Journal of Theoretical and Applied Information Technology
31st January 2022. Vol.100. No 2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MACHINE LEARNING-BASED AUGMENTED REALITY FOR IMPROVED TEXT GENERATION THROUGH RECURRENT NEURAL NETWORKS

[1]ANASSE HANAFI, [2]MOHAMED BOUHORMA, [3]LOTFI ELAACHAK

[1]Research Student/Assistant Professor, Abdelmalek Essaidi University, Department of LIST, Tangier, Morocco

[2]Professor. Abdelmalek Essaidi University, Department of LIST, Tangier, Morocco

[3]Professor, Abdelmalek Essaidi University, Department of LIST, Tangier, Morocco

E-mail : [1]anasse.hanafi94@gmail.com, [2]bouhorma@gmail.com, [3]lotfi1002@gmail.com

ABSTRACT

Machine learning (ML) is a large field of study that overlaps with and inherits ideas from many related fields such as artificial intelligence (AI). The main focus of the field is learning from previous experiences. Classification in ML is a supervised learning method, in which the computer program learns from the data given to it and make new classifications. There are many different types of classification tasks in ML and dedicated approaches to modeling that may be used for each. For example, classification predictive modeling involves assigning a class label to input samples, binary classification refers to predicting one of two classes and multi-class classification involves predicting one of more than two categories. Recurrent Neural Networks (RNNs) are very powerful sequence models for classification problems. However, in this paper, we will use RNNs as generative models, which means they can learn the sequences of a problem and then generate entirely a new sequence for the problem domain, also, we propose an architecture of a learning application that benefits from the advantages of both machine learning and augmented reality to provide a better learning experience and help teachers build adaptive educational content using the proposed model.

Keywords: *Artificial Intelligence, Machine Learning, Classification, Recurrent Neural Networks, Sequence Models, Generative Models.*

1. INTRODUCTION

Text classification is one of the important and common everyday jobs in controlled machine learning. It is about assigning a category or a class to documents, articles, books, papers, reviews, tweets or anything that contains text. It is a core task in natural language processing [1]. Many applications appeared to use text classification as the main task, examples include spam filtering, sentiment analysis, speech tagging, language detection, and many more. The aim of this paper is to present the use of a classification methodology as a generative model based on RNN algorithms, trained on textual dataset and able to generate entirely a new sequence of text for the problem domain. Thus, this paper presents an architecture of a mobile application that integrate the proposed generative model.

Generative models are one of the most promising approaches towards our goal. Actually, generative modeling is used in unsupervised machine learning to describe phenomena in data and enable computers to understand the real world. This AI understanding can be used to predict all manner of probabilities on a subject from modeled data [2].

An illustration of a generative model might be one that is trained on a set of images from the real world in order to generate similar ones. The model might take observations from a 500-gigabyte set of images and reduce them into 100-megabyte of weights. Weights can be thought of as reinforced neural connections. Through increased training, an algorithm comes to produce more realistic images.

To train a generative model, we first collect a large amount of data in some domain (e.g., think millions of images, sentences, or sounds, etc.) and then train the model to generate data like it. The trick is that

the neural networks we use as generative models have a number of parameters significantly smaller than the amount of the data we train them on, so the models are forced to discover and efficiently internalize the essence of the data in order to generate it. Generative models have many short-term applications. But in the long run, they hold the potential to automatically learn the natural features of a dataset, whether categories or dimensions or something else entirely.

2. STATE OF ART

With the rapid increase in computing capacities of computers and the evolution of learning techniques, the use of neural networks is becoming popular in various computer science field, especially within the image processing, automated translation and speech processing communities. In recent years, two types of neural networks have marked a technological breakthrough in the field of speech processing: so-called "deep" networks and recurrent networks.

The Artificial Neural Network (ANN) was introduced as a rudimentary model of information processing in the human brain. Thus, the basic structure of an ANN is a network of small computing nodes linked together by guided and balanced links (Figure 1).

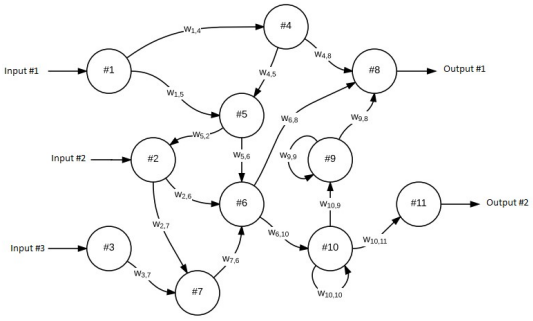

Fig. 1: Representations of an artificial neural network (ANN) in the form of a network of computation nodes connected by links directed and weighted by the coefficients.

The nodes represent the neurons and the weighted links represent the strength of the synaptic connections linking the neurons between them. In this representation, the neuron can be a summator of the potentials of the synaptic signals which reach it, and which in turn transmits information based on this sum via a non-linear transfer function. An ANN is activated by injecting data at all or part of the nodes and then propagating the information by following weighted links. Once the information is propagated, we can collect the activation levels of all or part of the nodes and use them as a prediction or a classification.

