University of Dallas in Texas

6V99 Spring 2020

NLP

Final Project – Due on April 25th, 2020

Dr. Elkhodari, Gasan

Student Name: Himaja Barla

**Use Case** - **Natural Language Generation in Python**

Natural Language Generation is the process of producing meaningful phrases and sentences in the form of natural language. In its essence, it automatically generates narratives that describe, summarize, or explain input structured data in a human-like manner at the speed of thousands of pages per second. NLG capabilities have become the de facto option as analytical platforms try to democratize data analytics and help anyone understand their data*.*Close to human narratives automatically explain insights that otherwise could be lost in tables, charts, and graphs via natural language and act as a companion throughout the data discovery process.

**Methodology**

Four sub-use cases have been used to properly interpret different Natural Language Generation tasks.

A brief description of the sub-use cases is as follows:

* Generating short sequences using recurrent neural networks and build a model to generate innovative baby names – *Baby name generator using simple RNN and Keras*
* Long Short-Term Memory network to generate longer texts – *Text generation in the author’s style of writing*
* Encoder-Decoder architecture for machine translation – *Neural Translation model to translate English sentences to French*
* Encoder-Decoder architecture for sentence auto-completion – *Generate Natural Language autocomplete sentences*

The description of each of the use cases is as follows:

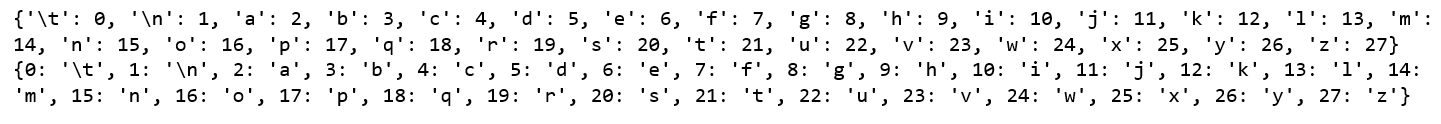
***Use Case1:*** *Baby name generator using simple RNN and Keras* – Generating short sequences using recurrent neural networks and build a model to generate innovative baby names

RNN’s are specially designed to make use of the ordered information present in Sequential data. Feed forward Neural Networks accept a fixed size input and produce a fixed size output using a fixed number of hidden layers in between. They assume that the input samples are independent of each other. They are a bad choice for sequential data. If the next character in the word is to be predicted, we better know which character came before it in the sequence. RNN’s addresses this concern. They are called recurrent because they perform same computations for every element in the sequence and the output depends on whatever elements came before. A recurrent neuron produces an output along with hidden state. A state can be thought of as memory of the network. It consolidates all the history information from the input data. The history and the current input are used to predict the output. The current input and the hidden state from previous time-step serve as the input for this time-step. Suppose we want to generate new names from scratch. The idea is to generate the next character given the current character and the history as input. RNN’s can learn future from the past.

*Corpus Used:* names data file which contains the list of all names is used to generate innovative baby names.

*Data Preparation:* Data set has only one column which has the list of all possible names. No preprocessing steps were required given the clean textual data.

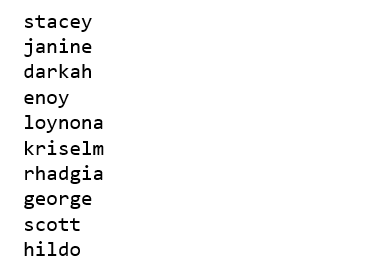
*Description:* The entire content in the dataset is encoded into numeric values, given the machine earning models only accepts numerical inputs. Character to integer and integer to character mapping is done to assign the numeric values to the characters and vice versa.



RNN model with Keras – A sequential model is created and an RNN layer of 50 units is added with return sequences set to true to make sure RNN outputs a sequence and not a single vector. The output layer is then passed to a dense layer with softmax activation to generate the output. The model is then compiled using ‘categorical cross-entropy’ loss and ‘adam’ optimizer.

The RNN model is then trained and keras is used to fit the trained model.

*Testing:* The model generated the following output for the baby names



RNN’s are not very effective for longer sequences so in the next use we work with LSTM to overcome the limitations of recurrent neural networks when input sequences span long intervals.

***Use Case2:*** *Text generation in the author’s style of writing* – Long Short-Term Memory network to generate longer texts

