Machine Learning I
(MATH80629)

Final report

Presented to
Laurent Charlin

by

Estefan Apablaza-Arancibia	(11271806)
Adrien Hernandez	(11269225)
Idris Selmi	(11219871)

Master of Science (M. SC.)

Data Science & Business Analytics
Québec, Canada

August 30, 2020

Contents

1 Introduction

The purpose of this project were to build neural network algorithms that will be used in the field of Natural Language Processing (NLP) in order to fit a specific dataset of human interactions.

Deep learning algorithms have been increasingly outperforming more traditional models in the past years in the task of text and language generation. Thus, we tried to employ different deep learning models in order to generate sarcastic sentences responding to neutral input comments written by humans.

All the experiments we conducted during this project were based on past study that mostly try to classify sarcastic comment. Therefore, it were a dataset already cleaned of sarcastic comments that we didn't need to werete time on data gathering.

During this project, many considerations were carefully analyzed in order to decide the usability of models and their parameters. Indeed, the generated sentences needed it to be syntactically and grammatically acceptable, as well as correlated to the input sentence meaning. That being said, the neutral input sentences were very important if we wanted our model to be useful. However, these constraints were particularly difficult to respect in the context of sarcasm generation, and we will show through our report how we tried to deal with it.

This paper is structured as follows: we start, in section 2, by giving a description of our project, including our research question, motivation and dataset. In section 3, we provided a survey of related work conducted in the field of text generation. We explain our process in section 4, including the pre-processing steps, the model architecture, the training steps and the predictions. Then, in section 5, we empirically report the results we obtained during this study and in section 5.1, we give a description of the different models that we tested and compared. Finally, in section 6, we summarized our findings and highlight possible future directions of investigation.

2 Description

2.1 Research Question

Our research question is: *Could create and tune a machine learning algorithm able to generate and output a sarcastic sentence as an answer to a neutral input sentence.* We tried to understand and focus our study on which words, type of sentences and machine learning model and parameters drive the algorithm to output sarcasm.

2.2 Why Sarcasm?

The idea of sarcasm may sometimes be abstract and difficult to express for humans themselves and we were interested to see how a machine learning algorithm could be tuned to deal with this.

2.3 Dataset

Our dataset has been firstly introduced at Princeton University in New Jersey and has been built for sarcasm research and training deep learning algorithms [1]. It contains approximately 1.3 million labelled comments in English scrapped from the famous website Reddit. The original features include **"Parent comment"**, which is the neutral parent comment. **"Comment"**, being the sarcastic comment written by a user to answer the Parent comment and **"Score"**, representing the comment score given by other Reddit users. We then added two additional features in order to control the sequence length: **Comment word count** and **Parent comment word count**.

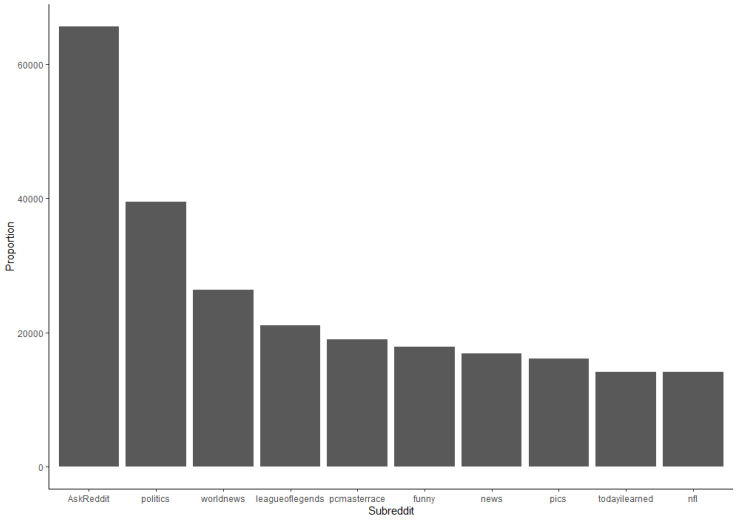


Figure (1) Top 10 Subreddits per number of comments

An example taken from the training dataset: **Parent comment:** *Yeah Ragen, you should be rewarded for signing up for a race that you will never participate in.* **Sarcastic comment:** *Absolutely people deserve to be rewarded for preparing for a significant task which will be a great accomplishment once finished.*

Being aware that sequence-to-sequence algorithms output quality were particularly sensitive to the quality of the data it were trained on, we tried to take into account the notion of garbage in, garbage out. We will give more details about this in the Prepossessing section.

