Twitter Bot using Neural Networks

Adrià Navarro Ramírez

1st July 2018

Abstract– This project aims to be a study of the application of Recurrent Neural Networks in the Natural Language Processing field. To do this, as a conductor, we are going to develop an application capable of generate text character per character after being trained with e-books first and with tweets later. At the end, the application will become a bot of Twitter that will write tweets autonomously, while an analysis of the texts generated will serve to understand how the Recurrent Neural Networks work and the possibilities beyond that tool.
Keywords– Recurrent Neural Network, Text Generation, Twitter Bot, Natural Language Processing.

1 Introduction

After some time with low levels of activity, we are now living important years in the development of Artificial Intelligence. There are some new techniques, and also machine learning engineers are taking some of the old ones to use them now. But why now? There are two main reasons to that increment of investment, papers and results, the new hardware and the social networks.

Data has always been essential, but in the last years it is even more important. The Internet created a new channel of information, but it is the emergence of Social Networks what has changed the vision of the data, because it generated a huge increment of the information we can collect, information that was not important until now. It does not mean that the usual data or information has lost its impact, companies and countries are generating more data than ever, but the opinion of people, their acts, their migrations, their every-day routines are now registered on the Social Networks, and doing aggregations of the data allows us to generate new information. That was the boost of Artificial Intelligence in the last years.

One of the greatest impact of that new paradigm was the enormous amount of data we can use. Deep Learning is useless with small amounts of it, but with tons of data it can produce amazing results. In that aspect, Social Networks are gold mines. For example, Twitter, the Social Network we are going to use in this TFG, generates 500 million tweets per day, and YouTube generates more than 100 hours of videos every minute.

1.1 The problem

With the new tools that have emerged, there is no reason to limit the domain of the problems we try to solve to the old problems, such as the usual regression or classification. Those problems can be solved in a new way using, for example, Neural Networks, but there are also new problems such as Object Detectors in images or music generation. The amount of possibilities has make the academic area of Machine Learning really exciting. There is a problem, though, that is actually one of the first problems that Artificial Intelligence tried to solve, and that is the Natural Language Processing. It is a wide topic, with some high level solutions. We all can think in the new voice assistants, and understand that their quality depends on their ability to understand what the customer is asking, and not in a comfortable text, but in an imperfect and noisy audio. Here the old paradigm does not work, especially because we don't know how to design a solution to understand language.

For this kind of problem, where we could not find a way to do it ourselves, Neural Networks have proved to be extremely useful. It is a simple concept: some numeric input flows through a path of neurons that perform a simple operation to end up finding the expected solution. This approximation to the solution, however, has its limitations.

We are trying to generate text character per character, meaning that after generate one character, we are going to repeat the process to generate the next one, forming words and sentences. If we try to use a Feedforward Neural Network, after specific character such as 'l', we would always generate the same character, lets say 'a', because during the training, the sequence 'la' is more usual than any other se-

• Contact e-mail: adrian.navarro@e-campus.uab.cat
• Branch of studies: Computació
• Tutors: Pau Riba (Computer Science), Josep Lladós (Computer Science)
• Course 2017/18

quence of 2 characters starting by 'l'. The objective should be to use not only the last character 'l', but the previous one, and so on, to have some kind of context or memory about what are we writing as a whole. That way, we could know that before the last character 'l' we wrote another 'l', and previous to this one, an 'e', preceded by 'h', forming the string 'hell'. Knowing this, the next character would not be 'a', but 'o', to form the word 'hello'. Feedforward Neural Networks can not perform those results.

1.2 State of Art

Nowadays, however, there are different applications that have some kind of understanding of the language. From an orthographic corrector to bots that can chat with customers to solve some of the frequent asked questions or even the virtual assistants such as Siri, Cortana or Alexa. The algorithms behind them started with some poor autonomous techniques (f.e. decision trees, where all the possibilities had to be contemplated by the developer), but now the Neural Networks have become the most widely tool for that, or at least, a combination of Neural Networks with old algorithms.

To overcome the limitations that Neural Networks present different types of Neural Networks were developed. One of them has proved to be extremely useful in the work with sequences of data. It includes text, which is a model of Natural Language represented as a sequence of characters or words, but also with spoken language (audio), music, or other problems where the sequence should be the input, such as weather prediction. This new type of Neural Network is called Recurrent Neural Network, and it is based on Markov Hidden Models.

