Cervantes 2.0

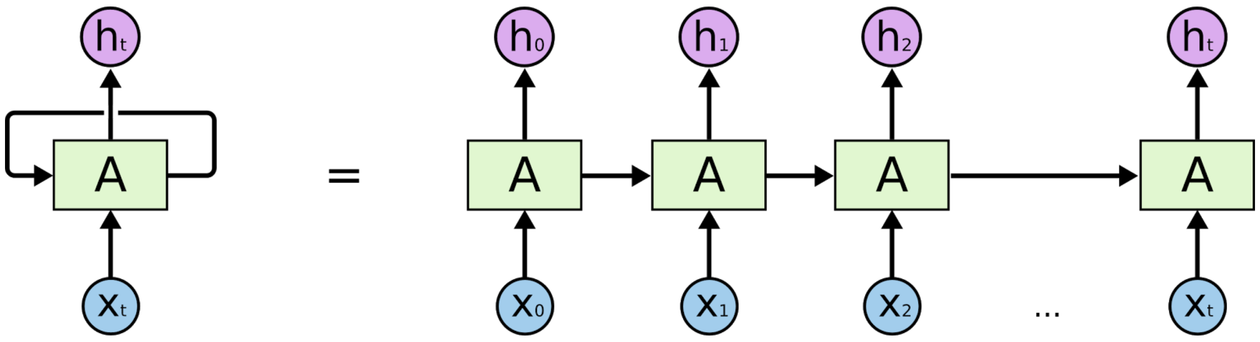
MLND - Capstone Project Report

Background

Deep Learning is a new area of Machine Learning, which has attracted a lot of attention lately due to the amazing results produced by Deep Learning models. With Deep Learning, it is now possible for an algorithm to predict things, classify images (objects) with great accuracy, detect fraudulent transactions, generate image, sound and text. These are tasks that were previously not possible to achieve by an algorithm and now perform better than a human.

In this project we will focus on Text Generation. Text Generation is part of Natural Language Processing and can be used to transcribe speech to text, perform machine translation, generate handwritten text, image captioning, generate new blog posts or news headlines.

RNNs are very effective when understanding sequence of elements and have been used in the past to generate text. I will use a Recurrent Neural Network to generate text inspired on the works of Cervantes.



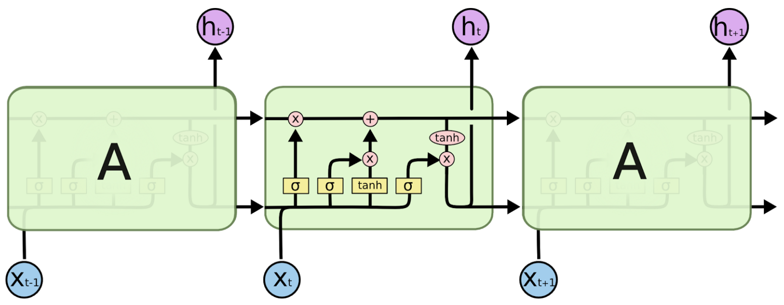
In order to generate text, we will look at a class of Neural Network where connections between units form a directed cycle, called Recurrent Neural Network (RNNs). RNNs use an internal memory to process sequences of elements and is able to learn from the syntactic structure of text. Our model will be able to generate text based on the text we train it with.

Problem

Miguel de Cervantes Saavedra, was a Spanish writer who is regarded as the greater writer in Spanish language. Famous for his novel, Don Quixote, considered one of the best fiction novels ever written.

Unfortunately, Cervantes passed away 500 years ago and he will not be publishing new novels any time soon…. But, wouldn't it be great if we could generate some text inspired on Don Quixote and other novels he published?

To solve our problem, we can use text from novels written by Cervantes in combination with the incredible power of Deep Learning, in particular RNNs, to generate text. Our deep learning model will be trained on existing Cervantes works and will output new text, based on the internal representation of the text it was trained on, in the Neural Network.



LSTM Cell

For our model to learn, we will use a special type of RNN called LSTMs (Long Short Term Memory), capable of learning long-term dependencies. LSTM can use its memory to generate complex, realistic sequences containing long-range structure, just like the sentences that we want to generate. It will be able to remember information for a period of time, which will help at generating text of better quality.

