INFO8010: The story-telling bot

Victor Mangeleer,[1] **Axelle Schyns**,[2] and **Lucie Navez**[3]
[1] *victor.mangeleer@student.uliege.be (s181670)*
[2] *axelle.schyns@student.uliege.be (s180598)*
[3] *lucie.navez@student.uliege.be (s180703)*

I. INTRODUCTION

Once upon a time, three students during lockdown in their respective room were very unhappy with the ending of the well known *Harry Potter* series of books. Seriously, seven books worth a month of reading for a small kiss between Ron and Hermione ? Who would ever be okay with that ?

As an ultimate attempt to save their crushed hopes, they arrived at the conclusion that they should try to generate the rest of the story by praying that deep learning would succeed where J-K Rowling had failed. In order to make their dream come true, they entered the mysterious and somewhat magical world of *Natural Language Generation* where they, poor muggle born, had to face many enemies: non decreasing loss, defective data loader and terrible results before having an opportunity at fulfilling their honorable quest. Indeed, such as Harry trying to defeat you-know-who, the path in front of them is tortuous. Anyway, these three students are us and, as you probably understood, our purpose in this project is to generate and flesh out the story of Harry Potter.

The motivation behind the idea of implementing our own neural network to generate more stories about Harry Potter is that it will be trained only on text related to the Wizarding World as well as being put down by J-K Rowling herself. Therefore, one can hope that the context of the story and the way it is written will be more precise. As an example, the following text has been generated by an open text generation API made available by deepAI and based on GPT-2.

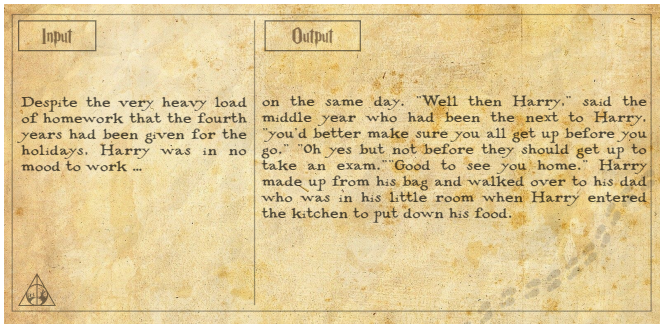

FIG. 1. Illustration of the story made by the text generation API of deepAI using GPT-2 has a backbone. The input sentence is on the right whereas the output of the neural network is on the left.

As it can be read in Fig. 1, the neural network managed to capture a bit of the sentence's context. However, since it has not been trained to understand deeply the world of Harry Potter, it started to talk about his dad which is supposed to be dead since he was one year old.

The opening gambit of this project is to simply learn more about the encoder-decoder architecture. Indeed, it was important to determine if the network will be trained to predict words characters per characters or complete words immediately. After testing the first approach, whose notebook is available on the *GitHub* page of the project, and seeing the results it was concluded that the main focus should be on a neural network using a *words-2-words architecture*.

Afterwards, a lot of time and thoughts were placed into *the conception of the dataset*. As a matter of fact, this step is often overlooked by the beginner computer science community even if it plays a major role in the outcome. Then, we started the design of the architecture of the network.

Once the architecture of the network was finished and in order to gain some insight regarding the *influence of the most relevant parameters* for the training of the story telling bot, several small networks have been created and trained to compare the effect of these parameters on their performances.

Eventually, in an effort to improve the training sessions, a *learning rate scheduler* as well as a layer of *attention* inside the network have been added.

Finally, all the observations made are going to be used to create the best version of the story telling bot possible, it's nickname is *Grope* and the text it has created will be displayed at the end of this report.

II. RELATED WORK

Some parts of this project were inspired by already existing works, since there exist many works related to Natural Language Processing and Natural Language Generation, either by retrieving existing structures or simply by using them as inspiration to code our own architecture. Thus, in the following section, the parts of the project that were conceived thanks to external sources or projects will be described.

While fields like Machine Translation or Abstractive Summarization currently attract most interest, several works have also looked into the automatic generation of text from the start of a sequence or paragraph. Indeed, you can find on internet several websites where you can automatically generate text like *Goose.ai* which allows to generate as many text as we want from a sentence given to it. In addition to that, there also exist models like *GPT-3*, which is used in OpenAI and can generate Python code and other incredible things. As a matter of facts, many papers are treating the subject and introducing different ways of tackling it such as [1], [2], [3], [4] and [5].

One of the main step of such a project is the pre-processing of data which was widely inspired by the YouTube videos of *Aladdin Persson*. Indeed, one of them [6] inspired the introduction of tokens in the sentences as described in section III. Another of its video ([7]) gave insight on the embedder size and inspired us to restrict our vocabulary to the words used in Harry Potter, which is around 20000 words, instead of the complete vocabulary provided by our embedder *GloVe*, which is around 40000 words. Actually, the architecture uses *GloVe*[8] as an embedder (see section IV A) since it provides an efficient way to represent words as vectors.

Furthemore, other sources from the web such as [9],[10],[11] inspired the steps that were followed for the rest of the dataset creation, like the tokenization, lower-casing, and stop-words removal. In the context of this project, additional pre-processing steps such as stemming and lemmatization were not relevant and thus were not implemented.

In the case of the conception of the SBOT, in order to understand more deeply how the encoder-decoder architecture works, the books [12], [13] and [14] were used. In addition to that, the attention mechanisms used in the architecture is based on the one described by [15].

Finally, for the metrics, multiple sources such as the papers [16], [17] and [18] were used. There are also the YouTube videos [19], [20], [21] and the websites/articles [22], [23], [24], [25], [26] which allowed us to guide us on what to use. In order to implement all these metrics, most of them were accessible throughout several GitHub repositories such as [27], [28], [29].

III. DATASET

A. Construction of the dataset

The first step consisted on obtaining the books and fortunately, one possessed the raw text files of the 7 volumes of the Harry Potter series on its GitHub [30]. The text contained in these books is the original one thus it contains a lot of useless or confusing information unusable for training, such as the numbers of the pages, that had to be pre-processed so that our model could learn optimally from it. According to our calculations, the sentence length distribution in each book of the series is as presented in Tab. I.

Title	**Sentences**
Harry Potter and the Philosopher's stone	6 497
Harry Potter and the Chamber of secrets	6 758
Harry Potter and the Azkaban's prisoner	9 207
Harry Potter and the Goblet of fire	14 747
Harry Potter and the Order of the Phoenix	17 517
Harry Potter and the Half Blood prince	12 305
Harry Potter and Deathly Hallows	14 765
Total	81 796

TABLE I. Collection of the Harry Potter series and the number of sentences inside each book.

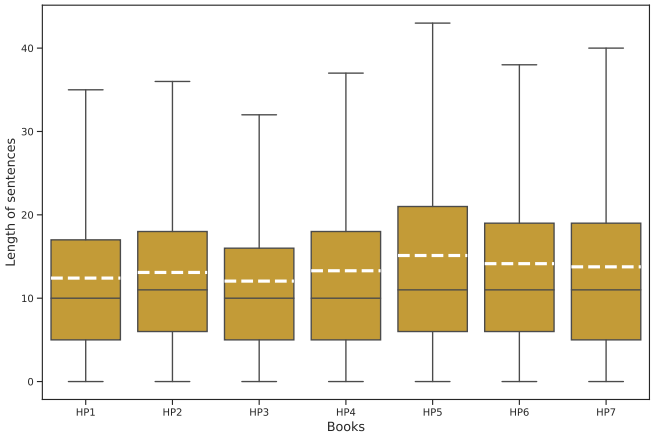

FIG. 2. Representation of the sentence length of each book in the Harry Potter series where the average sentence length is represented by a white dotted line and median is represented by the plain black line.

Therefore, a discussion regarding what to feed the model during training can be made. Indeed, there were several possibilities :

- Feeding words of a *single sentence*.
- Feeding words of a *paragraph* which is composed of an arbitrary number s of consecutive sentences.

In the first case scenario, the SBOT will be trained to predict the end of a sentence while knowing its beginning. However, in the second possibility, the network will also be trained to predict further sentences based on the context of the first.