Markov Hidden Models introduced the concept of state. The current state is a variable that will affect the probabilities of the outputs. The same concept is used in Recurrent Neural Networks. As we can see in Figure 1, a Recurrent Neural Network is a Forward Neural Network with a new component, the state, which will receive the previous state and modify it depending on the current inputs / outputs and then, send that result to the next iteration of the RNN execution. That state will affect the output, in a way that now the current inputs are not the only data that generates the output, but the previous iterations that propagate that state.

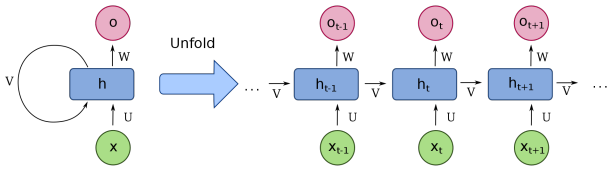

Figure 1: Scheme of a Recurrent Neural Network

Figure 1 shows a Recurrent Neural Network unfolded. In the right part of the figure we can see three iterations of the same RNN, with different inputs and outputs, and the propagation of the state from one iteration to the next one.

Because of this state that propagates in time, RNN simulates memory, and if Forward Neural Networks are equivalent to functions, Recurrent Neural Networks are equivalent to more complex programs, with variables that change during the execution.

So Siri, Cortana, Alexa... All of them are using RNN because they are the most advanced tool in Natural Language Processing nowadays.

1.3 Motivation

There are two main reasons that made me take the decision of start the Twitter Bot using Neural Networks project. The most important thing is the importance of the Natural Language Processing, not only in artificial intelligence but also in the human behaviour. This is a very exciting area of study, from the way a baby learns to talk to how different languages affect the way we think or perceive the reality. There are studies (Learning Features and Segments from Waveforms: A Statistical Model of Early Phonological Acquisition, by Ying Lin) that demonstrate that babies learn to group the sounds into the different phonemes of their language, and then, they learn the syllables by understanding that some phonemes make sequences that appear more often than others. They repeat the process to understand that some syllables generate words, and then they learn to associate meaning to the words. We can actually do the same process with the tools we already have, even if it is too early to generate the same results in language understanding, and Recurrent Neural Networks for Natural Language Processing opens that door.

The second reason is the fact that RNN are really popular nowadays. They brought a solution to problems that Forward Neural Networks were not able to solve, and even if it is not in NLP area, understanding how they work and being able to use them is a good opportunity for everyone interested in Deep Learning.

1.4 Objectives

Taking as a reference the reasons that started this project shown in the previous section, the academic objectives are to work with and learn about Natural Language Processing and Recurrent Neural Networks. To achieve this, we are going to create a text generator, and merging it with the importance of the Social Networks and all the data that we can take from them, the text generator will generate tweets autonomously and publish them on Twitter.

Summarising, the objectives of this TFG are the following ones:

- To study one of the general types of NN that have better performance nowadays (RNN).
- To approach NLP by using high-performance tools.
- To reach the state of art in autonomous generation of text.
- To manage the whole life-time of a project.

- To create a final product able to show the results of an academic research.
- To deal with a well known deep learning library

When the project is done, we expect to have similar results than the ones obtained by Karpathy in his project: The Unreasonable Effectiveness of Recurrent Neural Networks, which was a model to build this one.

1.5 Methodology and planning

The project is, at the end, a TFG, and that entails some time restrictions and requirements that have to be satisfied.

All the work comprised in the project started on February and will be delivered on July, completing a period of 5 months. The TFG structure includes a first task of study the subject of the project and then to make a planning of it. After define the technical tasks that should be done, we decided to make some of the tasks in parallel (the extraction of tweets can be authomatic, and the training of the model allow us to focus on documentation in the meanwhile) but most of them can not start until the previous one is done. The reason is the fact that the most important objective of this project is not the production of an application, but the study of its behaviour, and there is an analysis planned after every extraction of results. The Gantt diagram can be found in the annex, as well as the changes produced to the code or any material related to this project can be found in the dossier.