Repository

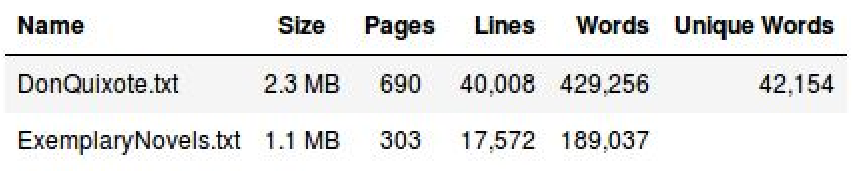
The solution explained below is available on this Github repository: Cervantes - Text Generation

Dataset

To train our model we will use the text from his most famous novel (Don Quixote) and other less known like Lady Cornelia, The Deceitful Marriage, The Little Gipsy Girl, etc. Also, we will not include any Plays, e.g. Numancia, to train our model as it's writing style differs from the novels and we want the generated text to follow the structure of a novel. All the novels are no longer protected under copyright and thanks to the Gutenberg Project, we are able to access all the text of these books.

Even though Miguel de Cervantes native language was Spanish, the text used to train our model will be in English. This is to make it easier for the reader to understand the input and output of our model.

Our Dataset is small as it is composed of only 2 files - Don Quixote and Exemplary Novels with a total size of 3.4 MB. Bigger datasets work better when training an RNN but for our case that is very specific it will be enough. Some additional information of the contents of the files below:



Name	Size	Pages	Lines	Words	Unique Words
DonQuixote.txt	2.3 MB	690	40,008	429,256	42,154
ExemplaryNovels.txt	1.1 MB	303	17,572	189,037	

- Note: Values in the table above will change after preprocessing.

There is some manual preprocessing that we will need to do as the text retrieved from Gutenberg Project contains additional content that is not necessary to train the model, for example:

- Preface
- Translator's Preface
- About the author
- Index
- Dedications
- Footnotes included in Exemplary Novels

Note: The files included in the dataset folder no longer contain the additional content mentioned above.

Dataset Stats

- Unique words: 39229
- Number of chapters: 135
- Average number of sentences in each chapters: 392.7925925925926
- Number of lines: 53162

- Average number of words in each line: 10.975941461946503

Extra Preprocessing

We need to prepare our data for our RNN, lets do some additional preprocessing:

- Lookup table: We need to create word embeddings to facilitate the training of our model.
- Tokenize punctuation: This is to simplify training for our neural network. Making it easy for it to distinguish between mad and mad!

Tokenize Punctuation

We'll be splitting the script into a word array using spaces as delimiters. However, punctuations like periods and exclamation marks make it hard for the neural network to distinguish between the word "mad" and "mad!".

Implement the function token_lookup to return a dict that will be used to tokenize symbols like "!" into "||Exclamation_Mark||". Create a dictionary for the following symbols where the symbol is the key and value is the token:

- Period (.)
- Comma (,)
- Quotation Mark (")
- Semicolon (;)
- Exclamation mark (!)
- Question mark (?)
- Left Parentheses (( )
- Right Parentheses ( ))
- Dash (–)
- Return (\n)

This dictionary will be used to token the symbols and add the delimiter (space) around it. This separates the symbols as it's own word, making it easier for the neural network to predict on the next word.

After preprocessing all data, it is saved in the local folder. This file is available in the repository as well - preprocess.p

Cervantes Neural Network

A GPU is suggested to train the Cervantes Neural Network as text generation takes a long time to train. The building blocks of the Cervantes Neural Network are include in cervantes_nn.py.

Functions included in cervantes_nn:

- get_inputs: Creates the TF Placeholders for the Neural Network
- get_init_cell: Creates our RNN cell and initialises it.
- get_embed: Applies embedding to our input data.
- build_rnn: Creates a RNN using a RNN cell
- build_nn: Apply embedding to input data using your get_embed function. Builds RNN using cell and the build_rnn function. Finally, it applies a fully connected layer with a linear activation.
- get_batches: Creates a generator that returns batches of data used during training

Building our Neural Network

The RNN cells used by our model are LSTM - https://www.tensorflow.org/tutorials/recurrent#lstm , which will process a word at a time and will calculate the probability of the next word when predicting.

To give the model more power, we can add multiple layers of LSTMs to process the data. In our Neural Network we use 2/3 RNN layers of size 256. The output of the first layer will become the input of the second and so on. To achieve this, we use a MultiRNNCell.

In order to prevent overfitting, we use dropout between the LSTM layers.

To calculate our models loss, we use [tf.contrib.seq2seq.sequence_loss] (https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/sequence_loss), which is the weighted average cross-entropy (log-perplexity) per symbol.

To optimise our model we use AdamOptimizer, which is a method for stochastic optimization - paper.