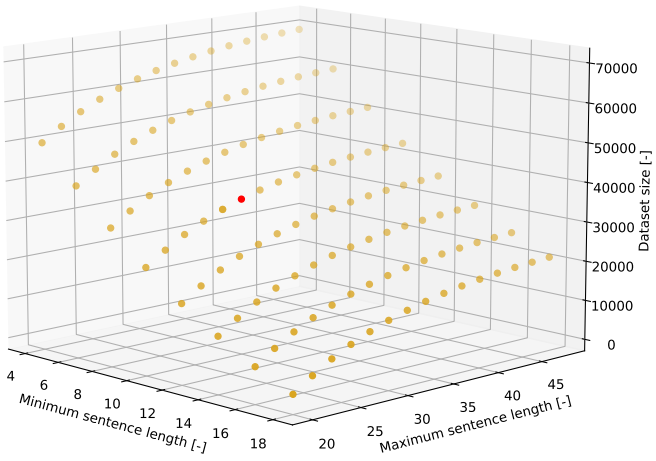

FIG. 3. Size of the dataset obtained for different pairs of values of the minimum sequence length and the maximum sequence length. The red dot indicates the value picked for training the network.

For the time being, the implementation is based on the first option, i.e. feeding words of a single sentence as input. Indeed, the reason behind this choice is that feeding words of a paragraph composed of an arbitrary fixed number of sentences creates input/output pairs that can get very big. This implies that the network capacity must be enlarged by increasing the hidden layer size in particular. Thus, the training get significantly longer and makes it impossible to obtain good results in a reasonable time. In conclusion, for now the network only aims to predict the end of a single sentence based on a sequence of words as input.

The next challenge was to define the size of this input sequence. As previously, there is a trade-off to make: the input size should be large enough so that the network can train significantly while not being too small. Otherwise, the network would not be able to sufficiently capture the context of the sentence. Even if the dataset must remain big enough, it must not exceed a certain size or else the same problem as previously arises and the training computation time becomes unreasonably long.

One can decide on which input size allows to get the best results by checking the Fig. 3. Indeed, as it can be seen in Fig. 2, most sentences of the books have a length between 5 and 20 words, and the mean is around 12 words per sentence. However, it can be observed that there are still many sentences for which this number is way smaller, or way bigger. Thus, by bounding the maximum sentence length to 20, a significant part of the dataset is lost which leads to the conclusion that a higher value would be appropriate to improve the training quality.

In conclusion, one chose to go with a minimum sentence length of 10 as well as a maximum sentence length of 30 words to ensure both a big enough dataset that would allow for the training to be significant and a small enough input and output pair size so that the training would not be excessively long. In this configuration, it can be seen in Fig. 3 that the size of the dataset is a bit more than 40000 sentences which means that around 60 % of the original set has been kept! Furthermore, for each sentence, the first 8 words will be considered as enough to capture the context of the sentence and they will be fed to the SBOT as input. For the predictions, the network has to make 22 of them, at least 2 predictions are expected to be words while the rest might be padding tokens or also words depending on the sentence size of the book is has been taken from.

B. Pre-processing of the data

As said before, the raw text books contained some information that one had to get rid off to ease the training and obtain better results. The following steps to pre-process the data were followed:

1. *Removing the English contractions* : This was necessary because not removing the contractions from the vocabulary would imply that some different words have the same meaning, for instance : "we'd" would be the same as "we" and "would", and the embedder would contain different entries for similar meanings (see section IV A). Therefore, reducing the embedder size is important since it defines the size of the last layer of the decoder and thus, the smaller the vocabulary, the smaller the last probability vector, that allows to make the predictions which leads to a more compact architecture of the network.

   Another reason to do so, is that a few contractions are recognized by the embedder, but not all of them. For these unknown contractions, the embedder lookup returns the `<UKN>` token : all these contractions will have the same "meaning" from the SBOT's point of view, and it would start learning this `<UKN>` token, since it would occur quite often. This kind of behaviour must be avoided.

2. *Removing useless characters* : In order to clean even further the dataset, the following characters have been removed from the raw text to avoid confusing the SBOT during its training :

   [, - ; : & \r " “ \n]

   It was deliberate to not remove the punctuation symbols such as "?", "!", and "." yet, because they were still needed for the next step of the pre-processing.

3. *Splitting the raw text into sentences* : This can be achieved by using the function `nltk.tokenize`. Indeed, it recognizes sentences by finding the punctuation symbols "?", "!", and "." and then split them into a huge list.

4. *Removing of the last punctuation symbols* : Indeed, the symbols "?", "!", and "." can now be removed since they will be replaced later on by tokens.

5. *Transforming the whole text to lowercase* : As a matter of fact, it allows to avoid duplicate entries due to an upper case at the beginning of sentences, for instance : "Castle" and "castle" would produce two different entries in the embedder, and, as explained above, it is preferable to minimize the embedder size.

Finally, after these steps of pre-processing, the final shape of the dataset is obtained by defining the input and output pairs. As explained in the former section, one decided to go with a single sentence. In order to delimit the sentences and put the dataset in a form that the SBOT could interpret, one must use tokens :

- `<SOS>` - *Start Of Sentence* : It allows to detect the start of an input sentence fed to the model, thus these tokens are placed at the very beginning of the input sentences. In addition to that, they will be used as first input by the decoder for it to make predictions.

- `<END>` - *End Of Sentence* : It allows to detect the end of sentence returned by the model, therefore these tokens are placed at the end of an output sentence. When the SBOT is making prediction, if it outputs an `<END>` token, this implies that, according to it, the sentence is finished.

- `<UKN>` - *UnKnowN* : When a sentence is embedded, if a word is unknown to the dictionary, its vector representation will be the one of the token `<UKN>`. Therefore, while training, the network can interpret it as anything.

- `<NTD>` - *Nothing To Do*: A token to pad the sentences that are smaller than the defined maximum sentence length.

The input/output pairs were then composed by taking the first `inputSize` words as the input, and the remaining words of the sentence as the output with a `<SOS>` token at the beginning, a `<END>` token at the end, and padded with `<NTD>` tokens so that it has a size `max_sentence_length`. The combination of all these operations are summarized and represented in Fig. 4.

IV. ARCHITECTURE

A. Embedder

First of all, the sentences that will be used to train our model must be transformed into numerical values. Indeed, it is not possible for a neural network to train on a data represented with words.

In order to convert a word into its associated vector representation, one can immediately think about using an *embedder*. In fact, an embedder layer in the domain of *natural language processing* is often used to create a mapping between words and their translation into a low-dimension vector space. If this layer is trained well, it can create very intelligent mappings by making similar words have a close representation in terms of vectors.

Instead of creating and training from scratch an embedder, the embedder *GloVe* has been chosen as a backbone for our recurrent neural network. The model is an unsupervised learning algorithm for obtaining vector representations for words and this is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. The version of *GloVe* chosen has been trained on millions of web pages and translates words into vectors of 50 components.

However, it is important to take into account that the vocabulary size of this embedder ($\sim$ 400000 words) exceeds by far the total vocabulary of all seven books combined ($\sim$ 13000 words). Therefore, since the size of the vocabulary plays a major role in the classification layer of the network, it is really important to reduce its dimension as much as possible. As a matter of fact, the final embedder that will be used corresponds to a truncated version of *GloVe* where only the words used in the books are inside the dictionary.

In the case of unknown words such as *muggles* or *you-know-who*, it made sense to create their own representation in the vector space defined by the embedder. Unfortunately, for the sake of time, the vectors have been initialized randomly and placed inside the dictionary. As a consequence, they will not be placed along similar words in terms of meaning and this might affect the results at the end of training. However, this is the best solution since the other possibility was to simply associate these words to a same vector representing unknowingness.

In addition to that, the tokens used for padding the sentences presented during the construction of the dataset also have their own unique representation inside the embedder.

He was a big, beefy man with hardly any neck, although he did have a very large mustache.

Mrs. Dursley was thin and blonde and she'd nearly twice the usual amount of neck.