2 DEVELOPMENT

2.1 The environment

To develop this project, we used the programming language Python, since it is a popular choice and it is also known for its compatibility with libraries focused on Artificial Intelligence.
We are also using one of the libraries supported by Python. One of the objectives of this project is to learn to use a popular library of deep learning, and we have chose Google's TensorFlow.
To use the resources available and knowing that deep learning requires lots of calculations that can be executed in parallel using specific hardware, we used the TensorFlow version that allows GPU usage.

2.2 Proposed Architecture

The Neural Network developed uses a first layer that receives inputs. Those inputs need to be numerical values, and in our case those values come from the characters of the text we are using. Each character will have a numerical value associated. The first step that the Neural Network performs is the transformation of those inputs in a One-Hot encoding format, in a way that a value is transformed into an array of n elements, being n the amount of different characters we are using. All the elements will be 0 except the one that correspond to the position associated to the character. In Figure 2 we can see a representation of that transformation.

Hello

Different characters = [H, e, l, o]

H = [1, 0, 0, 0]

e = [0, 1, 0, 0]

l = [0, 0, 1, 0]

l = [0, 0, 1, 0]

o = [0, 0, 0, 1]

Figure 2: Example of One-Hot Spot codification

Then, the input layer sends the inputs to the hidden layers. Those layers are composed by LSTM (Long Short Term Memory) cells and GRU (Gated Recurrent Unit) cells. Each one of those layers (and cells) executes the multiplication of the inputs and its weights, and adds the bias. Over that common operation, internally the cells are keeping the previous state and changing it in each iteration, modifying it depending on the inputs that the cell receives and the output that produces. The state is also used to modify the value generated for the cell.

After the hidden layers have created the new value, the output layer uses a softmax function over the value generated. It transforms the value into a new one in the range [0, 1]. Then the output layer selects the higher value as a 1 and the other values as 0, having as a result an array of n elements where n is the number of characters we are using, all of them as 0 except one equals to 1. The last step is to transform that array into a character again, using the same procedure than the input layer but reversed, so the position of the number 1 is the identifier of the character generated.

Finally, the Neural Network uses a softmax cross entropy function to evaluate the results generated with the labels during the training o validation phase. This function measures the distance between the generated character and the correct one. The Adam optimizer (Adaptive Moment Estimation, https://arxiv.org/abs/1412.6980) reduces the mean average of error of that last function. Adam is a good optimizer for Natural Language Processing since the problems in that field use to have a sparse gradient.

In Figure 3 we can see a scheme of the Neural Network unfold with an example of the word 'Hello' treated as an input sequence. It is important to note that, without the state, the last two iterations would be exactly the same and would produce the same output.

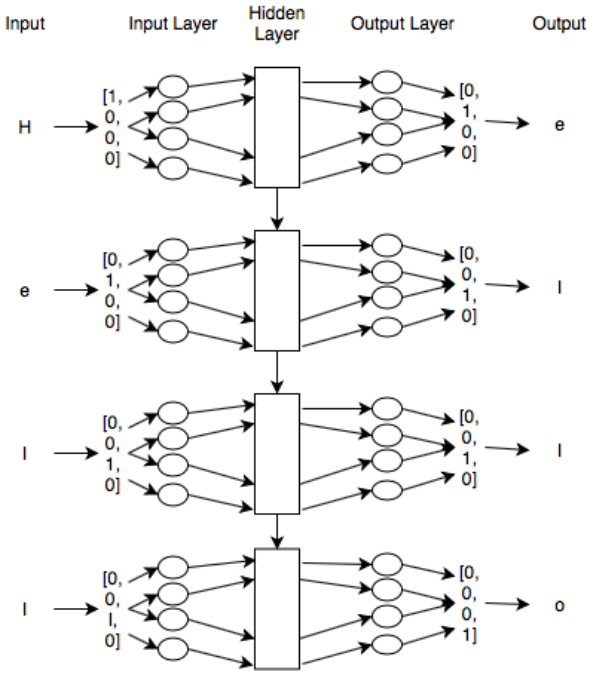

Figure 3: Recurrent Neural Network predicting the next character

2.3 Treatment of data

For this section we are going to suppose that the data that we are going to use to train the model is already in a txt file.

