Building a Deep Learning Model to Generate Human Readable Text Using Recurrent Neural Networks and LSTM

Anasse HANAFI (anasse.hanafi94@gmail.com)
Abdelmalak saadi University of Taniger
Mohammed BOUHORMA
Abdelmalak saadi University of Taniger
Lotfi ELAACHAK
Abdelmalak saadi University of Taniger

Research Article

Keywords: Artificial intelligence, machine learning, classification, Recurrent Neural Networks (RNNs), sequence models, generative models.

DOI: https://doi.org/10.21203/rs.3.rs-753724/v1

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, with the hope to better control the output of the generated text, because it is not always possible to learn the exact distribution of the data either implicitly or explicitly.

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 (Mcculloch and Pitts 1943). 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 goal of our work is not the classification of text itself, but as mentioned in the abstract, the use of a generative model to train on textual dataset, to generate entirely a new sequence of text for a problem domain.

Now days computers are able to understand the real world thanks to machine learning. One of the approaches in the unsupervised machine learning is generative modeling. These generative models are used to describe phenomena in data, and considered as an artificial intelligence, capable of predicting probabilities on a subject from the modeled data (Rosenblatt 1957)

Generative models have the ability to create data similar to the training data they have received because they have learned the distribution from which the data is provided. For example, if a generator model gets in its input a set of images of a certain object, the model should be able to understand what characteristics belong to a certain class and how they can be used to generate similar images.

To be able to create a generative model, the first required task is to collect a large amount of data in some domain (for example millions of images, sounds, financial data, metrological data, …etc.). The next step is to train the model to generate similar data.

Recurrent neural networks are considered as generative models, they have a small number of parameters compared to the collected data, and are forced to discover and efficiently internalize the essence of the data in order to generate it.

2. State Of Art

With the rapid increase in computing capacities of computers and the evolution learning techniques, the use of neural networks is very popular development in particular within the image processing and automated translation communities and speech processing. In recent years, two types 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 (Fig. 1)

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 then 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 preferably 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 system control, as a prediction or as 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 (Rosenblatt 1957), convolutional (LeCun and bottou 1998), Radial Basis Function (Lowe and Broomhead 1988) or the Kohonen maps (Kohonen 1982). But the most frequently used variant is the 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 and an output layer. For each of the layers, all of its nodes are connected to all the nodes of the previous layer. Figure (Fig. 2) gives a representation of a multi-layered perceptron with one hidden layer.

The origin of the multi-layered perceptron goes back to F. Rosenblatt who in 1957 was inspired work on formal neurons (McCulloch and Pitts 1943) and on Hebb's rule introduced in (Hebb 1949) to develop the model of the perceptron but also the method which allows this model of learning (LeCun and Bottou 1998).

This model has only one layer but already allows solving 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 function XOR). 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), 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). It was finally in the mid-1980s that this method was rediscovered by several research teams (Werbos 1982; Hornik 1989) and that it ended up being popularized.

In (1986), D. umelhart, G. Hinton and R. Williams show that using the back-propagation of the error gradient applied to a multi-layered perceptron we 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 and can approximate, with only one hidden layer and a sufficient number of neurons, any nonlinear and continuous function over a compact space with arbitrary precision (Lecun 1985). 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 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 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 (for example a classification) 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. Theoretical 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 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 two times. Because feed forward neural networks can’t store their inputs, they are considered weak at predicting next inputs they could obtain. Also, inputs are no 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 (Fig. 3) demonstrate the variance in information flow among a recurrent neural network and a feed-forward neural network.

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 storage or memory of recurrent neural network, this kind of network 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 (Fig. 4) demonstrates the tow notions 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.

3.2.1 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.

3.2.2 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 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.

LSTM are capable to remember their inputs thanks to their internal memory, also they are capable to update the stored value on this memory.

Regarding the importance or the weight of the stored value, the LSTM cell can choose whether to update or remove that value. Note that the weight of the value to store are made by the algorithm over time, in other word, the algorithm is capable to choose the right value to be stored.

All LSTM cells 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 cell. Figure (Fig. 5) illustrate an LSTM cell:

4. Methods

In this section we explained briefly our contribution, and we focused on demonstrating in details the steps we followed that allowed us to reach our objective.

4.1 Data collection, cleaning and analysis

In order to carry out our thesis, we have chosen to implement 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 when he answers the learner's questions.

To start our work, we have chosen the French language as the language of the data that will be used during the training phase of the model since we are working in the context of learning in a Moroccan university.

To begin our work, we started with the data collection phase. These data will be used as a learning database for our generative model.

Since we have chosen the field of chemistry / biology, we started by looking for university courses, scientific books, communications, articles, documents… any textual data related to the field of chemistry and biology.

Subsequently, we had to make a classification of all these data by their theme (or sub-discipline), the table (Table 1) represents the 10 sub-fields that we have chosen:

Table 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.

After classifying the data, we obtained a text file by discipline (e.g.: organic.txt, morphologie_anatomie.txt…).

In order to describe the data collected, we have carried out word processing in order to give a graphic representation. Below an example summary table (Table 2) of the treatments carried out:

Table 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

Text 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 or tables because it's not numerical. That's why 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 textual data. We have tried to apply this technique on our data, bellow an example of the image (Fig. 6) we've got:

As you can see in the picture above, the most frequent words are words in the category's 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 "TextKit" which will allow us to manipulate our data word by word.

Thanks to TextKit, we were able to perform the following 4 actions in the order listed:

- converts our "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
- store generated text in a new file as a new dataset to use

Find bellow TextKit command line used to execute these actions:

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

After storing the result of the command line in a new file "res.txt", we applied word cloud on that new file, bellow the generated image (Fig. 7) of the most frequent used words:

As you can see, after processing our data with TextKit, we get most frequent words related to the field of our interest.

4.2 Model architecture

The architecture of our model consists of a sequence of 4 layers (Fig. 8):

1. First LSTM layer
2. Dropout layer
3. Second LSTM layer

IV. Dense layer

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. 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 important links.

Dense layer, as suggests by the name, this 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.

While constructing the LSTM model there are hyperparameters that have to be setup correctly and adjusted so that we can get an accurate prediction when back testing our model. Reimers and Gurevych (2017) 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 (Fig. 9).

From the above Figure (Fig. 9) 0.3 is the suitable value to choose in our case, because that value has the smallest categorical cross entropy.

Each LSTM layer has a number of Cells, we want to find the most optimal number of cells for those two layers. We trained the network with different values on each cell in the LSTM layers, from the figure (Fig. 10) we can see that the more we use of cells, the more the CCE decreases.

5. Results

To be able to produce text understandable by the human being using RNN and LSTM algorithms, we started by creating a model and 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. With an example, the form of samples would be mush essayer to understand, so let’s take the following sentence: “An atom is the smallest unit of ordinary matter that forms a chemical element” the first input would be: “An atom is the smal” and the output will be the single character “l”. the second sample would be: “n atom is the small” and the output will be the single character “e”. Continuing with this procedure until we get to the last character of the chosen sentence. To have fairly correct predictions with the minimum possible errors, our model must learn on several samples.

