Trump Tweet Generator

Using Neural Network

Jacob Molin, Henrik Rosander, Linn Storesund
jacmo699@student.liu.se, henro926@student.liu.se, linst539@student.liu.se

January 26, 2021

1 Introduction

Machine learning is a technique that uses algorithms to automatically adapt, improve or change the behaviour of a system. In this study, machine learning and natural language processing will be used to create a model that generates text. The generator will work with a data set of Donald Trump's *Twitter* tweets and train a character-based model to predict the next character in a sequence. The desired result is a model that can produce a text sequence similar to Trump's tweet's with his vocabulary and grammar.

Twitter is used by 131 million people all over the world. Twitter is a micro blogging and social networking service developed in America. A person can create an account and post small text posts called "tweets" that other people can interact with by commenting, sharing, or liking the tweet. People can also follow each other's accounts which means subscribing to their tweets to receive them in your feed. A tweet cannot exceed 280 characters and all tweets are public.

Donald J. Trump is a 74-year-old politician, businessman and television personality who is the current president of the United States. He has used Twitter for 11 years and has tweeted very frequently, a total of 59 thousand tweets over the years. As seen in figure 1, he has 88.7 million followers that he can spread his opinions to all the hours of the day. The goal of this project is to create a text generator that learns how Trump tweets and can generate a text which is similar to a Donald Trump tweet.

Figure 1: Donald J. Trump's Twitter account [1].

2 Background

What is fascinating about text generation is that it must learn from scratch. It does not know English or where to put apostrophes, commas, periods and when to put capital letters in the text from start. The text generator is learning how sentences are built and the grammar on its own, based on the input data.

For this project we've been inspired by a blog post by Pranjal Srivastava posted on analyticsvidhya.com called "How to create a poet / writer using Deep Learning (Text Generation using Python)?" [2]. This blog post shortly describes and compares word-based and character-based text generator models. It also shows implementations character-based text generator models. Using this blog post as a guide line, we've worked with Keras and TensorFlow, developed by Google, and implemented in Python. Keras is written in Python and is a deep learning API. Keras behaves as an interface for the TensorFlow library and it is designed to be simple and consistent by using less code. The high level API handles the way developers create models and helps define layers, compiles models with loss and optimiser functions. Keras API also enables fast experimentation with deep neural networks and runs on top of other frameworks. Tensorflow is the most popular machine learning tool developed by Google's Brain team. It can be used for many tasks, but it is especially used for training and inference of deep neural networks.

3 Research questions

In this report these research questions will be answered:

- How can a text generator be built, that can write twitter posts in the style of Donald Trump, using machine learning?
- Which of the developed text generator models are most realistic?

4 Boundaries

Due to the time it takes to train neural network models we limited the size of the data set so that we could train multiple models and compare them in the time given for the project. We also chose to work with character-based text generation instead of word-based text generation for two reasons. The first one being that we were interested in creating models which learn to form words as well as sentences. With word-based text-generation that is not needed, since all words in the model's dictionary are given by the data set. The second one being the data set that we used was simply too small to sufficiently train a word-based model. More on this in section 6.

5 Data Set

The data set used in this project is a collection of most of Donald Trump's tweets from The Trump Archive [3]. When the data set was downloaded (26 November 2020) it contained 55 090 tweets. The data set is created by checking Donald Trump's Twitter feed every 60 seconds and uses scraping to collect the new tweets. Besides the actual text in the tweet, the data set collects information about if the tweet is a retweet, has been deleted (only applies for tweets after September 2016), device posted from, number of times retweeted, number of favorite and date posted, see figure 2 [3].