Then, the data (text) will be splitted into sequences of a length determined by the Neural Network. The length of the sequences represents the "memory" the Neural Network has, and it determines how long the hidden layers are going to send their states in the future iterations. The longer the length of those sequences, the longer the character will remain in the Network as a state. Then, all of these sequences need to be transformed into sequences of values instead of characters. To do this, we need to know how many different characters are in our text. This will determine the length of our vocabulary, and we will fill it with the characters, associating each one to a numeric value that will represent its position inside the vocabulary, as an index. For example, if our text is the string "Hello world!", we have 9 characters in our vocabulary: [H, e, l, o, , w, r, d, !] (Note that the blank space is also a character), and that vocabulary would be codified as {H:0, e:1, l:2, o:3, :4, w:5, r:6, d:7, !:8}.

Finally, the Neural Network accepts multiple sequences of text at the same time, called batches. It allows us to do the training in parallel branches, boosting all the process.

2.4 Generating Results

After the building of the network the treatment of the text that we use as input and the training, we save our model. TensorFlow allows us to easily save the graph, a package of the elements that compose our network, into an external file, to be able to load it in the future and use it quickly.

When we want to load the model we first build a new graph, with the same conditions that we had before (number of layers and cells per layer, for example), but since we are not going to train it now, we only accept one batch as the main input. Then we restore the session of TensorFlow to this new graph, and the trained weights will replace the new ones that were generated randomly. After that, we only need to use one character to start the generation of text, since the model will use the same function it used during the training to predict the next character, but instead of comparing it with the true character, we are just keeping it as the generated text, and use it as the input of the next iteration.

2.5 Twitter Bot

With all the previous steps we would have a text generator, but one of our objectives is to embed it into a Twitter Bot, in a way that it can generate tweets autonomously. It is a perfect excuse to use the data we can collect from Twitter and find an application to the text we generate.

There are 3 main branches in the building of the bot:

- **Creating the Twitter app.** First thing to do is to create a Twitter account. We need to go to Twitter and create an account using a phone number. Once we have it, we can create our app in apps.twitter.com. There we can generate our access tokens and keys, and we are going to need them to use the app. This app is linked to the account we used to create it.
- **Extracting tweets.** Since we are using Python to develop our project, we are also going to use Tweepy, a library that allows us to connect to the Twitter API. There we need to use the keys and tokens we got in the previous step to be able to access the app. Once we have access we can extract the tweets that an user has published or the tweets that contain a specific string (f.e. tweets with the hashtag Twitter).
- **Publish new tweets.** As in the previous step, we need to use the keys and tokens of our app to be able to publish a string as a new tweet in the Twitter account we used linked to the app.

So, having all configurated, we just need to pass the string generated by the Neural Network to a script that will access the Twitter app and publish it as a new tweet.

3 Text Generation

We have now created the structures we need, but to be able to generate text, we first need to get our data.

For the first tests we are going to use something simpler than tweets. Tweets have some extra work of treatment because they contain links, emojis, mentions, hashtags, they can be retweets... So we will first use a couple of e-books. In Project Gutenberg we can find lots of free e-books to download in different formats, in our case, in a

txt file. We are going to use *The Count of Montecristo* and *Hamlet* to train our model. The text is already in a good format, there is no need of treat it except for the first pages, that are not a part of the novels, but Project Gutenberg's notes.

The evaluation of the results is qualitative. We can use the loss function to measure the success of the text, to know how good they are, but at the end we want the algorithm to generate new text that a person could have written. The loss function will measure how close the text generated is to the known text and, the lower the loss, the higher the similarity to the original text.

So we start our test generating text from these two e-books checking the results at each step. The first one will be done using *The Count of Montecristo*, in a network with 3 hidden layers of 100 cells each one. After 5 epochs here we have an example of the text generated:

Hidden Layers	3
Cells	100
Epochs	5

a taaatntatnn ae e nean netaanen een e a ea teneeea n ae ta tneet tttnentttanta t n an tne ae antt at tee taa ann tett t ttet a ntaaannt an e attn

Results 1: THE COUNT OF MONTECRISTO

We can see that the text makes no sense yet. However, there is some interesting behaviour: we are generating some of the most used characters in English, such as *t*, *n* and *e*. We are also having blank spaces in a similar frequency than the one we could find in a normal text, making the words have different sizes and none of the words is too long.