For the purpose of presenting our results, we start by introducing the steps we followed to get to our end:

1. Preparing the dataset
2. Building the model
3. Training the model
4. Generating new text

To improve the training phase, we started by doing more cleaning processes on the dataset, the first thing was to reduce the number of words, eliminate 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.

After that, we started by constructing our sequences, all inputs of all samples will share the same length "sequence_length", and the outputs would be always one single character. That's our dataset.

To get a good understanding of what we have done so far, 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 be able to give convenable prediction.

So far, we have just completed the first step (Preparing the dataset) of our implementation, now we can start building the model.

As we saw in the method section, our model will be a sequence of four layers, two LSTM Layers speared by a dropout layer and a final Dense layer.

Now that we built our model, we started the training step. While the training process is running, we logged the values of loss and accuracy functions, bellow a graph (Fig. 11) presenting the evolution of those values during the training process (loss in blue, accuracy in orange).

1. 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.
2. 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.

There is other metrics that we can rely on to assess the performance of the model (precision & recall), the above two metrics are not fully sufficient, we definitely need to calculate these two metrics, but before that, what are these two metrics and how they work:

1. 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.
2. 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.

below an explanatory graph (Fig. 12) of the values of these two metrics obtained during the model training phase (recall in blue, precision in orange):

When the prediction on classes is very imbalanced, we can combine the above two metrics (prevision 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 for this metric (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 the model are very accurate.

Now that we built, trained and evaluated our model, we move to the next step which is generating the new text.

We first started by finding a sentence, that will be the start of our text which will generate subsequently. We consider this sentence as a seed of the final text. This seed allowed us to create the 1st sequence of model node inputs to predict the next character. We repeat this procedure on the sequence which will be shifted at each iteration by removing the 1st 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:

“Other by the nuclear force thin the electrons in that shell are called valence electrons the number of valence electrons form a group that is aligned in the same column of the table the hore more strong force which is mediated by gluons the protons and neutrons in turn are helium up to the nucleus

the greater the antiproton is a negatively charged equiribt of a sure to absorption of radiation of”

Note the above observations about the generated text:

- That is clearly English 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.
- Many of the words in sequence make sense (e.g., “the nuclear force “), and fewer do not (e.g., “hore, equiribt “).
- In general, the text looks realistic, but is still quite nonsensical.

6. Prototype

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 with a virtual teacher and 3D contents 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.

Architecture

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

1. Augmented reality mobile application
2. Web application
3. 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 developer 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 on the majority of mobile terminals.

This application will be able to detect real objects “in our case, images containing QR codes” and generate 3D objects “an avatar” that the user can interact with. 3D models or avatars will also be able to interact with the user by automatically generating text. To make this interaction more human, we used the models provided by the Android APIs to transform the text generated through our model into a speech that will be launched by the speakers of the mobile.

The second component is a web application that will be installed on a server and will run in the background.

The purpose behind this application is to receive requests “using the 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 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 simply a script based on the Python language, which will finally use the generative model for the generation of the text.

As you can see from the screen shot (Fig. 13), the developed prototype uses the camera of the device to render the 3D content merged with the real environment when the corresponding image (QR code) is detected.

Then the application uses the model to generate text that will be played by the speakers of the device.

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, we used it in generating new text based on the text itself.

In this work, we investigated deep learning approaches on textual data. We predict a character based on a string of characters to generate text. So that this text is understandable by humans, we have tried to train the model to generate several times, different metrics, and with different dataset.

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

Several courses require basic learning support, whether for reading itself or to learn a new concept or even improve your knowledge on a specific subject. Unfortunately, in a study class, the support may not be adapted to the learning capacities of all the learners (different skill levels, learner motivation level, learner interest, etc.). All these challenges are faced by teachers and their demands for more effort in their work. Learning apps that can deliver adaptable educational content can make it easier for teachers. A recent research has been carried out in this direction allowing the development of an application using a model capable automatically generating questions on different themes based on text, which simplifies the construction of the educational support as well as the controls and evaluations.

A study carried out by Mitkov and Ha (2003) which aims to generate questions in a precise form while using a model which is based on well-defined rules. An example of a rule is 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. They were also able to generate questions for multiple choice question type exams. The generated model was assessed by mediating the eminence of question produced from an online linguistics textbook. In another study, the automatically generated questions were revised, Mitkov and Ha (2006) confirmed that the combination of automatic generation of questions with manual correction may be more efficient than automatic generation without human supervision. What is needed assuming that generated questions will need human's verification.

Kunichika and al (2004) define a model that generate questions founded on syntactic and semantic investigates which are obtained using Definite Clause Grammar (Pereira and Warren, 1986). They verified their model by judging whether generated questions are "semantically correct," unfortunately their exact assessment criteria and process are not very clear.

Gates (2008) took a different approach to be able to generate text in the form of questions, the model created by gates triggered the questions the moment the reader is reading a text to inspire him to "look back" and re-read the text. Gates's model uses phrase structure parses and the Tregex tree searching language.

8. Conclusion

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 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:

1. Minimize the vocabulary scope by eliminating rare characters.
2. Create the model with additional LSTM and Dropout layers with more LSTM units, or even add Bidirectional layers.
3. Adjust some hyper parameters such as batch size, optimizer and even sequence length, and see which works perfectly.
4. Use a larger dataset.
5. Train on more epochs.

Declarations

Author contributions statement

All authors prepared and wrote the manuscript with figures.

All authors reviewed the manuscript.

References

[1] McCulloch, W. S. & Pitts, W. “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] Rosenblatt, F., “The perceptron, a perceiving and recognizing automaton,” Cornell Aeronautical Laboratory, 1957. 7

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

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

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

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

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

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

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

[10] LeCun, Y. “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] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. “Learning representations by back-propagating errors,” Nature, vol. 323, no. 6088, pp. 533–538, 1986. 8

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

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

[14] Reimers, N., Gurevych, I. “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. & 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] Kunichika, H., Katayama, T., Hirashima, T., and Takeuchi, A. (2004). Automated question generation methods for intelligent English learning systems and its evaluation. In Proc. of ICCE

[18] Gates, D. M. (2008). Generating reading comprehension look-back strategy questions from expository texts. Master’s thesis, Carnegie Mellon University

