Evaluation of Gated Recurrent Unit in Arabic Diacritization

Rajae Moumen[1], Raddouane Chiheb[2], Rdouan Faizi[3], Abdellatif El Afia[4]
ENSIAS
Mohammed V University
Rabat, Morocco

***Abstract*—Recurrent neural networks are powerful tools giving excellent results in various tasks, including Natural Language Processing tasks. In this paper, we use Gated Recurrent Unit, a recurrent neural network implementing a simple gating mechanism in order to improve the diacritization process of Arabic. Evaluation of Gated Recurrent Unit for diacritization is performed in comparison with the state-of-the art results obtained with Long-Short term memory a powerful RNN architecture giving the best-known results in diacritization. Evaluation covers two performance aspects, Error rate and training runtime.**

Keywords—Gated recurrent unit; long-short term memory; arabic diacritization

I. Introduction

Natural languages require different processing steps in order to perform Natural Language Processing (NLP) tasks, such as Text-to-speech synthesis (TTS), speech recognition, sentiment analysis, information retrieval, etc. In the case of Arabic, an additional preprocessing step is mandatory: Diacritization, or diacritic restoration. Diacritics are signs placed below or above a letter indicating a different phonetic value.

Arabic is a semitic language with two varieties: Classical and Modern. Classical Arabic is the pure language spoken by Arabs; Modern Standard Arabic (MSA) is an evolving variety with constant new terms to meet the modern innovations and changes. Generally, Arabic (Classical or MSA) is transcripted without diacritics, leading to different ambiguities at various linguistic levels as explained in [1].

According to [2], in over 77% of cases, a non-vocalized word can have several possible diacritizations and consequently different possible meanings.

TABLE I. POSSIBLE DIACRITIZATIONS FOR THE STRING "صدق"

Word	**Diacritized word**	**Meaning**
صَدَقَ	Sadaka	He is right
صدَّقَ	Sad~aka	He believes
صدِّق	Sud~ika	We believe him
صِدّْقُ	Sidku	Truth
صُدُقٌ	Sudku	Dowries

Table 1 gives an example of this aspect and lists some of the possible diacritization forms of the string "صدق" and the inferred meaning.

Arabic diacritization received a lot of interest and went through different models: Rule based models, statistical models and hybrid models.

Rule based models rely on existing linguistic rules formulated, in most cases by human experts. They have proven an acceptable efficiency in diacritization, given the lack of linguistic resources. The major drawback of rule-based models is the laborious, costly and time-consuming task to formulate and maintain rules that covers all rich linguistic aspects of Arabic. Moreover, Rule based models require strong linguistic knowledge.

Statistical models attempt to learn a diacritization model from diacritized texts; by predicting the probability of distribution of a sequence of words or characters. Authors in [3] present a review of these methods, using Hidden Markov chains, n-gram or finite state transducers.

The weakness of statistical models is their reliability on large corpus of fully diacritized text. Their strength is that no linguistic knowledge or tools like Pos Taggers or morphological analyzers are needed.

Hybrid methods come from the statement that the strength of linguistic knowledge combined to statistic methods would yield to better results.

Recurrent neural networks (RNN) language models have been used to solve diacritization problem as a statistical or hybrid method. The results are impressive and the error is proven to be asymptotic with a DER of 5.08% over all characters using long short-term memory (LSTM) a powerful RNN architecture proving its performance in various tasks. Moreover, RNN have been used successfully without relying on any linguistic tools, and solely on diacritized corpus.

However, the superior performance of RNN comes at the cost of expensive model training, reaching days or weeks in some experimental settings requiring a significant computation capability.

As a solution to these issues, authors introduced new RNN architectures with simple internal architectures. GRU, introduced in [4] is one of these RNN; in some tasks, it seems to perform better on the training runtime, and maintain a comparable accuracy to LSTM. To our knowledge, diacritization has never been addressed by GRU.

In this paper, we present the results of our evaluation of GRU to enhance the diacritization results regarding its performance in training runtime and error scoring. We use the performances scored by LSTM in diacritization as a baseline. We show that we achieve to maintain the state-of-the art results with better scoring on the training and runtime.

