Predictive Natural Language Processing Analysis Applied to Arabic

Karim El mokhtari[1], Monir Azmani[2], Abdelali Astito[2]

[1] Data Science Laboratory, Ryerson University, Toronto, Canada
[2] Laboratoire d'Informatique, Système et Télécommunications,
Université Abdelmalek Essaadi, Tangier, Morocco
elmkarim@ryerson.ca

Abstract. This paper aims at applying Recurrent Neural Networks (RNN) to Natural Language Processing using old texts such as sacred books. The objective of this work is to generate a text that follows the same structure and semantic of the studied corpus written in Arabic. This is done through various networks such as vanilla RNN, LSTM (Long Short Term Memory) and GRU (Gated Recurrent Unit). After training, the quality of the generated text is measured using BLEU scores. Besides, an exploratory analysis of the corpus is presented to exhibit some interesting findings at the words and chapters level.
Keywords: RNN, LSTM, GRU, NLP.

1 Introduction

Recurrent Neural Networks (RNN) networks can learn from sequences and predict sequences as well. They have a wide range of applications such as machine translation, caption generation and predicting from time series.

This paper aims particularly at applying RNNs in NLP (Natural Language Processing). The objective is to create a model that learns from old texts such as sacred books and generate a sequence of words that exhibits the same structure and semantic as the corpus. We are interested in the holy book of Quran and consider this work as a first paper to apply data science tools to learn and classify information from old texts.

We want to measure to what extent a machine can reproduce a text similar to an original known version. This measure includes the nature and frequency of words as well as their similarity to the context. Once the machine learns from the text, we want to show if it is possible to generate a document highly similar to the original. To achieve this objective, we are interested in many recurrent neural network

architectures, the vanilla RNN, the LSTM (Long Short Term Memory) and the GRU (Gated Recurrent Unit).

This paper is organized as follows. We first describe the corpus as well as summary statistics. Then, we expose the exploratory analysis that focuses on chapter length and word occurrence. We explain the prediction technique used in the following section; then we discuss the experimental results. Finally, we summarize the findings and discuss the prospects.

2 Datasets and Inputs

The dataset used here is the sacred book of the Quran. It can be downloaded in multiple text versions found in [1]. The Quran is an old text written in Arabic and is considered a holy book in Islamic culture [2]. The Arabic language is composed of 28 letters [3] that can be extended to 39 if we consider variants of some letters (like the letter "*alif*" (ا) and the variant "*alif*" with "*hamza*": (أ)). We are using a basic text version without any ornaments or chapter/verses numbers, so the following options are to be considered in the download page [1]:

1. Quran text: Simple Clean (do not include pause marks, do note include sajdah signs, do not include rub-el-Hizb signs)

2. File format: Text

The Quran corpus contains 114 chapters (or *Surahs*) of different lengths [4]. Each chapter consists of many verses. In the text file, verses are put in separate lines. Besides, chapters are classified as *Meccan* or *Medinan*. This classification refers to the two holy cities in the Islamic tradition. Below are summary statistics on this corpus:

- Number of chapters (*Surahs*): 114
  - Meccan chapters: 82
  - Medinan chapters: 20
- Unclassified: 12
- Number of verses (lines): 6,236
- Number of words: 77,437 words
- Number of characters: 323,671 character

3 Exploratory analysis

Before applying our machine learning model to the corpus, it is necessary to go through an exploratory analysis of the text to understand and show relevant characteristics of its nature.

Firstly, we compute the word count per chapter. The bar plot in Fig. 1 shows the word count by chapter.