Figures

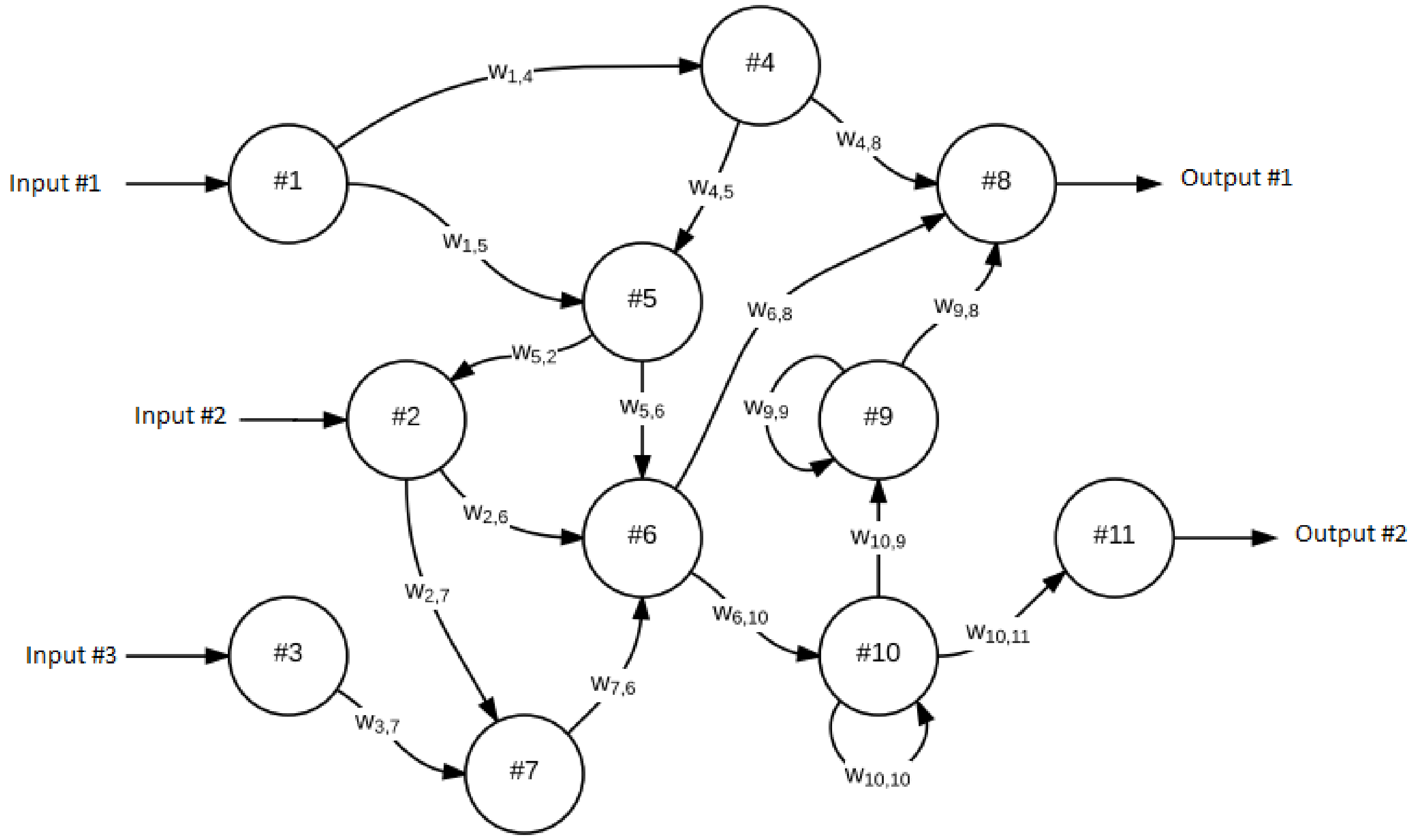

Figure 1

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 w(i,j)