2.4 Model and Algorithms

Our model consisted of a sequence-to-sequence model with an RNN encoder and an RNN decoder.

3 Related Work

3.1 RNN

Sarcasm generation can be put in place through the use of Recurrent Neural Networks. These networks, whose type of architecture allows to have feedback loops and a memory, have shown great results solving natural language processing tasks [2]. Moreover, unlike feed-forward neural networks, this type of algorithm also has the advantage to handle input and output data that don't have the same length (such as seq-to-seq models), which were key for our sarcasm generation task. However, as we discovered, the simple recurrent unit does not offer the best

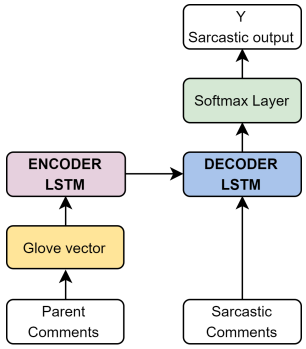


Figure (2) Sequence-to-sequence model

results mainly because of the effect of vanishing and exploding gradients that will cause the network to not being able to learn. In practice, we discovered that GRU and LSTMs networks became the algorithm used by default most of the time.

3.1.1 LSTM and GRU

LSTMs architecture were composed of a cell having 4 interactive layers: **a cell state, an input gate, an output gate and a forget gate**. giving the ability to add, remove or let the information get through the neural network algorithm. This architecture prevents the problem of vanishing gradient and therefore, prevents the model to stagnate during its learning phase. [3]

Gated recurrent units were similar to LSTMs as they also allow to add, remove or change the information inside the unit itself [4]. However, the GRU cells do not require as much computation and seem to offer a gain of computational requirements while offering quite similar results.

3.1.2 Bidirectional RNNs

As seen in the results gotten in the article [5], Bidirectional RNNs were a very simple architecture yet powerful in the context of natural language processing. Here, the purpose were to get the cell hidden state by reading the sequence from left to right and from right to left. In other words, we were calculating the forward and backward h state at the same time. The result of both computation were then concatenated to get the output desired. In NLP, this helps the model understanding the context of words coming at the beginning or at the end of sentences.

3.2 Sequence to Sequence Models

3.2.1 Encoder Decoder Models

Encoder decoder models were a more complex version of simple RNN, and were able to generate output sentences that were meaningful and that don't have necessarily the same length [6]. One-hot encoding each word were not a good representation for words. Vocabulary size is the number of distinct words. The feature vector size is the embedding dimension.

Teacher Forcing

In order to help our model to output better sentences, we used the teacher forcing method. This method showed impressive results in a work similar to our project in [7]. Matter of facts, we decided to try this method for our sarcasm generation algorithm. The analogy were to behave with the model as a teacher and his students. The teacher corrects each sequence which corresponds to training of our decoder RNN and eventually improves the word generation at each step.

Greedy Selection

The greedy selection were a very simple approach we decided to use when generating output sentences. This approach will only select the word having the biggest probably to appear after another word. This selection

were very fast as the decoder will automatically output the word being the most likely to appear. However, we were aware that this approach can be far from being the most optimal as some words such as "the" do not have a specific word that comes after.

Attention mechanism

Concretely the attention mechanism were a way to remember the encoder outputs relations to specific words. Without attention mechanism it were hard to remember which encoder states had the more impact. This were especially useful for tasks such as translation since it helps to create syntactically correct sentences. It also gives to the model a better understanding of what to put after a word like a verb or a noun like shown in the figure 3.

