Twitter Bot

Tim Marco Mennicken
Master student computer engineering
University of Applied Sciences Cologne
tim_marco.mennicken@smail.th-koeln.de

Sina Behdadkia
Master student computer engineering
University of Applied Sciences Cologne
sina.behdadkia@smail.th-koeln.de

Robert Rose
Master student computer engineering
University of Applied Sciences Cologne
robert.rose@th-koeln.de

***Abstract*—This article describes the development of a text generating neural network. It is trained on sets of short text messages extracted from the social networking service Twitter. The network creates a model with the ability of newly-creating texts which adopt the syntax of the training set.**

***Index Terms*—LSTM, neural network, Twitter**

I. Introduction

II. Data Acquisition

In order to train the neural network it is necessary to fetch data sets of existing Tweets[1]. Therefore two different tools are tested and compared with regard to the resulting data sets. The tested tools are the Twitter developer platform [1] (II-A) and GetOldTweets [2] (II-B). Especially the amount of fetched text messages is a crucial factor, as the neural network is expected to adapt better on bigger data sets. A requirement of obtaining at least 5000 Tweets for a valid data set is made.

A. Twitter developer platform

Twitter provides an official developer platform which provides API products, tools and resources which enable to automate the whole functionality of Twitter. The APIs are separated in several classes, some require a special access requirement which is not purchased in this work. Subsequent a standard API endpoint is used. In order to use the API products, users need to register for a Twitter developer account. This requires an exact description of the use case of the respecting API. After obtaining a valid developer account, it was necessary to generate a personalized API key as well as user and application access tokens. Twitter offer various wrapper libraries in different programming languages for simplified usage of the API products. In this project Tweepy (II-A1) is used. [3]

1) Tweepy: Tweepy is a Python wrapper for the Twitter API. It provides access to the entire Twitter RESTful API methods. After registration with the acquired API key and access token, Tweets can be fetched in form of Tweepy model class instances. Objects and method calls are named the same as the regarding instances of the Twitter API, i.e. one can use the terms of the official API reference.

2) Limitations: Each standard API endpoint underlies specific rate limits. That is, only a defined amount of requests are valid in a defined time window. Requests are separated in POST and GET endpoints. For data acquisition especially the GET (read) endpoints are of interest. The Twitter developer reference results in 900 possible status[2] requests per 15 minute window.
Furthermore the standard API endpoint has a limitation of only getting recent Tweets. Even when forcing the API to make requests in a loop, the oldest possible Tweets were published about a month ago with respect to the date of the request. Trying to get older Tweets results in blacklisting the API account, which prevents from passing the authentication stage get a response from the Twitter API.

3) Findings: The overall experience with the Twitter developer platform was not satisfying. Although it was possible to avoid the GET limitations with periodic calls in conjunction with pause times in between, it was not possible to evade the limitation of only getting recent Tweets. As purchasing a premium API is not an option in this project, a more convenient tool for getting a database is needed.

B. GetOldTweets

GetOldTweets [2] is a Python package to get old Tweets. It bypasses some limitations of the official Twitter API. Therefore it uses the how Twitter search through browsers work. The Twitter page scroll loader, which reveals a higher amount of Tweets depending on how far the user scrolls down, is used through calls to a JSON provider. Moreover there is no need to register with keys or access tokens.

1) Limitations: The GetOldTweets [2] package does not offer the possibility of publishing Tweets or getting a number of how often the regarding Tweet has been retweeted[3]. Basically it offers read only access without a lot of meta information.

2) Findings: As in this stage of the project it is most important to gather large data sets, GetOldTweets [2] is the preferred tool. The handling is simpler than that of the official Twitter API and it allows to fetch more Tweets in shorter time. Besides it is able to extract Tweets independent of the datum it was published.

[1]Tweets are short text messages limited to a specific count of characters. Initially only 140 characters were allowed, since 2017 the length of a Tweet was increased to 280 characters.

[2]Status in this context is equivalent to Tweet. Some naming of Twitter terms evolved over time without altering the API definition.

[3]A Retweet is a re-posting of a Tweet.

C. Implementation

The GetOldTweets [2] package will be implemented in a Python script. The script gets passed variable information, e.g. the username[4] of the person the Tweets will be fetched from, the amount of wanted Tweets, a period of time the Tweets were published and more via the command line. Please refer to the "get_tweets.py" script for all possible command line arguments.
The script creates a csv file containing for each Tweet the publication date and regarding text. Those are raw Tweets requiring further manipulation.