Many types of neural networks with very different properties have been developed since the emergence of formal neurons in the 1940s [1]. More specifically, we distinguish two types of networks: those whose connection graph has at least one cycle and those that do not. The first ones are said to be recurrent and the latter are said to be acyclic. Among acyclic networks we find networks of the perceptron type [2], convolutional [3], Radial Basis Function [4] or the Kohonen maps [5]. But the most frequently used variant is the multilayer perceptron (MLP).

We therefore begin by presenting the most classic of neural networks, the multi-layered perceptron which is an acyclic neural network (Feed-Forward Neural Network - FFNN) structured in layers. A multi-layered perceptron is thus composed of an input layer, one or more intermediate layers called hidden layers and an output layer. For each of the layers, all of its nodes are connected to all the nodes of the previous layer (Figure 2).

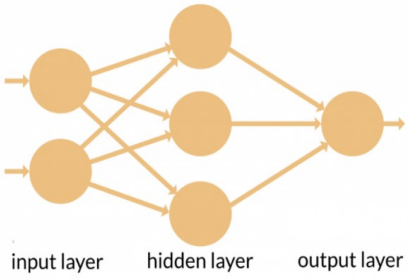

Fig. 2: Layered structured neural network commonly known as a multi-layered perceptron (MLP).

The origin of the multi-layered perceptron goes back to F. Rosenblatt who in 1957 was inspired by the work on formal neurons [1] and by Hebb's rule [6] to develop the model of the perceptron.

This model has only one layer but has already solved simple tasks of classifying geometric symbols. It is however impossible with the method formulated by F. Rosenblatt to train a system having several layers which turns out to be very restrictive a few years later. Indeed, in 1969 M. Minsky and S. Papert demonstrate by rigorous analysis that the perceptron is unable to learn functions if they are not linearly separable (like for example, the Boolean XOR function). They even

go a little further by demonstrating that it is necessary to have at least one additional layer of neurons to be able to solve this problem. But there is then no method to train a multi-layered perceptron.

It took more than ten years for researchers to develop a learning technique that allows to adjust the parameters of a multi-layered perceptron. Indeed, it was P. Werbos who first proposed the idea of applying to ANN the error gradient back-propagation technique developed over the years 1960. During his thesis that he defended in 1974 [7], he analyzed the relevance of this method but, given the lack of interest in the scientific community for ANN following the publication of M. Minsky and S. Papert, he did not publish any result on the subject before 1982 [8]. It was finally in the mid-1980s that this method was rediscovered by several research teams [8–13] and that it ended up being popularized.

In 1986, D. umelhart, G. Hinton and R. Williams show in [9] that using the back-propagation of the error gradient applied to a multi-layered perceptron can finally exceed the limits of the perceptron which were raised by M. Minsky and S. Papert in 1969. In particular, the multi-layered perceptron makes it possible to treat complex nonlinear problems, with only one hidden layer and a sufficient number of neurons, any nonlinear and continuous function over a compact space with arbitrary precision [10]. The multi-layered perceptron is thus called a universal approximator of functions.

In the case of many sequences' classification tasks, to make a decision at a given moment, it is interesting to know the past and all or part of the future of the sequence. For example, in automatic speech processing task, to be able to determine which phoneme is being pronounced at time t, it is very useful to know which phonemes follow it as well as which precede it. The RNNs we described in the previous section process sequences in temporal order and therefore do not have access to future information.

To tackle this problem, we can add future information to the input data or introduce a delay between an input vector and its target so that the RNN has time to process some future information before having to make a decision. However, these different approaches add constraints on learning either by increasing the number of parameters in the input layer, or by forcing the RNN to learn the chosen delay to give its response at the right time. And in both cases, we do not solve the problem because we introduce hyperparameters (number of input vectors to aggregate or response time) which can be difficult to adjust. In addition, the asymmetry between the past and the future in terms of the exploitable context is not in any way eliminated.

In 1997, Schuster and Paliwal introduced in [11] an elegant solution called bidirectional Recurrent Neural Network (BiRNN) which consists in presenting each sequence to be processed to two RNNs of the same type but with different parameters:

- ✓ The first one processes the sequence in the natural order
- ✓ The second one processes the sequence in reverse order.

The two sequences obtained at the output of the two RNNs are then concatenated before being put at the input of a multi-layer perception (MLP) which for each initial input vector then produces an output which is based on all the context passed via the exit of the first RNN and all the future context via the exit of the second. It is thus possible to fully exploit the capabilities of the RNNs over the entire sequence for each time step.

3. THEORICAL BACKGROUND

3.1 Recurrent Neural networks

Recurrent neural networks (RNNs) are the cutting-edge algorithm for sequential data. This is because it is the first algorithm that remembers its input, due to its internal memory, which makes it ideally suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the incredible achievements of Deep Learning (DL) in recent years.

In order to understand how RNNs works, we absolutely need to have a general understanding of a normal feed-forward neural network and also understand what are sequential data. The simplest definition of sequential data is any kind of data where the order is important. For example, the DNA sequence can be considered as a sequential data.

