Deep Learning

Text Generation using Recurrent Neural Networks

April 5, 2019

Lavanya Mandadapu

Contents

1 Introduction

Text Generation is a type of Language Modelling problem. Language Modelling is the core problem for a number of of natural language processing tasks such as speech to text, conversational system, and text summarization. A trained language model learns the likelihood of occurrence of a word based on the previous sequence of words used in the text. Language models can be operated at character level, n-gram level, sentence level or even paragraph level. In this particular work we are going to create a language model to generate natural text by implementing and training Recurrent Neural Networks. Entire code for this work is available at `https://github.com/lavanya463/TextGen_RNN`

2 Description, Pre-processing and Preparation of data

The raw data for this project is taken from projects Gutenberg [1]. A set of five novels has been used for this work. These are crime novels by A. Conan Doyle featuring Detective Sherlock Holmes. The main idea of this work is to implement an RNN, that will be trained on the above mentioned novels, so, it can generate sequence of words that can be interpreted as Conan style of writing. The selection of five novels are because, in order to obtain acceptable results it is recommended to use a file size of at least 2 MB [2].

A good data preparation can result in a well learned Model. For this work, all the novels have been combined as a single novel with a size of 2.5 MB. After reviewing the text, there are a lot of empty lines in the file. When the model is trained with the data containing empty lines, it has produced letters with a lot of spaces. In order to avoid this, empty lines are removed. There are also some special characters in the file, these are not actually part of the novel, rather these are the descriptions of Gutenberg projects provided at the start of every novel. These special symbols have been replaced with a blank space. We also want our network to learn how the upper and lower case letters appear, and also the usage of punctuation, therefore, these are not removed from the data.

Once the pre-processing of the data is finished. The content in the data is splitted and stored into an array of characters. All the unique characters available in this array is stored into an separate array. By doing this way, the data array contains all the examples and character array contains the unique letters in the data, acts as features. Like other learning models, it is good practise to feed numerical training data into the networks. To do this, a dictionary that maps the characters and indices is created. However, in order to predict we need the reverse mapping, so another dictionary with reverse mapping is also created.

In this work, Keras is used to create and train the network, so, actual input form that Keras will accept has been prepared. The input form needed is:

$(number\_of\_sequences, length\_of\_sequence, number\_of\_features)$

The last dimension is the number of the features, in this case the length of the characters array. Next, the length of sequence are the total time steps of our networks. This can be seen as how long we want our model to learn at a time. The first dimension is the number of sequences, which will be obtained by dividing the length of our data by the length of each sequence. In this work, target sequence will have the same length as the input sequence. However, we can also obtain a target sequence with different length.

3 Recurrent Neural Network

The basic models, LSTM, GRU and SimpleRNN have been used in this work. LSTM is known for its ability to deal with long sequences. A three layer LSTM model, dipicted in the figure 1 has produced impressive results, anyhow the networks with more layers are also tested.

In our model, we expect a sequence for the output, unlike the normal neural network. Therefore, we need to make *return_sequences* as true. As we are setting *return_sequences* as true, the output is now a three-dimension vector. When it is given as input into the Dense layer, it gave an error, as the Dense layer only accepts two-dimension input. In order to input a three-dimension vector, a wrapper layer called *TimeDistributed* is used. This layer will help us maintain output's shape, so that we can achieve a sequence as output in the end.

For activation soft-max is used in all the experiments. Soft-max layer is soft version of the max-output layer and also resilient to outliers. A dropout of 0.2 is used after the third layer in LSTM and after second layer in GRU. This can be seen in the figures 1 and 2. Dropout is a regularization technique to reduce the over fitting in neural networks. As the name implies, dropout drops some of the units, both hidden and visible, in neural network.

4 Training and Experiments description

In this section, various experiments performed in a total of 7 categories will be explained.

- **Length of window size**: The length of the window size determines the length of data that our model will learn at a time. Different lengths, starting from 30 to 150 are used. When a too low window size is used, the results are not descriptive, this can be seen because the sequence is being disturbed and model could not able to learn those. A sequence length of above 50 has given more prominent results, for most of the experiments a sequence length of 100 is used. When we see the data, we can observe that Conan style of writing contains a larger sequences. This can be another reason for obtaining bad results when window size is low.