II. Related Work

A. RNN Performance

The first motivation of this study is to enhance the performances of Arabic diacritization. The evaluation of GRU on Arabic diacritization is conducted in comparison with LSTM. In literature, many studies have made this evaluation in various tasks. For example, in [5], authors used a probabilistic approach to determine which RNN architecture is optimal. They evaluated thousand different RNN architectures and identified some that outperform LSTM and GRU in some tasks but not all. In [6], authors compared LSTM and variants over large-scale tasks such as speech recognition, handwriting recognition, etc. Authors conclude that variants of LSTM do not improve significantly the performance.

In [7], an empirical comparison between LSTM and GRU is performed for music and speech modeling. The study did not conclude the superiority of one on the other and then considers GRU to be a better choice since it uses less parameters.

We find the studies in literature inconclusive and insufficient to generalize over all tasks, taking into consideration the considerable differences that might remain in experimental settings and characteristics of the addressed problem.

B. Arabic Diacritization

The Arabic diacritization problem is mainly addressed as a classification problem over seven classes corresponding to the possible diacritics.

Diacritization is divided into two sub-types: The morphological diacritization giving satisfying results reaching an error of 3% to 4%, while syntactic diacritization is still to be improved with a rate of 9.9%.

Earlier approaches in diacritization are rule-based models, like in [8], where morphological analyzer is used for semi-automatic diacritization. Work in [9] presents “Alserag”, another rule based system working through three modules: Morphological analysis, syntactic analysis and morph-phonological module. The system scores 8.68% as diacritization error rate (DER) and 18.63% as word error rate (WER). The main drawback of rule-based models is their high development cost; and the fact that creating linguistic resources such as corpuses are laborious task that need to be reproduced over the studied language.

More recent studies benefit from the evolution of machine learning to learn a diacritization model from vocalized text, the study in [3] presents an overview of these models: Using Maximum Entropy, Hidden Markov Models (HMM) and weighted finite state machine.

In [10], a maximum entropy model is trained for sequence classification to restore the diacritic of each character. They used The Arabic treebank corpus, containing 600 documents form newspapers with over 340k words. The system achieves a DER of 5.5% and a WER of 18%.

More recently, Machine-learning models tend to rely on less and less external resources such as in the study in [11] where the model relies solely on diacritized text. To our knowledge, among systems depending on no external resources, this study gives the best results scored to now with a DER of 8.14% for the end-case character, and 5.08% for all characters.

III. Recurrent Neural Networks

RNN are computational approach based on large connected neural unit forming a directed graph.

In basic RNN we assume that $w = (w_1, \dots, w_T)$ is the input sequence, $h = (h_1, \dots, h_T)$ the hidden sequence and $y = (y_1, \dots, y_T)$ the output sequence, the basic RNN computes h and y by executing the equations (1) and (2) from t=1 to T iterations:

$$h_t = f(w_{hh}h_{t-1} + w_{ih}x_t + b_h) \quad (1)$$

$$y_t = w_{h0}h_t + b_0 \quad (2)$$

f is the activation function for hidden layers, w the matrix weights and b the bias vector, h the hidden bias vector.

RNN are suitable for capturing dependencies among sequential data types. The problem of RNN is that they remain weak on long-term dependencies as studied in [12] where authors proved the difficulty to capture long-term dependencies because of the “vanishing” or “exploding” of stochastic gradients.

Gated recurrent neural networks (GNN) have been proposed to resolve this problem; LSTM and GRU belong to the category of GNN.

A. Long Short-Term Memory (LSTM)

Introduced in [13], LSTM are special case of RNN capable to resolve long-term dependencies issue encountered in standard RNN by using a gating mechanism.

LSTM has the property to remember patterns selectively. Making them suitable for a number of sequence learning problems such as language modelling, translation, speech recognition, and Arabic diacritization.

Study in [11] uses LSTM to build a language-independent diacritizer trained solely from vocalized text without referring to any external tools. Authors run several RNN architectures and achieve with a 3-layer-Bidirectional LSTM to reach a DER of 4.85%.