III. Data Preparation

In order to archive good results with the trained models, the fetched Tweets will be prepared before starting the training. This includes filtering unwanted content and manipulating punctuation of the Tweets.

A. Particular Content

Tweets can contain particular contents as weblinks, picture or video links, punctuation symbols, arbitrary special characters, retweets, hashtags[5] and references to Twitter usernames. In order to make the newly-created Tweets as authentic as possible, some of these contents need to be filtered out before passing the data set to the neural network. Therefore the following cosiderations are made.

- Weblinks are connected to the content of the regarding Tweet. The generated Tweets will, in best case scenarios, newly-create a statement. An improper weblink would impair the authenticity of the generated Tweet.
- Picture and video links have the same connection to the content of the regarding Tweet as weblinks.
- Punctuation symbols are part of valid sentences. The neural network will not be taught the grammatical rules of punctuation symbols, rather it will learn to use punctuation based on frequency of appearance. Hence only the most common punctuation symbols will be adapted.
- Special characters e.g. emoticons can be widely interpreted. It is hard to make an algorithm understand the regarding statement and when to use them.
- Retweets bear reference to other Tweets. As the trained model will have no or little connection to the Twitter universe, it has no point to include retweets.
- Hashtags are widely used in Tweets and are likely to be set multiple times in various Tweets of one user.
- Username references comply with the same considerations as hashtags.

With respect to the considerations only selected punctuation symbols[6], hashtags and references to usernames are fed into the neural network. The remaining particular contents will be filtered out before saving the data to the csv file. Therefore regular expressions are used to search all gathered Tweets for outstanding patterns and remove unwanted content.

[4]A username appears in the respecting profile URL and is unique to the user.

[5]A hashtag, written with a # symbol, is used to index keywords or topics on Twitter.

[6]point, comma and colon

B. Space characters

Before training, the neural network will separate the input data set at each space character to generate a list of tokens. Punctuation symbols that are directly behind words, will distort this process. For example would "house" and "house." be interpreted as different words by the network. To prevent punctuation symbols from causing errors, a space needs to be added before each punctuation symbol. This will cause the network to notice each punctuation symbol as a unique word. Besides multiple consequent space characters, which possibly exist because of previous string manipulation, are collapsed to one space character.

C. Data preparation flow

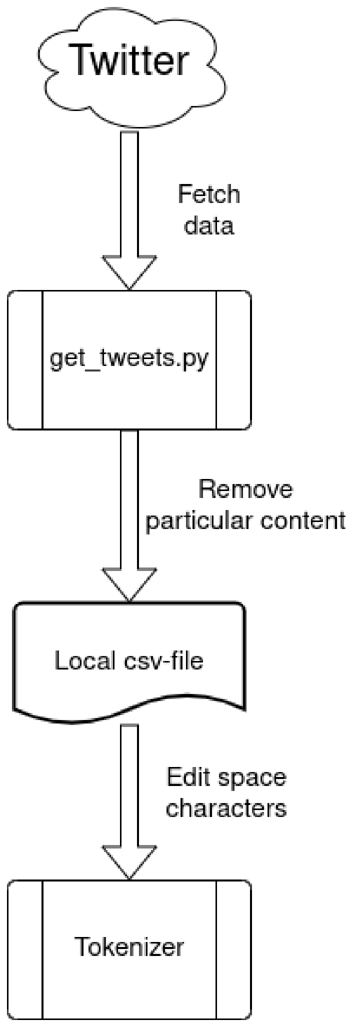

Fig. 1. Data preparation sequence.

IV. Post Processing

The generated Tweets will be modified slightly in order to increase authenticity and readability. Therefore space characters before punctuation symbols will be removed, as those were only needed in the training process. Furthermore all words are written lowercase, which is not desirable. The first words of all generated sentences are made uppercase with help of the natural language toolkit [4].

