Quote Generation with LSTM

Recurrent Neural Networks

Group 8:

Jasmine Thai

Keenan Flynn

Martin Yap

Hoyun Yoon

Professor Albishri

Python Deep Learning

08 May 2022

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

I. Introduction

People are not machines who can churn out new novels, plays, articles, and quotes 24/7 for the whole duration of their lives. For each generation, there will be great people who will leave their mark on the world and inevitably part ways onto the great beyond. As unique people that were impacted by the happenings of that singular moment in time, it is impossible for future generations to perfectly mimic their works and they only draw inspirations. It is a spectrum between uniqueness and resemblance of old works that people allot themselves to whether they know it or not.

II. Problem Statement

Eventually, people will find themselves with Writer's block where all their favorite author's books have been read and the faucet they drew inspiration from runs dry. With an ever-ravenous population hungry for fresh content, we have decided to propose a remedy to writer's block and something whimsical for the bored. In addition, burnout can be a problem for people who need to produce lots of content for their media community. Creation of quotable text for use in media settings, boredom with already-existing quotes, and lack of fresh creative quotes are problems our users might face.

III. Solutions

Our project is in the domain of NLP but is better described as Natural Language Creation. This is a project that employs text generation using an LSTM neural network. The LSTM will be trained on a dataset of quotes and the interface will output a randomly generated quote. The user will have the option to enter a starting string that the LSTM will generate around. They will have an additional option of turning up the temperature of the model. Toggling the temperature will make the output more random. What separates us from our competitors is our option of adjusting the temperature.

IV. Dataset and Preprocessing

A. Datasets

There are two datasets that were used to train the quote generator model. Both datasets provided good amounts of quotes compared to other datasets found online. These datasets were also easy to integrate into the python code and preprocess into datasets that the model would be able to understand.

The quotes dataset was created by the user nelsonic and made available on the GitHub website to be easily accessible. This dataset is a curated list of quotes that are used to inspire action and is contained within a single JSON file so that they can be easily used for any project. The dataset includes approximately 1,600 entries, each with an inspirational quote and the respective author.

The Quotables dataset was created by the user alvations and is publicly available on the GitHub website. The Quotables dataset is a single text file where each entry has an author and quote. This dataset contains 39,269 quotes that were made by 3,112 people. For the project, the Quotables dataset had to be converted to a JSON file to allow us to merge it with the quotes dataset and to make it easier to process the quote information. The process to convert the text file into a JSON file was by converting the text file into an excel file and labeling the columns for the data to identify the authors and the quotes. An online converter was then used to convert the excel file into a JSON file that could be used with the project code and merged with the quotes dataset JSON file.

B. Preprocessing

To preprocess the datasets, incomplete entries were removed from them by identifying if both the author and text fields for an entry were filled. Next, the objects in the datasets were converted into strings to allow the model to understand the entries and distinguish between the author and text of the entry. Since the quotes in the datasets vary in length, a definite sequence length needed to be identified before using the quote sequences to train the model. The lengths of all the quotes were found and then used to identify the appropriate quote length that would cover most of the quote use-cases. In the case of the datasets used, a quote length of 1000 was found to be optimal for covering most of the quote use-cases.

The next step for preprocessing the datasets was to convert the texts to numbers since recurrent neural networks do not understand human language. To accomplish this task, the Tokenizer method from the Keras preprocessing text library was used. A stop-character for both the quote and author were also defined to help identify the end of the quote and author, which is then later used to help identify the end of quotes and authors being generated. The stop-characters and the datasets were then fitted to the tokenizer using the fit on texts method. After fitting the tokenizer, we were able to define the vocabulary size by finding the length of the word count in the tokenizer and adding one to the count.

Next, the datasets had to be vectorized using the tokenizer's texts_to_sequences method. These sequences were then padded using the pad_sequences method to have them all be the same length and were padded with the stop characters. By padding the sequences with the stop characters, the model is expected to learn that whenever it sees the stop characters, it means that it is the end of the quote or author sequence. In the next part, the datasets needed to be converted from numpy arrays to TensorFlow datasets using the from tensor slices method in the TensorFlow data dataset library. For each of the sequences, they needed to form input and target texts by duplicating and shifting them. After

forming the input and target texts, the datasets were split into batches that could be used to feed into the model for training.