The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere.

input	output
<SOS> he was a big beefy man with hardly any	neck although he did have a very large mustache <END> <NTD>
<SOS> mrs dursley was thin and blonde and she had	nearly twice the usual amount of neck <END> <NTD> <NTD> <NTD>
<SOS> the dursleys had a small son called dudley and	in their opinion there was no finer boy anywhere <END> <NTD>

FIG. 4. Constructions of the input and output pairs based on the initial raw text. In the original text, the characters and contractions that are going to be modified by pre-processing are highlighted in **yellow**. The dataset pairs are composed as follows : the input is highlighted in **teal**, and the output is highlighted in **yellow**. Finally, it is important to notice that the `<SOS>`, `<NTD>` and `<END>` tokens have been added to the original text and are highlighted in **burgundy**.

B. Encoder

Once the words are embedded, the sentence must be encoded using an encoder layer represented by a gated recurrent unit. The purpose of an encoding layer is to capture the context of the sentence and represent it with the recurrent state h^* of the GRU.

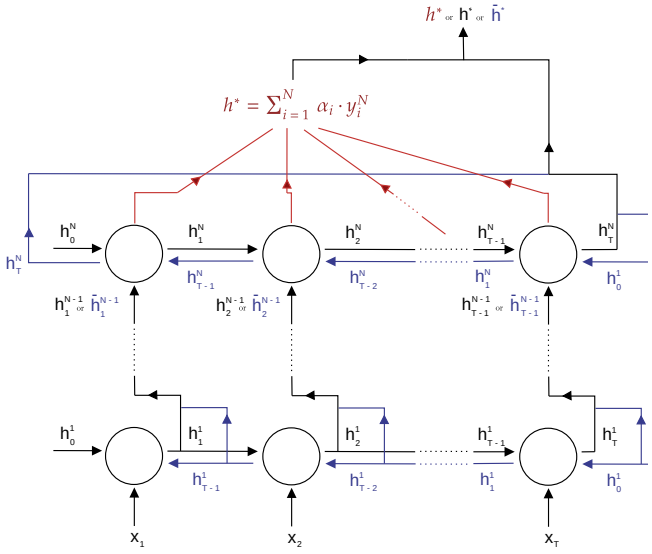

FIG. 5. Representation of the *Encoder's architecture* where the computation of the recurrent state is shown in *black* for unidirectional, *black* and *blue* for bidirectional and *red* for the attention mechanism. The variable X is the embedded input, h_i^N is the i^{th} recurrent state of the layer number N of the GRU, y_i the outputs of the encoder and α a weight used to compute the recurrent state related to the attention mechanism.

As it can be observed in Fig 5, the context vector h^* can be computed in multiple ways. Indeed, for a sentence of T words, one finds that the encoder can be:

- *Unidirectional* : The sequence, in this case the words composing the sentence, is read from left to right. Therefore, the recurrent state is supposed to capture the context of the sentence by only looking at it once. In this configuration, the last recurrent state h^* of the N^{th} layer of the GRU will be passed to the decoder. Therefore, from a mathematical point of view, one has:

$$h^* = h_T^N = \phi(x_T,\ h_{T-1}^N;\ \theta) \tag{1}$$

where ϕ is an activation function, x_T the last input of the sequence and θ the parameters of the model.

- *Bi-directional* : in order to capture even more the context of the sentence, the sequence is also ran through the encoder from right to left. Thus, the context vector have information regarding what is before and after each word.

In this configuration, the last recurrent state $\bar{h}^*$ of the N^{th} layer of the GRU will be passed to the decoder and it consist of the concatenation of the h obtained by reading the sentence in one way as well as the other. Hence, from a mathematical point of view, one has:

$$\bar{h}^* = (\overrightarrow{h}_T^N,\ \overleftarrow{h}_T^N) \tag{2}$$

where $\overrightarrow{h}_T^N$ is the recurrent state obtained by reading from left to right (*black* path) whereas $\overleftarrow{h}_T^N$ the recurrent state obtained by feeding the sequence from right to left (*blue* path) to the encoder.

- *Attention*: For long sentences, the network struggle to capture the context and generate meaningful long sentences. One of the reason explaining this problem is that it is foolish to assume that the last recurrent state can contain by itself all the information regarding the sequence fed to the GRU.

  Thus, a relatively simple solution consist of making a weighted sum of the different outputs of the encoder to create the context vector. The approach used in our architecture is therefore based on the one described by *Luong et al* [15] and [13], it can be explained as follows:

  1. The first step consist in computing a score using the encoder's outputs y and the current hidden state h emitted by the decoder. Thus, one has:

  $$e_{i,j} = y_{i,k}^T \cdot h_{k,j} \tag{3}$$

  2. Then, the weights used to determine the final recurrent state h^* are determined by applying a *softmax* on the scoring vector:

  $$\alpha_i = \frac{\exp(e_i)}{\sum_{k=0}^{j} \exp(e_{i,k})} \tag{4}$$

  3. Finally, the attention recurrent state is computed by making a weighted sum of the encoder's output as follows:

  $$h^*_{\text{att}} = \sum^{i} \alpha_i \cdot y_i \tag{5}$$

  where $y_i = \psi(h_i)$ is an output of the encoder with ψ an activation function.

After some research and testing, the number of layers inside the encoder has been fixed to $N_{\text{enc.}} = 2$.

C. Decoder

Now that the context vector h^* is created, the next part consist of decoding it and predict the next element of the sequence, i.e. the next word of the sentence. Once again, the neural network architecture used is a GRU with $N_{\text{dec.}} = 2$. As it can be observed in Fig 6, several mechanisms take place inside the decoder. Indeed, one finds that:

- In order to improve the training speed of the decoder, some *teacher forcing* is used. The method is simple, once the decoder makes a prediction $\hat{y}_i$, it uses it as its next input to predict the next word and so on. However, since the output word is not necessarily a good one, the next predictions might be wrong in the long run due to a snowball effect. Therefore, a simple technique consist of feeding as input either the one predicted by the decoder or the exact one y_i with a certain probability. As a rule of thumb, this probability is set to 0.5.

- Once the GRU has made a prediction $\hat{y}_i^N$, it must pass through a fully connected layer. Indeed, the size of the output layer is determined by the total dimension of the vocabulary inside the embedder. Afterwards, a softmax is applied to the vector to turn its values into probabilities. The index of the highest probability inside the vector corresponds to the ID of the predicted word by the SBOT. Finally, the word that is output is y_i^*.

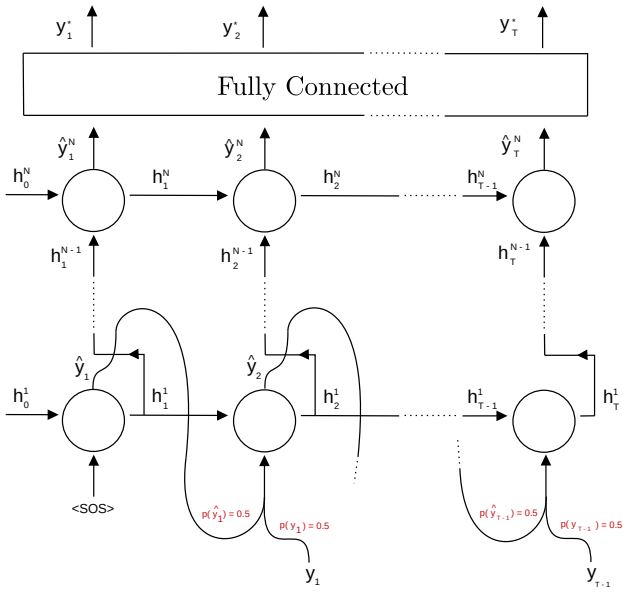

FIG. 6. Representation of the *Decoder's architecture* where the variable $\hat{y}_1^N$ is the i^{th} output made by the layer number N of the GRU, y_i is the exact word that should be predicted and y_i^* is the final i^{th} prediction of the decoder once it has ran through the fully connected layer.