In the work in [14], the problem is approached with bidirectional LSTM considering diacritization as a sequence transcription problem. The system does not require any previous treatment (lexical, morphological or syntactic) The WER scored in this study is 5.82%.

B. Gated Recurrent Unit (GRU)

First proposed in [4], GRU is generally incorrectly considered as a special-case of LSTM, because of the fact that global architecture is quite similar to LSTM. In fact, GRU is quite different form LSTM .It defines two gating signals instead of three in LSTM, an update signal and a reset gate.

GRU has no cell state. Unlike LSTM, it exposes the memory content at each time step. The transition between the previous memory content and the new memory contents is made using leaky integration controlled by the update gate.

GRU has shown its efficiency in many studies like [15] and [16], it achieves promising results in classification tasks and reduces the training runtime since few iterations are needed to update the hidden states and the internal structure of a cell is simplified.

However, GRU still comes second to LSTM in terms of performance. Therefore, GRU is mainly used in situations where fast training is needed with limited computation capability.

IV. Approach

A. Experimental Setup

Our goal is to set up an environment based on GRU to maintain state-of-the art results and enhance the computation efficiency. For this purpose, we use as a comparison pattern the approach used in [11], giving the best results to our knowledge in same conditions, relying solely on diacritized text.

We compare the error rate and runtime of both GRU and LSTM over same datasets and experimental setup. For this purpose, we use the setup described in Fig. 1. The GNN in the figure stands for the network used, namely GRU and LSTM.

We use single recurrent layer for our networks for limited computation capacity in one hand, and in the other hand to omit potential issues related to multilayer deep learning architectures.

The network needs exclusively two inputs for training, represented in two separate documents, the first one contains the diacritized text, and the second contains the equivalent undiacritized text.

In the whole process, we use the Buckwalter Arabic Transileration, an ASCII scheme, representing Arabic orthography strictly one-to-one.

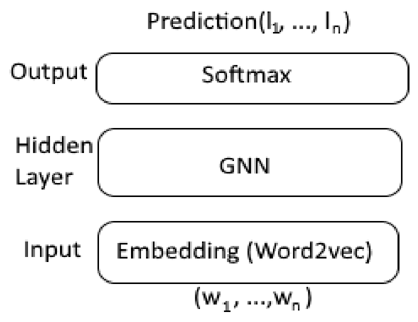

Fig. 1. Architecture of the GNN for Arabic Diacritization.

We tried different environments for experiments, and the results might be quite different to an environment to another; in this paper, we present the results of experiments with Python, Numpy and Theano [17]. We use Theano in order to parallelize computation of GPU and giving the best possible results with the limited computation capacity.

We first go through Word embedding, i.e. mapping the characters sequence into letter vectors. We use for this purpose word2vec [18] implemented with Theano and Lasagne Framework.

We use GRU as it has been introduced in [4] and we define the states as following in the equations (3), (4), (5) and (6):

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot s_t \quad (3)$$

$$s_t = g(W_h x_t + U_h(r_t * h_{t-1}) + b_h) \quad (4)$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad (5)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad (6)$$

$\odot$ stands for element-wise multiplication.

x_t being the input vector

h_t being the output vector

z_t being the update gate vector

r_t being the reset gate vector

W, U and b the parameter matrices and vector

g being the chosen function, here we use *Tanh* as it has been proved to converge faster in practice.

We implement LSTM as following in (7), (8), (9), (10), (11) and (12):

$$f_t = \sigma\left(W_f.[h_{t-1}, x_t] + b_f\right) \quad (7)$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \quad (8)$$

$$C'_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \quad (9)$$

$$C_t = (f_t * C_{t-1} + i_t * C'_{t-1}) \quad (10)$$

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \quad (11)$$

$$h_t = o_t * \tanh(C_t) \quad (12)$$

x_t being input vector

h_{t-1}being the previous cell output

C_{t-1} being the previous cell memory

H_t being the current cell output

C_t being the current cell Memory

W, U = Weight vectors for gates (forget: f; candidate: c; i/p gate: i; o/p gate: o)

For training, test and dev, we use the data described in the section IV-*B*.

We use a softmax output layer because it ensures that the sum of possible output is 1, moreover, it defines a distribution overall output target classes, suitable to the addressed problem.