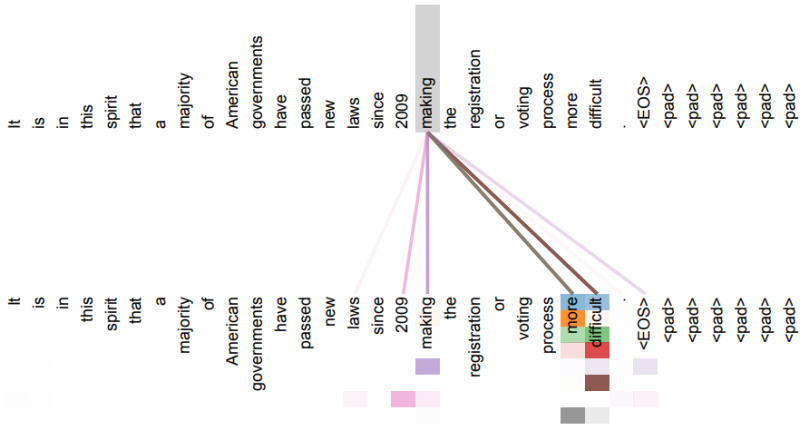


Figure (3) Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'.[8]

3.2.2 Transformer Model

This recent model got impressive results in the task of text generation, outperforming stat of art algorithms such as RNNs. However, it were a very different encoder-decoder model, which were not based on recurrence but were fully based on attention mechanism [8]. This model should be able to perform very well for our task of sarcasm generation.

3.3 Word Embedding

Word embedding were essential in natural language processing as it allows to give a representation for words into a vector space that a machine learning algorithm can understand. In this way, each word becomes a n-dimension vector which, once combined, form a n-dimension matrix representing itself the original sentence.

3.3.1 GloVe

GloVe [9] stands for Global Vectors and it were for Word Representation. It exists several pre-trained glove models that were available online and allow to get a great word embedding without having to train our own glove model. We decided to use the "pre-trained Glove on Twitter data 27B 100d corpus". This model, trained on 27 billion tweets, creates for each word a vector of size 100 dimensions, where the Euclidean distance between each of the words depends on their meaning[9].

The main dissimilarity between the techniques of Word2Vec and GloVe were their quality of association

between different databases. They both grasp the association between each element via a different calculated vector. For some databases the GloVe seems more appropriate. GloVe captures global statistics and local statistics from a corpus, in order to find words vectors. Indeed, it focuses more on an entire sentence to determine associations while Word2Vec combines these elements independently of the corpus.

3.4 Sentence Generation Using RNNs

Finally, all this concept were combined in an elaborated scientific article that pretty much generates punch line jokes instead of sarcasm. Globally, they were able to get an interesting output by using a combination of some algorithms such as Attention Mechanism, Glove and Encoder-Decoder LSTM models. Their evaluation part were interesting as they try to deal with the difficulties to measure the performance of a good sentence generated. Indeed, they chose to have human ratings as the main evaluation of their model's output. Moreover, their conclusion gives several interesting information about the importance of having good quality data for input if we were hoping for a good-quality output. They mention the fact that their corpus were not large enough and that their model only learned jokes from one author, which leads to a lack of diversity.

4 Process

4.1 Pre-Processing

The "garbage in garbage out"(GIGO) concept were the misuse of imperfect data to create or build scientific models that in the end were useless because the data were simply inaccurate. The important part were really the imperfect data. In order to prevent or at least to minimize the GIGO we had to concentrate our effort in standardizing the data which in our case were sentences. In python, the NLTK library were vastly use to handle some part of this process by taking care of the UPPER CASE word, the negations handling, the heavy punctuation, part-of-speech tagging, interjections and many more. An example of raw sentence from the dataset [**"hahahaha oh man, I'm guessing the downvotes are legit"**] were modified in order to look like this [**"haha", "oh", "man", "I", "am", "guessing", "the", "downvotes", "are", "legit"**]. We also tried as much as possible to remove swear words sentences but they were still a lot of offensive comments .

