Arabic Diacritization with Recurrent Neural Networks

Yonatan Belinkov and **James Glass**
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
{belinkov, glass}@mit.edu

Abstract

Arabic, Hebrew, and similar languages are typically written without diacritics, leading to ambiguity and posing a major challenge for core language processing tasks like speech recognition. Previous approaches to automatic diacritization employed a variety of machine learning techniques. However, they typically rely on existing tools like morphological analyzers and therefore cannot be easily extended to new genres and languages. We develop a recurrent neural network with long short-term memory layers for predicting diacritics in Arabic text. Our language-independent approach is trained solely from diacritized text without relying on external tools. We show experimentally that our model can rival state-of-the-art methods that have access to additional resources.

1 Introduction

Hebrew, Arabic, and other languages based on the Arabic script usually represent only consonants in writing and do not mark vowels. In such writing systems, diacritics are used for marking short vowels, gemination, and other phonetic units. In practice, diacritics are usually restricted to specific settings such as language teaching or to religious texts. Faced with a non-diacritized word, readers infer missing diacritics based on their prior knowledge and the context of the word in order to resolve ambiguities. For example, Maamouri et al. (2006) mention several types of ambiguity for the Arabic string عـلـم *Elm*, both within and across part-of-speech tags, and at a grammatical level. In practice, a morphological analyzer like MADA (Habash et al., 2009) produces at least 13 different diacritized forms for this word, a subset of which is shown in Table 1.[1]

Word	Gloss
Ealima	he knew
Eulima	it was known
Eal~ama	he taught
Eilomu	knowledge (def.nom)
...	...
EilomK	knowledge (indef.gen)
Ealamu	flag (def.nom)
...	...
EalamK	flag (indef.gen)

Table 1: Possible diacritized forms for عَلَم *Elm*.

The ambiguity in Arabic orthography presents a problem for many language processing tasks, including acoustic modeling for speech recognition, language modeling, text-to-speech, and morphological analysis. Automatic methods for diacritization aim to restore diacritics in a non-diacritized text. While earlier work used rule-based methods, more recent studies attempted to learn a diacritization model from diacritized text. A variety of methods have been used, including hidden Markov models, finite-state transducers, and maximum entropy – see the review in (Zitouni and Sarikaya, 2009) – and more recently, deep neural networks (Al Sallab et al., 2014). In addition to learning from diacritized text, these methods typically rely on external resources such as part-of-speech taggers and morphological analyzers like the MADA tool (Habash and Rambow, 2007). However, building such resources is a labor-intensive task and cannot be easily extended to new languages, dialects, and domains.

[1] Arabic transliteration follows the Buckwalter scheme: http://www.qamus.org/transliteration.htm.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 2281–2285,
Lisbon, Portugal, 17-21 September 2015.

Diacritic	Transliteration	Transcription
دَ	*a*	/a/
دُ	*u*	/u/
دِ	*i*	/i/
دً	*F*	/an/
دٌ	*N*	/un
دٍ	*K*	/in/
دّ	~	Gemination
دْ	*o*	No vowel

Table 2: Arabic diacritics.

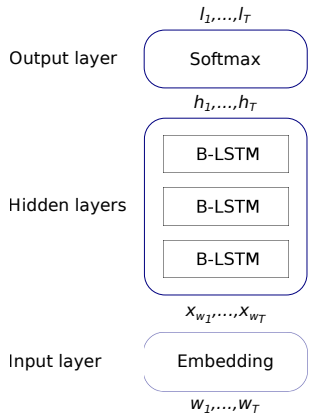

Figure 1: An illustration of our network topology.

In this work, we propose a diacritization method based solely on diacritized text. We treat the problem as a sequence classification task, where each character has a corresponding diacritic label. The sequence is modeled with a recurrent neural network whose input is a sequence of characters and whose output is a probability distribution over the diacritics. Any RNN architecture can be used in this framework; here we focus on long short-term memory (LSTM) networks, which have shown recent success in a number of NLP tasks. We experiment with several architectures and show that we can achieve state-of-the-art results, without relying on external resources. Error analysis demonstrates the benefit of using LSTM over simpler neural networks.

2 Linguistic Background

Languages based on the Arabic script typically employ an *abjad* writing system, where each symbol represents a consonant while vowels and other phonetic units, commonly known as diacritics, are usually omitted in writing. In modern standard and classical Arabic, these include the short vowels *a*, *u*, and *i*, the case endings *F*, *N*, and *K*, the gemination marker ~, and the silence marker *o*.[2] Table 2, modified from (Habash et al., 2007), lists the diacritics. Importantly, the gemination marker ~ can combine with short vowels and case endings (e.g. Table 1, row 3).

3 Approach

We define the following sequence classification task, similarly to (Zitouni and Sarikaya, 2009).

[2]We also include the low-frequency superscript Alif ‘ that is usually ignored due to its limitation to fixed lexical items.

Let $\mathbf{w} = (w_1, ..., w_T)$ denote a sequence of characters, where each character w_t is associated with a label l_t. A label may represent 0, 1, or more diacritics, depending on the language. Assume further that each character w in the alphabet is represented as a real-valued vector x_w. This character embedding may be learned during training or fixed.