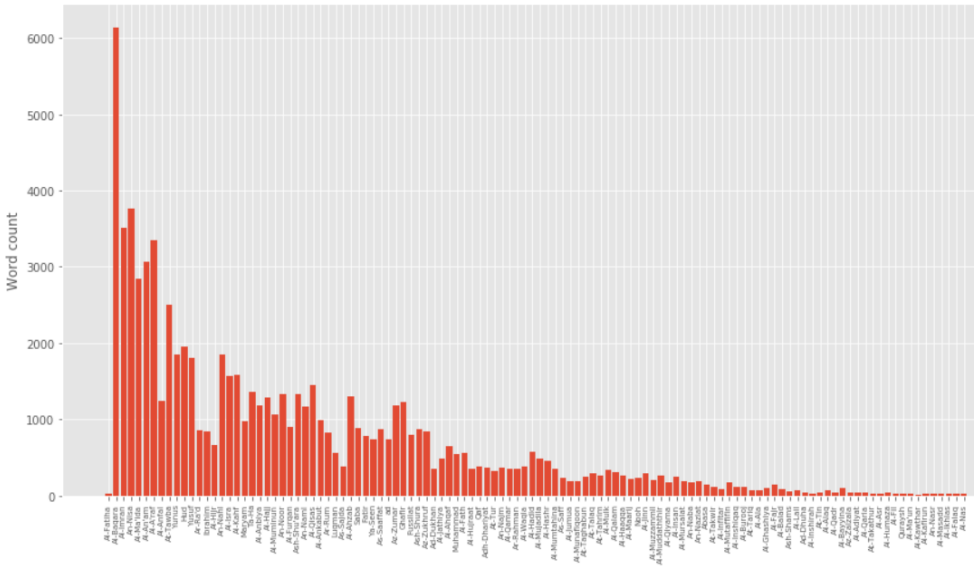

Fig. 1. Word count by chapter. A decreasing trend is observed.

We can easily observe a decreasing trend of the word count from the beginning to the end of the corpus. This may suggest that the chapters were arranged according to their length in the Quran. The last thirty chapters are all below 100 words. The second chapter named "Al-Baqara" is by far the lengthiest with 6144 words, followed by the fourth chapter "An-Nissa" with 3767 words. The shortest chapter of the corpus is "Al-Kawthar" that contains only 14 words.

Fig. 2 shows a tree map of the top 60 lengthiest chapters of the Quran. The area of the rectangle associated with each chapter is proportional to its word count.

Fig. 2. Tree map of the top 60 lengthy chapters

A second analysis shows the words frequency over the whole corpus. A customized list of stop-words was created to remove the frequent words that do not associate with any context.

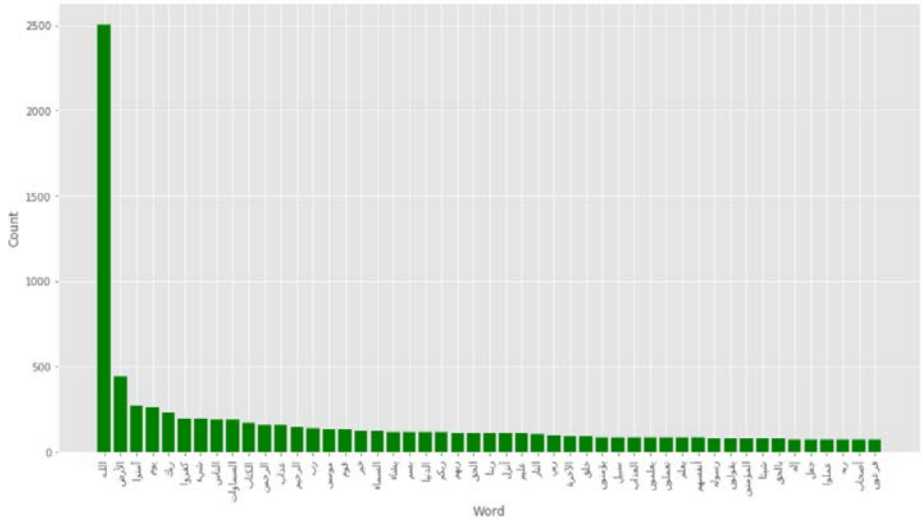

Fig. 3. Word frequency over the corpus

It is not surprising to notice in Fig. 3 that the most frequent word in the Quran is the word "الله" (Allah) with 2505 occurrences – it may well be considered as a stop word –. The second frequent word "الأرض" (earth) is far below the word "Allah" count with 444 occurrences, followed by the words "آمنوا" (those who believe) with 270 occurrences, "يوم" (day) with 261 occurrences, "ربك" (your Lord) with 231 occurrences and "كفروا" (those who do not believe) with 191 occurrences.

These frequencies seem logical when we know that the topics discussed in the Quran deal mainly with faith as well as believers and non-believers behaviour and traits according to the Islamic religion. The word "Earth" is also related to the topic of belief since the message of this whole book is destined to people living on Earth.

We illustrate in Fig. 4 the repartition of the top 100 frequent words in the Quran.