B. *Dataset*

The Dataset used in this work is a part of the corpus “Tashkeela” introduced in [19]. The corpus contains 75 millions of fully vocalized words extracted from 97 books mainly religious and media online from different sources in classical and modern Arabic language. We use the approach described in [10] by splitting the part of the corpus into three parts, Train/Dev/Test as described in Table 2.

TABLE II. DATA STATISTICS

	Training	**Dev**	**Test**
Words	560k	90k	80k

V. RESULTS AND DISCUSSION

A. *Evaluation Metrics*

To evaluate the accuracy of GRU we adopt the same metrics used in most of the diacritization studies, for instance [14], [11], [3]. The metrics are the Diacritics Error rate (DER) that compares the diacritization of the predicted word with the input word at a character level. In addition to the word error rate (WER), that compares the predicted word to the original at a word level, in other terms, if an error is detected on a character, the whole word is considered incorrectly diacritized.

To evaluate the performance of GRU, we consider the average epoch runtime.

B. *Diacritic Error*

Table 3 lists the results of the networks error rates, measured with diacritic error rate (DER) over all diacritics DER (ALL) and DER over the last character only DER (Last) on the dev dataset.

TABLE III. DER FOR GRU AND LSTM OVER DEV DATASET

	DER (ALL)	**DER(Last)**
LSTM	7.15	10.50
GRU	7.31	10.79

Experiments show that LSTM achieves better results than GRU in both the DER of all characters and the DER of the last character; however, the results obtained by GRU are still good.

a) Qualitative Analysis

To identify the errors we get from the system implemented with GRU, we proceeded in a direct way by classifying a sample set of the words incorrectly diacritized.

For classification, we use the diacritization scheme error proposed in [20].

Table 4 and Table 5 show the distribution of errors among the sub-categories. We notice that both networks behave in the same way, and distribution over categories is quite the same; we notice that most of the errors are made in Form/Spelling, this is to be expected as it has been reported in works like [21] that the diacritization tools are less performant in case-ending diacritics.

TABLE IV. GRU ANNOTATED ERROR CATEGORIES

Error Category	**Sub-category**	**Nbr of Errors**
Form/Spelling	Shadda	8
	Tanween	11
Grammar	Active-Passive Voice	4
Diacritization	Missing short Vowel	5
	Confused short Vowel	2
Morphology	Partial inflection	1
	Full-inflection	0
Overall		31

TABLE V. LSTM ANNOTATED ERROR CATEGORIES

Error Category	**Sub-category**	**Nbr of Errors**
Form/Spelling	Shadda	6
	Tanween	10
Grammar	Active-Passive Voice	3
Diacritization	Missing short Vowel	5
	Confused short Vowel	1
Morphology	Partial inflection	1
	Full-inflection	0
Overall		26

C. *Training Performances*

Fig. 2 compares average Epoch Runtime of GRU and LSTM.

The results do not include the embedding process training.

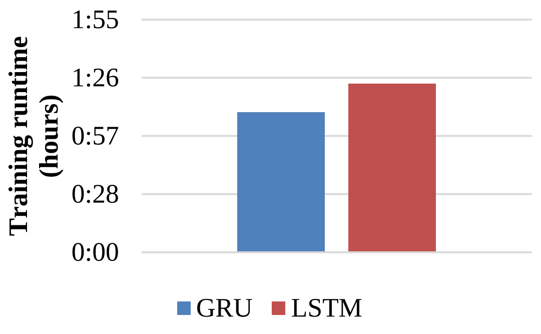

Fig. 2. Average Epoch Runtime of GRU & LSTM.

D. *Discussion*

We notice that GRU scores better results and outperforms LSTM in training. The Training time is reduced by 18.82%.

The evaluation showed that GRU gives comparable results to LSTM in diacritization accuracy and improves the training process. Consequently, we assume that GRU gives satisfactory results in Arabic diacritization. However, we consider our study to be completed by running different experiment settings over different datasets.