Due to the time it takes to train the deep neural network model we decided to use a subset of the data set. The actual data set used to train the models are all the tweets from March 2020. Four examples of tweets from the data set can be seen in figure 3.

	id	text	isRetweet	isDeleted	device	favorites	retweets	date
1130	713033155989610496	Endorsements for Lyin' Ted Cruz- https://t.co/...	f	f	Twitter Web Client	8395	3209	2016-03-24 16:02:06
1131	711046719421550592	Landing in Phoenix now. Tomorrow's events will...	f	f	Twitter for iPhone	8040	2056	2016-03-19 04:28:43
1132	708428041232490496	Great news! Thank you Governor Ralph DLG Torre...	f	f	Twitter for iPhone	8599	3126	2016-03-11 23:03:01
1133	706530975992381441	Thank you Michigan! #Trump2016 https://t.co/tt...	f	f	Twitter for iPhone	9628	3421	2016-03-06 17:24:46
1134	706153159085821952	#CaucusForTrump #Trump2016 https://t.co/pTUlD...	f	f	Twitter for iPhone	5885	1628	2016-03-05 16:23:27
30692	714899439706574848	MAKE AMERICA GREAT AGAIN!https://t.co/0w4ldD7dW3	f	f	Twitter for iPhone	11340	4946	2016-03-29 19:38:03
30800	710050536259764226	Will be interviewed on @NewDay on @CNN at 7:15...	f	f	Twitter for Android	5012	1017	2016-03-16 10:30:14
30838	710050169505640448	Will be interviewed on @GMA at 7:00 A.M. Big w...	f	f	Twitter for Android	5463	1076	2016-03-16 10:28:47
30987	704880376892813312	Thank you Arkansas! #Trump2016#SuperTuesday	f	f	Twitter for iPhone	13186	3814	2016-03-02 04:05:52
31431	704846146657722368	Thank you Virginia! #Trump2016#SuperTuesday	f	f	Twitter for iPhone	16200	5289	2016-03-02 01:49:51
33415	704834185471598592	Thank you Alabama! #Trump2016#SuperTuesday	f	f	Twitter for iPhone	16229	5296	2016-03-02 01:02:19
34237	704834141565657088	Thank you Tennessee! #Trump2016#SuperTuesday	f	f	Twitter for iPhone	16460	5573	2016-03-02 01:02:09
35247	704834059915104256	Thank you Massachusetts! #Trump2016 #SuperTuesday	f	f	Twitter for iPhone	16781	5621	2016-03-02 01:01:49
35550	704818842153971712	Thank you Georgia!#SuperTuesday #Trump2016	f	f	Twitter for iPhone	17638	6137	2016-03-02 00:01:21
40398	715668280153911296	THANK YOU, NEW YORK! #Trump2016 https://t.co/O...	f	f	Twitter for iPhone	10923	4028	2016-03-31 22:33:09

length df: 55090

Figure 2: 15 rows of the entire data set.

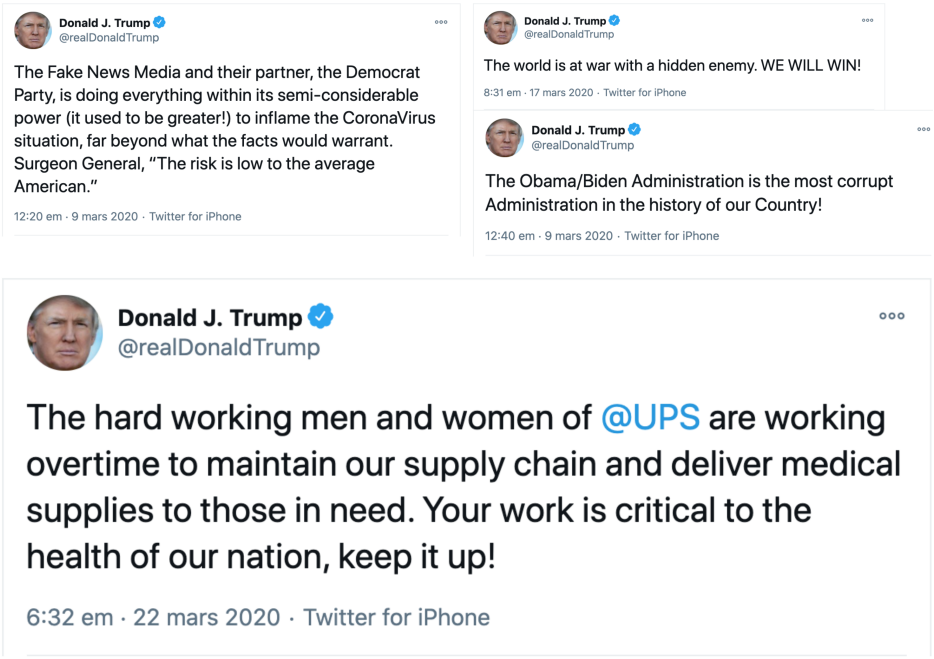

Figure 3: Four of Donald Trump's tweets from March 2020 which are in the data set [1].