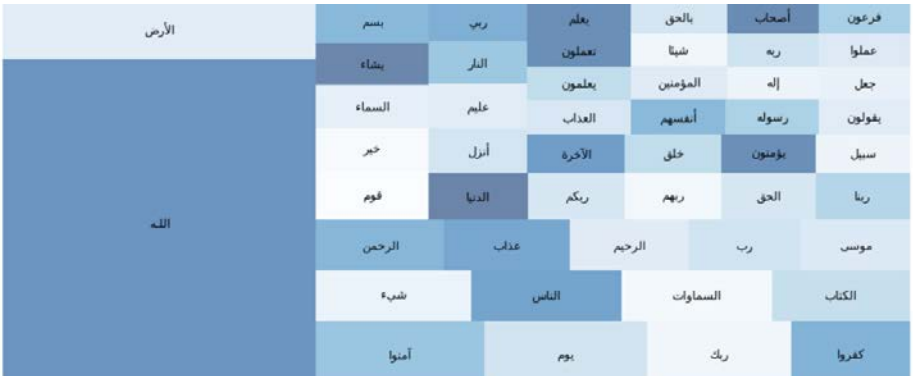

Fig. 4. Tree map of words frequency over the corpus

This figure shows that many words and their opposite occur with frequencies of the same order of magnitude such as:

Word	Meaning	Frequency
آمنوا	those who believe	270
الدنيا	world	115
خير	good/well-being	119

Opposite Word	Meaning	Frequency
كفروا	those who do not believe	191
الآخرة	hereafter	90
العذاب	torment	86

The two most frequent proper names in the Quran are "موسى" (Moses) with 134 occurrences, followed by "فرعون" (Pharaoh) with 72 occurrences. This suggests that the story of Moses associated with the character of Pharaoh is the most popular among the stories narrated in the Quran.

Another interesting visualization of word frequencies is the word cloud. This illustration, shown in 5, displays graphically the occurrence of the top 200 words by drawing each word with a size proportional to its frequency.

This figure pictures the main topics that the reader may find in the Quran. As more space is available in this kind of visualization, we can read new words still dealing with the topic of faith such as: "الشيطان" (devil/Satan),"الملائكة" (angels),"السماء" (sky/heaven),"هدى" (guidance) and "قلوبهم" (their heart). Frequent bi-grams are also depicted such as "يوم القيامة" (the day of judgement),"الحياة الدنيا" (worldly life),"الرحمن الرحيم" (the merciful and gracious) and "السماوات والأرض" (the heavens and the earth).

Finally, we analyzed the entire set of words in the Quran using Word2Vec [9]. This algorithm creates a set of word embeddings by sliding a five-words-sized window over the whole text and training a neural network over the words of each window. During training, the neural network calculates an embedding vector that considers the

co-occurrence of words in the five-words windows. The resulting embedding shows interesting properties. First, it allows switching from a sparse vector word representation to a dense representation using lower dimension real number vectors. Secondly, the vectors can be used to calculate distances between words. Those distances show quite interesting relationships.

Below we describe two examples of the results obtained with the Word2Vec network trained on the Quran corpus.

1. Predicting words that are similar to a specific word. The calculation is based on the cosine distance between word-vectors.
   For example, the closet words to the word "خير" (good/well-being) are:

Word	Meaning	Cosine similarity
كبير	big/enormous	0.324
يبين	show you	0.295
دار	home	0.272
وجه	face	0.263
كاذبين	those who lie (antonym)	0.258
مؤمن	believer	0.258
جهة	side	0.249
متاع	goods	0.248
ماء	water	0.248
اخرى	other	0.236
بشرا	human being	0.228
عدل	Justice/fairness	0.226

   These words usually come in contexts where the word "خير" (good/well-being) exists as well. The similarity score is higher for words that co-occur more often with the word "خير", such as "كبير" (enormous) describing the nature of well-being in the context of Heaven or reward for good deeds.

2. Calculating distances between words and using them in predicting new words. For example, Word2Vec allows calculating the distance between two word-vectors $\mathbf{w_1}$ and $\mathbf{w_2}$. The resulting distance can be added to another word-vector $\mathbf{w_3}$ to predict a new vector $\mathbf{w_4}$. As the distance between pairs ($\mathbf{w_1}$,$\mathbf{w_2}$) and ($\mathbf{w_3}$,$\mathbf{w_4}$) is equal, we can expect the relationship between $\mathbf{w_1}$ and $\mathbf{w_2}$ to be similar to the one between $\mathbf{w_3}$ and $\mathbf{w_4}$.
   For instance, let's consider the pair "موسى" (Moses) and "هارون" (Aaron). As they are brothers according to the Quranic context, the distance between them is probably associated to the brotherhood relationship. Therefore, when we add this distance to another name such as "اسماعيل" (Ismail), Word2Vec returns "اسحاق" (Isaac) who is Ismail's brother according to the Quranic context.