D. Loss function

In the end, the problem faced is falls into the class of categorization. Therefore, it makes sense to use a *CrossEntropy* as our loss function. However, since the size of the vocabulary is relatively huge, a better version of this loss is the *SparseCategoricalCrossEntropy* which allows to use as exact output for comparaison the ID, i.e. index, of the word. This will save a lot of GPU memory during training. One last really important thing is the use of masks. Indeed, in addition to words, the SBOT must learn to predict end-of-sentence tokens since it allows it to tell to the user that it is done making prediction.

However, each pair of input and output of the dataset possess a lot of nothing-to-do tokens in order to reach *max sentence length*. The problem is that, without a mask and since sometimes a third of the output pair is composed of these tokens, in the long run the network learns to predict them. As a matter of fact, after training for a while, they were the only thing output by the network.

The SBOT learns to predict them because when the cross entropy was computed, it took into consideration these tokens. Therefore, the solution is to apply a mask which allows the network to only train its parameters to predict words or end-of-sentence tokens.

E. Learning rate scheduler

During training, the model happened to have at several occasions a steady, or even increasing, loss after quite a few epochs. In order to remedy to that problem, one added a learning rate scheduler called ReduceLROnPlateau from Pytorch as it seemed to be the perfect fit to the model. Indeed, what this scheduler does is basically finding out when a *plateau* in loss is happening and reduces the learning rate at that moment so that the model can start learning again. In the implementation chosen, the parameters were:

- *Mode* = min : It means that the scheduler consider that a plateau is reached by detecting when the loss stops decreasing.
- *Factor* = 0.5 : This means that when a plateau is detected, the learning rate will be decreased by a factor of 2.
- *Patience* = 5 : The scheduler will wait 5 epochs of steady loss before applying the reduction to the learning rate.
- *Threshold* = 1e-2 : The loss is considered steady ($\rightarrow$ plateau) if its decrease is less than 1e-2.

V. METRICS

The best metric for Natural language generation in term of evaluation of the quality is the *Human assessment*. Indeed, as the text generated is intended for 'humans', they are the best to evaluate if it fits their taste or not.

However, such assessment is quite costly and may take quite quickly a lot of time. To remedy to that, other metrics have been developed to evaluate output of *NLG* tasks. These metrics do not cost as much as human assessment and can be applied on a large amount of data. Furthermore, even if not all of them fit perfectly to every possible type of tasks, they still can be used in any situation. However, while they are really good in the most renowned types of tasks in *NLG* such as:

- *Machine Translation* (MT)
- *Question Answering* (QA)
- *Question Generation* (QG)
- *Abstractive Summarization* (AS)
- *Dialogue Generation* (DG)
- *Data To Text Generation* (D2T)
- *Image Captioning* (IC)

and they present one common inconvenient regarding our situation: all the metrics need to compare the generated text to a reference. As in our case, the text that is being generated has no constraints other than being globally coherent to the Harry Potter theme, binding this text to a very specific reference will add bias to the results obtained via the metrics. Indeed, the metrics will attribute a very low score if the output is completely different from the reference, while actually the output is really well structured and coherent, it just does not fit perfectly to the reference. However, it would still be nice to have other metrics than human assessment to evaluate our model. The metrics can be separated into several groups and it is important to notice that the one concerning characters based metrics will be discarded since our network focus on generating words.

The first group of metrics is called the *N-gram Based metric*. Indeed, it uses n-grams (sequence of items) to computes the similarity between the reference and the output. In this group, two metrics are really well known and widely used, it is *Bleu* and *Rouge*. Both metrics have a similar use but *Rouge* focus on recall while *Bleu* focus on precision. The way they work is as follows:

- *Bleu-n*: It counts the number of n-grams from the candidate that appear in the reference (clipped count, i.e. if the n-gram appear 6 times in candidate and only one in reference, the number of match will be 1), and divide it by the total number of n-grams in the reference. The value of n allows to shift the focus between precision (small n) and word ordering (bigger n).
- *Rouge-n*: It works in the other way around of *Bleu*: number of n-grams of the reference that appear in the candidate, divided by the number of n-grams in the reference.

- *Rouge-L* : Instead of looking for n-grams, it looks for the longest common sub-sequence of the candidate and the reference, and divides it by the number of words in the reference.

The disadvantages of these metrics are that it does not include context/meaning nor sentence structure in the evaluation. Finally, it is important to know that the returned value of these metrics should be between 0 and 1 where 1 is the best possible value.

The second group is called *Embedding based metrics*, the metrics compute a similarity/dissimilarity score between the embedding of the reference and the candidate. All sort of embeddings can be used, at many different levels which allow for more personalisation of the metric. The most used metric in this group is BERTscore but greedy matching will also be investigated. The way they work is as follows:

- *BERTscore*: It uses a pre-trained model to first embed the candidate and the reference (it tokenizes the sequence in words, creates a vector of values for each word which will depend of the words around it, thus same words won't have the same vector, thanks to a Transformer encoder). Once the embedding has been done, it uses pairwise cosine similarity metric defined as:

$$M_{i,j} = \frac{\mathbf{x}_i^T \hat{\mathbf{x}}_j}{||\mathbf{x}_i|| ||\hat{\mathbf{x}}_j||}$$

where x_i is the reference token and $\hat{x}_j$ the candidate token. By computing all these values, one will be able to give rise to a matrix of the similarity between the candidate and the reference. From this matrix, defining x as the reference and $\hat{x}$ as the candidate, one obtains:

  - *Recall*

$$R = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^T \hat{x}_j$$

  - *Precision*

$$P = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^T \hat{x}_j$$

  - *F1 Score*

$$F1 = 2 \cdot \frac{P \cdot R}{P + R}$$

In addition to that, some optional weights can be added to adjust the relative importance of each word compared to the other. Thanks to its embedding, it captures the context, information from the sequence, unlike the previously discussed metrics. Those 3 measures vary in a range of 0 to 1, with 1 being the best value possible.

- *Greedy matching* : For each token w of the reference sequence (resp token $\hat{w}$ of the candidate sequence), it computes the cosine similarity of its word embedding with the word embedding of each token $\hat{w}$ (resp w), takes the maximum and average over all w (resp $\hat{w}$). then it adds the 2 quantity. From a mathematical point of view, one has:

$$Gm = \frac{Gm(r, \hat{r}) + Gm(\hat{r}, r)}{2}$$

where,

$$Gm(r, \hat{r}) = \frac{\sum_w \max_{\hat{w}} \cos\_\sin(e_w, e_{\hat{w}})}{r^{-1}}$$

$$Gm(\hat{r}, r) = \frac{\sum_{\hat{w}} \max_w \cos\_\sin(e_{\hat{w}, e_w})}{\hat{r}^{-1}}$$

with e the embedding, r the reference and $\hat{r}$ the candidate. In addition to that, one can also retrieve the score for the metrics *Embedded Average* and *Vector Extrema* but they will no be detailed them here. Again, all those values are in the range 0 to 1, 1 being the best value.

The last group of metrics is called *Learned functions*, it tries to find a mapping from the candidate and the reference to a human rating by using a pre-trained regression model. For this group, the following metrics were found:

- *Bleurt* : This metric actually used a Bert model with a linear layer on top. Indeed, *Bleurt* will first use a fully trained model on synthetic data then fine tune its results thanks to human rating such that after, from a candidate sequence and a reference, it will be able to predict the rating a human would give, rating between 0 and 1, 1 being the best value.
- *Bartscore* : It takes into account the linguistic quality but also the semantic overlap and the context by considering the evaluation as being a generation task. The *Bartscore* is computed on the basis of the following formula:

$$\sum_{t=1}^{m} w_t \log(p(\hat{r}_t | \hat{r}_{<t}, r, \theta))$$

which consists in computing the probability of obtaining a candidate $\hat{r}$ from a reference r. This metric is more for comparing different models together as there is not really a range of value, the smaller the value, the worst the model.

VI. RESULTS

The purpose of this section is to analyze and understand the influence of the different parameters important to training. In the end, all the knowledge acquired will be put to use for the construction of the biggest network of all named *Hagrid*.

In order to assess the performance of all the models that are going to be presented, their characteristics as well as their score on all the metrics presented in the section V are gathered in the Tab. II.