At this point, the model needs more epochs to keep learning, so we are going to repeat the training with 2000 epochs:

Hidden Layers	3
Cells	100
Epochs	2000

Monte Cristo would have dished that an aristion of his happy to say he had a study, and to the churqoised a private offer month, as if the corblina would be seeked back its ore into myshischessible and the seat, and a precaution of the picle. He had suppided to barden towards his power.

"Well," returned Morrel.

"That means is that have been a place."

"Yes, you moving to the morning, and who was see, and you have tell you to say, it is it to that is a murderer," said the daughter, "this inside of the cause of any way to me." The count was served his heally at the chimney; he would not half story.

"What deserted!" said Monte Cristo with meeting that they had no doubt to him a servants; "that is an endure and so to me as a third. I could not befolence here. Albert, with your favuntarian and carried?"

"Indeed, and I was something made there."

Results 2: THE COUNT OF MONTECRISTO

Here we have two samples of text. Now we can see something way more similar to what a human being would have written. The model has learnt how to join the characters to create words, and almost all of them are perfectly correct in English. It has also learn some of the names that appear in the novel, such as Monte Cristo, Morrel or Albert. Not only we have the correct size of the words, but also of the sentences and even the paragraphs, having placed correctly the punctuation signs. It is also remarkable how the Neural Network creates dialogues opening and closing quotes, and, although the sentences barely have sense, we can appreciate the structure of a novel here, for example on *"Well," returned Morrel.* Note also how it uses third person in the narration and first person in the dialogues (text between quotes). And finally, we even have an answer in the second text, where the first paragraphs ends in a question mark and the second one, also a dialog, starts with *Indeed*.

We have seen how the architecture of the Neural Network was able to generate decent text, and the number of epochs determined the success of it. Let's try to generate some Shakespeare text now, training the model with *Hamlet* and repeating the previous features:

Hidden Layers	3
Cells	100
Epochs	2000

HAMLET. O God! Haw, his musidenbous, and my dry Yee Swaller to hell, As if I, with the King lits do not, and of such am I hear-laons to this work wi'twrak. For the lover well sudnt-Kin thought the mother. There's a spees, to down our something at a bingty of his fine That, fast, in this effect of elser's ears. The pall, as I take her lay, And in grief destine awh, think it both trick offices; he trick is bornue of accilate, within the carriages, Let the braw miss and makes the thoush can. Look nor out of what were nothing burices hat I will season grave, And this struck, I without another.

OSRIC. The King, Queen, there hath by the rices there is not, That indeed, and what is a plots are hour wise of power was not tell too conceit.

KING. We have no merch, Art fortune, that do you goe? If a copyidies of enemit finds the choice if it that mark yet the till this time to you frees' may call the means, to writ Thanks.

HAMLET. Why, we sleard, my lord.

HAMLET. I would not berefain one every his mother. Fetthedtory contracted home, think is no room in the appluw'd The owherathy and of halus cry, thee, let the stould tell you these. Your sent, sir, as you heavens, work'd shall the harm in't, And thou must be your possies black. It off, be as a lois for most false whee to play.

Results 3: HAMLET

We have similar results to the *Montecristo* version. Most of the words are correct (in old English), but it still makes sentences that make no sense. The names of the characters and the general structure is good enough, so it is able to recreate not only the novel style but also a theatre script.

So now we are going to produce a text with the same architecture for the Neural Network, still *Hamlet* for the training, but we are increasing the number of epochs from 2000 to 6000:

Hidden Layers	3
Cells	100
Epochs	6000

HAMLET. Gentremen! A good freel', And I'll so same his head That will the play's show hither must not of his son it.

HORATIO. Not do, my lord?

HAMLET. I a twell, you well. S you an attends to you the habit between her come to my peoind A well you falls and eyes.

QUEEN. He world in.

POLONIUS. My lord, I have seen the contreft, yet your craye come. A cannonit word look acceit that I see any worth, fer ginds; comes percomen.

Results 4: HAMLET

Increasing the number of epochs did not improve the results because the architecture or the data limited the quality of the text generated. But now we know we can generate text, so let's try to replicate the same quality with tweets and try to improve them.