- **Size of the input file**: For all the experiments, combined novel file has been used. But in order to understand the impact of the size of input file, some experiments using only the first novel has been performed. This resulted in producing a sequence with spelling mistakes and unknown words. This is because with less data, the model will not be able to learn better. That is why, it is always good to have larger data for better results.

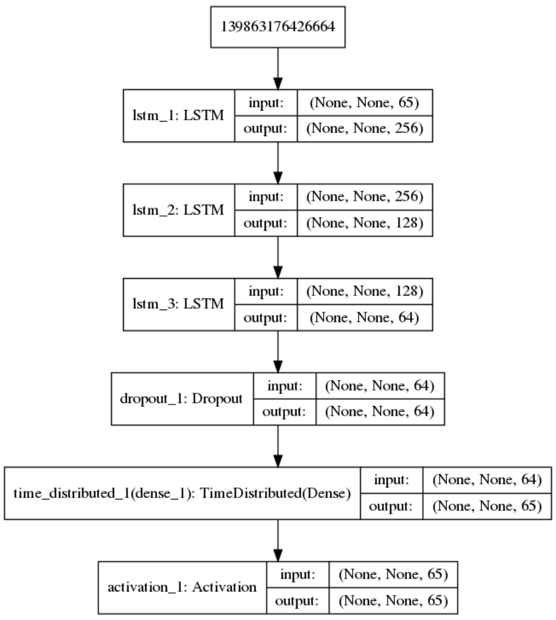

Figure 1: LSTM model used in this work

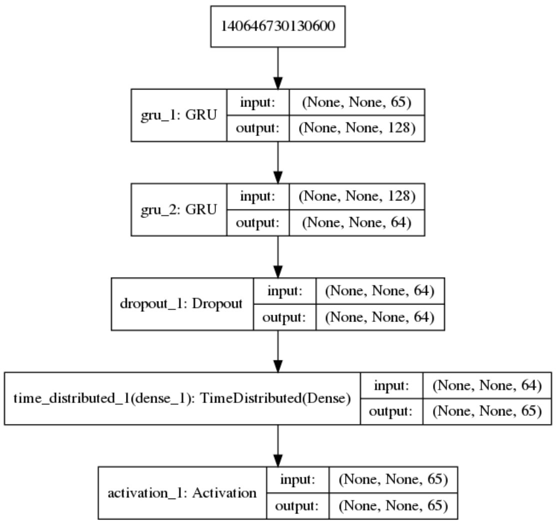

Figure 2: GRU model used in this work

- **The type of RNN**: LSTM, GRU and Simple RNN are tested in this work. A simple RNN cannot remember the sequence for long time due to vanishing gradient problem. Especially, in the case of text generation the model should know what to forget and when to forget and has to remember certain sequences for long time. To allow all of these things, LSTM's (Long Short Term Memory) is generally used. Infact, the best result in this work is obtained with LSTM. Many state-of-art works and comparisions [3] in text generation have obtained good results using both LSTM and GRU. In our case as well, the results obtained using both the models are similar. However, after testing different types of RNN, LSTM has been used for next set of experiments.

- **Number of layers**: An LSTM network with three RNN layers, dropout, dense and activation has given acceptable results. This architecture is dipicted in figure 1. When more layers are tested, they have incorporated complexity into the network. It is not necessary in this particular case as we obtain similar sequences in both cases. However, the network with high complexity takes more memory and is prone to over fitting. When a single RNN layer is tested, it tends to underfit. In order to decide the good number of hidden units for the layers, values with powers of 2 are tested and it has been observed that initial layer with 256 hidden units seems to be reasonable. 128 and 64 hidden units are given to the next layers respectively. When tested with GRU two GRU layers with 128 and 64 is used (figure 2). For simple RNN, similar configuration of the LSTM has been tested.

- **Batch size**: Initially a larger batch size (1000) has been used for experiments. A smaller batch size is taking a long time and when tested with same configuration has resulted in the repetition of the sequence and sometimes has given more or less similar results. Therefore, a larger batch size has been fixed. Another advantage of using a larger batch size is that, network can be trained for the conversations. In this way, it can produce a sequence that can interpret as conversations. Later, in results section, we can see that model has generated text that can be interpreted as conversation.