VI. Conclusion

In this paper, we presented a new approach for Arabic diacritization using gated recurrent unit. We showed that GRU outperforms LSTM in training runtime and gives an error rate comparable to LSTM. In future work, we intend to evaluate our approach over different datasets and integer the diacritization tool into a text-to-speech synthesizer for Arabic.

References

[1] A Azmi and R. S.Al Majed, «A survey of automatic Arabic diacritization techniques» *Natural language Engineering,* vol. 3, n° 121, pp. 477-495, 2013.

[2] M. Boudchiche and A. Mazroui, «Evaluation of the ambiguity caused by the absence of diacritical marks in Arabic texts: statistical study» *5th International Conference on Information & Communication Technology and Accessibility (ICTA)*, Marrakech, Morocco, Dec. 2015.

[3] I. Zitouni and R. Sarikaya, «Arabic diacritic restoration approach based on Maximum Entropy Models» *Computer Speech and Language,* vol. 23, n° 13, pp. 257-276, July 2009.

[4] K. Cho, B. v. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation» *Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, October 2014.

[5] R. Jozefowicz, W. Zaremban and I. Sutskever, «An Empirical exploration of recurrent network architectures» *Proceedings of the 32nd International Conference on Machine Learning*, Lille, 2015.

[6] K. e. A. Greff, «A search space odyssey.» *IEEE Trans. Neural Networks. lean. Syst.,* 2016.

[7] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, «Empirical evaluation of gated reccurent neural networks on sequence modeling» *ArXiv preprint,* 2014.

[8] T. A. El-Sadany and M. A. Hashish, «Semi-Automatic Vowelization of Arabic Verbs» *10th National computer conference, Utilization of computers in development*, Jeddah; Saudi Arabia, 1988.

[9] S. Alansary, «Alserag: An Automatic Diacritization System for Arabic» *Intelligent Natural Language Processing: Trends and Applications,* 2017, pp. 523-543.

[10] I. Zitouni, J. S. Sorensen and R. Sarikaya, «Maximum entropy based restoration of Arabic diacritics» *21st International Conference on Computational Linguistics*, Sydney, Australia, July 2006.

[11] Y. Belinkov and J. Glass, «Arabic diacritization with Reccurent Neural Networks» *Conference on Empirical Methods in Natural Language Processing,* Lisbon, Portugal, September 2015.

[12] Y. Bengio, P. Simard and P. Frasconi, «Learning long-term dependencies with gradient descent is difficult» *IEEE Transactions on Neural Networks,* vol. 5, n° 12, pp. 157 - 166, Mar 1994.

[13] S. H. a. J. Schmidhuber, «Long Short-Term Memory» *Neural Comput. 9*, 1997, pp. 735-1780.

[14] G. A. Abandah, A. Graves, B. Al-Shagoor, A. Arabiyat, F. Jamour and M. Al-Taee, «Automatic diacritization of Arabic text using recurrent neural networks» *International Journal on Document Analysis and Recognition,* vol. Volume 18, pp. 183-197, June 2015.

[15] D. Bahdanau, K. Cho and Y. Bengio, «Neural machine translation by jointly learning to align and translate» *International Conference on Learning Representations*, Vancouver, 2015.

[16] A. Graves, «Generating sequences with recurrent neural networks» *Neural and Evolutionary Computing*.

[17] T. D. Team, « Theano: A Python framework for fast computation of mathematical expressions» 2016.

[18] M. Tomas, S. Ilya, C. Kai, C. Greg and D. Jeffrey, «Distributed Representations of Words and Phrases and their Compositionality» *Advances in neural information processing systems*, 2013.

[19] T. Zerrouki and A. Balla, «Tashkeela: Novel corpus of Arabic vocalized texts, data for auto-diacritization systems» *Data in brief,* February 2017.

[20] G. Abuhakema, R. Faraj, A. Feldman and E. Fitzpatrick, «Annotating an arabic learner corpus for error» *Proceedings of the International Conference on Language Resources and Evaluation*, Marrakech, Morocco, 2008.

[21] N. Habash, R. Gabbard, O. Rambow, S. Kulick and M. Marcus, «Determining Case in Arabic: Learning Complex Linguistic Behavior Requires Complex Linguistic Features.» *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning,*, Prague, Czech Republic, 2007.