Those distances can also be shown graphically; however, as embedding vectors are multidimensional, if we want to plot words in a two-dimension space, we need to

reduce their dimensionality. We can apply methods such as PCA [10] or t-SNE [11] to capture the essential components in the embedding vector and reduce down the representation to 2D.

We apply t-SNE to all embeddings and show the 20 closest words to the word "فرعون" (Pharaoh) in Fig. 6.

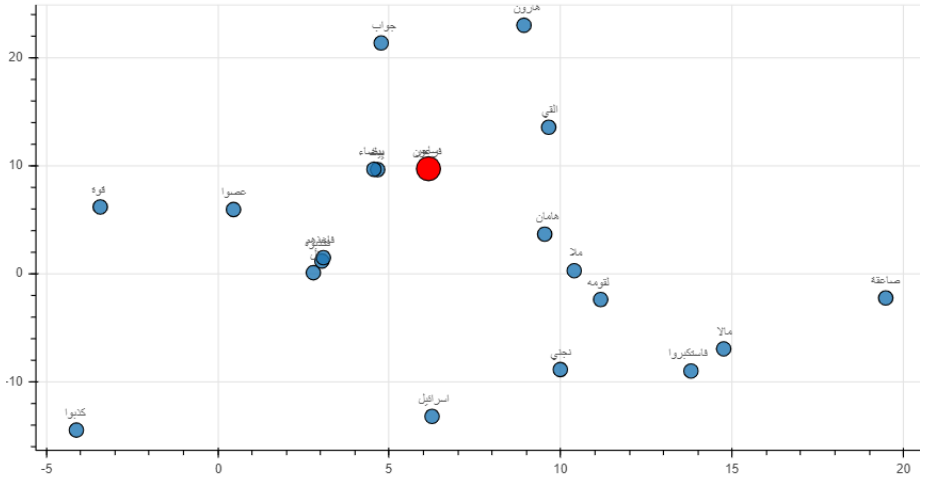

Fig. 5. The twenty closest words to the word "فرعون" (Pharaoh). Word embeddings are reduced to 2D using t-SNE. Numbers on x and y axis do not hold any specific meaning. The distance between dots indicates similarity.

4 Theoretical concepts of Recurrent Neural Networks (RNN)

RNNs are a special type of neural network where the output is calculated from the inputs and from the previous state of the network. They were applied successfully to Natural Language Processing in many applications such as language translation [12, 13], image captioning [14] and chat bots [15] .

In the vanilla RNN depicted in Fig. 7, we maintain a hidden state representing the features in the previous time sequence. Hence, to make a word prediction at time step t, we take both input X_t and the hidden state from the previous time step h_{t-1} to compute h_t. So, generally, an RNN makes a prediction based on the hidden state in the previous time step and current input by applying the following rule: $h_t = f(X_t, h_{t-1})$.

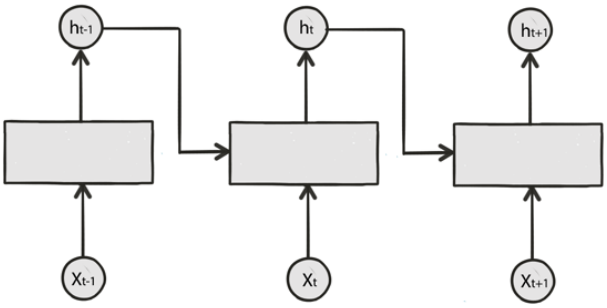

Fig. 6. RNN structure

The main disadvantage of this structure is its reliance only on the hidden state of the last time step. In many practical cases, such as language translation, the prediction at time step t requires data from older time steps. The RNN has no mechanism to store old data. This weak dependency with the past is due to the "vanishing gradient" issue. Indeed, when unrolled, an RNN can be considered as a deep neural network. Backpropagation using gradient descent gives more weight to recent layers (or time steps) than farther ones (older time steps).

To mitigate this problem and reinforce the dependency with old inputs/states, the LSTM or Long Short Term Memory [5], depicted in Fig. 8 defines two different memory cells noted *C* and *h*.

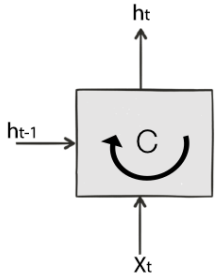

Fig. 7. LSTM general structure

The internal cell *C* retains the relevant data that the LSTM learns to memorize from all past time steps. The state *h* is only a short version of *C* and is used as the current output. To learn what to store and erase from the cell *C*, the LSTM uses three gates called forget, input and output. The role of each gate is described as follows:

- Forget gate G_{forget}: controls what part of the previous cell state is kept.
- Input gate G_{input}: controls what part of the new computed information is added to the cell state *C*.
- Out gate: controls what part of the cell state is exposed as hidden state.