4.2 Architecture

The figure 2 represented an overview of our desired model. In reality, most of the encoder decoder model required more detailed design thinking. Mainly because of the limitation of the library use such as Keras. We tried different configurations and got multiple errors because of the embedding size or the layer were not correctly configured. A nice tool that we found were the utility functions provided to plot Keras model such as the figure 4. This helped us tremendously and helped us understand the encapsulation of each layer of the model that were building little by little.

4.3 Training

Steps

The steps in order to train the model were not trivial since we had a lot of different way to do it theoretically but were not always feasible with Keras. Here is a bird view of the steps engaged:

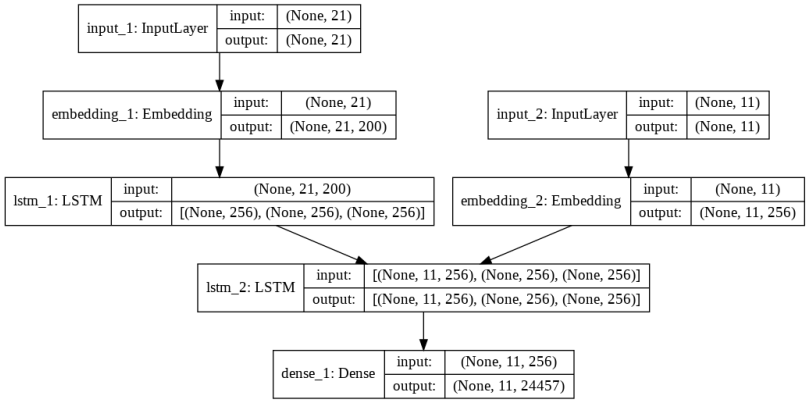


Figure (4) Keras model representation

1. **Dataset separation** : Considering that data were already pre-process, the dataset were split in two : one part for the training and the second one for the validation.
2. **Tokenization** : Each input and output sentence were decomposed in word format and assign to a number. This were the tokenzation process.
3. **Labelling** : Some specific labels labelled were added in order to indicate the start of a sentence ($< SOS >$) and the end of the sentence($< EOS >$).
4. **Padding** : Unfortunately, Keras works with constant-sized sequences and ,therefore, some padding adjustment needed to be done in the input and output using the maximum length.
5. **Word embedding** : Loaded the pre-trained word embedding (GloVe [9]) and labelled the existing word found in the word vector representation.
6. **Mapping** : Created a word mapping between the real words and the corresponding index.
7. **Filling** : Created an embedding layer filled with the words founded in the encoder entry only.
8. **Training** : At this step, we had everything to start the training of our model. One important aspect of the training were that since we were using the teaching forcing method we needed to transfer the weights value to the LSTM with only one input.

Environment

At the beginning the teams were using its personal computer to train the model. The latency was tangible because one epoch to train the model were approximately 10 min. During the course, we discovered that Google offers a system called Colab. Not only were we not using our machines but they offer super powerful machines with Graphical Processor units (GPU) and Tensor Processor Unit(TPU). Using the Colab, machine we were able to compute 20x faster than our own machines about 30 sec/epochs.

4.4 Prediction

Steps

The steps, in order to predict, were a bit different and we had to learn how to use ".h5" file format which were an hierarchical data format that encapsulates trained models.

1. **Model loading** : This part were basically to load the model and distribute it to each layer (e.g. encode inputs, decoder inputs, embedding, dense, etc.). We needed to do that because we reconstructed a new LSTM (1 input instead of one hot vector) and assigned the weight of the initial trained LSTM.
2. **Mapping** : We reconstructed the mapping between the real words and the corresponding index in the matrix. This helped us reconstruct human readable sentences.
3. **Tokenization, Padding** :The two processes were the same as before but for the new input.
4. **Prediction loop** : In this loop we predicted one word at a time until the model generated the end of the sentence($< EOS >$). We used the mapping to give a meaning to the index and create a sentence.