While the text of the e-books was easy to use, the tweets extracted using the Tweepy script through the Twitter app were not. We first need to clean the tweets, remove the links and the emojis that difficult the codification and add complexity to the text. We are also getting just unique tweets, since the retweets would add the same text multiple times and we have no interest on it. So after cleaning the tweets to format the text, we are also adding two characters to each tweet. Those characters are $\tilde{C}$ and $\tilde{N}$ (will not appear in English tweets) and will determine the start and the end of the tweet. That way the character we are going to send to the model to start the text will be Ç, and the tweet will end when the Neural Network generates Ñ. So there will be no bias when we start the text with a specific character and we will know when to stop the tweet, instead of finishing at 140 characters in the middle of a word.

The tweets extracted contained some words related to the FIFA World Cup of Russia 2018, since it is a popular topic right now. Some of the words used to do the search and the extraction of the tweets were *FIFA*, *football*, *Russia2018*, *Messi* or *Cristiano*.

So here we can see the results that the Neural Network produced after being trained with the tweets, having 3 hidden layers, 100 cells per layer and 2000 epochs:

Hidden Layers	3
Cells	100
Epochs	2000
@markchets1 @Sheuraaled @Colandan_hea @SportsMagust Spain is a player. And what a great to take better today! It's the field feeling.	
@Kayden_Kross Argentina should be the only game to the best.	
Watching Iniesta was martial aftern	
#Wedlic #FifaWorldCup2018 #ridger #Russia2018	
I liked a @YouTube video	

Results 5: TWEETS

We have the same quality in the tweets. We can see the same pattern: correct words, names (Iniesta, Spain or mentions such as @YouTube) and size, but sentences with no meaning, except, perhaps, the last tweet *I liked a @YouTube video*, which is the text that appears by default when someone shares a YouTube video on Twitter. Our next step will be to increase the amount of cells per layer, so from 100 we are going to reach 700, and we are going to reduce the number of epochs from 2000 to 25, since more cells require more time to train:

Hidden Layers	3
Cells	700
Epochs	25
World Cup madness all against Iniesta. #Russia2018	
The mom magician carearos stepping up his goal against Nigeria	
Messi hasn't give the best in this world have step denthed at it is the best midfielder of the days of the match	
Rojo save yout it will be the goving to talk about his stil but to talk about to put him a Russian state.	

Results 6: TWEETS

We are keeping good results and we have even improved the weak point of our text: the sentences. Now they have higher quality, the text makes more sense, although they are still not as good as the original tweets. Let's see what happens if we increase even more the number of cells and the epochs:

Hidden Layers	3
Cells	1000
Epochs	50
Nigeria today against Sweden.That's why Germany are best in the world.Thanks to Toni kross.Well done Germany.	
@FIFAWorldCup @Budweiser @woodyinho Toni kross made free kick	
@zmanbrianzane I get your then but you look at the centre midfield becousers of blessing about Bristiano Ronaldo's Penalty Miss Against Iran	
But FIFA Rule 18 article 26 section 6 states that thy shall not award more than a penalty to an African team at the World Cup of the defenders.	

Results 7: TWEETS

Now the sentences make sense, but it is due to overfitting. The model has become really good at copying the original tweets, and then the text make sense and the loss was really low. The objective of the project is to be able to create new text, not replicate the old one, so for us this is now not an option.

If we train the model with tweets and the Neural Network has 10 hidden layers instead of 3 we get the following type of tweet:

Hidden Layers	10
Cells	100
Epochs	800
Coool Nannna aaaCnacclNa	

Results 8: TWEETS

All the generated tweets are similar to this last one. Increasing the number of layers use to be useful if we need to increase the level of abstraction of a Neural Network. In computer vision's Object Detection problem, multiple layers are used because each layer can be focused on different characteristics of the image, such as vertical lines or corners. In our case we have enough level of abstraction and increasing the layers only increases the number of epochs we need to reach the same results.

3.1 The best model

So, which is the best model we trained? The Neural Network with 3 hidden layers and 1000 cells per layer obtained the best results, but it was just copying the training data. Since the objective is to be able to generate new text, it is better to have worse text if it is original. For that reason, the best model is the last one we used with the e-books, and the first we used with the tweets: 3 layers, 100 cells per layer and about 2000 epochs to train.

Now that we have chosen a model, we just need to embed it in the Twitter bot and execute it to generate new

tweets and publish them.