It is demonstrated that this process allows diminishing the impact of the vanishing gradient as the gates operate as on/off switches for choosing specific data to memorize or discard from the cell *C* [5].

The three gates are neural networks using sigmoid activation functions $\sigma(.)$:

$$G_{forget} = \sigma(W_{fx}X_t + W_{fh}h_{t-1} + b_f) \quad (1)$$

$$G_{input} = \sigma(W_{ix}X_t + W_{ih}h_{t-1} + b_i) \quad (2)$$

$$G_{\text{out}} = \sigma(\mathrm{W_{ox}X_t + W_{oh}h_{t-1} + b_o}) \quad (3)$$

For each gate, *W* is the matrix of conventional learned weights and *b* the bias parameter. The following equations define how the new values of *C* and *h* are calculated using the three gates:

$$\tilde{C} = \tanh(W_{cx}X_t + W_{ch}h_{t-1} + b_c) \quad (4)$$

$$C_t = G_{forget} \,.\, C_{t-1} + G_{input} \,.\, \tilde{C} \quad (5)$$

$$h_t = G_{out} \,.\, \tanh(C_t) \quad (6)$$

The GRU (Gated Recurrent Units) [6] is a simplified version of the LSTM. It inherits some concepts from both LSTM and vanilla RNN. A GRU does not maintain a cell state *C* and uses two gates instead of three. The equations of the two gates are as follows:

$$G_r = \sigma(W_{rx}X_t + W_{rh}h_{t-1} + b) \quad (7)$$

$$G_{update} = \sigma(W_{ux}X_t + W_{uh}h_{t-1} + b) \quad (8)$$

A proposed new state $\tilde{h}_t$ is calculated using gate G_r:

$$\tilde{h}_t = \tanh(W_{hx}X_t + W_{hh}.(G_r \,.\, h_{t-1}) + b) \quad (9)$$

The new hidden state is computed by:

$$h_t = (1 - G_u).h_{t-1} + G_{update} \,.\, \tilde{h}_t \quad (10)$$

G_u defines what data from the old state h_{t-1} to be kept and what new data from $\tilde{h}_t$ to be introduced in h_t.

5 Learning model architecture

The concept we want to implement in this work is to learn the structure and semantics of the Quran using various architectures of the RNN, namely vanilla RNN, LSTM and GRU. Then from any subset of the corpus, predict a new sequence of words with the same learned structure and semantics.

Fig. 9 depicts the unrolled learning architecture using the RNN network. We denote

the words of the corpus by w_i where i is the index of the word in the corpus. This index ranges from 0 (first word) to N-1 (last word). N is the number of words in the full corpus.

During the initial step of every epoch, the first group of 100 words (w_0 ... w_{99}) is fed into the RNN that predicts the following word ($\hat{w}_{100}$). The RNN updates its hidden state h_0 that is used in the following step. This hidden state is supposed to summarize all important data from the past and allows the RNN to predict the following word ($\hat{w}_{101}$) by using this hidden state h_0 along with the next group of 100 words (w_1... w_{100}).

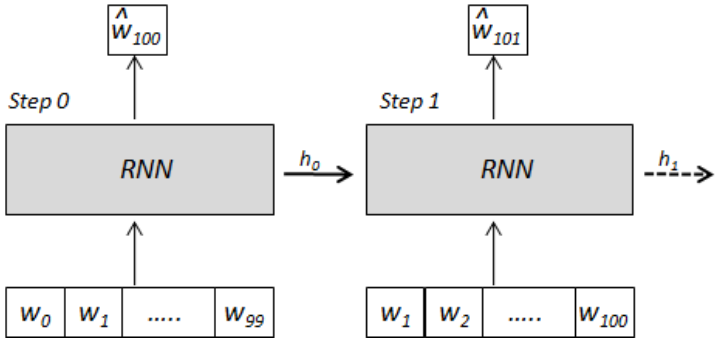

Fig. 8. RNN learning model architecture

The training in one epoch continues up to the last group of 100 words: (w_{N-100} ... w_{N-1}).

Sampling the RNN consists in providing a random sequence of 100 words from the original text in the input, let's denote it (w_0 ... w_{99}) and let the RNN predict the following word $\hat{w}_{100}$. In the next step, we use as input 99 words from the original group (w_1 ... w_{99}) and add the predicted word as the 100th in the input vector. Fig. 10 illustrates this process.