V. MODEL

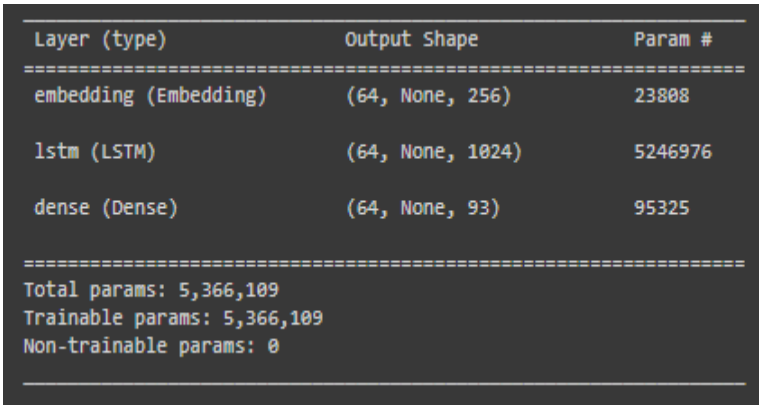
Text generation falls into the deep learning category of generative models. Generative models take samples as training data and try to generate new samples from the same distribution. This falls into the realm of unsupervised learning.

Recurrent Neural Networks are a category of deep neural networks that weighs the sequence of an input. This is particularly important in the realm of natural language processing where ideas are transferred over time in the form of consecutive sentences. A character-level recurrent neural network can learn the concepts of grammar and punctuation given enough training data and time. It is also able to learn distinct sections of a textual input, in our case this will be a quote and the author of the quote.

In this project we propose the use of an LSTM. LSTMs have a stateful memory which allows them to be particularly useful in generating text in long sequences.

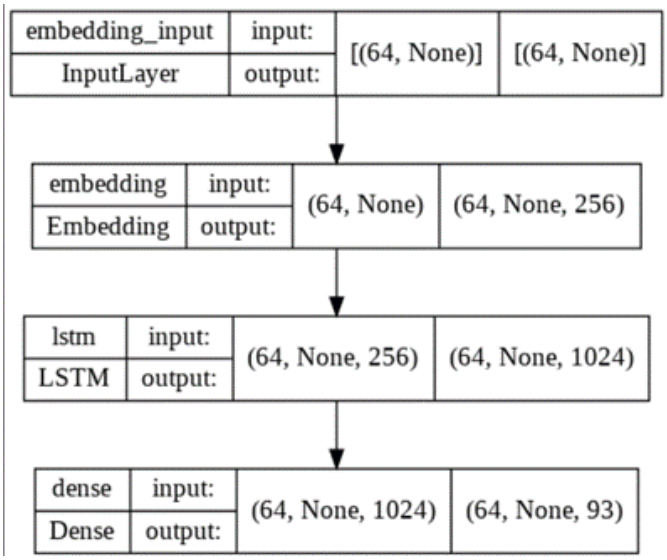
To build our generative LSTM, we implement a build_model() function which creates a Keras Sequential model. The build_model() functions create a neural network with the following layers:

- Embedding layer: This layer is an input layer that creates a trainable lookup table which will map the vectors that represent each character into a vector with the specified number of embedding dimensions.
- LSTM layer: Recurrent layer which contains RNN units that memorize the dataset. This layer looks up the embedding for each character and generates a logit which predicts the likelihood of the next character.
- Output Dense layer: Has 93 outputs. This layer will output the character that the LSTM layer predicts as having the highest probability of being next in the sequence.



Layer (type)	Output Shape	Param #
embedding (Embedding)	(64, None, 256)	23808
lstm (LSTM)	(64, None, 1024)	5246976
dense (Dense)	(64, None, 93)	95325

Total params: 5,366,109
Trainable params: 5,366,109
Non-trainable params: 0



The build_model() function takes in the following inputs:

- Vocabulary size: The LSTM that we are building is a character-level network. That means that the vocabulary size is actually the number of distinct characters in the input set. We have 93 distinct characters in our input set.
- Embedding Dimensions: This is the number of dimensions that each embedding is represented by. We chose an arbitrary number of 256 as our number of dimensions.
- RNN units: The number of cells which do the memorization in the LSTM layer.
- Batch Size: The number of quotes that are trained on at a single time.