The techniques used to transfer data by RNN's and feed forward neural network allowed these two neural networks to get their names. The data pass only in a single direction, from the input layer to the output layer, passing by all the hidden layers, and never touch a node twice. Because feed forward neural networks can't store their inputs, they are considered weak at predicting next inputs they could obtain. Also, inputs are not ordered and all previous stats are forgotten and cannot be memorized.

In a recurrent neural network, the information pass through a circle. When there is a taken decision or a prediction, it contemplates the current input and also what it has learned from the inputs it received before. Figure 3 demonstrate the variance in information flow among a recurrent neural network and a feed-forward neural network.

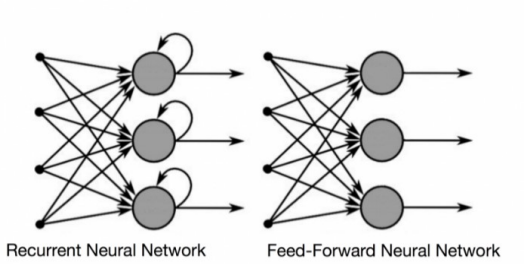

Fig. 3: RNN & feed-forward neural networks information flows

An example is the best manner to demonstrate the notion of a RNN memory, let's set the string "chemistry" as an input to our feed forward neural network, the algorithm will process that string character by character. When it processes the character "m" it has already forgotten about "c", "h" and "e". So, it's almost impossible for this neural network to make a prediction on the upcoming characters.

However, thanks to the internal memory storage of recurrent neural network, this kind of networks has the ability to remember the previous inputs. The produced output or result will be copied and looped back into the network as a new input. The strong point of RNN algorithms is that they have the ability to maintain critical and relevant information on future data, thanks to inputs from network nodes.

A feed-forward neural networks use a matrix of weights that will be applied to the inputs to produce the outputs. This is also the case for recurrent neural networks. In addition, these networks have the capacity to readapt this weight matrix through a backpropagation through time (BPTT) and a gradient descent.

For a better understanding of the idea of backpropagation through time, we will need to understand the two notions of forward and backpropagation. Mostly, in neural networks, forward propagation allows to have an output of the model and check if this output is correct. While the backward propagation makes it possible to find the deviation of the error by comparing it with the weight, which will subsequently allow this value to be subtracted from the weight.

These deviations are then processed by gradient descent, an algorithm that makes it possible to iteratively minimize a given function. Then it regulates the weights, in order to minimize the error. By these two notions, the neural network is capable to learn during the training phase. Figure 4 demonstrates the tow notions in a feedback neural network:

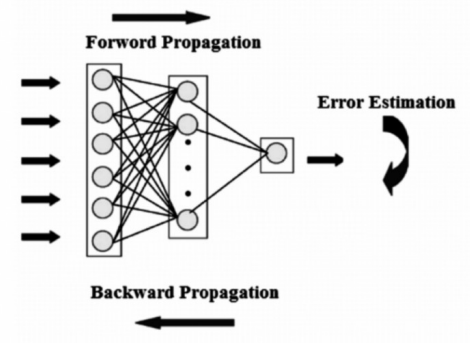

Fig. 4: Forward and backpropagation propagations in a feedback neural network.

3.2 Issues of standard RNN's

There are two main difficulties that RNNs have confronted, to understand those difficulties we need to explain what is a gradient.
A gradient is a fractional derivation compared to its input. In other words, the gradient calculates the percentage of the output's changes of a function when inputs are changed.

✓ Exploding gradients:
Sometimes, with no significant reason, the algorithm allocates a wrong high importance to the

wights, in this situation, we have an exploded gradient. This issue can be easily solved by squashing or truncating the gradients.

✓ Vanishing gradients:

From time to time, we could get small values for the gradient, which cause the model to stop learning or take too long time to give a significant result. This issue was harder to solve than the first one, but thanks to LSTM algorithms, this issue was solved.

3.3 Long Short Time Memory

Long Short-Term Memory (LSTM) is an artificial recurrent neural network. Different than the standard feed forward neural networks, they extend memory which make them capable to store significant information. The layers of a recurrent neural network compose the building units of LSTM algorithms and are called LSTM network. Using the LSTM in any sequential task will guarantee that long-term information and context are maintained.

In comparison with the standard type of RNN, LSTM units are not interconnected, but are considered as a block of memory that are connected in layers, each block is responsible of managing its state, its output and the flow of the information thanks to the gateways. Regarding the weight of the stored information, the LSTM blocks can choose whether to update or remove that information.

All LSTM blocks are composed of tree gates (input gate, output gate and forget gate), all these gates decide together whether to accept the new input, let it update the output or swape the memory of the block (Figure 5).

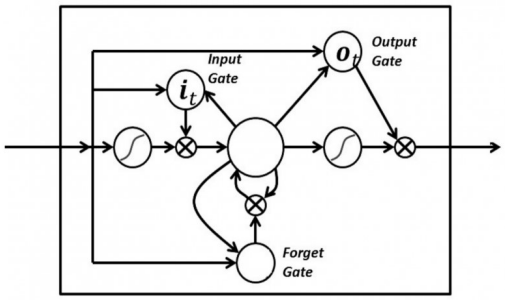

Fig. 5: Illustration of a recurrent neural network with three gates: input, output and forget

4. METHODOLOGY

In this section we explain our contribution, by demonstrating in details the steps followed. To the best of our knowledge, no related work has been reported addressing such a type of methodology with accurate prediction.