Our neural network has the following structure, illustrated in Figure 1:

- Input layer: mapping the letter sequence $\mathbf{w}$ to a vector sequence $\mathbf{x}$.
- Hidden layer(s): mapping the vector sequence $\mathbf{x}$ to a hidden sequence $\mathbf{h}$.
- Output layer: mapping each hidden vector h_t to a probability distribution over labels l.

During training, each sequence is fed into this network to create a prediction for each character. As errors are back-propagated down the network, the weights at each layer are updated. During testing, the learned weights are used in a forward step to compute a prediction over the labels. We always take the best predicted label for evaluation.

Hidden layer Our main system relies on long short-term memory networks (LSTM) (Hochreiter and Schmidhuber, 1997; Graves et al., 2013). Here we describe a single LSTM layer and refer to Graves et al. (2013) for the extension to bidirectional LSTM (B-LSTM) and to multiple layers. The LSTM computes the hidden representation for

	Train	Dev	Test
Words	470K	81K	80K
Letters	2.6M	438K	434K

Table 3: Arabic diacritization corpus statistics.

input x_t with the following iterative process:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t \odot c_{t-1} + \\
&\quad i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

where σ is the sigmoid function, $\odot$ is element-wise multiplication, and i, f, o, and c are input, forget, output, and memory cell activation vectors. The crucial element is the memory cell c that is able to store and reuse long term dependencies over the sequence. The W matrices and b bias vectors are learned during training.

Implementation details The input layer maps the character sequence to a sequence of letter vectors, initialized randomly. We also tried initializing with letter vectors trained from raw text with `word2vec` (Mikolov et al., 2013a; Mikolov et al., 2013b), but did not notice any improvement, probably due to the small letter vocabulary size. The input layer also stacks previous and future letter vectors, enabling the model to learn contextual information. We use a letter embedding size of 10 and a window size of 5 characters, so the input size is 110.

We experiment with several types of hidden layers, ranging from one feed-forward layer to multiple B-LSTM layers. We also add a linear projection after the input layer. This has the effect of learning a new representation for the letter embeddings. The output layer is a Softmax over labels:

$$P(l|w_t) = \frac{\exp(y_t[l])}{\sum_{l'} \exp(y_t[l'])}$$

where $y_t = W_{hy}h_t + b_y$ and $y_t[l]$ is the l^{th} element of y_t.

Training is done with stochastic gradient descent with momentum, optimizing the cross-entropy objective function. Layer sizes and other hyper-parameters are tuned on the Dev set. Our implementation is based on Currennt (Weninger et al., 2015).

	DER		
Model	All	End	# params
Feed-forward	11.76	22.90	63K
Feed-forward (large)	11.55	23.40	908K
LSTM	6.98	10.36	838K
B-LSTM	6.16	9.85	518K
2-layer B-LSTM	5.77	9.18	916K
3-layer B-LSTM	5.08	8.14	1,498K

Table 4: Diacrtic error rates (DERs) on the Dev set, over all diacritics and only at word ending.

MaxEnt (only lexical)	8.1
MaxEnt (full)	5.1
3-layer B-LSTM	4.85

Table 5: Results (DER) on the Test set. MaxEnt results from (Zitouni and Sarikaya, 2009)

4 Experiments

Data We extract diacritizied and non-diacritized texts from the Arabic treebank, following the Train/Dev/Test split in (Zitouni and Sarikaya, 2009). Table 3 provides statistics for the corpus.

Every character in our corpus has a label corresponding to 0, 1, or 2 diacritics, in the case of the gemination marker combining with another diacritic. Thus the label set almost doubles. We opted for this formulation due to its simplicity and generalizability to other languages, even though previous work reported improved results by first predicting gemination and then all other diacritics (Zitouni and Sarikaya, 2009).

Results Table 4 shows the results of our models on the Dev set in terms of the diacritic error rate (DER). Clearly, LSTM models perform much better than simple feed-forward networks. To make the comparison fair, we increased the number of parameters in the feed-forward model to match that of the LSTM. In this setting, the LSTM is still much better, indicating that it is far more successful at exploiting the larger parameter set. Interestingly, the bidirectional LSTM works better than a unidirectional one, despite having less parameters. Finally, deeper models achieve the best results.