5 Results

This section intent to illustrate the different models created and some concept that we used in order to benchmark them. The results were not always comprehensible.

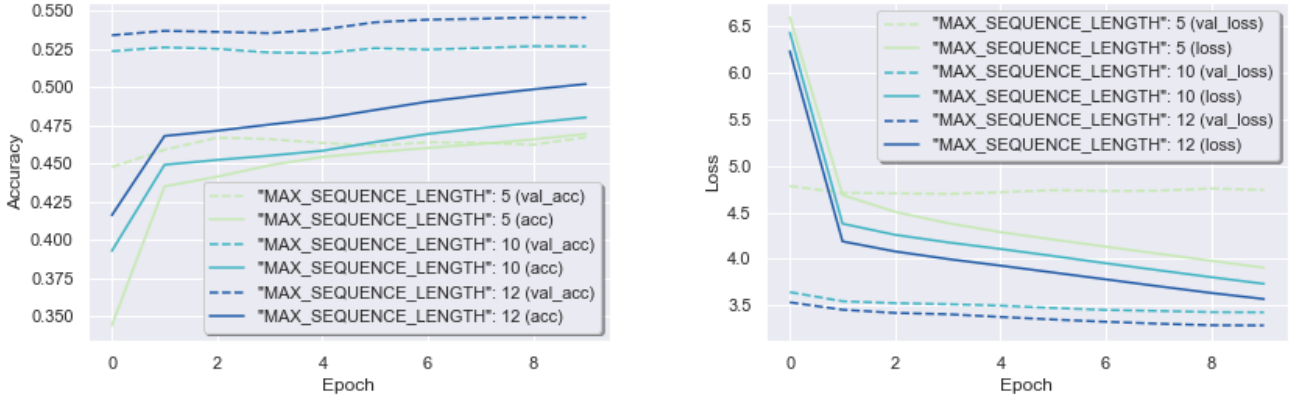
5.1 Model Benchmark

In order to compare which model were the best to start generating we started comparing the models by their capacity. At first we tried multiple combinations of scenarios recommended by the literature review. It were the easiest and fastest way to know if we were getting a more proficient model. However, these assumptions were made with the idea that the train could be validated but the reality were not really the case.

In order to compare different models, we tried multiple configuration modifying the hyper-parameter values. Each time there were an increase in the capacity we decided that it would be the right choice. The hyperparameters were tested in this order : **Sequence Length**, **Latent dimension**, **Optimizers** and lastly **Embedding dimension**.

Sequence Length

The sentences were not all the same length therefore the dataset had to be trimmed. In an ideal world, with unlimited resources we would have use everything an sized according to our longest sentence (around 900 words) but this were not the case. We tried different scenarios going from 5 words to 12 like shown in the figure 5. The conclusion of this hyperparameter were that we needed to aim has high as possible since we clearly saw an increase in the capacity of the model.