A. Influence of the capacity

The capacity of a model defines its ability to find a good model in the space of all possibilites regardless of its complexity. Therefore, it is interesting to first observe the evolution of the train loss with respect to the size of the recurrent state h_{size} since this will be the only tunable parameter to increase the model's capacity.

As it can be seen in Fig. 7, the model is not able to learn properly for the hidden size 128 and 512 since the cross entropy reaches a tray of respectively 4.5 and 3.2. However, for the biggest hidden size, the model started to learn even more starting at the epoch 100. It is important to take into consideration that, due to an unfortunate event, the evolution curve from 100 to 150 epochs for both hidden size 128 and 512 were lost but they do converges.

In the Fig. 8, the hidden size 128 also converges to the high value of 4.5 for the cross entropy. However, in the case of the hidden size 512 and 1024, they were able to converge more quickly than for the unidirectional architecture to a cross entropy of 1 and 0.3.

Finally, the Fig. 9 gives a quick comparison between the predictions made by the different models for the same input sentence.

B. Influence of directionality

A typical context vector in an *recurrent neural network* is created by reading the sequence from left to right. However, it can be interesting to know why a words is used knowing the future ones that will follow it, this corresponds to a case when bi-directionality can be introduced. The evolution of the train loss with respect to the number of epochs and the direcionality of the model confirms this sayings, since one can observe that the cross-entropy loss in generally smaller for bi-directional networks. Indeed, in the context of text generation, not only the previous words of a sentence are important, but also the future ones.

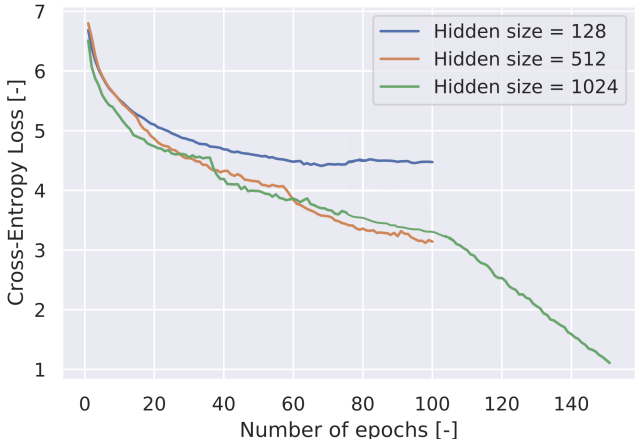

FIG. 7. Evolution of the train loss for a *unidirectional* architecture where the capacity of the model is increased by changing the hidden size of the recurrent state h

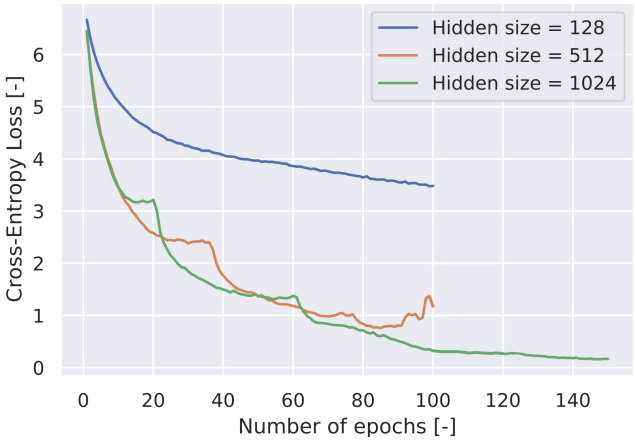

FIG. 8. Evolution of the train loss for a *bi-directional* architecture when the capacity of the model is increased by changing the hidden size of the recurrent state h

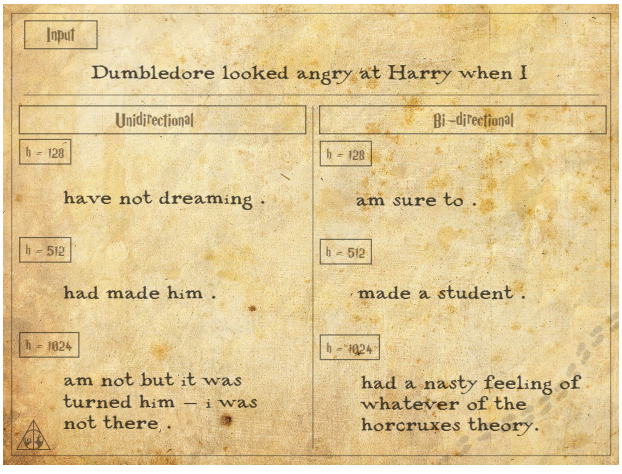

FIG. 9. Illustration of the output made by the SBOT with different value of h_{size} for the same input sentence.

Thus, it seems logical that bi-directional networks can reach a smaller cross-entropy loss, and thus, a higher accuracy. However, even if the loss is small, this does not implies that the models generalize well to new context as it can be seen by the metrics in Tab. II.

From Fig. 11, one can observe that the final cross-entropy loss is still very high, even after 100 epochs. This is due to the capacity of the network, which is too small, as it is explained in section VI A. Nevertheless, what is interesting here is the fact that the bi-directional model can reach a smaller cross-entropy anyway. Another observation is that the bi-directional network cross-entropy is always decreasing, while the unidirectional one reaches a plateau around the 60^{th} epoch, which demonstrates that introducing bi-directionality is necessary in our case to get better results.

For a higher capacity network, as shown in Fig. 12, the observations remain the same : the bi-directionality allows to get better results since the cross-entropy loss can get smaller than the unidirectional one. The slight rise of the cross-entropy for the bi-directional network around the 90^{th} epoch is not significant. In this case, the cross-entropy gets significantly smaller than in Fig. 11, but struggles to get below 1.

Finally, on this final Fig. 13, one can see that the same behavior occurs once more, but the final value of the cross-entropy loss almost reaches zero, which is a very good value. Overall, the bi-directionality allows to get better results, due to the nature of the task accomplished for this project. From this result, one can also conclude that all the effort put into the dataset was not wasted since the network is able to learn. Indeed, an illustration of the results obtained by the SBOT are shown in Fig. 10.

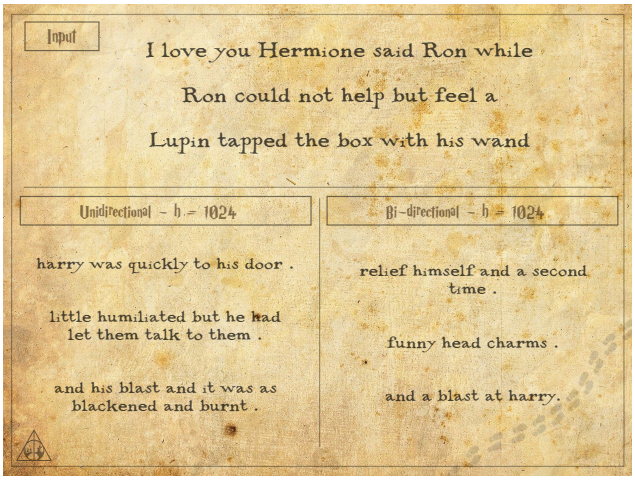

FIG. 10. Illustration of the output made by the SBOT for a *uni-directional* and *bi-directional* architecture where $h_{\text{size}} = 1024$ for the same input sentences.

