

Machine Learning Engineer Nanodegree

Capstone project

Karim El mokhtari

March 18th, 2019

Project overview

Recurrent Neural Networks (RNN) networks are able to 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 project aims particularly at applying RNNs in NLP (Natural Language Processing). The objective is to create a model that can learn from old texts such as sacred books, and generate text that follows the same structure and semantic. I am interested in the holy book of Quran since I am originating from an Arabic culture and would like to use this work as a first step to apply machine learning to learn and classify information from old texts.

In this work, many networks are tested: SimpleRNN (or vanilla RNN), LSTM (Long Short Term Memory) and GRU (Gated Recurrent Unit).

A general overview of RNNs can be found [here](#)¹.

Problem Statement

The objective of this project is to measure to what extent a machine can reproduce a text similar to an original known text. This measure includes the nature of words, their frequency and the similarity of the words to the context. Once the machine learns from the text, it is possible to generate a text that is similar to the original. The expected architectures to be used are RNNs.

Datasets and Inputs

The dataset is the sacred book of Quran. It can be downloaded in many text versions found in [this link](#)². The Quran is an old text written in Arabic and is a holy book in the Islamic culture³. The Arabic language is composed of 28 letters⁴ that can be

¹ <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>

² <http://tanzil.net/docs/download>

³ <https://en.wikipedia.org/wiki/Quran>

⁴ https://en.wikipedia.org/wiki/Arabic_alphabet

extended to 39 if we consider variants of some letters (like the letter “*alif*” (ا) and its variant “*alif*” with “*hamza*”: (أ)).

The Quran corpus is public and can be downloaded at this URL:

<http://tanzil.net/docs/download>

I am 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. Quran text: Simple Clean (do not include pause marks, do not include sajdah signs, do not include rub-el-hizb signs)
2. File format: Text

The Quran corpus is organized in 114 chapters (or *Surahs*) of different lengths⁵. Each chapter consists of many verses. In the text file, verses are put in different lines. In addition, chapters are classified as *Meccan* or *Medinan*. This 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

The text file is available in the github account related to this project. The encoding used is UTF8.

Github account: https://github.com/elmkarim/arabic_text_rnn

Benchmark Model

I compare the performance of both LSTM⁶ and GRU⁷ 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 RNN⁸ which is a simple recurrent network that consists only of a hidden state and does not use any additional gates for long-term dependency.

⁵ https://en.wikipedia.org/wiki/List_of_surahs_in_the_Quran

⁶ S. Hochreiter and J. Schmidhuber: Long short-term memory. Neural computation, 9(8):1735-1780, (1997)

⁷ Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio: Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation. Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724-1734 (2014)

⁸ <https://arxiv.org/pdf/1506.00019.pdf>

Evaluation Metrics

The metric that is used to measure the similarity between the predicted text and the original is the BLEU score⁹ 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 text which has been 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 are not taken into account.

From a sequence of 100 words from the original text, we will produce a sequence of 10 words using the model. Each predicted word is added to the input sequence for the next timestep as further explained in the section "Algorithms and techniques". We compare the sequence of 10 words with the true sequence from the original text by computing BLEU-1, BLEU-2 and BLEU-3 scores. BLEU-1 score considers only the precision at unigram level, while BLEU-3 considers tri-grams.

Project Design

Three structures will be studied: vanilla RNN, LSTM and GRU. Words in the original text are tokenized. No Lemmatizing or stemming are applied as I was not able to find these tools for the Arabic language. The text 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 schema. 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 78245. The number of patterns of 100 consecutive words is 78145 patterns from which the RNN, LSTM and GRU will learn.

The training and test sets are respectively 80% and 20% of the total dataset. So 62516 data points are used for training and 15629 are considered for testing. Validation sets are chosen randomly at each epoch during training and will consist of 10% of the training data.

Algorithms and Techniques

Figure 1 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) with N the number of words in the full corpus.

During the first step of every epoch, the first group of 100 words ($w_0 \dots w_{99}$) is fed into the RNN that makes a prediction of the following word (\hat{w}_{100}). The RNN updates

⁹ K. Papineni, S. Roukos, T. Ward, and W. J. Zhu: BLEU: A method for automatic evaluation of machine translation. In ACL (2002)

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 ($w_1 \dots w_{100}$).