Before going through this section, here is a brief description of what we want to accomplish. Our contribution consists of implementing a prototype of an augmented reality application playing the role of a learning assistant through an intelligent 3D avatar, this Avatar will have the ability to respond to questions in the field of chemistry / biology through a learning model. This learning model is a generative model for generating human-understandable sequences of characters. The sequences or the character chains generated will be represented in the augmented reality application by the avatar.

4.1 Data collection, cleaning and analysis

Since we are working in the context of learning in Moroccan universities, we have chosen the second official language of Morocco (which is the French language) as the language of the data that will be used during the training phase of the model. Then, we started by looking for university courses, scientific books, communications, articles, documents, any textual data related to the field of chemistry and biology.

Tab. 1: Collected data fields and description

Field	Discipline	Description
Chemistry	Organics	This discipline interest in structure, properties and also reactions of organic composites
	Inorganique	This discipline is the opposite of the first one and it's aims to study properties, formation and synthesis of composites that don't contain carbon-hydrogen bonds
	Analytique	This discipline is interested essentially in identifying

		matters
	Physical	This discipline tries to apply physics concepts on macroscopic phenomena in chemistry such as motion and force
Biology	Anatomy morphology	Description of the inner and outer form of living things
	Psychology	Behaviors of living beings taken individually and among themselves
	Embryology	Development, living things
	Psychology	Analysis of the functions and functioning of living beings
	Botany, zoology, anthropology	Classification of living things
	Ecology, Ethology, Sociologies	Analysis and study of societies and organization of living beings

We classified the collected data by its theme (or sub-discipline). Table 1 represents the 10 sub-disciplines of the collected data. We obtained a text file for each discipline (e.g., organic.txt, psychology.txt).

In order to describe the data collected, we have carried out word processing for each text file. Table 2 represents the results of the treatments carried out.

Tab. 2: Statistics on collected data

File name	Number of lines	Total number of used characters	Total number of used words
Organique.txt	636	53757	8031
psychology.txt	957	75671	10351
Ecology ethology socilogi.txt	532	35954	6942

Textual data which is not processed and cleaned can't revel its insight. in our case, it's difficult to represent this textual data in graphs because it's not numerical. in order to give a graphical description of the collected data, we researched tools and techniques for visualizing textual data, and we found a way to graphically represent this type of data using a python library named "Word Cloud".

Word cloud is a technique that use the frequency of a word and associate it to the size and opacity of that word to create an image in order to visualize the textual data. Figure 6 shows the graphical representation of the collected data

Fig. 6: Textual data visualization: generated image using word cloud python library

As you can see, the most frequent words are words in the category of pronouns and determinants. With the given dataset, Word cloud does not give a clear view on the most frequent words that interest us. To try fix this problem, we used another tool called "TextKit" which will allow us to manipulate our data word by word.

Thanks to TextKit, we were able to perform the following 3 treatments in the order listed:

- ✓ Convert "organique.txt" text file into a token document where each token is a word.
- ✓ Take away stop words, in natural language processing, stop words are words so common that they provide little information about a document, and so are often removed.
- ✓ Filter words that are less than a certain number of characters long

Find bellow TextKit command line used to execute these treatments:

textkit text2words atome-fr.txt | textkit filterwords | textkit filterlengths -m 5 > output.txt

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

From the command above, the result was redirected to a new text file (output.txt). This output file was then used as an input to the word cloud library. Figure 7 shows a graphical representation of the dataset after cleaning.

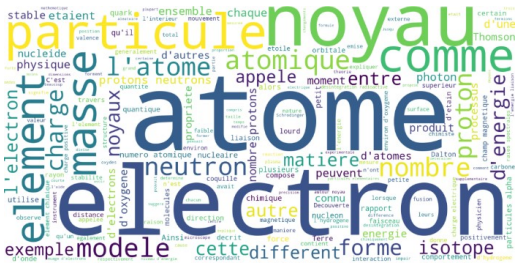

Fig. 7: Textual data visualization: generated image using word cloud python library

4.2 Model architecture

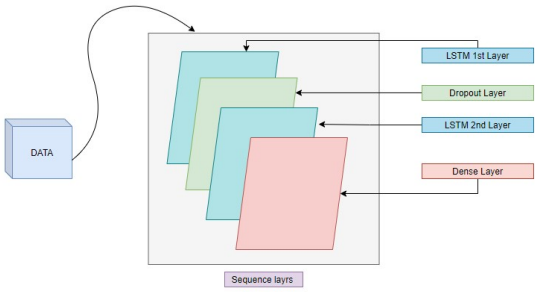

Fig. 8: Proposed model architecture

The architecture of proposed model consists of a sequence of 4 layers (Figure 8). The LSTM layers are the main layers of our model, they are the ones that will allow the model to learn thanks to their internal memory. The idea behind using dropout layer is to prevent the overfitting effect. For example, if the dropout is set to 0.2, then for each iteration within each epoch, each node in that layer has a 20 percent probability of being removed from the network. In this way, the model is capable to learn deeper and create important links.