Training our Neural Network

To train our model, we first need to create a word embedding of our input. The word IDs will be embedded into a dense representation before feeding to the LSTM. This allows the model to efficiently represent the knowledge about particular words.

The embedding matrix will be initialized randomly and the model will learn to differentiate the meaning of words just by looking at the data.

```
embedding = tf.Variable(tf.random_uniform((vocab_size, embed_dim)),
name="embedding")
    embed = tf.nn.embedding_lookup(embedding, input_data)
```

Hyperparameters

The following parameters are used to train the Neural Network:

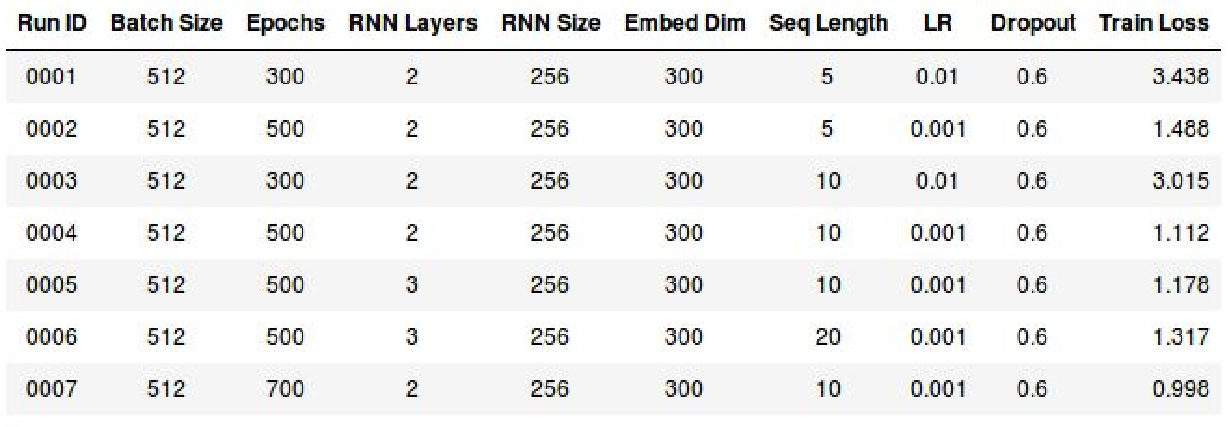
- batch_size: The number of training examples in one pass.
- num_epochs: One pass of all the training examples.
- rnn_layer_size: Number of RNN layers
- rnn_size: Size of the RNNs.
- embed_dim: Size of the embedding.
- seq_length: Number of words included in every sequence, e.g. sequence of five words.
- learning_rate: How fast/slow the Neural Network will train.
- dropout: Simple way to prevents an RNN from overfitting - link.
- show_every_n_batches: Number of batches the neural network should print progress.
- save_every_n_epochs: Number of epochs the neural network should save progress.

Parameters used in our best performing model:

```
# Batch Size
batch_size = 512
# Number of Epochs
num_epochs = 700
# RNN Layers
rnn_layer_size = 2
# RNN Size
rnn_size = 256
# Embedding Dimension Size
# Using 300 as it is commonly used in Google's news word vectors and the
GloVe vectors
embed_dim = 300
# Sequence Length
seq_length = 10
# Learning Rate
learning_rate = 0.001
```

Training results

The table below captures the results of training the Cervantes Neural Network with different hyperparameters:



Run ID	Batch Size	Epochs	RNN Layers	RNN Size	Embed Dim	Seq Length	LR	Dropout	Train Loss
0001	512	300	2	256	300	5	0.01	0.6	3.438
0002	512	500	2	256	300	5	0.001	0.6	1.488
0003	512	300	2	256	300	10	0.01	0.6	3.015
0004	512	500	2	256	300	10	0.001	0.6	1.112
0005	512	500	3	256	300	10	0.001	0.6	1.178
0006	512	500	3	256	300	20	0.001	0.6	1.317
0007	512	700	2	256	300	10	0.001	0.6	0.998

For a detailed view of the training loss, checkout the training logs included with the project.

Training Loss

We can see that a learning rate of 0.01 is too large to train our Neural Network. When we trained it with 0.01, we were never able to achieve a train loss < 3. Another indicator of this is that the learning plateaus in both runs (0001, 0003); in 0001 it plateaus at around epoch 100 and in 0003 at around epoch 180.