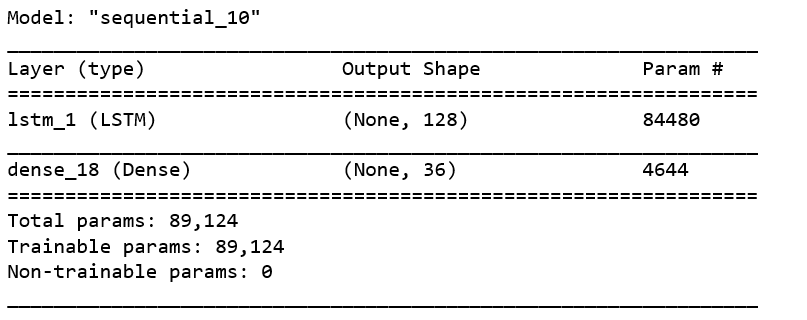
Simple RNN’s struggle while modeling long-sequences. So, LSTM’s have been used which doesn’t suffer from vanishing and exploding gradient problems and as a result can handle longer sequences. LSTM’s are specially designed to handle long-term dependencies. RNN’s just have one state to capture historical information. This is not sufficient to capture long term dependencies. So, LSTM’s use an additional state to capture long-term dependencies. Thus, they have two states one to capture long-term history and other to capture the short-term history. These states are called the hidden and cell states, respectively. At each time-step, LSTM will accept the input and the hidden and cell states from the last time-steps. Depending on the input data it may forget or add new information in the hidden and cell states and pass it to the next time-step. The hidden state can also be used as the output if needed. To understand how effective LSTM’s are to capture long-term dependencies let’s go through this case study where we’ll generate text that imitates Shakespeare’s unique style of writing based on a dataset of selected literary works of Shakespeare.

*Corpus Used:* Shakespeare dataset has selected literary works of Shakespeare. This dataset is used to generate a text that imitates Shakespeare’s unique style of writing

*Data Preparation:* Data is loaded into spacy and is convert into lower case. All the punctuations are removed. Also, tokenization is used for data preprocessing.

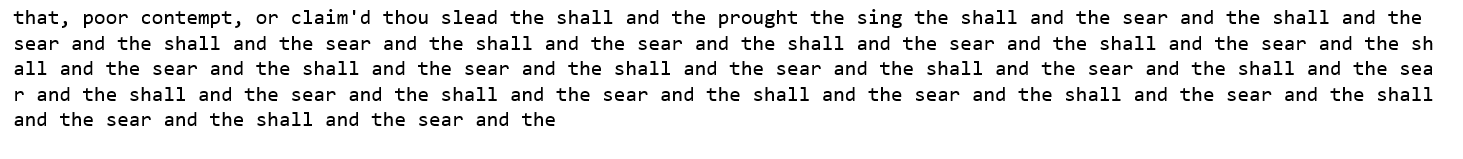
*Description:* The entire content in the dataset is encoded into numeric values, given the machine earning models only accepts numerical inputs. Character to integer and integer to character mapping is done to assign the numeric values to the characters and vice versa.

A sequential model is created and an LSTM layer of 128 units is added as an input layer and a Dense layer is added as an output layer. The model is then compiled using ‘categorical cross-entropy’ loss and ‘adam’ optimizer



The model is then fit and generate\_text function is used to generate the new text that imitates Shakespeare’s style of writing.

*Testing:* The model generated the following output:



Increasing the sample size and the number of iterations will lead to better predictions.

***Use Case3:*** *Neural Translation Model to translate English sentences to French* – Encoder-Decoder architecture for machine translation

There are applications where we need to generate full meaningful sequence given another sequence as input which cannot be done by RNN or LSTM. So, we use sequence to sequence models to generate a whole new sequence given another sequence as input. They aim to map a fixed length input with fixed length output where the lengths of input and output may differ. Some applications of sequence to sequence are translations from one language to another, automated question answer systems, text summarization, grammar correction etc. All these tasks are special cases of natural language generation. We work on translating text data from English to French using Encoder-Decoder Network.

*Corpus Used:* The dataset is in the form of a text file which contains English sentence followed by French sentence separated by a tab. It is extracted from <http://www.manythings.org/anki/>

*Data Preparation:* Data is loaded into spacy and is convert into lower case. All the punctuations are removed. Also, tokenization is used for data preprocessing.

*Description:* The entire content in the dataset is encoded into numeric values, given the machine earning models only accepts numerical inputs. Character to integer and integer to character mapping is done to assign the numeric values to the characters and vice versa.

Encoder-Decoder architecture consists of two separate neural networks.

Encoder:

* The encoder accepts the input sentences and summarizes the information in its state vectors
* Encoders are implemented using LSTMs and here the states refer to the cell and hidden states from the LSTM layer
* During training, the encoder learns these states from data. Intuitively, we can think of the states as a summarization of all the useful information from the input
* The encoder output is ignored

Decoder:

* The decoder is also implemented using LSTM's and the initial hidden and cell states are initialized to the encoder final states
* Intuitively, the decoder gets to know about all the useful information from the input from these states. the decoder uses this information to generate the output
* The final decoder states are ignored
* The output of the decoder is compared with the target sequence to calculate the error which is minimized during the training process by updating the weights of the encoder and decoder networks
* The input to the decoder at each time-step is the predicted output from the previous time-step as usual
* However, during training, the input to the decoder at each time-step is the actual output from the previous step instead of the predicted output
* This technique is known as 'Teacher-Forcing' which helps the model to learn faster

Now that we know how the encoder and decoder works, let's apply this to the case study of the machine translation

Encoder

* The encoder will accept the English sentences, the number of time-steps in the encoder will be the length of the English sentences
* As we have sentences of varying lengths, the length of the longest English sentence can be taken as the step-size
* Shorter sentences can be padded with zeros at the end
* Encoder summarizes all the necessary information from the English sentences in its state vectors which are then passed to the decoder
* Encoder outputs are ignored