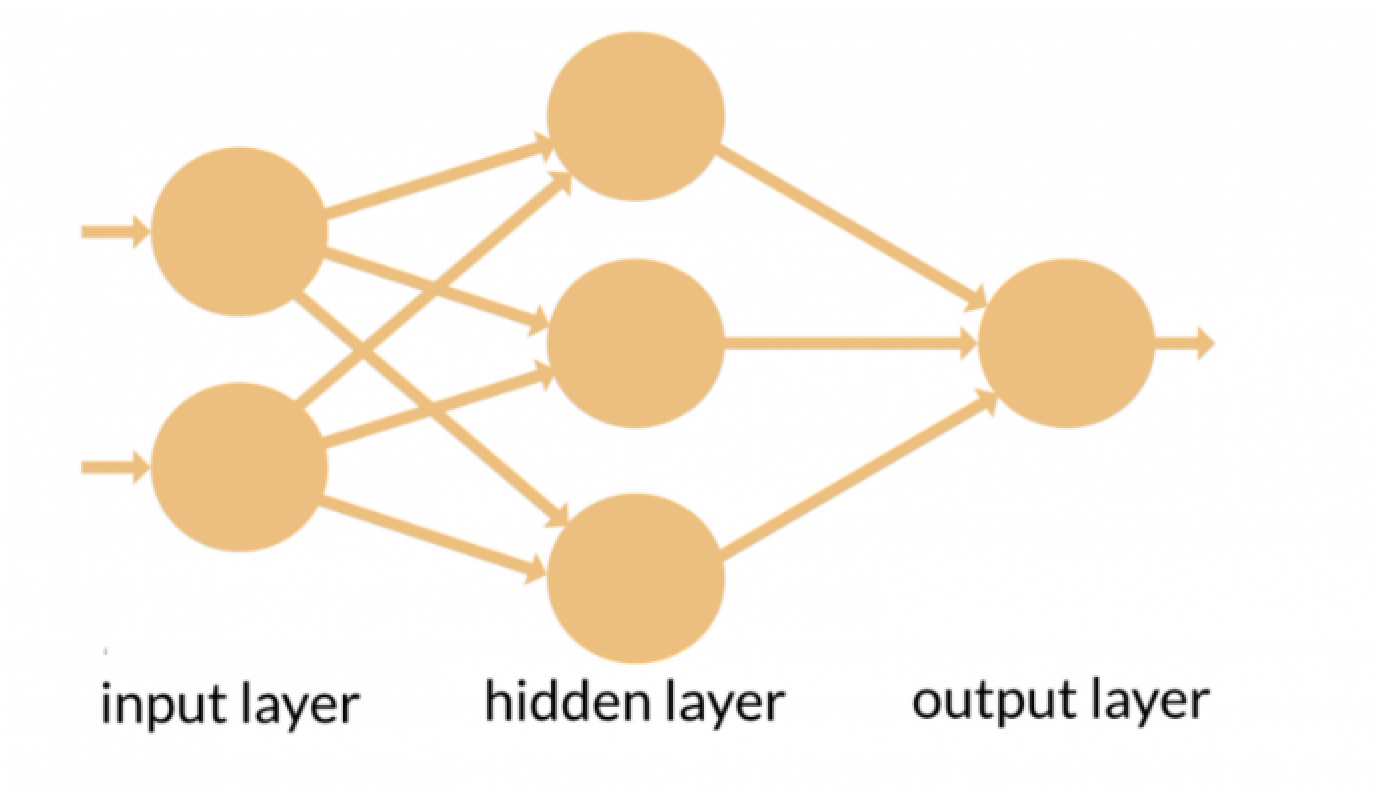

Figure 2

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

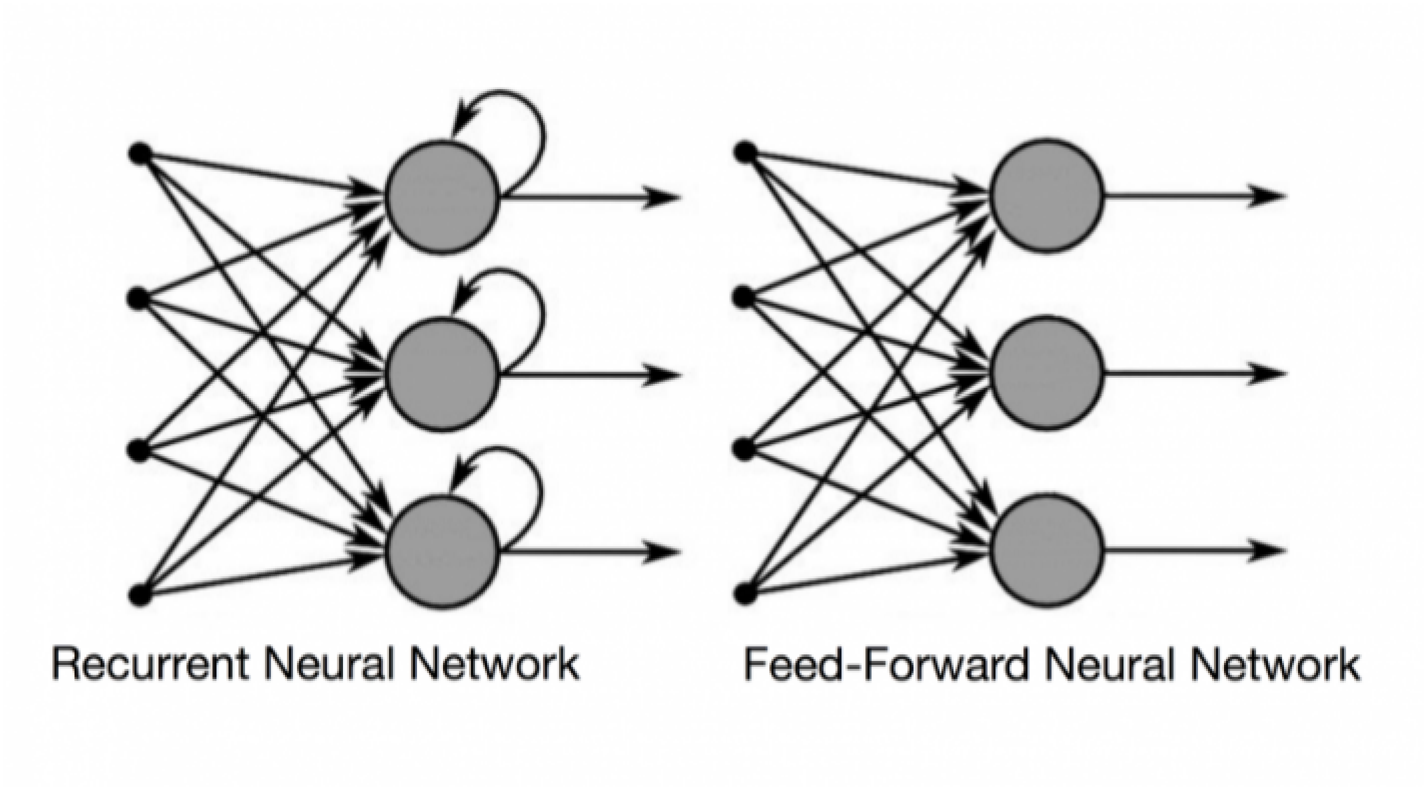

Figure 3

RNN & feed-forward neural networks information flows

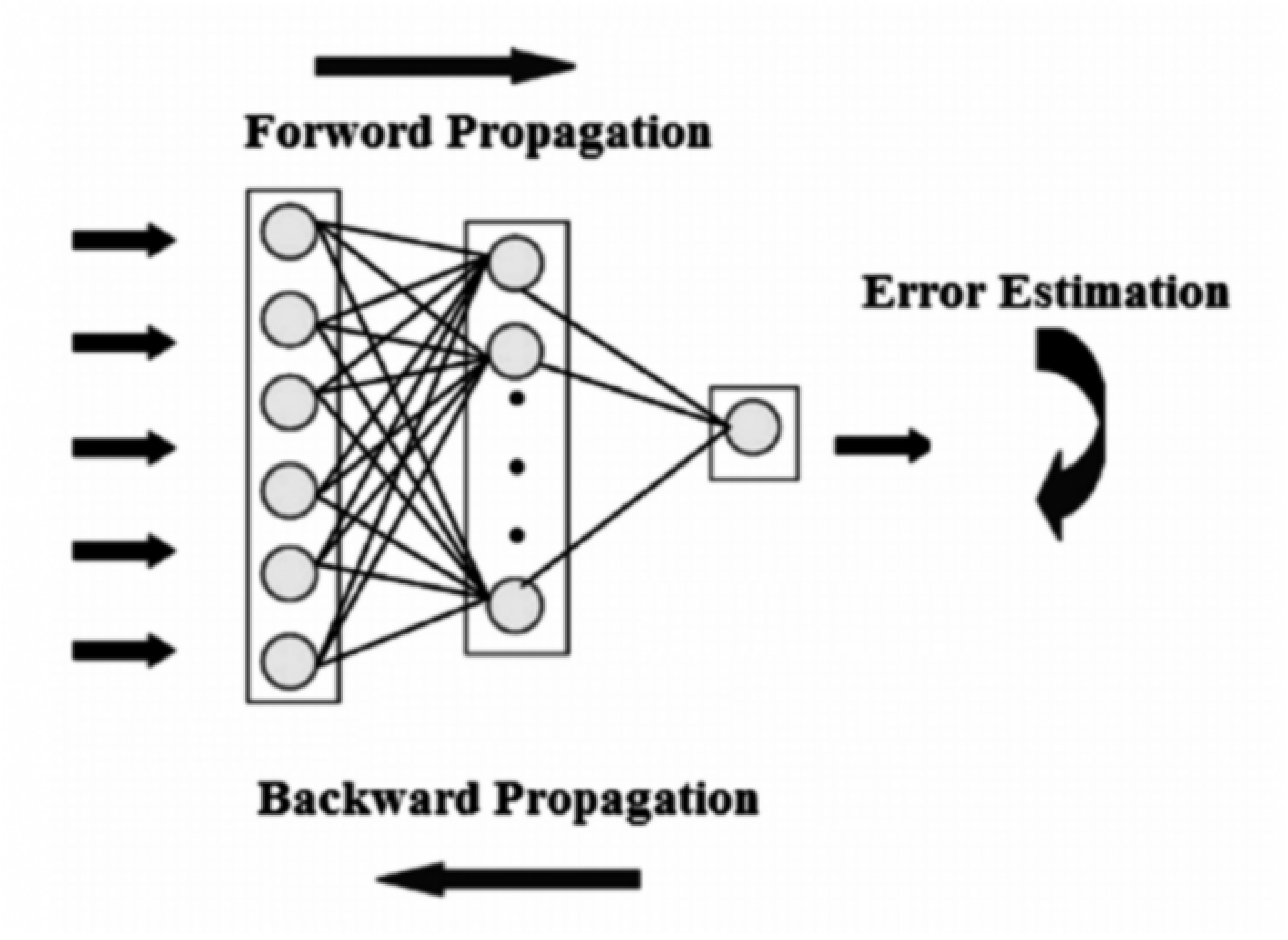

Figure 4

Forward and backpropagation propagations in a feedback neural network.