The dense layer is entirely connected by nodes of the layer. Each node of each layer receives in its input all other nodes of the previous layer, they're densely associated. In that way, the learning will be more efficient, meaning that the model will learn all possible features from all the associations of the features of the previous layer.

Regarding the distribution of the collected and cleaned data into learning and validation dataset, we chose a method called cross-validation, this method has proved its effectiveness since it uses all the available data in an optimal way. In general, a large training and a small validation set in each iteration. For the construction of our model, we chose 80 percent of the data for training and 20 percent for validation.

While constructing the model, there are hyperparameters that have to be setup correctly and adjusted so that we can get an accurate prediction when back testing the model. This paper [14] have stimulated us on how we could organize experimental trials to find the best hyperparameters that will lead to higher accuracy and decrease the chance of overfitting the data, related to if no experimental trials was done to our model and data.

When performing the experimental trials, we will construct our LSTM model with default hyperparameters that fit our case best according to diverse papers that we find extremely valuable. We will then take each hyperparameter and attempt to detect an ideal value to that hyperparameter.

The best value for a hyperparameter is found by evaluating the LSTM model by back testing it with the test data. We will then calculate the "Categorical Cross Entropy (CCE)" loss function.

Cross entropy is generally used in multi-class classification tasks and it's used to measure the variance between two probability distributions. In the context of machine learning, it is the quantity of error for categorical multi-class classification problems

After all potential values for that specific hyperparameter has been evaluated we will plot on the "X" axis the hyperparameter value and on the "Y" axis the CCE value. We can then see that the hyperparameter value with lowest CCE is the most optimal one.

Figure 9 shows that 0.3 is the suitable value to choose for the dropout layer in our case because that value has the smallest categorical cross entropy.

Journal of Theoretical and Applied Information Technology
31st January 2022. Vol.100. No 2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

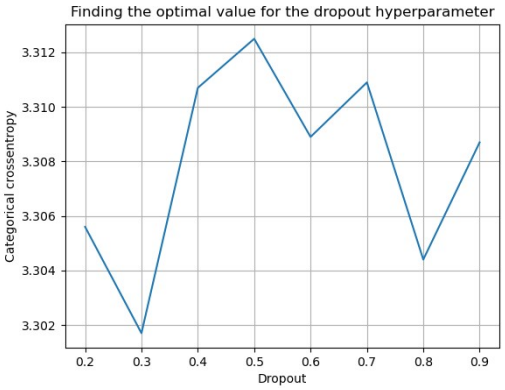

Fig. 9: Evaluation of the categorical cross entropy with different values of dropout hyperparameter

Each LSTM layer has a number of memory blocks or cells, to find the most optimal number of cells for those two layers. We trained the network with different values, from the figure 10, we can see that the more we use of cells, the more the CCE decreases (256 was chosen as the value of the number of cells of the two LSTM layers. We noticed, the more we increase this parameter, the more time the model needs to learn).

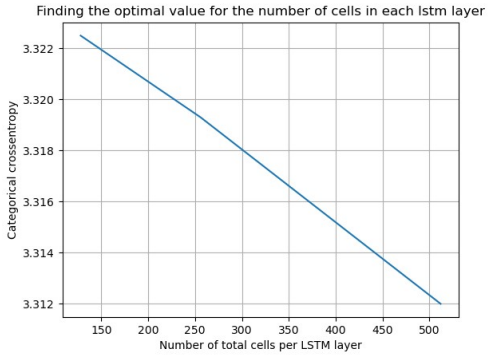

Fig. 10: Evaluation of the categorical cross entropy with different number of cells in each LSTM layer

To sum up (see Figure 11), the proposed model is a sequential model which is a linear stack of layers. The first layer is an LSTM layer with 256 memory block and it returns sequences. This is done to guarantee that the succeeding LSTM layer obtain sequences and not just randomly dispersed data. A dropout layer is applied after the first LSTM layer to avoid overfitting of the model by ignoring randomly selected neurons during training, and hence decreases the sensitivity to the specific weights of individual neurons. 30 percent is used as a good compromise between maintaining model accuracy and avoiding overfitting. Finally, we have the last layer as a fully connected layer with a "softmax" activation and neurons equal to the number of unique characters.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100, 256)	303104
dropout (Dropout)	(None, 100, 256)	0
lstm_1 (LSTM)	(None, 256)	525312
dense (Dense)	(None, 39)	10023

Fig. 11: The proposed model layers output shapes

Now that we have a clear understanding on how we built our model, the next step is the training phase. In order to improve this phase, we started by doing more cleaning processes on the dataset, the first thing was to reduce the number of words by eliminating upper-case characters and rare characters, such as punctuations, and replacing successive white spaces with one space character. We consider now that our dataset is clean enough, but we can't model characters directly, so we decided to translate characters to numbers (assign an integer to each character) and the samples will be based on those integers. The next step was the construction of the sequences, all inputs of all samples will share the same length "sequence_length", and the outputs would be always one single character.