V. Experiments

At the first place as mentioned before we started to keep our dataset update, regarding to that aim we've made our own tweetBot, which is already able to gather the most recent tweets of any account.
In this step we tried to find a general approach, how it should be down as well. After analysing some articles we've choosed RNN against CNN models, because of the nature of our problem, which is about sequences data. The relationship between characters in a single word is also an important issue.
Schematically, a RNN layer uses a for loop to iterate over the timesteps of a sequence, while maintaining an internal state that encodes information about the timesteps it has seen so far. RNN models helps to find the most related character / word regarding to existed dataset.
First of all we ran the character level model. However promising they might sound, character level models do run against intuition. Words have semantic meaning, characters don't and apriori it's not obvious we can expect a model to learn anything about the semantic contents of a piece of text by going over the characters.
However we built a single LSTM model at first to see how it will work. So that at the end of every epoch because of a for loop printed the results, while is predicting next character appropriate with the most recent character.
The results of this method was sometimes not understandable words, or even not a word at all, which bring us to that point to give a try to word level model.
Word level neural language model can predict next word based on words already seen on the sequences.
Neural network models are a preferred method for developing statistical language models because they can use a distributed representation where different words with similar meanings have similar representation and because they can use a large context of recently observed words when making predictions.
The language model is statistical and will predict the probability of each word given an input sequence of text. The predicted word will be fed in as input to in turn generate the next word.
A key design decision is how long the input sequences should be. They need to be long enough to allow the model to learn the context for the words to predict. This input length will also define the length of seed text used to generate new sequences when we use the model.
There is no correct answer. With enough time and resources, we could explore the ability of the model to learn with differently sized input sequences.
Instead, we will pick a length of 50 words for the length of the input sequences, somewhat arbitrarily.
We could process the data so that the model only ever deals with self-contained sentences and pad or truncate the text to meet this requirement for each input sequence.
Regarding to the model design, we are transforming the raw text into sequences of 50 input words to 1 output word.
In this point we trained three types of RNN models:

1) Consisting embedding layer
2) Without embedding layer
3) Bidirectional LSTM

1.1) The model we trained is a neural language model. It has a few unique characteristics:

- It uses a distributed representation for words so that different words with similar meanings will have a similar representation.
- It learns the representation at the same time as learning the model.
- It learns to predict the probability for the next word using the context of the last 100 words.

Specifically, we will use an Embedding Layer to learn the representation of words, and a Long Short-Term Memory (LSTM) recurrent neural network to learn to predict words based on their context.
We've used two LSTM hidden layers with 100 memory cells each. More memory cells and a deeper network may achieve better results.
A dense fully connected layer with 100 neurons connects to the LSTM hidden layers to interpret the features extracted from the sequence. The output layer predicts the next word as a single vector the size of the vocabulary with a probability for each word in the vocabulary. A softmax activation function is used to ensure the outputs have the characteristics of normalized probabilities.
Technically, the model is learning a multi-class classification and categorical cross_entropy is the suitable loss function for this type of problem. The efficient Adam implementation to mini-batch gradient descent is used and accuracy is evaluated of the model.
Finally, the model is fit on the data for 100 training epochs with a modest batch size of 128 to speed things up.

A. Result

Generating with seed: "remember, i am self-funding my campaign,"

emember, i am self-funding my campaign, boegfeth and suds timina to honest about me, "(bad about his ventizes fiptive orce fush i tell wite throuse? 2pcx, amer

So we've some results,which are not satisfying and not all fo them have a certain meaning, but still much better than character level.

We specify the kind of model we want to make (a sequential one), and then add our first layer. We'll do dropout to prevent overfitting, followed by another layer or two. Then we'll add the final layer, a densely connected layer that will output a probability about what the next character in the sequence will be.

B. Result

At this point we've some meaningful sentences,which are grammatically correct and we can understand the prediction as well.
Instead, at the end we've still a loop of a sentence,which repeats the same words.

VI. Results

VII. Conclusion

We can probably have better results by increasing the size of the RNN, tuning the model to limit variance, etc.
Applying these modifications could increase the capacity for the neural network to generate good phrases, less fuzzy.
So, yes, LSTM can be use to generate not-so-bad text, without great instruction, bu we cannot say that it is a real text, with a global meaning. Regarding the sense of a paragraph (even with a network trained longer than what we did), we are far behind something readable. After few words, sentences does not mean anything, as a whole.

References

[1] Twitter, "Twitter Developers" [Online]. Available: https://developer.twitter.com/en.html. [Accessed: 21- Jan- 2020].
[2] Jefferson Henrique, "GetOldTweets-python - A project written in Python to get old tweets" [Online]. Available: https://github.com/Jefferson-Henrique/GetOldTweets-python. [Accessed: 21- Jan- 2020].
[3] Aaron Hill, Joshua Rosslein and Harmon, "Tweepy - Twitter for Python" [Online]. Available: https://github.com/tweepy/tweepy. [Accessed: 21- Jan- 2020].
[4] NLTK Project, "Natural Language Toolkit 3.4.5" [Online]. Available: https://www.nltk.org/. [Accessed: 21- Jan- 2020].