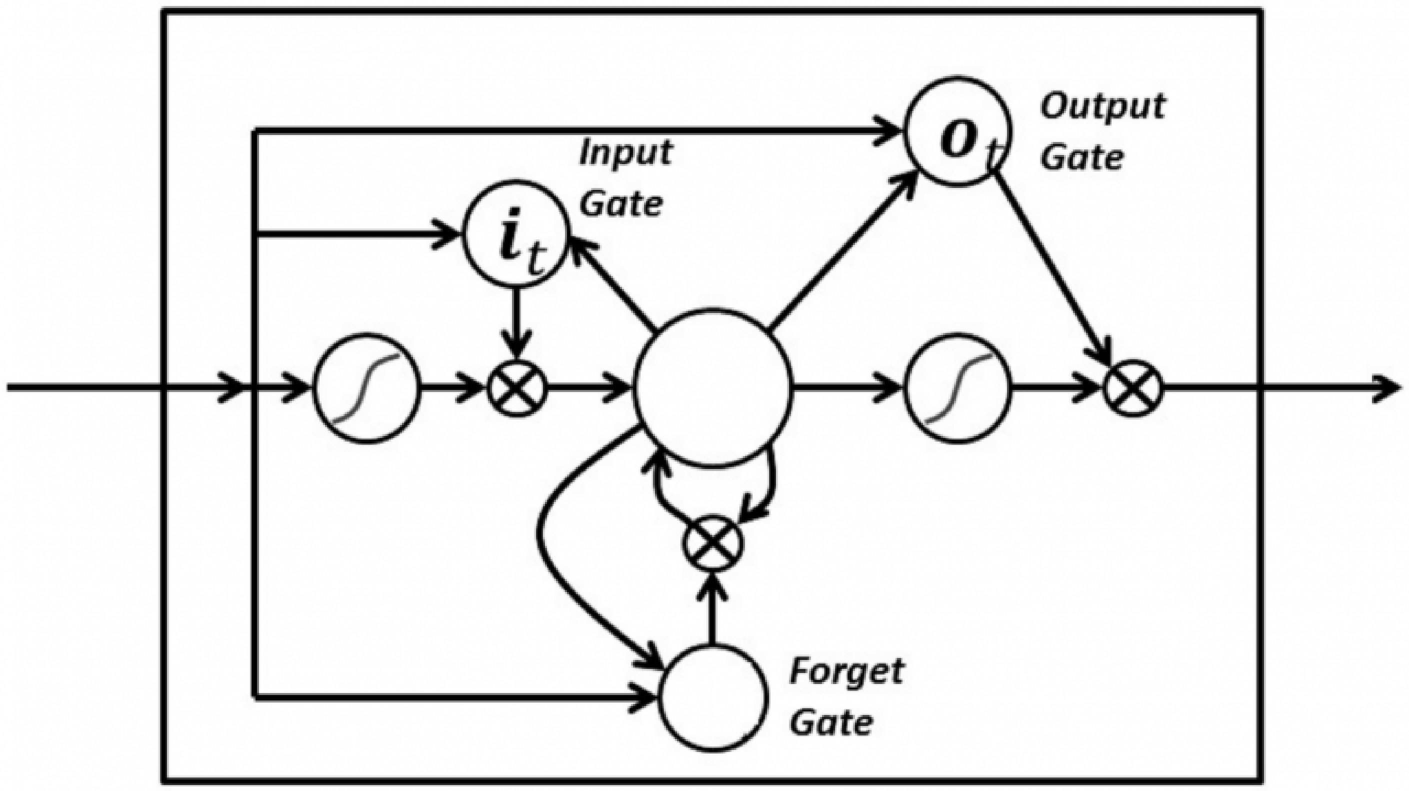

Figure 5

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

de
le
et
est
en
pour
une
la
un
par
atome
électron
dans
sont
plus
avec
qui
particule
les électrons
atomique
élément
noyau
proton
neutron
comme
isotope
masse
modèle
nucléaire
les atomes
énergie
chimique
photon
nucléide