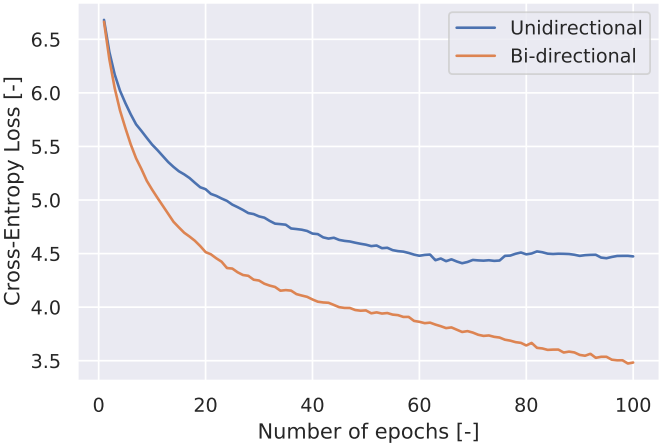

FIG. 11. Evolution of the train loss for a *uni-directional* and *bi-directional* architecture where $h_{\text{size}} = 128$

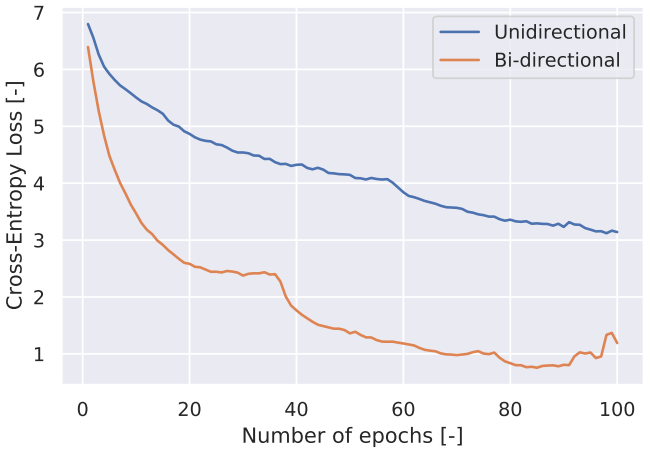

FIG. 12. Evolution of the train loss for a *uni-directional* and *bi-directional* architecture where $h_{\text{size}} = 512$

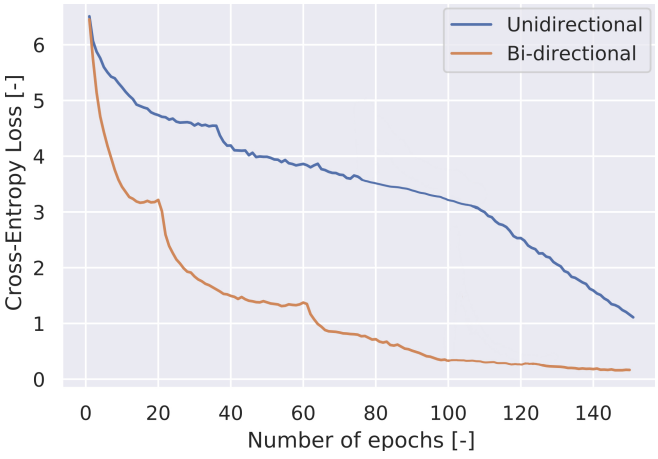

FIG. 13. Evolution of the train loss for a *uni-directional* and *bi-directional* architecture where $h_{\text{size}} = 1024$

C. Influence of teacher forcing

As explained before, the purpose of Teacher-Forcing is to increase the training speed. Therefore, one can observe in Fig. 15 that the shape of the curves are in their global shape very similar. However, the curve corresponding to the one where the network uses teacher forcing is shifted downwards, which shows that the mechanism allows the model to learn significantly faster than without using it.

Even if in the end, the loss of both models is relatively low, the difference in the quality of the outputs they make is quite noticeable. This observation is illustrated in Fig. 14 where the two models are tested upon the same sentences.

D. Influence of attention

In addition to making the training faster, the words predicted by the network should also be more accurate. As it can be seen in Fig. 16, the cross-entropy of the unidirectional model using attention drops faster and reaches a smaller value in the end.

In the case of the bi-directional model shown in Fig. 17, one can observe that the cross-entropy of the model using attention globally has almost the same behavior as for the model that does not use attention. However, as it has been said, the real difference can be observed in Fig. 18 by having a look at the outputs produced by the models for the same input. From our personal opinion, the results obtained with the models using attention are a little more funnier and accurate.

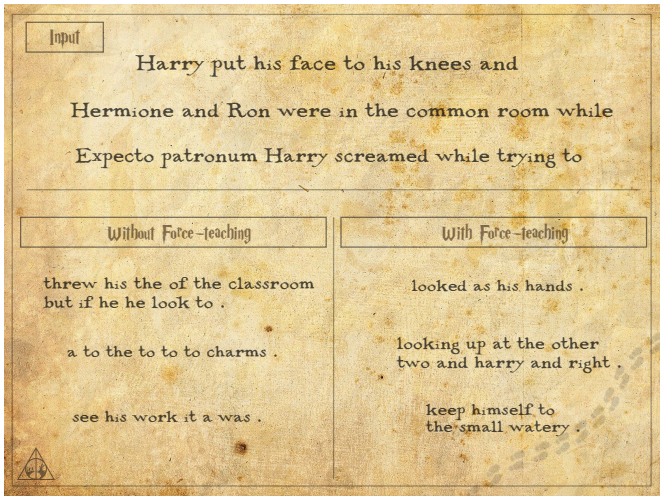

FIG. 14. Illustration of the output made by the SBOT for a *bi-directional* architecture with and without using Teacher-Forcing where $h_{\text{size}} = 1024$ for the same input sentences.

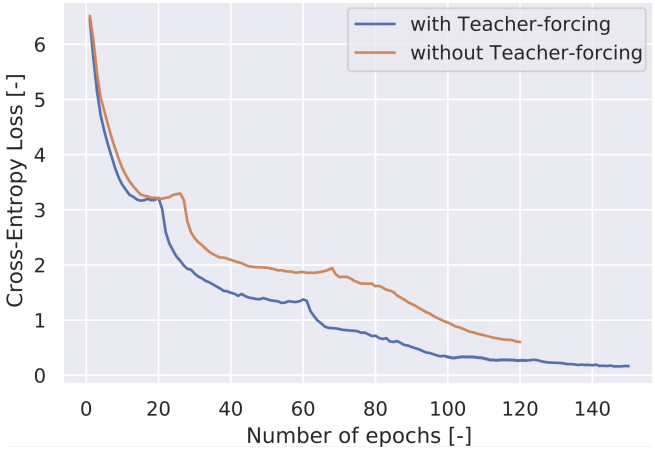

FIG. 15. Evolution of the train loss for a *bi-directional* architecture with and without using Teacher-Forcing where $h_{\text{size}} = 1024$

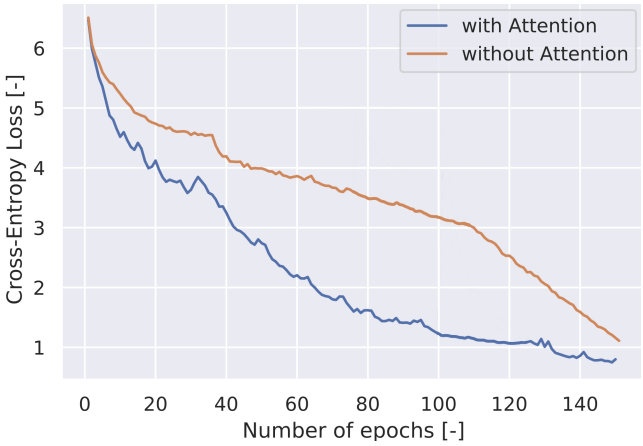

FIG. 16. Evolution of the train loss for a *uni-directional* architecture with and without using attention where $h_{\text{size}} = 1024$

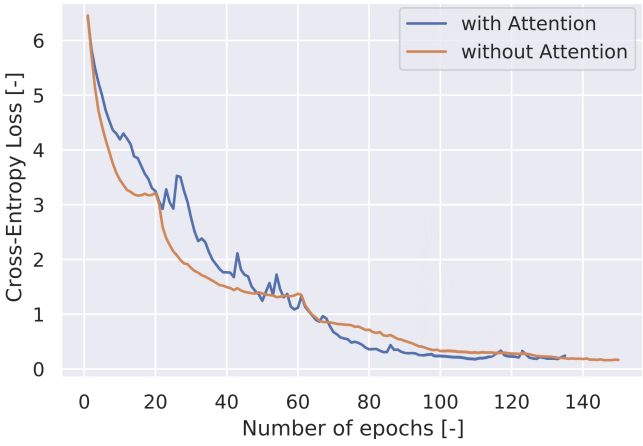

FIG. 17. Evolution of the train loss for a *bi-directional* architecture with and without using attention where $h_{\text{size}} = 1024$

E. Influence of the learning rate

In the beginning of this project, the learning rate used was $\gamma = 0.01$ and for a long time the results were not satisfying (steady or even increasing loss). Indeed, the theoretical concept that uses the learning rate is the gradient descent which aims to move towards a local minimum in order to optimized the model parameters. In our situation, the problem has been identified as the learning rate being too big which leads the gradient descent to diverge. This can produce some kind of oscillations because the step taken each iteration is too large and that never allows the method to reach the minimum of the function. Therefore a simple solution was to start the training with $\gamma = 0.001$. The next problem to face was a steady loss which was solve by introducing a scheduler to adapt dynamically the learning rate during the training (more details about the scheduler are given in section IV E).