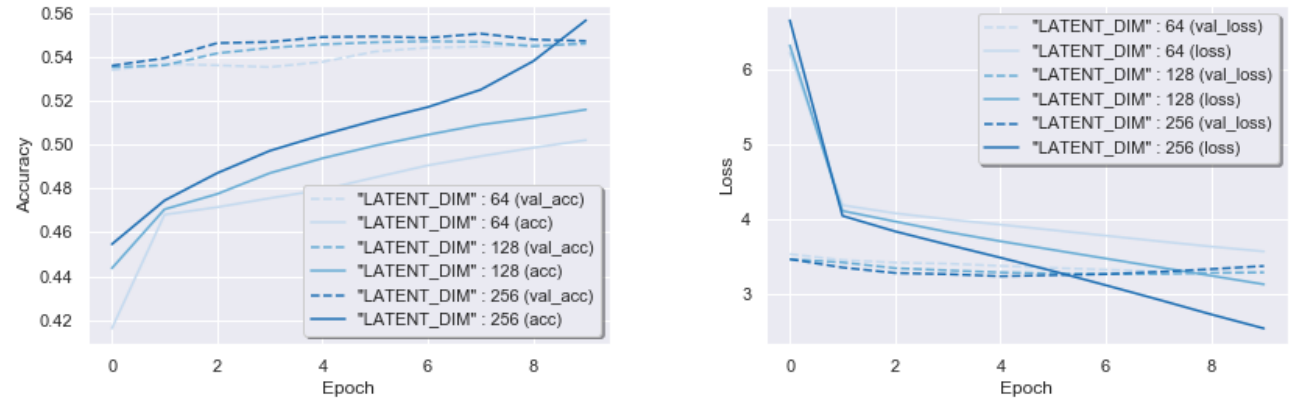


(a) Accuracy (b) Loss

Figure (5) Sequence length hyper parameter

Latent dimension

The latent dimension were basically how much compress information were giving from one LSTM (encoder) to another (decoder). We assumed at the beginning of this project that for this application the bigger the dimension the better. However, in reality it were the opposite. Our understanding on this topic were that the latent space between the two models were a data compression and the general idea were that the smaller it were the more it generalized the training data and gives a higher learning capacity [10].

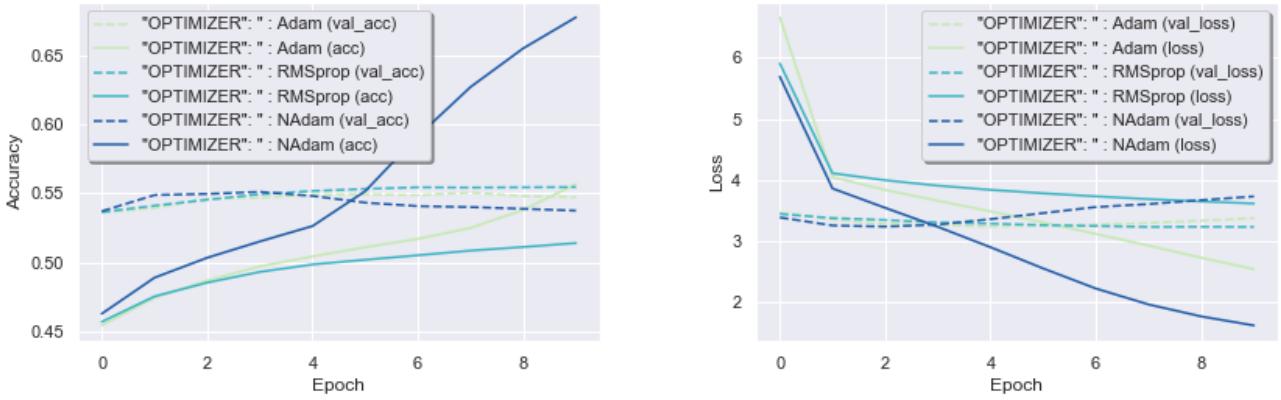


(a) Accuracy (b) Loss

Figure (6) Latent dimensions hyperparameter

Optimizer

Multiple optimizer exists in the documentation [11] to regularize the training process. At the beginning, the team tried without any optimizer and quickly realized that our model capacity decreased drastically. It were a shame that we were not able to try all of them because but we decided to go with the most popular optimizer being : *RMSprop* , *Adam* and *Nadam*. The Keras library recommends the *RMSprop* optimizer as the default parameter and our model seemed to have increased in capacity (Fig. 7).