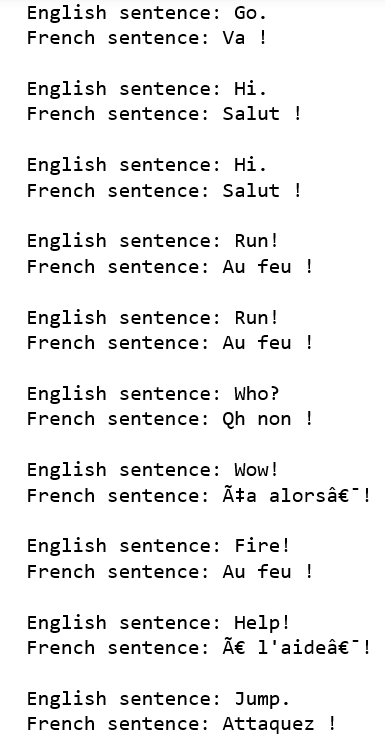
Decoder

* The initial states of the decoder are the final states from the encoder
* The encoder consolidates all the useful information from the English sentences in its state vectors which are needed in the decoder to generate the translated French sentence
* The decoder inputs during the training are the French sentences because of teacher-forcing
* Decoder outputs are the translated sentences
* Decoder states are ignored
* Similar to the encoder, as we have sentences of varying lengths, the number of time-steps in the decoder can be set to the length of the longest French sentence

There are two inputs to the network - English sentences for the encoder and French sentences for the decoder. The targets are the French sentences.

All these vectors are 3-Dimensional - the first dimension being the number of sentences, the second being the number of time steps which is the length of the longest English or French sentence and the third being the length of the one-hot encoded vector for the characters which is the respective vocabulary size. Model is compiled and fit to generate the required output.

*Testing:* The following is output generated:



Model complexity can be increased to improve the model performance by increasing the number of hidden layers in the encoder and decoder and by increasing the nodes in each layer.

***Use Case4:*** *Generate Natural Language autocomplete sentences* – Encoder-Decoder architecture for sentence auto-completion

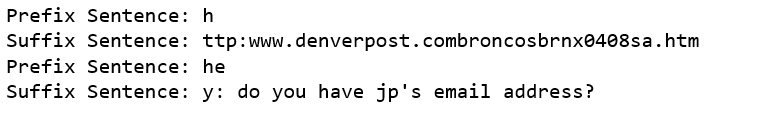
Encoder-Decoder network is used for sentence auto-completion which can be found in popular email applications like Gmail and instant messaging applications like WhatsApp. Given an incomplete sentence as input, sentence auto-completion is the task of generating a new sequence that can serve as possible ending of the input sentence. So, the input is basically a prefix of the whole sentence and the output is the corresponding suffix.

*Corpus Used:* Real email messages from the Enron email dataset to build the model to autocomplete emails. The original dataset contains whole email message including the headers, sender, recipients and body. We only consider sentences from the email body.

*Data Preparation:* These are preprocessed so that they can be used to train an encoder-decoder model. As a first step, we need to find out the input and target sequences from the original sentences. We can divide each sentence into two at each character position. This will generate one prefix and the corresponding suffix for each character in the sentence. We can create two empty lists to store the prefixes and suffixes. Then we can iterate over each character position in each email message to break the sentence into one prefix and one suffix. We also need to append the start and end token to the suffixes. Finally, we can append them to the existing lists of prefixes and suffixes. With this simple trick we reduce the sentence autocompletion task to sequence to sequence generation task.

*Description:* The entire content in the dataset is encoded into numeric values, given the machine earning models only accepts numerical inputs. Character to integer and integer to character mapping is done to assign the numeric values to the characters and vice versa. Since we broke this sentence autocompletion task to sequence to sequence generation task, now the problem is similar to use case 3.

*Testing:* The following is the output generated:



Accuracy of the model can be improved by increasing the model complexity, training for more epochs and training with bigger data set.

**Python Code**

Jupyter notebook has been attached for reference

**Conclusion**

One of the hardest problems in the area of Natural Language Processing and Artificial Intelligence is automatically generating language that is coherent and understandable to humans. Teaching machines how to converse as humans do, falls under the broad umbrella of Natural Language Generation. In this project, an overview of this important and thriving area, traditional approaches, statistical approaches and also approaches that use deep neural networks have been considered. A comprehensive review towards building simple RNN using Keras package to generate innovative baby names, text generation in author’s style of writing, Neural translational model to translate English sentences to French, sentence autocompletion have been provided which are important applications of natural language generation. Notably, three important areas of further research towards building more effective dialogue systems have been identified: 1) incorporating larger context, including conversation context and world knowledge; 2) adding personality in the NLG system; and 3) overcoming dull and generic responses that affect the quality of system-produced responses.

**Future Work**

In recent times, advanced concepts like attention and transformers emerged are being heavily used to understand language data