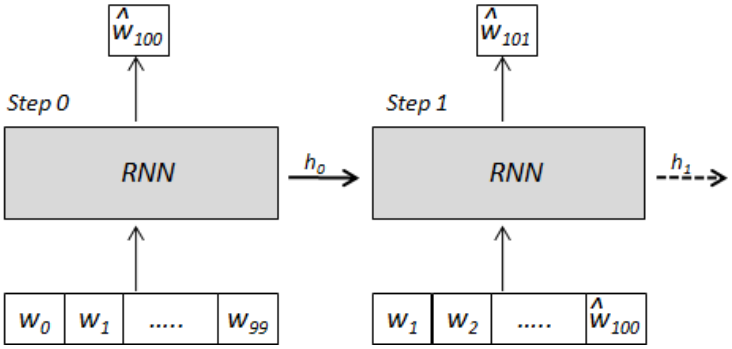

Fig. 9. Sampling the RNN

We continue sampling the RNN by adding at each time step the predicted word from

the last step to the input vector. After 100 time steps, all the words in the input vector are coming from the RNN predictions during the last 100 time steps. In this case, we can test to which extent the RNN is able to predict the next word correctly even if no original text is provided at its input.

Benchmark Model

We compare the performance of both LSTM [5] and GRU [6] in learning and predicting from the original text using the BLEU metric explained below. The performance of these two models are compared to a basic RNN [7] which is a simple recurrent network that consists only of a hidden state and does not use any additional gates for long-term dependency.

Evaluation Metrics

The metric that is used to measure the similarity between the predicted text and the original is the BLEU score [8] applied on 1, 2 and 3-grams. BLEU score is widely used to assess the quality of machine translation models. It evaluates the quality of a text translated from a natural language to another by comparing the machine's output with that of a human. BLEU's output is a number between 0 and 1 that indicates the similarity between the reference and generated text, with values closer to one representing more similar texts. Clarity of grammatical correctness is not taken into account.

From a sequence of 100 words from the original text, we produce a sequence of 10 words using the model. Each predicted word is added to the input sequence for the next time step as explained above. We compare the sequence of 10 words with the correct sequence from the original text by computing BLEU-1, BLEU-2 and BLEU-3 scores. BLEU-1 score considers only the precision at the unigram level, while BLEU-3 considers tri-grams.

Design

Words in the original text are tokenized. No Lemmatizing or stemming is applied. The corpus is divided into groups of 101 words. The RNN is trained on 100 words; then it is given the 101st word to predict in a supervised learning scheme. The dataset is generated using a sliding window of 101 words starting from the beginning of the text to the end.

The actual number of words in the dataset is 78,245. The number of patterns of 100 consecutive words is 78,145 patterns. The training and test sets are respectively 80% and 20% of the total dataset. So 62,516 data points are used for training and 15,629 are considered for testing. Validation sets are chosen randomly at each epoch during training and will consist of 10% of the training data.

Implementation

The implementation of the model was done with Python 3.6 and Keras library. The

training was performed on a cluster of four NVIDIA P100 GPUs with 16GB of memory.

To fine-tune the model, the following parameters were used:

- Size of the hidden layer of the RNN: we experimented with 500, 1000 and 2000 units.
- A dropout layer associated with the RNN: we experimented with different values ranging from 0.1 to 0.5.
- Batch size: ranging from 32 to 128

A softmax layer is added to the output of the RNN to calculate the probability of each word. We used the "categorical cross-entropy" loss function and the "Adam" optimizer [16]. This optimizer is used instead of the classical stochastic gradient descent to update network weights iteratively based on training data.

Data is split into vectors of 100-word tokens denoted $\boldsymbol{X_i}$ that are used as inputs to the RNN; the model is trained to predict the following word token denoted $\boldsymbol{y_i}$. The different vectors of 100 words are created by shifting a window of size 100 over the entire corpus as shown in Fig. 11.

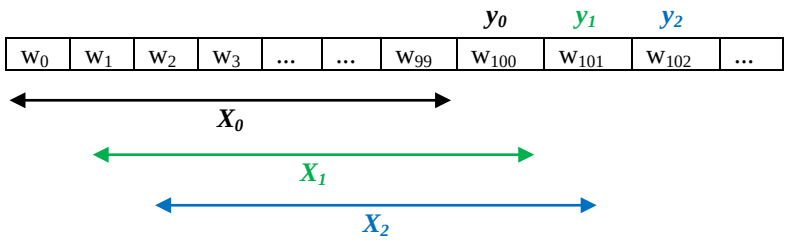

Fig. 10. Splitting the corpus into input vector X_i and target values y_i

The dataset is then split into training set and test set with a 80/20 proportion.