Figure 6

Textual data visualization: generated image using word cloud python library

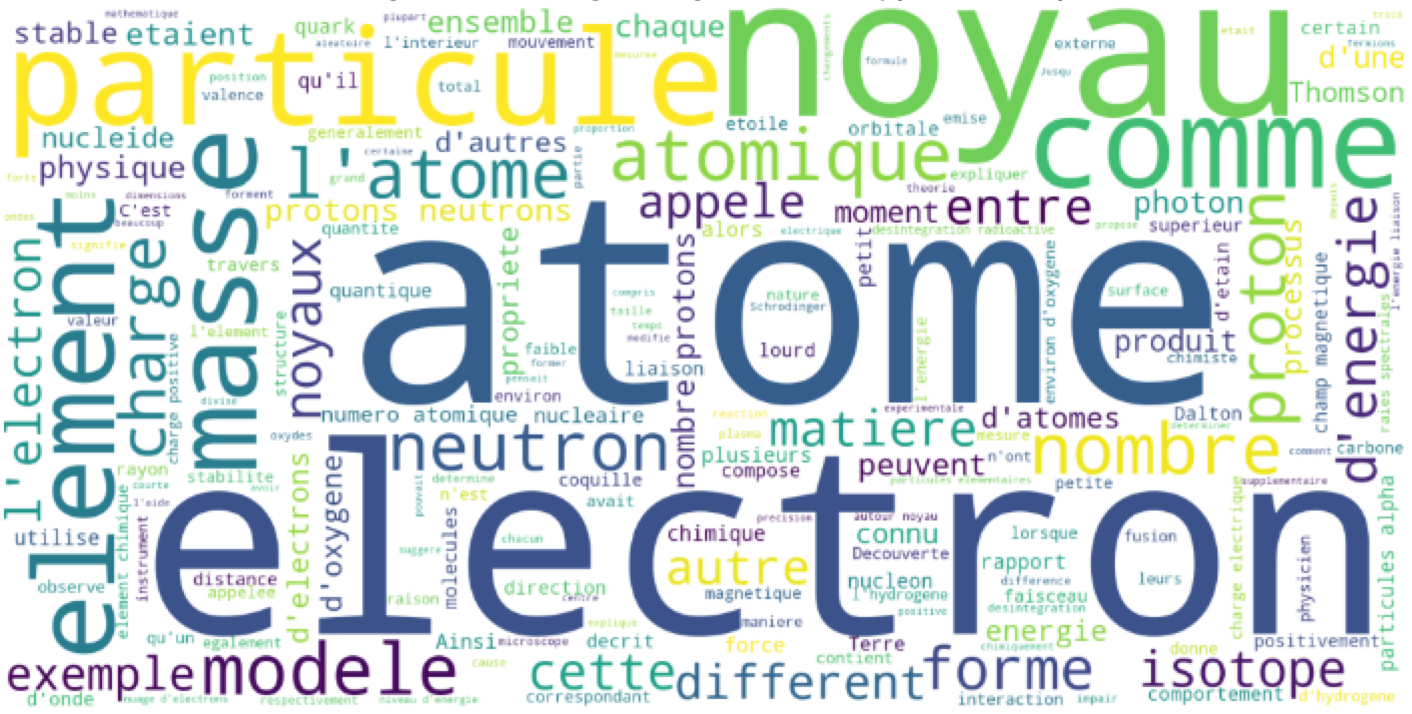

Figure 7

Textual data visualization: generated image using word cloud python library

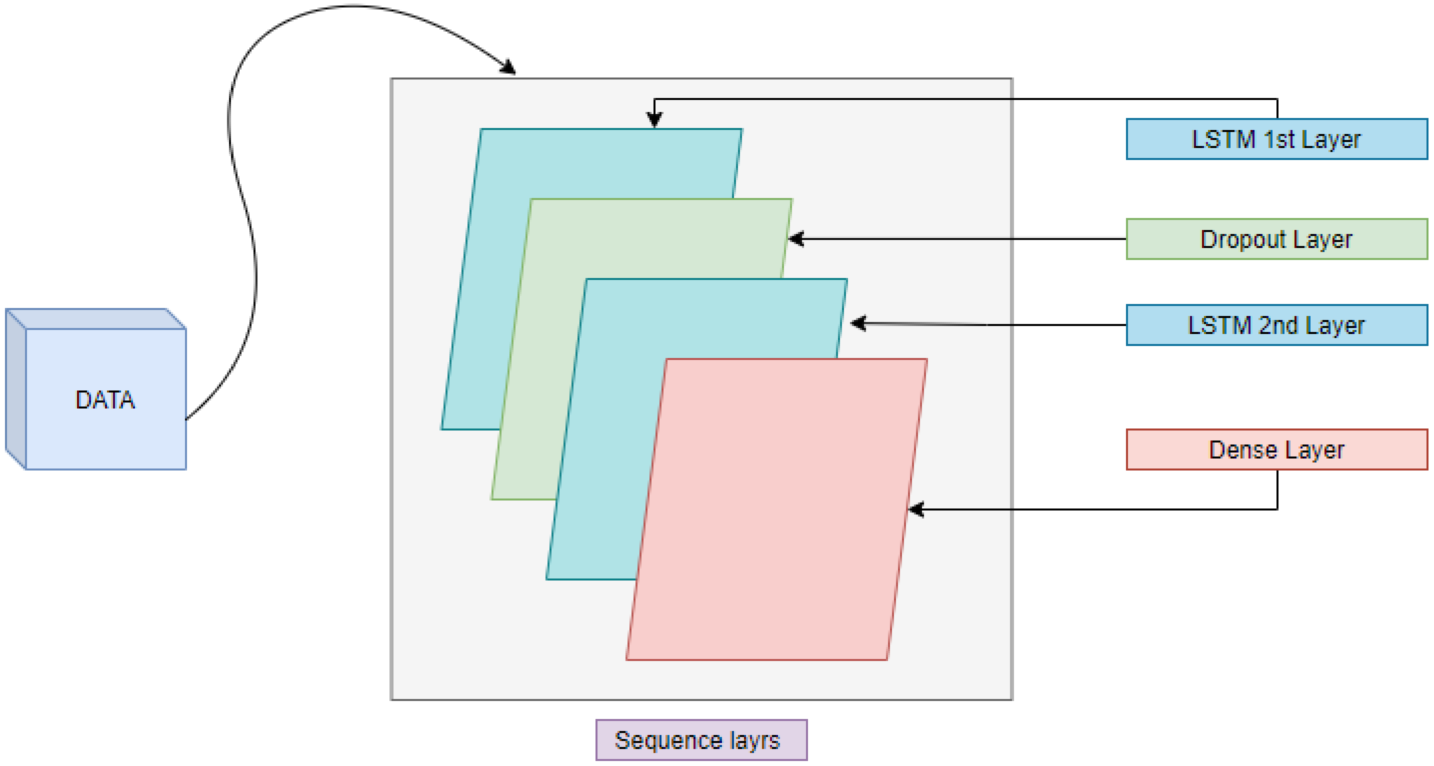

Figure 8