6 Data Pre-Processing

Before creating and training the model the text in the data set was cleaned. We first removed all retweets and responses to Trump's tweets, which are also in the data set. After that we removed links, odd characters and fixed certain wrong formatting (like putting a white space between the last word of the tweet and the "#" of the following hashtag). All the characters were also lowered so "Hello" and "hello" didn't count as two separate words. The cleaned up data set contained 447 tweets, 10 963 words, 3088 unique words, 63 999 characters and 62 unique characters. This was combined into one long text string.

The data set we were working with was relatively small when comparing unique words with total amount of words. Unique words in the data set constitute about 28% of the whole data set, which is probably too much to get a well trained model from. We chose to create a character-based model instead. The unique characters constitute about 0.01% of all the characters in the data set, which means that each character occurs frequently enough in the data set to be able to obtain a well-trained model. The downside of using a character-based approach versus a word-based approach is that the model has to learn how to create grammatically correct words beyond learning to create sequences of words, which a word-based model gets for free. This means that the accuracy in general is lower with character-based models, since a much larger network is needed to create an accurate result. The next step was to create character mapping. Every unique character in the data set was mapped to a unique number, or index, and these were put in a dictionary. This made the training process smoother.

The data was then formatted into a training array X_{train} and a target array Y_{target} so it could be accepted into the model. Each index of X_{train} should contain a sequence of a set length of characters from the data set that are to be considered before generating the next character. Y_{target} contains the target that is a correct generating after the corresponding sequence in X_{train}. This was done for the entire data set, where a sequence is moved one character at a time. An example using the text string "trump tweet" and a sequence length of four is shown in table 1.

X_{train}	Y_{target}
['t', 'r', 'u', 'm']	['p']
['r', 'u', 'm', 'p']	[' ']
['u', 'm', 'p', ' ']	['t']
['m', 'p', ' ', 't']	['w']
['p', ' ', 't', 'w']	['e']

Table 1: An example of five iterations of the sequence learning method.

Lastly X_{train} and Y_{target} were modified into a form that is accepted by the model. We used a LSTM model which has the input format ($number\_of\_sequences$, $length\_of\_sequence$, $number\_of\_features$). Therefore X_{train} and Y_{target} were modified using the code below. In the code *characters* is a sorted list containing all the unique characters in the data set. *characters* is therefore the dictionary that the model use to generate text.

```
import numpy as np
characters = sorted(list(set(text)))
for i in range(0, length-seq_length, 1):
        sequence = text[i:i + seq_length]
        label = text[i + seq_length]
        X_train.append([char_to_n[char] for char in sequence])
        Y_target.append(char_to_n[label])
X_modified = np.reshape(X_train, (len(X_train), seq_length, 1))
X_modified = X_modified / float(len(characters))
Y_modified = np_utils.to_categorical(Y_target)
```

7 Modeling

When the data is ready to use, a model is created for generating the output. In this project a sequential model from keras was used. The Sequential model consists of a linear stack of layers. The network structure consists of LSTM (Long short-term memory) layers, which is of artificial recurrent neural network (RNN) architecture.[5][6] To instantiate a LSTM layer on the model there are some parameters to be declared. These are the units, which have the dimensionality of the output space. The second parameter is the input shape, which will be used to generate the dropout mask, which consists of zeros and ones. The third parameter, if there are going to be multiple layers, is a callback which tells the model that the same data is going to be used again.

When the layers are set up, the next step is to fit the model accordingly shape the weights of the model. This is where the training phase begins, and therefore some more attributes in the fitting need to be introduced. When using Keras' built-in Tensorflow, a batch size and amount of epochs needs to be defined. The batch size is the parameter which tells the model how many samples that will be propagated through the network during each iteration. Another parameter is the epoch, which is used to define how many times we must iterate over the data set.

Using GPU-acceleration from NVIDIA, supported by Tensorflow through using CUDA, a platform which allows computation being sent to the GPU (Graphical Processing Unit) instead of only computing on the CPU [4]. This allows faster time for fitting the model, however, it requires the user to have a NVIDIA graphics card.