The Twitter account used in this project is **@BotTfg**.

4 Conclusions and future work

During the realisation of this project we have tried to solve a problem in Natural Language Processing: the generation of text. We have seen what Recurrent Neural Networks are and why they are the best option nowadays to try to solve that problem. We have used the facilities that a Social Network offers to extract data. We also build different architectures of a Recurrent Neural Network using a popular library. We generated and evaluated the text that those architectures produced to understand how the model works, which limitations it has and how to improve the results.

The generation of text is a subject that is still being investigated. We have succeed in our objectives, but the quality of the text generated may be not as good as we expected. Why? There are several things that we could improve to have better text, such as the quality and amount of the input data. But the main reason is the difference between building words and building sentences. Both are sequences of characters, but while we want the model to use exactly the words that already exist (we do not want it to create new words because they will probably be wrong), we want it to generate sentences that were not in the training data. Generating completely new sentences and making them make sense is, in our project, the same concept than producing new words. If we would generate text word per word instead of character per character we would not need to learn the words, but how to put them together in sentences, and the resulting text would be probably better, but the sentences would be still simple. We could improve it which larger datasets, so the model would end up learning the concept of the structure of the language (putting an adjective before a noun, for example), but to reach the highest level of text generation we should need to stop generating in a blind way, without knowing the meaning of the words, sentences and expressions. To reach the highest level we should merge other fields like object detection and audio treatment to our model, allowing it to associate images to words or replace the text by the voice, and also simulating basic needs or desires as we have. The model then could learn the name of an object we show to a camera, and learn to talk (or write) to satisfy the simulation of basic needs, just as a baby learns the concept of food because he is hungry and can see the things that satisfy that need, associate it to the word and generate a sentence to ask for food when he is hungry again, and not randomly.

The future work, then, should not be focused on improving this artificial learning of the language, this is good enough, but to merge the different fields that emulates the human intelligence into one new, more general, area of study.

References

[1] Official website of TensorFlow https://www.tensorflow.org/

[2] Official website of Python https://www.python.org/

[3] Official website of NVIDIA http://www.nvidia.es/page/home.html

[4] Python Programming Tutorials, Harrison (Sentdex). https://pythonprogramming.net/rnn-tensorflow-python-machine-learning-tutorial/

[5] Build A Tweet Bot With Python, waleadesina, https://scotch.io/tutorials/build-a-tweet-bot-with-python

[6] Official documentation site of Tweepy, http://docs.tweepy.org/en/v3.5.0/cursor_tutorial.html

[7] Mining Twitter Data with Python (Part 1: Collecting data), Marco Bonzanini, https://marcobonzanini.com/2015/03/02/mining-twitter-data-with-python-part-1/

[8] Siraj Raval's Youtube channel, Siraj Raval. https://www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A/featured

[9] The Trump Deep BS Quote RNN Generator, Killian's Blog. http://killianlevacher.github.io/blog/posts/post-2016-03-01/post.html

[10] Creating A Text Generator Using Recurrent Neural Network, Chun's Machine Learning Page. https://chunml.github.io/ChunML.github.io/project/Creating-Text-Generator-Using-Recurrent-Neural-Network/

[11] Training an RNN to generate Trump Tweets, David Merrick. https://www.david-merrick.com/2017/06/12/training-an-rnn-to-generate-trump-tweets/

[12] The Unreasonable Effectiveness of Recurrent Neural Networks, Andrej Karpathy. http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[13] Yet another text generation project, JC Testud. https://towardsdatascience.com/yet-another-text-generation-project-5cfb59b26255

[14] Learning Features and Segments from Waveforms: A Statistical Model of Early Phonological Acquisition, by Ying Li. University of California, 2005. http://phonetics.linguistics.ucla.edu/research/YLin$_Diss.pdf$

[15] Vanilla Char-RNN using TensorFlow, by vinhkhuc. https://gist.github.com/vinhkhuc/7ec5bf797308279dc587

[16] Creating A Text Generator Using Recurrent Neural Network, by Trung Tran. November 14, 2016. https://chunml.github.io/ChunML.github.io/project/Creating-Text-Generator-Using-Recurrent-Neural-Network/

ANNEX

A.1 Original Gantt diagram of the planning of the project