RNN learning architecture

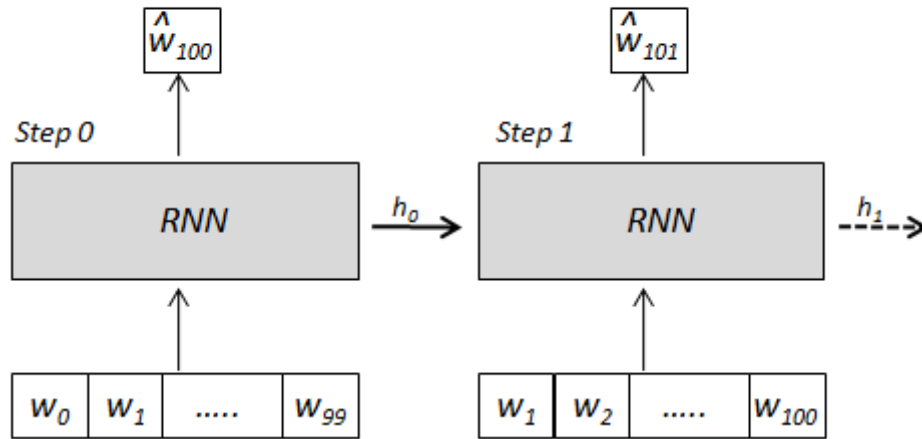


Figure 1: RNN learning architecture

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

Sampling the RNN will consist in providing a random group of 100 words from the original text in the input, let's denote it ($w_0 \dots w_{99}$) and let the RNN predict the following word \hat{w}_{100} . In the next step, we will use as input 99 words from the original group ($w_1 \dots w_{100}$) and add the predicted word as the 100th in the input vector. Figure 2 illustrates this process.

Sampling the RNN

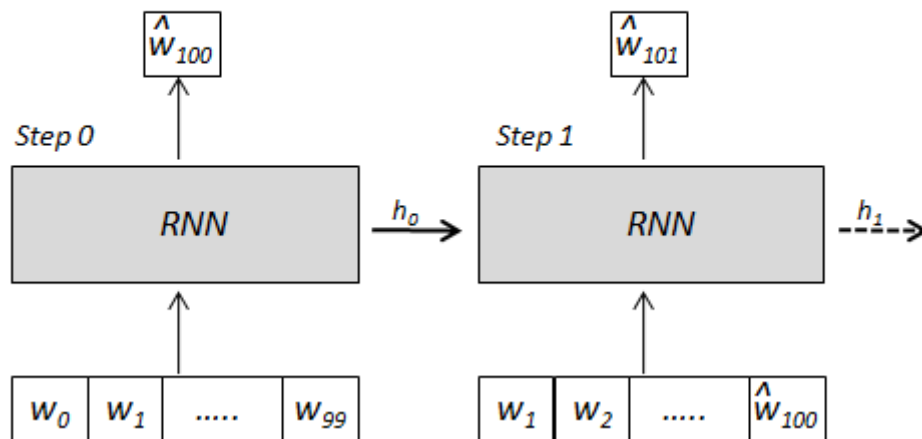


Figure 2: Sampling the RNN

We can continue sampling the RNN by adding at each timestep the predicted word from the last step to the input vector. After 100 timesteps, all the words in the input

vector will be coming from the RNN predictions during the last 100 timesteps. In this case, we can test to which extent the RNN will be able to predict correctly the next word even if no original text is provided in its input.

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.

In order to fine tune the model, the following parameters were used:

- Size of the hidden layer of the RNN: I experimented with 500, 1000 and 2000.
- A dropout layer associated with the RNN: I 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. Therefore, I used the “Adam” optimizer with a “categorical_crossentropy” loss function. Adam optimizer is used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data.

I created a class called “ModelRNN” in a separate file to create different models based on a Simple RNN, LSTM or GRU. Below the method that creates the model:

```
def create_model(self, hidden_layer, input_shape, output_shape):
    self.model = Sequential()
    self.model.add(self.model_type(hidden_layer, input_shape=input_shape))
    self.model.add(Dropout(0.2))
    self.model.add(Dense(output_shape, activation='softmax'))
    self.model.compile(loss='categorical_crossentropy', optimizer='adam')
    self.model.summary()
    return(self.model)
```

Data is split into vectors of 100 word tokens X_i that are used as inputs to the RNN, the model is trained to predict the following word token y_i . The different vectors of 100 words are created by shifting a window of size 100 over the entire corpus as shown below (figure 3):