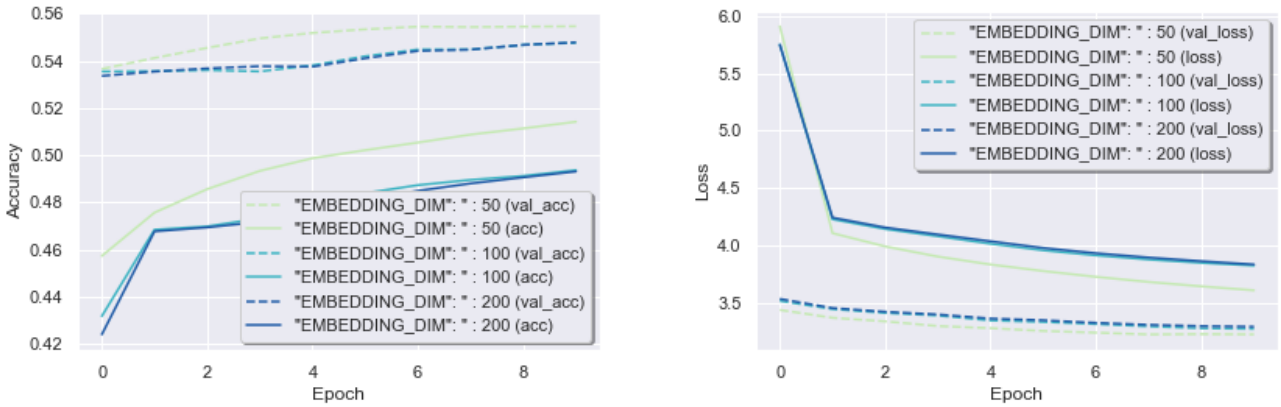


(a) Accuracy (b) Loss

Figure (7) Optimizer type hyper parameter

Embedding dimension

As explained earlier in section 3.3, we selected the model trained on multiple messages. This model were available on different dimension going from 25 to 300. We realized that this value did not affect much the capacity of the model when the value were set to 100 and 200 but rather decreased if set to 50. The way we interpreted this were base on the fact that after a certain amount of precision (value set to 100) the additional vector value doesn't provide information to the model and were therefore were useless.



(a) Accuracy (b) Loss

Figure (8) Embedding dimension hyper parameter

Model selected

In summary, the number of hyperparameter combination were pretty overwhelming and it were not trivial to keep track of all the effects of different combinations. However, after multiple attempts in reorganizing the dataset, modifying the hyper parameter values and different settings we finally selected our best model based on the capacity. The figure 9 shows a fraction of these attempts and highlights the selected model. In order to achieve a good amount of epoch before over fitting, we also included in the code the feature called "early stopping" which were pretty self-explanatory. Basically, it accepts a number of maximum epoch (e.g. 200 epochs) and iterates until the validation accuracy and the accuracy converge. The final hyperparameter value were shown in table 1.

Figure (9) Best model selected

Hyperparameter	Value
Maximum Sequence length	6 Words
Latent dimension	64
Optimizer	RMSprop
Embedding dimension	200
Batch size	64

Table (1) Hyperparameter

They were multiple hyperparameter that we did not discuss such as the batch size which defines the number of samples that will be propagated through the network. We did not change this value but the major consideration were to set it as a binary base number in order to fill the GPU memory batch size accordingly[12]. Another hyperparameter that we explored were the dropout at different values (0.1 to 0.5) but, unfortunately, it wasn't affecting the capacity as much as the rest.

5.2 Sarcasm Generated

Once we had a selected model, we needed to verify the generated sarcasm. The table 3 shows some output sentences. We tried our best to select the most politically correct responses per se:

Input:	Sarcasm:
"What an awful human being"	"well it's not very american of course"
"Should have gotten a bike"	"come on dude, we all have forgotten"
"It's all Chinese to me"	"it's a woman"
"Today were a good day"	"typical chelsea fan"

Table (2) Sarcasm generated examples

Even though we selected the model based on the capacity aspect we still had to enhance the bot's responses. The model capacity were not good enough to score a sarcastic bot sentence. We explored some multiple combinations of parameters to see how it affected the quality of the sentence generated based on the presence of sarcasm, the sentence syntax, the correlation between input and output, and finally, how offensive it is. It were very hard to subjectively determine whether our results were considered good since sarcasm were a fuzzy concept, but we had the opportunity to obtain particularly good results. Here are some outputs scored with the criteria explained earlier:

