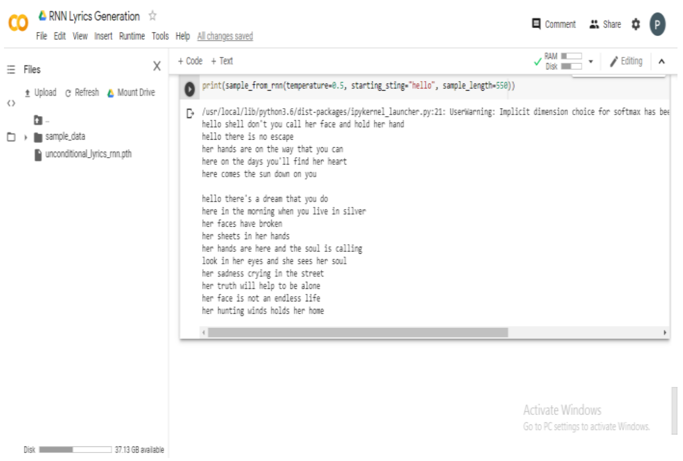
Architecture:



Architecture of the system

4 Experiment:

On successfully training our generated model in the google colaboratory, the effectivness, grammtical errors, spelling mistakes and special symbols were reduced and thus by decreasing the time complexity. The user provides the input string. The input string is sampled with the initial condition and it is trained to generate the lyrics that starts with the input string provided by the user. The generated lyrics are seen by the user.



```
print(sample_from_rnn(temperature=0.5, starting_sting="hello", sample_length=550))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:21: UserWarning: Implicit dimension choice for softmax has bee
hello shell don't you call her face and hold her hand
hello there is no escape
her hands are on the way that you can
here on the days you'll find her heart
here comes the sun down on you

hello there's a dream that you do
here in the morning when you live in silver
her faces have broken
her sheets in her hands
her hands are here and the soul is calling
look in her eyes and she sees her soul
her sadness crying in the street
her truth will help to be alone
her face is not an endless life
her hunting winds holds her home
```

Generated lyrics with input string

Conclusion:

In this we trained simple generative model for text and generates the lyrics based on user given input. Therefore, the aim of this project is to generate lyrics that may help songwriters in their task. Also, it generates lyrics according to certain grammatical rules based on parts-of-speech and statistical constraints.

References

1. https://medium.com/@shivambansal36/language-modelling-text-generation-using-lstms-deep-learning-for-nlp-ed36b224b275
2. https://www.youtube.com/watch?v=BSpXCRTOLJA
3. https://towardsdatascience.com/ai-generates-taylor-swifts-song-lyrics-6fd92a03ef7e
4. https://datascience.stackexchange.com/questions/43176/recurrent-neural-network-lstm-not-converging-during-optimization
5. https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/
6. https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594
7. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/