Proposed model architecture

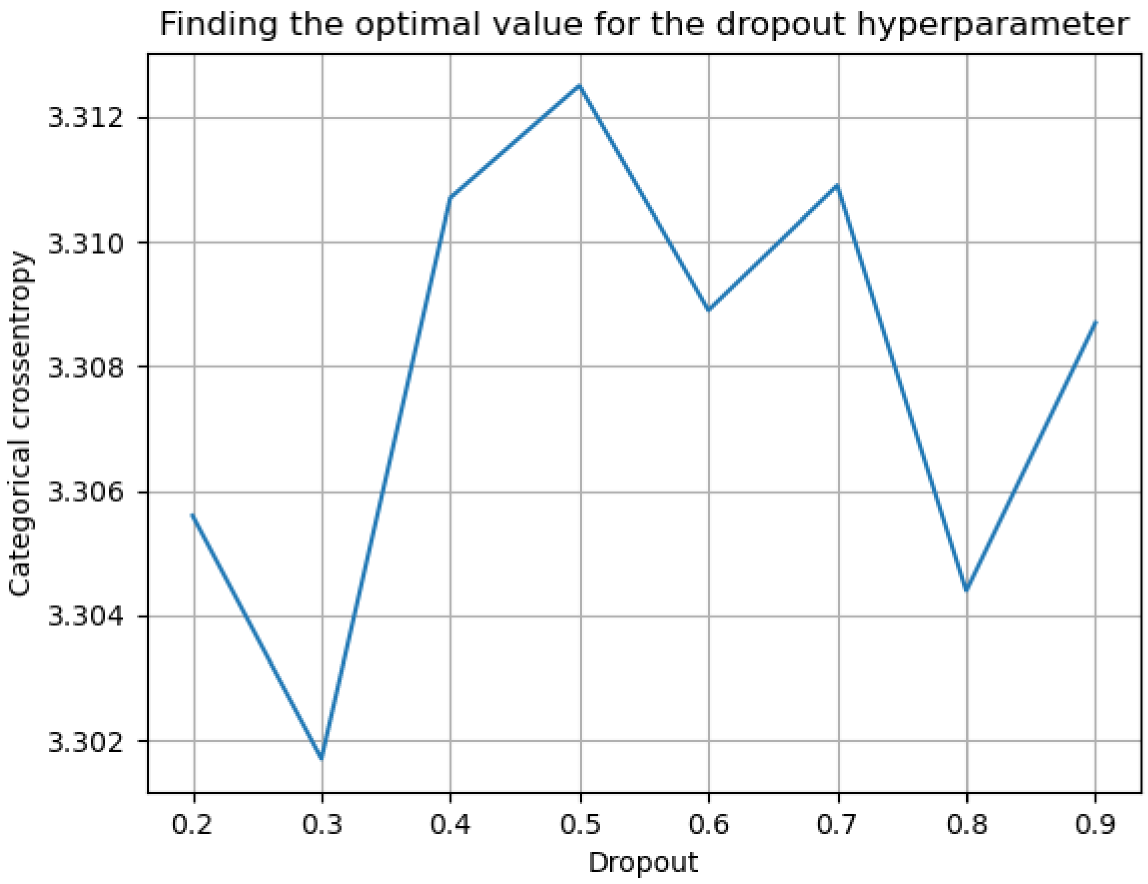

Figure 9

Evaluation of the dropout hyperparameter

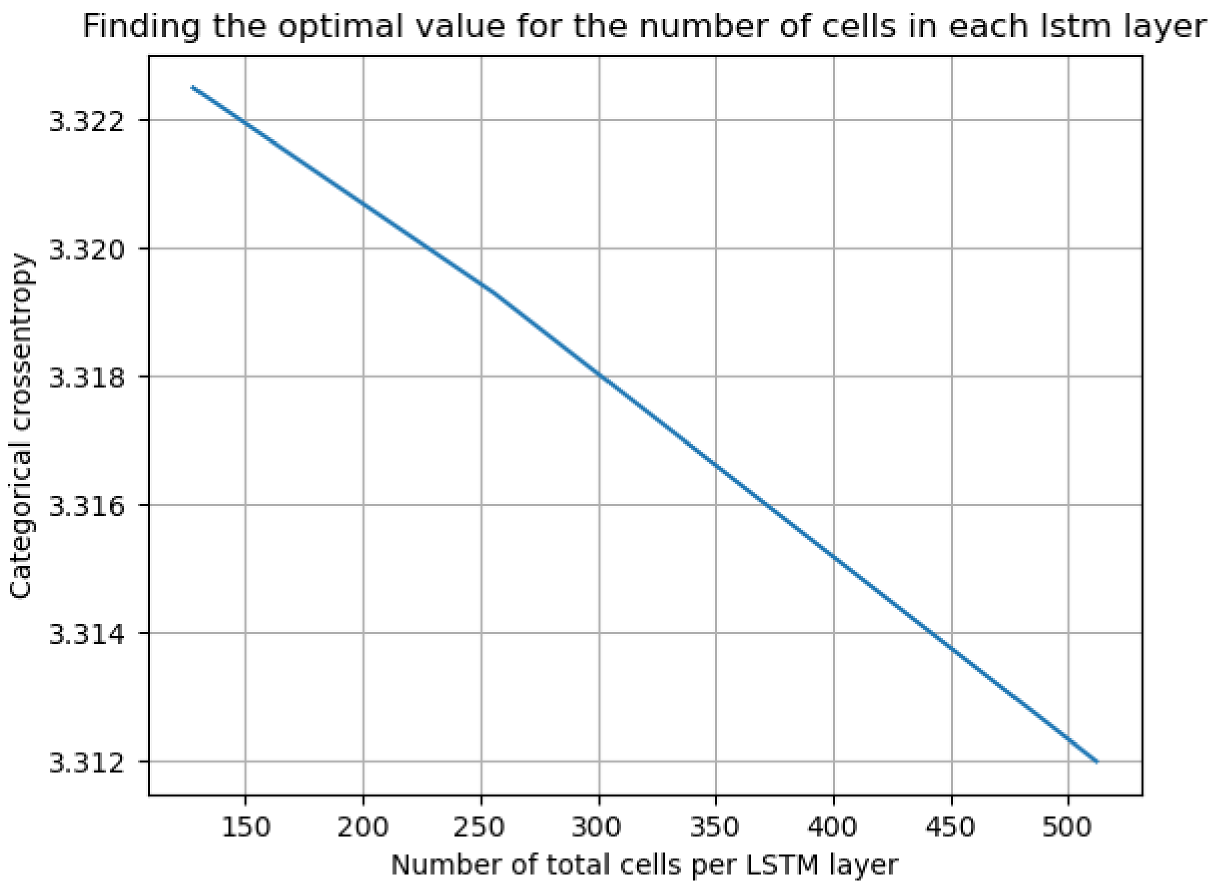

Figure 10

Evaluation of the number of cells in each LSTM layer hyperparameter

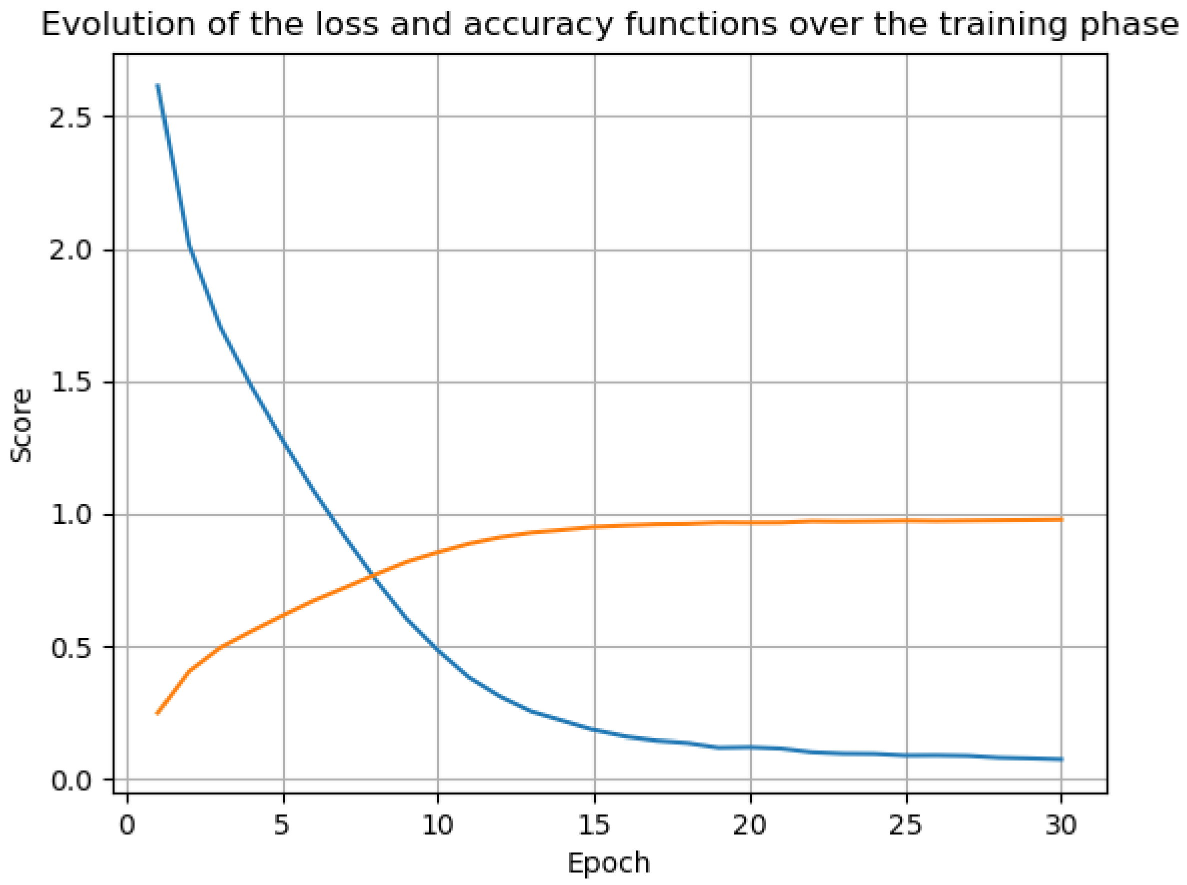