Input: **"What an awful human being"**					
Model	Sarcasm:	Syntax	Correlated	Sarcastic	Offensive
1	"because that makes it a real way"	0	0	1	0
2	"but it's not a real man, the white people can be"	0	1	1	1
3	"but it's okay because you said no offense"	1	0	0	0
4	"yeah, but that's not nearly as important	1	1	0	0
5	**"well it's not very american of course"**	1	1	1	0

Table (3) Comparison of the quality of sarcastic outputs between models

We verified that these sentences were not in the original database. Of course, this brief analysis were completely subjective and left to the discretion of the jury. For example, the model we selected were number 5 from the table 3. Some member of the team found it offensive but not the majority so the final score for *offensive* were 0. Another group people might have a completely different score chart.

6 Conclusion

This project were a short but very concise approach of text generation using recurrent neural networks. We were conscious that our model were pretty simple, and that much more complex methods can be applied in order to obtain a very impacting result for the task of sarcasm generation. However, the fact that we decided to focus on an NLP task allowed us to understand a lot of new concepts that were unknown to our team.

6.1 Findings Highlight

We discovered that our encoder-decoder model were quite sensitive to the hyperparameters we selected. Indeed, the quality were varying a lot depending on what we were modifying in the code. Moreover, we

realized the importance of having a "clean" dataset to train our model if we wish to improve our model as some of the output sentences created were offensive, or even discriminatory. Moreover, we suspect this problem to not only come from the dataset itself but also from the pretrained GloVe we used to embed our words.

6.2 Limitation

The biggest limitation for this project were the time constraint as we were trying so many new things we've never tried before. It required us several weeks to be able to get a first output, as it were the first time we used Keras and Tensorflow. In addition, the level of the team being quite heterogeneous, it were difficult to bring added value for certain members. The limitations of our model also come from the dataset itself since it were not composed of easy English sentences, but contains a lot of urban slang and sayings not always understandable out of the context of the specific topics were they were written. Moreover, we had to only use a small part of the entire dataset as it would have required much more computational power.

6.3 Future works

The text generation is something relatively new and the technique are still evolving. It is clear that our project those not have a concrete viable business future but, in an ideal world, if we had more time to create a sarcastic bot, this would have been some interesting path to explore :

- Add deepness in the RNN by adding another layer of LSTM and using a bidirectional structure
- Add attention mechanisms into the LSTM models
- Other recent models also using attention such as Transformers could be very interesting to try
- Change the greedy search by a beam search approach, in order to improve the quality and the diversity of the sentences generated
- Bridging Meanings Through a Shared Space. In other words, to have one encoder but multiple decoder
- An alternative of teacher forcing method is reinforcement learning which gives a reward and negative reward(punishment)
- Create our own word embedding directly based on the comments from Reddit

References

[1] M. Khodak, N. Saunshi, and K. Vodrahalli, "A large self-annotated corpus for sarcasm," 2017. 2

[2] H. Ren and Q. Yang, "Neural joke generation," 2017. 2

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997. 3

[4] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. 3

[5] K. Al-Sabahi, Z. Zuping, and Y. Kang, "Bidirectional attentional encoder-decoder model and bidirectional beam search for abstractive summarization," 2018. 3

[6] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. 3

[7] A. Lamb, A. Goyal, Y. Zhang, S. Zhang, A. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," 2016. 3

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. 4

[9] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162 4, 6

[10] S. Wiewel, M. Becher, and N. Thuerey, "Latent-space physics: Towards learning the temporal evolution of fluid flow," 2018. 8

[11] F. Chollet *et al.*, "Keras," https://keras.io, 2015. 9

[12] Y. Kochura, Y. Gordienko, V. Taran, N. Gordienko, A. Rokovyi, O. Alienin, and S. Stirenko, "Batch size influence on performance of graphic and tensor processing units during training and inference phases," *Advances in Intelligent Systems and Computing*, p. 658–668, Mar 2019. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-16621-2_61 10