To have a good understanding on how we constructed the sequences, let's do a demonstration. First let's assign the number 10 to the variable "sequence_length", and secondly let's take the following dataset: "chaque atome est composé de" (it would be represented by integers in the algorithm, just for the purpose of explaining we decided to simplify and use characters directly). Here are the tree first sample that will be generated based on the previous chosen dataset: ("chaque ato", "m"), ("haque atom", "e") and ("aque atome", " "). Note that each sample is composed of one input and one output. In that way we increase the number of samples and the model will be able to give convenable prediction with the minimum possible errors.

5. RESULTS

To be able to generate new text understandable by the human being, the model must learn on several samples. Showing to it many samples to learn features between inputs and

outputs. Each sample is composed of two things, the first one is a chain of characters (the input), and the second one is the next character (the output) that give sense to the hole characters combined together or that correct the phrase grammatically.

The model is now ready and the samples are created, we move to the next step which is the training phase. While the training process is running, in order to assess the performance of the model, we logged the values of loss and accuracy functions, Figure 12 represents the evolution of those two functions during the training process.

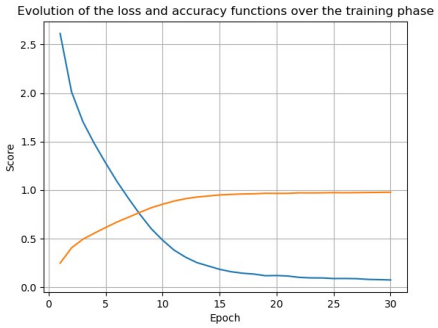

Fig. 12: Evolution of the loss and accuracy metrics over epochs (blue trait represents the loss function. The orange trait represents the accuracy function)

- ✓ The loss function lets you know how the model behaves on the test and validation datasets. Technically it's the sum of the errors identified in that dataset.

- ✓ The accuracy metric let you know assess the performance of the model, technically it's the number of correct predictions divided by the total number of predictions.

The above two metrics are not fully sufficient to assess the performance of the model, there are other metrics that we can rely on, precision and recall functions. We definitely need to calculate these two metrics, but before that, what are these two metrics and how they work:

- ✓ Precision is a metric which allows to calculate the number of positively correct predictions, this metric is very important because it helps to know the performance of the model when there are false positives. For example, assuming we are making predictions about a dangerous disease, if the value of that metric is low, then many patients will be identified as sick with a false diagnosis.
- ✓ The recall metrics gives the sum of true positives that were found or identified, different than the precision metric, the recall acts only on correct positives predicted out of all positive predictions. It gives some knowledge on the coverage of all positive predictions.

Figure 13 is an explanatory graph of the values of these two metrics obtained during the model training phase.

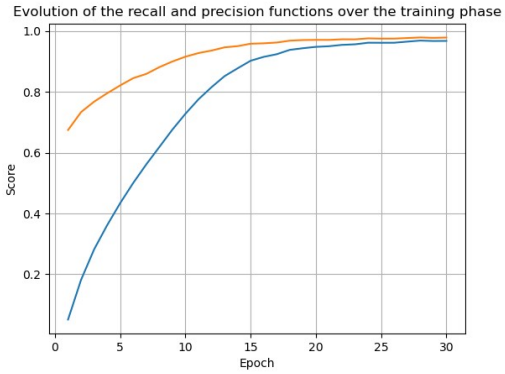

Fig. 13: Evolution of the recall and precision metrics over epochs (blue trait represents the recall function. The orange trait represents the precision function)

We can combine the above two metrics (precision and recall) and get a new useful metric. It's called F1 and it measures the accuracy of the model.

When the false positives and the false negatives are few, F1 has a value close to 1, in this case the model is considered as perfect. In case the F1 score is close to 0, the model is a total letdown.

Bellow an example of calculation of this metric (calculated for the last epoch):

$$\mathrm{F1} = 2 * \frac{precision * recall}{precision + recall} = 2 * \frac{0.9783 * 0.9673}{0.9783 + 0.9673} = 0.97$$

The F1 score is close to 1, the prediction made by our model are very accurate.

Now that we built, trained and evaluated the model, we move to the next step which is the generation of the text. The two figures below (Figures 14 & 15) represent the algorithm used during this phase of text generation.

```python
import numpy as np
import pickle
import tqdm
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Activation
import os

sequence_length = 100
# dataset file path
FILE_PATH = "data/atom.txt"
BASENAME = os.path.basename(FILE_PATH)
# load vocab dictionaries
char2int = pickle.load(open(f"results/atom.txt-char2int.pickle", "rb"))
int2char = pickle.load(open(f"results/atom.txt-int2char.pickle", "rb"))
vocab_size = len(char2int)

# building the model
model = Sequential([
    LSTM(256, input_shape=(sequence_length, vocab_size), return_sequences=True),
    Dropout(0.3),
    LSTM(256),
    Dense(vocab_size, activation="softmax"),
])
# load the optimal weights
model.load_weights(f"results/atom.txt-100_6.h5")
```