8 Evaluation

By using the trained model, text can now be generated from the training data. By comparing the weights of each character after each other, an output is generated based on the parameters. A problem that was encountered was that the runtime of the model turned out to be very time consuming, which led to trial and error was a very limited option for the text generating. In figure 4 there are multiple results of using the same training data with different values of the parameters. The parameter *drop* represents the dropout regularization of the neural network. This is done by probabilistically removing inputs (or connection) to a layer. This helps with preventing overfitting of the model, i.e. preventing the model from becoming too specified on the training data, instead of being generalized. The result have two parts as you can see in figure 4. The grey part is not generated, it is a beginning of a tweet from the data set, which the new generated text is based on. The black part is the generated part, how the tweet will continue if the model got to decide and therefore the relevant part to look at when analysing the result.

- Base model **(1 layer, seq_length: 150, units 400, drop: 20%, epochs: 1, batch_size: 100)**:
  - ing american patriots who love our country, cherish our values, respect our laws, and always put america first! thank you for a wonderful evening!! #ke te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te te t
- Better model **(3 layers, seq_length: 150, units: 400, drop: 20%, epochs: 10, batch_size: 50)** ~6.5h training:
  - h our values, respect our laws, and always put america first! thank you for a wonderful evening!! #kag2020 great story. thank you to mr. young of jone the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the toe the t
- Biggest model **(3 layers, seq_length: 150, units 300, drop: 20%, epochs: 20, batch_size: 18)** :
  - our values, respect our laws, and always put america first! thank you for a wonderful evening!! #kag2020 great story. thank you to mr. young of jone to toeether. we will prevail! thank you to the great soate of the ueietes of the ueietes of the uece for the doronavirus tituation and sooelimn to se

Figure 4: Different results due to different values of the parameters.

Depending on the computer and hardware you're using it can be very time consuming to train these types of models. A lot of time can be lost by training models with a faulty setup of the data and we therefore recommend cleaning up the data set a lot, maybe even removing characters like dots, commas and parentheses, to get a better result. It's also a good idea to train a small network in the beginning, just to see if everything works, before scaling up the network. In our project we didn't get a very satisfying result. Our model probably needed more training time and more data to get a better result. It seems to be much harder than we thought to train and generate with a good result. If we would do it again we would work more with word-based prediction, so the model doesn't have to learn how to build grammatically correct words. All in all it was a fun project when things worked, but a little sad when it didn't, since there was such a long waiting time for the model to train.

9 Summary

In this section the research questions from section 3 will be answered.

- **How can a text generator be built that can write twitter posts in the style of Donald Trump using machine learning?**
  We built a machine learning model by using Python, Keras API and TensorFlow. With these tools a program can train a model by using external data to create a text based on Trump's vocabulary and grammar. Training a basic models does not require much time or a more powerful hardware than a decent laptop, but the result from the model will most likely not resemble the original text data. To train a deeper and better model to get better result, more time, more powerful hardware and more text data is needed.
- **Which text generator model is most realistic?**
  The model used in the "Biggest model", see figure 4, is the most realistic one. Even though the text have some misspelling you can understand the meaning of the sentence. Next time we will train a model with words instead of characters and with more data. That way a more realistic text would probably be produced.

References

[1] Donald J. Trump's Twitter Account [Twitter]
https://twitter.com/realdonaldtrump (Retrieved 2020-11-28)

[2] How to create a poet / writer using Deep Learning (Text Generation using Python)? [Analytics Vidhya]
https://www.analyticsvidhya.com/blog/2018/03/text-generation-using-python-nlp/
(Retrieved 2020-11-28)

[3] Data set of Donald Trump's tweets [The Trump Archive]
https://www.thetrumparchive.com/ (Retrieved 2020-11-26)

[4] CUDA Zone [NVIDIA]
https://developer.nvidia.com/cuda-zone (Retrieved 2020-12-13)

[5] Your First Deep Learning Project in Python with Keras Step-By-Step [machinelearningmastery]
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
(Retrieved 2020-01-20)

[6] Understanding RNN and LSTM [Afshine Amidi and Shervine Amidi]
https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e