The training was made for all three networks vanilla RNN, LSTM and GRU over the training set. A validation set of 10% of the training data is used during training.

6 Experimental results

After experimenting with different combinations of models parameters, the following parameters were chosen as they performed the best among the tested combinations:

- Batch size: 128
- Hidden layer size for the RNN: 1000. Higher values increase training time and complexity with no result improvement
- Dropout: 0.2. No significant improvement was detected by changing dropout.

To evaluate the performance of each one of the three networks (vanilla RNN, LSTM and GRU), two assessments were performed:

- The first one is done by measuring the accuracy of word prediction. In this case, for each input vector $\mathbf{X_i}$, we compare the real output token that stands for the next word in the corpus $\mathbf{y_i}$ with the one predicted by the model denoted $\hat{y}_i$. The accuracy is the number of the correctly predicted words over the number of predicted words

- The second assessment uses BLEU scores. In this case, for each input vector $\boldsymbol{X_i}$ from the corpus, we predict from the model a sequence of 10 words then we compare it with the correct sequence of 10 words that follows vector $\boldsymbol{X_i}$ in the corpus. BLEU scores calculate the similarity at different grams levels. In our case, we considered 1, 2 and 3-grams similarity.

Assessment 1: Accuracy of prediction

We calculate the accuracy on both training and test sets. The results are shown below.

Table 1. Accuracy on training and test sets

	Accuracy on training set	**Accuracy on test set**
Vanilla RNN	2.6%	1.8%
LSTM	89.3%	8.5%
GRU	77.4%	6.4%

We note that the vanilla RNN performed poorly by predicting only 2.6% from the training set and 1.8% from the test set. The LSTM did well compared to the GRU by predicting correctly 89.3% of the words in the training set and 8.5% in the test set.

These results show that the four-gates structure of the LSTM allows a better learning and a higher dependency with previous words which is very important in learning the context of each sentence and predicting a word that is associated to this context. The GRU also performs well because of its two-gate architecture that offers a better dependency with past words when compared with the vanilla RNN that only considers the last hidden state to predict the next word.

We note that the overall performance of the three models is poor over the test set which is a set of sentences that were not included in training. This may suggest that the current configuration was not able to learn enough from the corpus to make more accurate predictions. Another hypothesis is that the complexity of the corpus is high and the three models were not able to generalize over the whole Quran corpus.

Assessment 2: Evaluation of BLEU scores

The BLEU scores calculated over unigrams, bigrams and trigrams for the three models are shown on the following tables.

Table 2. BLEU score for 1000 samples from the training set

	BLEU-1	BLEU-2	BLEU-3
Vanilla RNN	0.030	0.173	0.349
LSTM	0.620	0.507	0.461
GRU	0.533	0.395	0.342

Table 3. BLEU score for the test set

	BLEU-1	BLEU-2	BLEU-3
Vanilla RNN	0.026	0.162	0.336
LSTM	0.435	0.333	0.304
GRU	0.368	0.241	0.206

We note the same observations as for the prediction accuracy assessment. The vanilla RNN performed poorly while the LSTM showed the best results. Besides, the performance of all models is lower on the test set than on the training set which is normal.

Nevertheless, using the test set, we note that the performance of the LSTM model is better in terms of the BLEU scores than in terms of prediction accuracy. This can be explained as follows using an example.

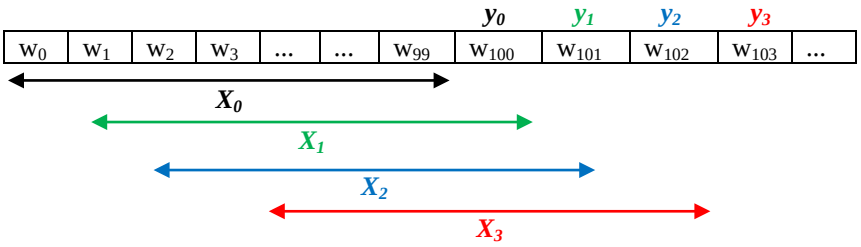

Fig. 11. Explaining why LSTM performs better in terms of BLEU score than accuracy score