Fig. 14: Text generation algorithm – part 1

```python

# specify the seed to first characters to generate
seed = "Les électrons d'un atome sont attirés par les protons d'un noyau atomique sont attirés par chacun"

s = seed
# generate 400 characters
n_chars = 400
generated = ""
for i in tqdm.tqdm(range(n_chars), "Generating text"):
    # make the input sequence
    X = np.zeros((1, sequence_length, vocab_size))
    for t, char in enumerate(seed):
        X[0, (sequence_length - len(seed)) + t, char2int[char.lower()]] = 1
    # predict the next character
    predicted = model.predict(X, verbose=0)[0]
    # converting the vector to an integer
    next_index = np.argmax(predicted)
    # converting the integer to a character
    next_char = int2char[next_index]
    # add the character to results
    generated += next_char
    # shift seed and the predicted character
    seed = seed[1:] + next_char

print("Seed:", s)
print("Generated text:")
print("##BEGIN##")
print(generated)
print("##END##")
```

Fig. 15: Text generation algorithm – part 2

We first started by finding a sentence, that will be the start of the text which will generate subsequently. We consider this sentence as a seed of the final text. This seed allowed us to create the first sequence of model node inputs to predict the next character. We predict the next character on the sequence which will be shifted at each iteration by removing the first character and adding the last predicted character at the end of the sequence, we were able to generate text using the model.

Here is an interesting generated text:

“Autre par la force atomique, les électrons dans cette couche sont appelés électrons de valence le nombre d'électrons de valence forment un groupe qui est aligné dans la même colonne du tableau la force plus forte qui est médiée par les gluons les protons et les neutrons sont leurs tour hliym jusqu'au noyau plus l'antiproton est un equibt chargé négativement d'une absorption sur du rayonnement de”

Note the above observations about the generated text:

- ✓ That is clearly French but as you may notice, most of the sentences doesn't make sense.
- ✓ It generally conforms to the line format observed in the original text.
- ✓ Almost all the words in sequence make sense (e.g., “les électrons dans cette couche sont appelés “), and fewer do not (e.g., “hliym, equibt “).
- ✓ In general, the text looks realistic, but is still quite nonsensical.

6. PROTOTYPE

As described briefly in the beginning of the methodology section, we dedicate this part to provide a prototype of an architecture that can handle both machine learning and augmented reality capabilities. Our aim is to help students or learners through an augmented reality application to understand the subject they are trying to study. We believe that augmented reality is an innovative way to help learners, as the application will interact with the learner via a virtual avatar that can generate content with the help of machine learning.

There are several development kits that allow the creation of augmented reality applications, this was not a problem for our goal. But the real challenge was how we could create an application with augmented reality capabilities while integrating the use of the generative model that we implemented and discussed in the previous parts.

It was absolutely necessary that we first prepare a technical architecture that will allow us to define the components of our application, taking into account the diversity of technologies in the development kits.

6.1 Architecture

The architecture of our prototype is based on three main components:

- ✓ an augmented reality mobile application
- ✓ a web server-based application

- ✓ a python script that will use the model we implemented

The first component is the mobile application with augmented reality capabilities, which is based on the AR Core Framework developed by google. Using this Framework gives the developers flexibility to designate their application as they wish since it relies on the architecture of an android application. Thus, we can target a very large number of users since the android platform is the most used operating system in the majority of mobile terminals.

The mobile application will be able to detect real scene objects and generate 3D virtual models and avatars that the user can interact with. 3D models will also be able to respond to the user by automatically generating text. Thus, we can get the help of the operating system services to transform the generated text into a speech that will be launched by the speakers of the mobile providing a better experience regarding the human machine interaction.

The second component is a web server-based application that will be installed on a server and runs on the background. The purpose behind this component is to receive requests "using HTTP protocol" that will be sent from the mobile application. These requests will carry information on the language, the domain, the size, …etc., of the text to be generated, roughly the generative model to be used.

This web application will be able to process these requests and get the help of the third component of this architecture. The third component is a script based on the Python language, which will finally use the generative model to generate new text.

7. DISCUSSION

Deep learning models such as LSTM have proven to be effective in the tasks of translation, handwriting generation, image generation using attention models, image caption and video to text, however, in this paper, we proposed a model based on LSTM and we used it in generating new understandable text based on the text itself. We investigated a deep learning approach on textual data, we predict a character based on a string of characters to generate new text. So that this text is understandable by humans, we've tried to train the model several times, taking into consideration different metrics, and datasets.

The values obtained from the calculated metrics and those chosen for the hyperparameters show clearly that our model makes good predictions. In fact, more than 97 percent of the generated words are correct. We believe that this is the main advantage of the proposed methodology.

Based on the proposed prototype, and in order to highlight our work, we present an interesting use case:

In the context of learning, several courses require a basic learning support, whether for learning a new concept or to improve one's knowledge on a specific subject. Unfortunately, in a study class, the support may not be adapted to the learning capacities of all learners (different skill levels, learner motivation and interest level). All these challenges are faced by teachers and demand more effort in their work. The proposed model can be used to automatically generate educational content and facilitate the construction of an adaptative educational support. Moreover, the proposed prototype architecture can easily handle the diversity of technologies and offers an innovative way to improve the learning experience with the help of machine learning and augmented reality capabilities.