- **Number of epochs**: A range of 10 epochs to 1000 epochs have been tested. And for lower number of epochs like 10 or less than 60, the results are just some random characters packed together. Between 60 to 300 the output sequences did not make any sense at all, there are number of repetition of the sentences and words with mistakes. This implicitly mean that model needs more training, so it needs more epochs. Above 300 the results appeared to be at least good enough to read and understand. Obviously, the number of epochs have increased the time. But, when tested with 1000 epochs, it has produced fully readable sentences covering the upper and lower case letters and punctuation's.

- **Optimizers**: Adagrad, Adam, RMSprop and SGD with nesterov momentum have been used in order to check whether optimizers can help the network convergence. That is to check whether we can obtain acceptable results within less number of epochs. Inspired by most of the papers on text generation, for this work, optimizer RMSprop with a learning rate of 0.001 is used. SGD with Nesterov momentum has given similar results. In classic SGD we don't compute the exact derivative of our loss function. Instead, we're estimating it on a small batch. Which means we're not always going in the optimal direction, because our derivatives are noisy. So, using momentum which gives exponentially weighed averages can provide us a better estimate which is closer to the actual derivative. Both Adagrad and Adam also performed good. The main advantage with these two are that they made the training faster as they converged earlier.

5 Results and Analysis

In this section we can see some of the generated text produced by the implemented RNN. The prediction is done in a method called as *generate_text*. A random initial point is selected and text will be generated from that point. It is to be noted that due to this randomization, some sentences can start with some strange single letters, but later the words should be in interpretable way.

Best Result : This is one of the best results that has obtained during the course of experiments. Text 1 depicts this result. The result is quite interesting. We can see how the model has learned the trend of writing a capital letter after a full stop. Another interesting thing is how it has generated text one after the other with spaces. We can also observe how the punctuation's are being used. In some sentences the entire text is not making much sense, it even has some spelling mistakes like *corursul* etc. But still, we can acknowledge some of the correct sentences in the text. We can also see typical conversational style in the text.

Text 1:

" Dr. Watson, said he, and I will be a strange way to make a success.

You have not showing there at last?

No, sir.

He was a man of strong character out of the same story of the moor. The theed the last of them which has been some surprise and set at Rose, said the baronet. I have no doubt that I am not received them as to the most of them with the other windows of the south of course of the third from the convict of the corursul price. He was a man of steam landing of the stables of cour"

Some of the other interesting sentences obtained are

" Z When I look at the treasure. I am incestantly about it. For the due finding them staring with a cr "

"_ the secret of the stairs, and like to have her wits turned at the singing and shouting

Bad results: Text 2 is obtained with a simple LSTM network of 10 epochs. We can see that the words are repeated continuously to the length of generation (that we will specify). This is a naive simple replication which assumes that next state is the current state. This kind of assumption is not good in text generation. Text 3 is obtained by training a three layer LSTM network with 100 epochs. Unlike the previous text, now there are sentences which are repeated. In this stage also, the model is not good enough to generate a complete text in descriptive way.

Text 1: *" tore tos tos tos tos tos tos to"*

Text 2: *" the form of the most interesting than the boy was a small station of the moor. I was a small station of the moor. I was a small station of the moor. I was a small station of the moor. I was a small station of the moor."*

All the texts that are generated by different experiments mentioned in previous section are available in the git repository.

6 Conclusions and Future Work

The results obtained by the trained model are appreciable. This can be considered as a base and many more complex scenarios can be learned. for example, we can feed a model with a large amount of C or C++ codes, so the network can learn the structure and can help us in debugging or recommending snippets. Text generation can be used in many applications, like in machine translation or to switch the writing styles.

References

[1] Projects Gutenberg for ebooks:. www.gutenberg.org.

[2] Creating A Text Generator Using Recurrent Neural Network by Trung Tran. https://chunml.github.io/ChunML.github.io/project/Creating-Text-Generator-Using-Recurrent-Neural-Network/.

[3] Illustrated Guide to LSTM's and GRU's by Michael Nguyen:. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.