In Fig. 12, let's suppose that we picked randomly for our training set tuples (X_0, y_0), (X_2,y_2) and (X_3,y_3) and we picked the tuple (X_1, y_1) for our test set. In reality, those four tuples share more than 95% of the words; only one word changes between a tuple and the next one. Therefore, during training, the LSTM learns the long term dependency between words by training on (X_0, y_0), (X_2,y_2) and (X_3,y_3). And even though it may incorrectly predict the word for the test tuple (X_1, y_1), if we go on predicting the word following y_1, the LSTM may predict it correctly as it is already trained on (X_2,y_2). Thus, the LSTM can catch up and continue predicting the rest of the sequence correctly even if it fails to predict the word belonging to the test set. Such long term dependency is lower with the GPU and much lower with the vanilla RNN structure that's why the LSTM outperforms both of them.

We can find below an example of a generated sequence predicted by the LSTM from

the test set and compared with the same sequence in the corpus:

Correct sequence from the corpus

من	لا	يؤمن	بها	واتبع	هواه	فتردى	وما	تلك	بيمينك

Sequence generated by the LSTM

لا	لا	يؤمن	بها	واتبع	هواه	فتردى	وما	تلك	بيمينك

As explained above, the sequence is generated by the LSTM by applying an input vector from the test set. Therefore, the first predicted word has a higher probability of being incorrect (coloured in red). When we carry on predicting, the following words are correct, and this is explained by the long term dependency exhibited by the LSTM.

7 Conclusion

This work aims to learn from sequences of words of an Arabic corpus that is the sacred book of Quran and create models to predict a text that is similar to it. The main idea is to divide the corpus into groups of 100 words and train a recurrent neural network to predict the next word that follows each group. This work covered three different networks namely the vanilla RNN, the LSTM and the GRU.

The evaluation metrics are prediction accuracy and BLEU scores. The results showed that the vanilla RNN performed poorly on training and test sets. The LSTM and GRU performed better as they use a gating mechanism to memorize the long term dependency with past words. The LSTM performed the best even though the prediction accuracy with the test set was low. This may be explained by the semantic complexity of the corpus that the LSTM and GRU were not able to learn.

The three networks were also assessed in terms of BLEU score. In this context, each network predicted a sequence of 10 words from a vector of 100 words; the generated vector was compared with the correct sequence of 10 words existing in the corpus. In this second assessment, the vanilla RNN did poorly as well while the LSTM and GRU showed good performance, the LSTM still performing better. Indeed the similarity between the 10-words sequence generated by the LSTM was 43% at unigram level compared the original text and 30.4% at trigram levels. This observation shows that even if the LSTM makes a wrong prediction of the first word in the sequence, its long term dependency with past words allows better predictions of subsequent words.

An interesting prospect of this work is to use the GAN (Generative Adversarial Network) structure to generate sentences from the same corpus by using another concept based on coupling a generator with a discriminator.

8 References

1. http://tanzil.net/docs/download

2. https://en.wikipedia.org/wiki/Quran
3. https://en.wikipedia.org/wiki/Arabic_alphabet
4. https://en.wikipedia.org/wiki/List_of_surahs_in_the_Quran
5. S. Hochreiter and J. Schmidhuber: Long short-term memory. Neural computation, 9(8):17351780, (1997)
6. K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259, 2014.
7. Lipton, Zachary C., Berkowitz, John, and Elkan, Charles. A critical review of recurrent neural networks for sequence learning. arXiv:1506.00019, 2015.
8. K. Papineni, S. Roukos, T. Ward, and W. J. Zhu: BLEU: A method for automatic evaluation of machine translation. In ACL (2002)
9. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality, NIPS 2013.
10. K. Pearson. On lines and planes of closest fit to systems of points in space. Philiosophical Magazine, 2:559–572, 1901.
11. L.J.P. van der Maaten and G.E. Hinton. Visualizing data using t-SNE. Journal of Machine Learning Research, 9(Nov):2431–2456, 2008.
12. 1. Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc VV: Sequence to sequence learning with neural networks. In NIPS, pp. 31043112 (2014).
13. Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua: Neural machine translation by jointly learning to align and translate. arXiv:1409.0473 (2014)
14. A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth: Every picture tells a story: Generating sentences from images. In ECCV (2010)
15. M. T. Mutiwokuziva, M. W. Chanda, P. Kadebu, A. Mukwazvure and T. T. Gotora, "A neural-network based chat bot," 2017 2nd International Conference on Communication and Electronics Systems (ICCES), Coimbatore, 2017, pp. 212-217.
16. Kingma, Diederik P. and Ba, Jimmy. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG], December 2014.