Finally, our LSTM model was compiled with a sparse categorical cross-entropy (with logits) loss function and an Adam optimizer. We use sparse categorical cross-entropy as there are 93 possible outputs.

Our model was trained over a series of days. Due to the nature of a large dataset and a high number of parameters, the training process was slow. To combat this slow training, a series of callbacks were implemented to assist the team. The Early Stopping callback and the Model Checkpoint callback were applied, and the checkpoints were stored in a Google Drive folder that automatically connected each time the program was run.

Because there is no validation set for this type of problem, the LSTM was solely evaluated on Loss. This loss was reduced to XXX before the team was satisfied with the outputs of the model.

The weights of the trained LSTM model were saved and exported into a separate application python file. This python file implements a lightweight version of the LSTM model which takes a singular input only and will be run when the user enters an input string. The tokenizer which created the word embeddings also needed to be imported into the application file. This was done with the Pickle library which creates a .pickle file which can copy the Keras tokenizer to be read in a different file.

The LSTM model generates text through a generate_text() function in the application python file. This function takes the user inputs of starting

string, temperature, and input length. The starting string is a user input that sets the theme for the generated text. The temperature is a measure of how random the output will be. A high (~1.0-1.5) temperature generates more random text than a lower temperature (~0.5-1.0). The input length is the max length of the output string.

VI. Applications

A. Remedy for Burn-Out

For a writer or creator to produce a daily feed or article every day, it is a job for them, but also a lot of stress. If their stress continues to build up, they will eventually burn out and, in extreme cases, even commit suicide.

The pain of creation for those who have to create these new ideas and concepts tends to be personal and can be hard for those who have not experienced it to understand.

Therefore, our web application will be of some help to those who are suffering from such creative pains. When writing on a topic, we hope that it will help them in their activities by using our application to find or create relevant quotes.

Also, people who need to create a large amount of content will be able to create it by making various quotes using our web application. In addition, it will also shorten the time to create content using our app and eliminate endless worries about what to create.

B. Remedy for boredom

For writers and content creators, boredom is one of the worst words to hear. Readers always crave new things and want to see something different. Of course, writers and creators know those needs, so creating something new is like homework every day for them. Worst case scenario, stale content leads readers or viewers to unsubscribe and never come back.

In the boredom of creating similar texts that are repeated every day, our application will be a great tool to create new ones to solve their homework.

By creating quotes through simple keywords and settings, you will get new things that you did not know before. In doing so, the reader will be able to satisfy their craving for something new and entice readers to revisit.

In addition, our application does not simply generate random quotes, but uses keywords to create quotes with desired content, so it will be fresh and useful to writers or content creators who want to create specific content.

References

1. BOMMAKANTI, HARI CHARAN “Novel Quote Generator using LSTM” Kaggle, 8 April 2020, https://www.kaggle.com/code/haricharanbk/novel-quote-generator-using-lstm

2. Python Flask Tutorial: Full-Featured Web App Part 1 - Getting Started https://www.youtube.com/watch?v=MwZwr5Tvyxo

3. Trekhleb, Oleksii “Generating cooking recipes using TensorFlow and LSTM Recurrent Neural Network: A step-by-step guide” Towards Data Science, 18 June 2020 https://towardsdatascience.com/generating-cooking-recipes-using-tensorflow-and-lstm-recurrent-neural-network-a7bf242acad3

4. Understanding Word Embeddings from scratch | LSTM model https://towardsdatascience.com/word-embeddings-and-the-chamber-of-secrets-lstm-gru-tf-keras-de3f5c21bf16

5. Weber, Ben “Deploying Keras Deep Learning Models with Flask” Towards Data Science, 31 July 2018 https://towardsdatascience.com/deploying-keras-deep-learning-models-with-flask-5da4181436a2