The training loss improved when we use a learning rate of 0.001. The lower learning rate improves our Neural network performance by -2.0. As the training is not plateauing, we are also able to train it longer. This is why we increase the epochs of run 0002 to 500.

Sequence Length

Our basic RNN was trained with a sequence length of 5. The sequence length, is the number of words to be included in every sequence.

We can see an improvement when increasing the sequence length to 10. This means that our RNN will use a longer sequence to train our Neural Network. Which ends up improving significantly the quality of text generated by our network.

With a sequence length of 5, our text didn't made much sense, the sentences are short and the paragraphs are not well structured.

When using a model trainer with sequence length of 10, we can notice that the text makes much more sense, and the quality of the sentences and paragraphs improves significantly.

Train some more

Our best result in the the first 6 runs was run 0004. If we train our network longer with the same parameters, we achieve a train loss of less than 1.

Conclusion

Even though our train loss for the last run is less than 1, we can see in the samples below that the text generated by run 0004 and run 0007 models are similar is several ways:

- They are both able to open and close quotations
- The text makes more sense when compared with run 0001, which is expected as the sequence length (10) used to train both models is longer than run 0001 (5)
- Paraghaps are well formed.
- Sentence length is similar and close to the average sentence length.

Text Samples:

- Run 0004: “ Senor,” said Sancho,” I mean to know from this perilous journey in the ugly which has been bound; “ At any rate, Dulcinea,” replied the actor
- Run 0007: Quixote or cost him his squire, unless indeed his wife might follow him Don Quixote bade Sancho he settled three days with open his heart in fixing his affections should comply with Preciosa

Generated Text Samples

Run 0001
Quixote,” and my profession abideth about her Majesty then, the cloth was frantic, that it is so hast been, that she has not spread of showing and tear resting with him, for the persuasion, leaving him by force or get up by full great achievements of what book, Leaves the pastime?”

” I would lay myself upon him with a burnished hand to the waist. The thieves laughed at the other little intelligence, and where the redress, in the service of some dwarf, and we were in the line of mutual Sancho, this, I have not such matters of us with I not heard by which is the best world, Non when the bano seated myself and Sancho Panza asked the same of his story, they were excellent wife at once, and here the green intentions of the Judge so cautiously gave her. The blush, and heard them came to over a payment with shepherd, laying round it he ordered her, and

Run 0004
Quixote,” and am the idea thou wilt give more good quickly to see now thou hast won the good thing, as I have told them not.”

” Senor,” said Sancho,” I mean to know from this perilous journey in the ugly which has been bound; for it is that? What are it in me; but the knight-errant should come free?”
” At any rate, Dulcinea,” replied the actor held in nonsense of your wife, master the devil who is so generous that I can go to the house of Luscinda, Preciosa, and more will by everything the wrong the beauty itself ought to wash its course.

” What could mean be” The greatest who is long in his own behalf so long as I have chosen to Uchali, who shall not go to defend your ass will run wrong, for the mercy I was now in a coach that, after having sold their riches and language that Preciosa asked

Run 0007
Quixote or cost him his squire, unless indeed his wife might follow him or with great respect. Their master would be, the blind parents describes, or look free, and all the rest with your life shall be imagined; the gipsy had rule come to his heart. The extreme day I have heard of her good; in the moment of this my house and soul to find the paper; though, as it were, we believe it is because they who are dead, and bind me to undergo the exertion thou great desire, for indeed he said to him cannot read it aloud.

Don Quixote bade Sancho he settled three days with open his heart in fixing his affections should comply with Preciosa, and to keep secret whatever will be no wish on, as it was queens as his master had heard him, he strove to sing him, maintaining the sun favouring befall), which was not of it, both of them, served in her courtesy for the

References

- NLP Tokenization - https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html
- Vector Representations of Words - https://www.tensorflow.org/tutorials/word2vec#motivation_why_learn_word_embeddings
- Recurrent Neural Networks - https://www.tensorflow.org/tutorials/recurrent
- Alex Graves - Generating Sequences With Recurrent Neural Networks https://arxiv.org/pdf/1308.0850.pdf
- Christopher Olah - Understanding LSTM Networks http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Prasad Kawthekar, Raunaq Rewari, Suvrat Bhooshan - Evaluating Generative Models for Text Generation - https://web.stanford.edu/class/cs224n/reports/2737434.pdf
- Ilya Sutskever, James Martens, Geoffrey Hinton - Generating Text with Recurrent Neural Networks - http://www.cs.utoronto.ca/~ilya/pubs/2011/LANG-RNN.pdf