Figure 11

Evolution of the loss and accuracy metrics over epochs

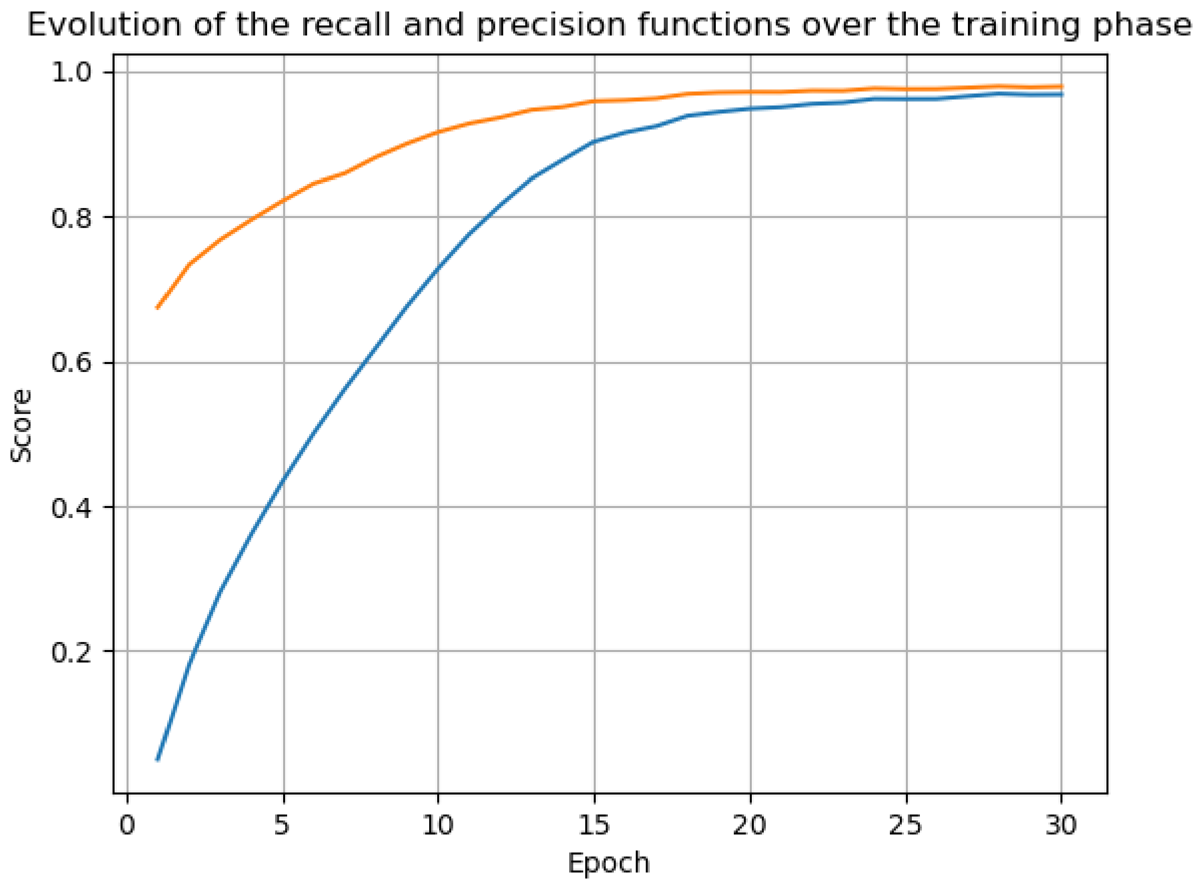

Figure 12

Evolution of the recall and precision metrics over epochs

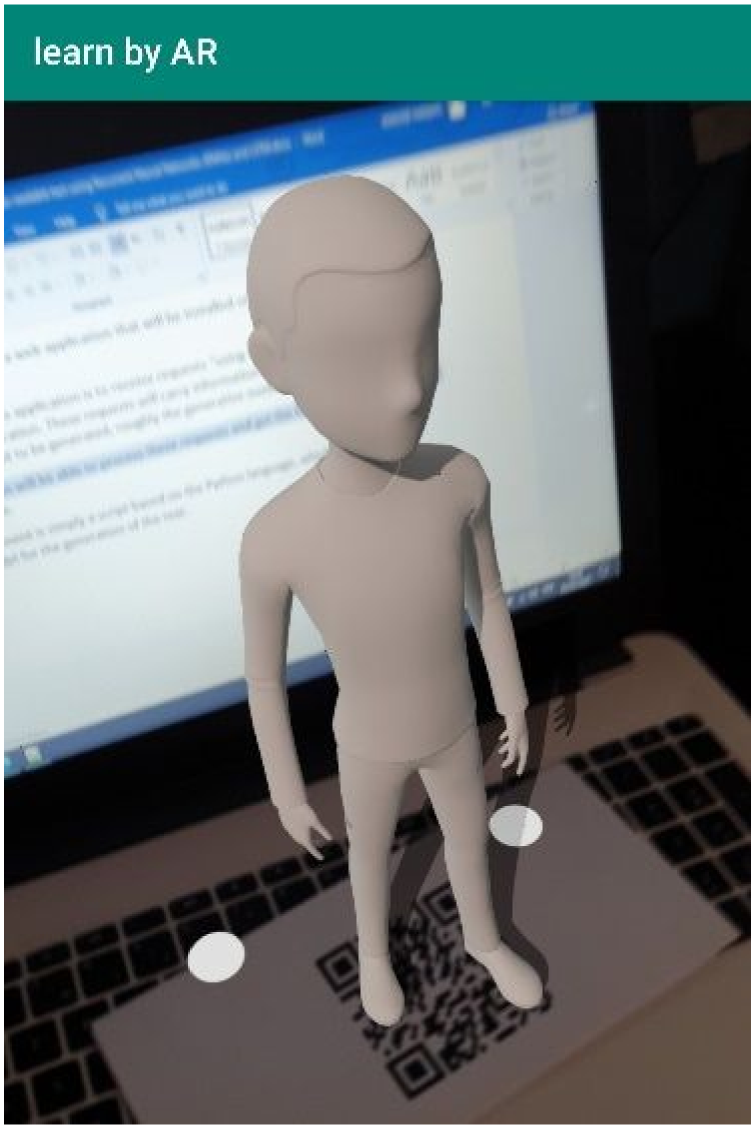

Figure 13

Screen shot of the developed prototype