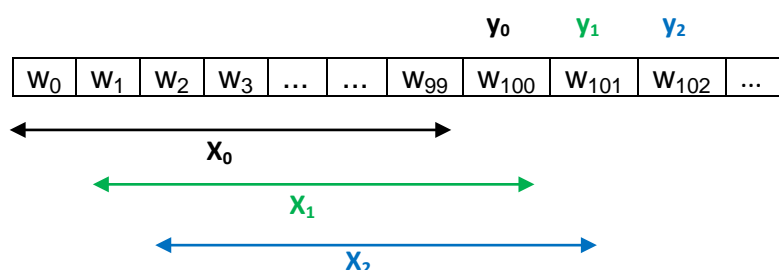


Figure 3: Splitting the corpus into input vector X_i and target values y_i

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

The training was made for all three networks SimpleRNN, LSTM and GRU over the training set. A validation set of 10% of training data is created by Keras during training. Checkpoints were also added in order to save the best weights.

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 without any improvement in the results
- Dropout: 0.2. Actually no big changes were detected by changing the dropout value.

In order to evaluate the performance of each one of the three networks (simple RNN, LSTM and GRU), two assessments were performed:

- The first one is 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 that denoted $\hat{\mathbf{y}}_i$. The accuracy is the number of the correctly predicted words over the number of predicted words
- The second assessment is done using BLEU scores. In this case, for each input vector \mathbf{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 following the vector \mathbf{X}_i in the corpus. BLEU score calculate the similarity at different grams levels. In our case, we considered 1, 2 and 3-grams similarity. The BLEU scores are calculated using the NLTK library.

Assessment 1: Accuracy of prediction

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

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

Table 1: Accuracy on training and test sets

We note that the simple 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 to learn the context of each sentence and predict a word that is adapted to this context. The GRU performs also 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 generalize the training over the whole Quran corpus.

Evaluation of BLEU scores

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

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

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

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

Table 3: BLEU score for the test set

We note the same observations as for the prediction accuracy assessment. The RNN performed poorly while the LSTM showed the best performed results. In addition, 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.

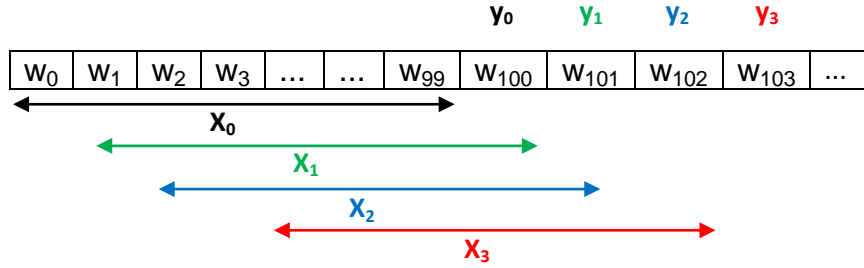


Figure 4: Explaining why LSTM performs better in terms of BLEU score than accuracy score

In Figure 4, let's suppose that we picked randomly for our training set the 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, 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 predict incorrectly 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_4, y_4) . Thus, the LSTM can catch up his error and continue predicting correctly the rest of the sequence even if it fails to predict the word belonging to the test set. Such long term dependency is lower with the GPU and much more 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

بیمینک	تلك	وما	فتردى	هواه	واتبع	بها	يؤمن	لا	من
--------	-----	-----	-------	------	-------	-----	------	----	----

Sequence generated by the LSTM

بیمینک	تلك	وما	فتردى	هواه	واتبع	بها	يؤمن	لا	لا
--------	-----	-----	-------	------	-------	-----	------	----	----

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 learned by the LSTM.

Conclusion

The aim of this project is to learn from sequences of words of an Arabic corpus that is the sacred book of Quran and create models able 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 following each group. The project covered three different recurrent neural networks that are the vanilla RNN, the LSTM and the GRU.

The evaluation metric used 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 mechanism of gating 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 predicts a sequence of 10 words from a vector of 100 words, the generated vector is compared with the correct sequence of 10 words existing in the corpus. In this second assessment, the vanilla RNN did poorly as well, in the other hand, the LSTM and GRU showed good performance, the LSTM still performing better. Indeed the similarity between the 10-words sequence generated by the LSTM has 43% of similarity at unigram level compared the original text and 30.4% of similarity at trigram levels. This observation shows that even if the LSTM predicts incorrectly the first word in the sequence, its long term dependency with past words allows to make better predictions in subsequent words.

An interesting prospect of this work is to test 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. I tried actually to train this structure using an LSTM in both generator and discriminator models, unfortunately, I was not able to make this architecture converge. The source files are provided in my github account.