F. Conclusion and creation of Hagrid

After assessing the influence of each parameter, one concluded that the winning combination should be obtained by combining a *high capacity*, *Bi-directionality*, *Teacher forcing*, an *attention mechanism*, a small γ and a *scheduler*. This combination corresponds to the model *Hagrid*, which is evaluated in the Tab. II. The evolution of the loss is represented in the Fig.19 and as it can be seen, after quite a long training, the cross-entropy loss finally converged towards around 0.2. In addition to that, some of the predictions made by *Hagrid* are represented in Fig. 20.

VII. DISCUSSION

Throughout this project, we learned a lot about the world of *natural language processing*. Indeed, even if results of the metrics in the Tab.II are not as good as expected, the fact that we are able to make a network of neurons predict words and even sometimes sentences with a bit of meaning is mesmerizing.

The biggest problem of our project is that we were not able to fully tackle its over fitting to the books. As a matter of fact, the network is able to produce complete sentences while capturing a little bit the context of the input sentence but most of the time, the output sentence feels like something coming right from the book. Therefore, if this project was to be continued, the best improvement that one could make is for the *story telling bot* to pick the most probable word it outputs among a distribution of the most probable ones after this one. Actually, this might allow to produce more randomness to the sentences and creates even better results.

Now regarding the results obtained for each metric for each model grouped in Tab. II. First thing to notice is that none of the models actually yield good results, all metrics only have small values. Actually, this is due to what was explained in the metric part: those metrics put more focus on real correspondence between what was generated and what was the reference than on if what was generated was of good quality. However, even if the results are overall not terrible, it is still possible to retrieve information about them. Indeed, the model with the parameters that appeared to be the best, *Hagrid*, has the best results compared to the other models.

In term of human assessment, the quality of the outputs is not as good as one would have hoped but stays quite honorable when taking into account the context of this project. As a matter of fact, given the time and our past experience in the world of deep learning, better results would have been quite surprising ! So overall, what was obtained is quite good.

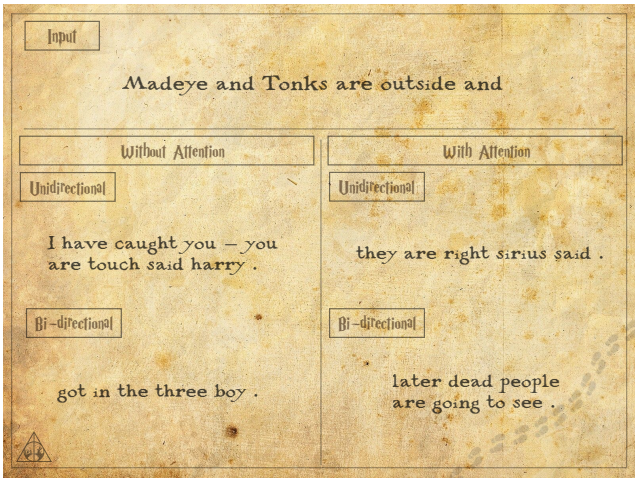

FIG. 18. Illustration of the output made by the SBOT for an *uni-directional* and *bi-directional* architecture with and without using attention where $h_{\text{size}} = 1024$ for the same input sentences.

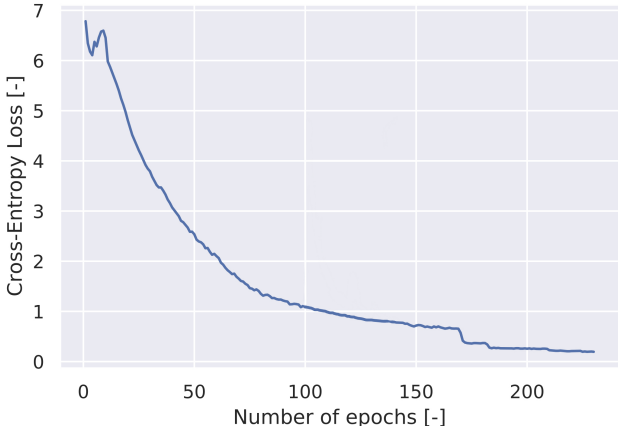

FIG. 19. Evolution of the train loss for *Hagrid*.

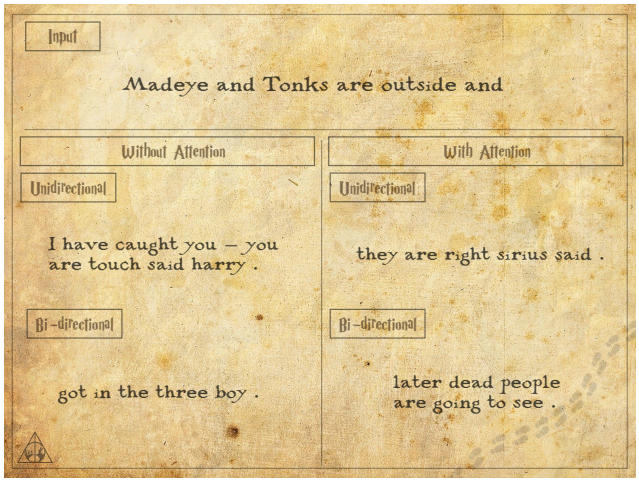

FIG. 20. Illustration of the outputs made by *Hagrid*.

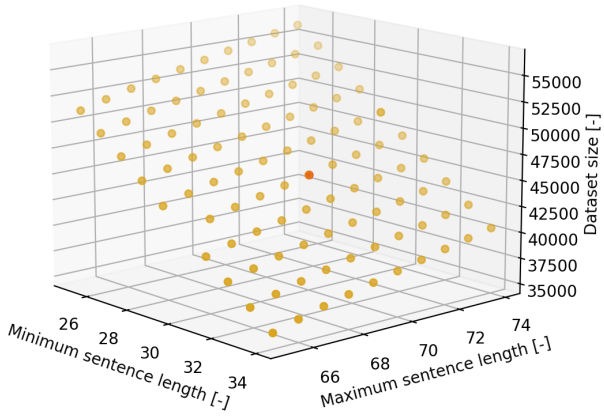

FIG. 21. Size of the dataset obtained for different pairs of values of the *minimum sentence length* and *maximum sentence length*. The paragraph size is fixed to $s = 3$ sentences and the red dot indicates the value picked for training the network.

VIII. GOING FURTHER WITH GROPE

As a bonus for this project, one decided to try and overcome the fact that the previously discussed models could only take a few words of a sentence as input and was limited to produce the end of one sentence. Indeed, whatever the version of the *story telling bot* used, it was not able to make correct sentences after predicting the end of the first one.

In order to overcome this constraint, the format of the dataset has to be modified, so that the input and output pairs are not anymore the beginning and end of a sentence but instead of a paragraph. Actually, these paragraph will be created by assembling together s consecutive sentences. As for the former dataset, their size will be defined by a minimum and maximum paragraph length as described in section III A.

Once again, one needs to find a combination of parameters which allows to have a consequent dataset with respect to its original size while keeping in mind that the training time should not take too long. Therefore, the evolution of the the size of the dataset with respect to the *minimum sentence length* and *maximum sentence length* is represented in Fig. 21.