We present now some of the methods to generate text that have been already discussed.

Recent research conducted by [17], aims to generate meaningful and diverse questions from natural language sentences, based on a neural encoder-decoder model. the role of the encoder is to produce an input representation taking the response into account, which is passed to the decoder to generate a response-driven question. The proposed model is a sequence-to-sequence model which enrich the encoder with response and lexical functionalities to generate response-oriented questions. while the decoder is based on the attention mechanism. The authors performed a preliminary study on generating neural questions from text with the SQuAD dataset, the results of experience found that their method can produce fluid and diverse questions. Regarding the proposed method, our major concern lies in one aspect, the attention mechanism can only deal with fixed-length text strings. The text has to be split into a certain number of segments or chunks before being fed into the system as input, while the chunking of text causes context fragmentation. For example, if a sentence is split from the middle, then a significant

amount of context is lost. In other words, the text is split without respecting the sentence or any other semantic boundary.

Another study, led by Mitkov and Ha [15] aims to provide an alternative to the long and demanding activity of developing multiple choice tests and proposes a new NLP (Natural Language Processing) approach to generate questions from narrative texts (e.g., manuals, encyclopedias). The proposed methodology is based on premises and the system is built on separate components, which perform the following tasks: extracting terms, selecting distractors and generating questions. More specifically, the model is based on well-defined rules, for example, the generation of a question in the following form "what do/does/did the subject verb?" from a sentence in the form of a subject followed by a verb and an object. However, in another study [16], led by the same authors, the automatically generated questions were revised and the authors confirm that the model is not efficient without a manual correction. Regarding this methodology, our main concern lies in the complexity of the proposed model and the requirement of the dataset premises.

8. CONCLUSION & RECOMMENDATION

We now have a good understanding of how a neural network works, which allows us to adopt the right algorithm to use for a given machine learning problem. More specifically, we learned what the difference between a Feed-Forward Neural Network and a recurrent neural network is, when should we use a Recurrent Neural Network, how back-propagation and forward-propagation work over time, what are the key problems with recurrent neural network and how an LSTM solve these issues.

We've described in details the steps followed to create the model for text generation, we provided an architecture prototype of a mobile application which uses the proposed model and benefits from the advantages of augmented reality to highlight our work.

We also talked about generative models. As we continue to advance these models and scale up the training and the datasets, we can expect to eventually generate samples that represent entirely comprehensible sentences.

However, the deeper promise of this work is that, in the process of training generative models, we will endow the computer with an understanding of the textual information and what it is made up of.

In order to further improve the generated model, we can:

- ✓ minimize the vocabulary scope by eliminating rare characters.
- ✓ create the model with additional LSTM and dropout layers with more LSTM units, or even add bidirectional layers.
- ✓ adjust some hyper parameters such as batch size, optimizer and even sequence length.
- ✓ use a larger dataset.
- ✓ train on more epochs.

REFRENCES

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics, vol. 5, no. 4, pp. 115– 133, 1943. 7

[2] F. Rosenblatt, "The perceptron, a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, 1957. 7

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998. 7

[4] D. Lowe and D. Broomhead, "Multivariable functional interpolation and adaptive networks," Complex systems, vol. 2, no. 3, pp. 321–355, 1988. 7

[5] T. Kohonen, "Self-organized formation of topologically correct feature maps," Biological cybernetics, vol. 43, no. 1, pp. 59–69, 1982. 7

[6] D. O. Hebb, "The organization of behavior: A neuropsychological theory," Psychology Press, 1949. 7

[7] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Harvard University, 1974. 8

[8] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in System modeling and optimization, pp. 762–770, Springer, 1982. 8

[9] D. B. Parker, "Learning logic," MIT, 1985. 8

[10] Y. LeCun, "Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks)," in Proceedings of Cognitiva 85, Paris, France, 1985.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

propagating errors," Nature, vol. 323, no. 6088, pp. 533–538, 1986. 8

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., DTIC Document, 1986. 8, 12

[13] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural networks, vol. 2, no. 5, pp. 359–366, 1989. 8

[14] N. Reimers, I. Gurevych "Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging "Association for Computational Linguistics, pp. 338–348, 2017. 9

[15] Mitkov, R. and Ha, L. A. (2003). Computer-aided generation of multiple-choice tests. In Proc. of the HLT-NAACL workshop on Building educational applications using natural language processing.

[16] Mitkov, R., Ha, L. A., and Karamanis, N. (2006). A computer-aided environment for generating multiple-choice test items. Natural Language Engineering, 12(2).

[17] Zhou Q., Yang N., Wei F., Tan C., Bao H., Zhou M. (2018) Neural Question Generation from Text: A Preliminary Study. In: Huang X., Jiang J., Zhao D., Feng Y., Hong Y. (eds) Natural Language Processing and Chinese Computing. NLPCC 2017. Lecture Notes in Computer Science, vol 10619. Springer, Cham. https://doi.org/10.1007/978-3-319-73618-1_56