From the figure, one can observe that a good choice for these parameters is to take *minimum sentence length* and *maximum sentence length* equal respectively to 30 and 70. Therefore, it will possible to feed 20 words as inputs, make the network predict at least 10 words and possibly 40 more of them.

For the sake of accelerating the training, *Grope* uses *Hagrid* as a pre-trained network. The metrics obtained for this model are available in the table II.

The loss managed to reach 5.76 in 2 days of training (over 26 epochs) which is still not low enough to expect good results. The training is tremendously long for this model since the starting point of the loss was 8.166.

The training is still going on at the moment this report is being written, but it will not reach satisfying results before the final deadline. Thus, if the reader wants to have a look at the final version of our trained model, it will be accessible in the *GitLab* created for this project when ready.

IX. ACKNOWLEDGEMENTS

First of all, we would like to thank *Mr. Louppe* which was the best investor that one could ever imagine for the realization of this project. Even if the results do not meet our expectations, the knowledge acquired throughout all these trainings is priceless.

Furthermore, we would also like to thank *Mr.Lambrechts* whose discussions during lunchtime were very insightful and significantly contributed to the direction this project took.

Name	Epochs	h_{size}	r_{tf}	Bidirectional	Attention	Loss	F1	Bleu-1	Bleu-4	Rouge-L	Gm	Bleurt	Bartscore
Remus	100	128	0.5	✗	✗	4.4737	0.18	0.12	0	0.23	0.62	0.12	-5.33
Sirius	100	128	0.5	✓	✗	3.4830	0.19	0.15	0	0.25	0.61	0.17	-5.34
Albus	100	512	0.5	✗	✗	3.1407	0.13	0.15	0	0.22	0.6	0.15	-5.34
Severus	100	512	0.5	✓	✗	1.19	0.18	0.23	0	0.27	0.65	0.19	-5.09
Hermione	150	1024	0.5	✗	✗		0.15	0.16	0	0.2	0.59	0.16	-5.45
Dobby	150	1024	0.5	✓	✗		0.18	0.25	0	0.27	0.64	0.25	-5.22
Draco	150	1024	0.5	✗	✓		0.21	0.2	0	0.24	0.63	0.19	-4.79
Neville	150	1024	0.5	✓	✓		0.23	0.22	0	0.24	0.64	0.29	-5.21
Minerva	120	1024	0	✓	✗		0.16	0.19	0.05	0.31	0.68	0.15	-5.17
Hagrid	300	2048	0.5	✓	✓								
Grope	300	2048	0.5	✓	✓								

TABLE II. Collection of the trained models and their parameters among with the results obtained for the different metrics and the final loss obtained during the training. *Hagrid* and *Grope* have the same parameters but do not use the same dataset: *Hagrid* is fed words of a single sentence as input, and *Grope* is given words of multiple sentences as input, i.e. paragraphs. In the case of the metrics, they were obtained by evaluating the models on 50 random and unseen sentences among the books.

X. REFERENCES

[1] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le and R. Salakhutdinov, Transformer-xl: Attentive language models beyond a fixed-length context, *arXiv preprint arXiv:1901.02860*, 2019.

[2] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, *arXiv preprint arXiv:1810.04805*, 2018.

[3] Y. Kikuchi, G. Neubig, R. Sasano, H. Takamura and M. Okumura, Controlling output length in neural encoder-decoders, *arXiv preprint arXiv:1609.09552*, 2016.

[4] A. Fan, M. Lewis and Y. Dauphin, Hierarchical neural story generation, *arXiv preprint arXiv:1805.04833*, 2018.

[5] Y. Qu, P. Liu, W. Song, L. Liu and M. Cheng, A text generation and prediction system: pre-training on new corpora using BERT and GPT-2, in *2020 IEEE 10th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, IEEE, 2020, 323–326.

[6] Aladdin Persson, *Pytorch Torchtext Tutorial 1: Custom Datasets and loading JSON/CSV/TSV files*, 2020. url : `https://www.youtube.com/watch?v=KRgq4VnCr7I`.

[7] Aladdin Persson, *Pytorch Seq2Seq Tutorial for Machine Translation*, 2020. url : `https://www.youtube.com/watch?v=EoGUlvhRYpk`.

[8] J. Pennington, R. Socher and C. D. Manning, Glove: Global vectors for word representation, in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, 1532–1543.

[9] Harshith, *Text Preprocessing in Natural Language Processing*, 2019.

[10] J. Weng, *NLP Text Preprocessing: A Practical Guide and Template*, 2019.

[11] Deepanshi, *Text Preprocessing in NLP with Python codes*, 2021.

[12] I. Pointer, *Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications*. O'Reilly Media, 2019.

[13] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2019.

[14] S Robertson, NLP From Scratch: Translation With A Sequence to Sequence Network And Attention, 2017.

[15] M.-T. Luong, H. Pham and C. D. Manning, Effective approaches to attention-based neural machine translation, *arXiv preprint arXiv:1508.04025*, 2015.

[16] T. Zhang*, V. Kishore*, F. Wu*, K. Q. Weinberger and Y. Artzi, BERTScore: Evaluating Text Generation with BERT. url : `https://arxiv.org/pdf/1904.09675.pdf`.

[17] I. V. S. M. N. L. C. J. P. Chia-Wei Liu Ryan Lowe, How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation. url : `https://arxiv.org/pdf/1603.08023.pdf`.

[18] BARTScore: Evaluating Generated Text as Text Generation. url : `https://nips.cc/media/neurips-2021/Slides/28542.pdf`.

[19] HuggingFace, What is the BLEU metric?. url : `https://www.youtube.com/watch?v=M05L1DhFqcw`.

[20] HuggingFace, What is Rouge Metric?. url : `https://www.youtube.com/watch?v=TMshhnrEXlg`.

[21] Y. Kilcher, BLEURT: Learning Robust Metrics for Text Generation (Paper Explained). url : `https://www.youtube.com/watch?v=rl4nUngiR2k`.

[22] R. Khandelwal, BLEU — Bilingual Evaluation Understudy. url : `https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1`.

[23] Wikipédia, Bleu - Wikipédia. url : https://en.wikipedia.org/wiki/BLEU.

[24] A. B. SAI and all, A Survey of Evaluation Metrics Used for NLG Systems. url : https://arxiv.org/pdf/2008.12009.pdf.

[25] D. Çavuşoğlu, Evaluation Metrics: Assessing the quality of NLG outputs. url : https://towardsdatascience.com/evaluation-metrics-assessing-the-quality-of-nlg-outputs-39749a115ff3.

[26] H. Face, Bleurt metric. url : https://huggingface.co/metrics/bleurt.

[27] T. Zhang*, V. Kishore*, F. Wu*, K. Q. Weinberger and Y. Artzi, BERTScore - Github. url : https://github.com/Tiiiger/bert_score.

[28] Metrics - Github. url : https://github.com/Maluuba/nlg-eval.

[29] Bart metric - Github. url : https://github.com/neulab/BARTScore.

[30] FORMCEPT, *Harry Potter raw text books*, 2017. url : https://github.com/formcept/whiteboard/tree/master/nbviewer/notebooks/data/harrypotter.

[31] Ben Trevett, *Sequence to Sequence Learning with Neural Networks*, 2021. url : https://github.com/bentrevett/pytorch-seq2seq/blob/master/1\%20-\%20Sequence\%20to\%20Sequence\%20Learning\%20with\%20Neural\%20Networks.ipynb.

[32] Aladdin Persson, *How to build custom Datasets for Text in Pytorch*, 2020. url : https://www.youtube.com/watch?v=9sHcLvVXsns.

[33] R Khandelwal, *Attention: Sequence 2 sequence model with attention mechanism*, 2020.

[34] W. Haverals and L. Geybels, Putting the Sorting Hat on JK Rowling's Reader: A digital inquiry into the age of the implied readership of the Harry Potter series, *Journal of Cultural Analytics*, volume 6(1):24077, 24077, 2021.

[35] G. Research, BLEURT: Learning Robust Metrics for Text Generation. url : https://aclanthology.org/2020.acl-main.704.pdf.