On the Test set (Table 5), our 3-layer B-LSTM model beats the lexical variant of Zitouni and Sarikaya (2009) by 3.25% DER, a 40% error reduction. Moreover, we outperform their best model, which also used a segmenter and part-of-

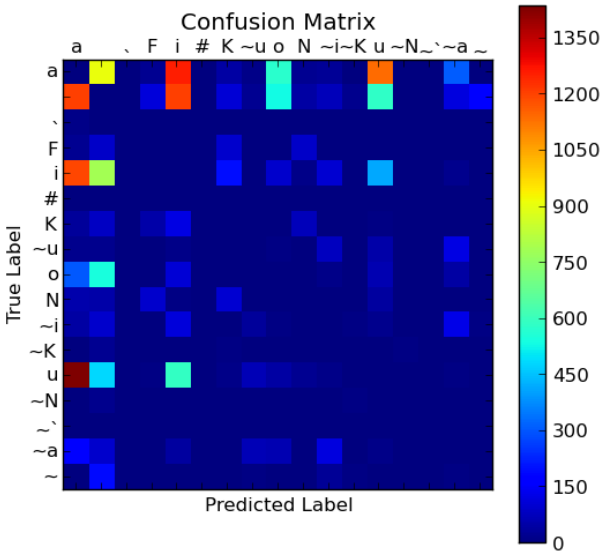

Figure 2: A confusion matrix of errors made by our system. "#" marks word boundary. Best viewed in color.

Model	Diacritization
Gold	AiEotabara Almudiyru AlEAm~u l " Aln~ahAri " juborAn tuwayoniy~ Aan Alt~a$okiylAti AlqaDA}iy~apa jA'at lita-moyiyEi milaf~i maHaT~api Al " Aim . tiy . fiy . "
Feed-forward	AiEotabara Almudiyru AlEAm~u l " Aln~ahAr_ " jaborAn tuwayoniy_ Aan Alt~a$okiylAti AlqaDA}iy~api jA'at litamayiyEi malaf~i maHaT~api Al " A_m . tiy . fiy . "
LSTM	AiEotabara Almudiyru AlEAm~u l_ " Aln~ahAri " juborAn t_w_yoniy_ Ain Alt~a$okiylAti AlqaDA}iy~apa jA'at litamoyiyEa milaf~i maHaT~api Al " Aim . tiy . fiy . "
B-LSTM	AiEotabara Almudiyru AlEAm~u l " Aln~ahAri " juborAn t_wayoniy_ Aan Alt~a$okiylAti AlqaDA}iy~apa jA'at lita-moyiyEi milaf~i maHaT~api Al " Aim . tiy . fiy . "

Figure 3: Sample errors by selected diacritization models. Wrong predicted diacritics are underlined and in red; missing diacritics are noted by underscore. Translation: "The editor of An Nahar, Gebran Tueni, thought that the judicial formations came to dilute the issue of MTV station".

speech tagger. This shows that our model can effectively learn to diacritize without relying on any resources other than diacritized text.

Finally, some studies report work on a Train/Test data split, without a dedicated Dev set (Zitouni et al., 2006; Habash and Rambow, 2007; Rashwan et al., 2011; Al Sallab et al., 2014). We were reluctant to follow this setting so we performed all development on the Dev set of (Zitouni and Sarikaya, 2009). Still, we ran our best model on the Train/Test split and achieved a DER of 5.39% on all diacritics and 8.74% on case endings. The first result is behind the state-of-the-art (Al Sallab et al., 2014) by 2% but the second one is better by 3%. Given that we did not tune the system for this data set, this result is encouraging.

Error Analysis A quantitative analysis of the errors produced by one of our models on the Dev set is shown in Figure 2. The heat map denotes the number of errors produced. The major source of errors comes from confusing the short vowels *a*, *i*, and *u*, among themselves and with no diacritic. This is expected due to the high rate of short vowels in Arabic compared to other diacritics. It also explains why methods that take the confusion matrix into account in their classification algorithm do quite well (Al Sallab et al., 2014).

We also analyzed some errors qualitatively. Figure 3 shows the errors produced by several of our diacritization models on a sample sentence. Interestingly, the simple feed-forward model fails to predict the correct case ending on the word *AlqaDA}iy~ap* ("judicial"), while both LSTM models succeed. This may indicate that LSTM indeed captures the kind of long-distance dependencies that are responsible for case marking. Other errors are more difficult to explain, but note that all models struggle with the proper name *tuwayoniy~* ("Tueini"), which is difficult to solve without external resources.

5 Conclusion

In this work, we develop a recurrent neural network that predicts diacritics in non-diacritized texts. Our model is language agnostic: it is trained solely from diacritized text without relying on additional resources. Using LSTM units, we demonstrate that our model can effectively learn to diacritize Arabic texts and rivals state-of-the-art methods that rely on language-specific tools.

In future work, we intend to incorporate our diacritization system in a speech recognizer. Recent work has shown improvements in Arabic speech recognition by diacritizing with MADA (Al Hanai and Glass, 2014). Since creating such tools is a labor-intensive task, we expect our diacritization approach to promote the development of speech recognizers for other languages and dialects.

Acknowledgments

This research was supported by the Qatar Computing Research Institute (QCRI).

References

Tuka Al Hanai and James Glass. 2014. Lexical Modeling for Arabic ASR: A Systematic Approach. In *Proceedings of INTERSPEECH*.

Ahmad Al Sallab, Mohsen Rashwan, Hazem M. Raafat, and Ahmed Rafea. 2014. Automatic Arabic diacritics restoration based on deep nets. In *Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP)*.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of ICASSP*.

Nizar Habash and Owen Rambow. 2007. Arabic Diacritization through Full Morphological Tagging. In *Proceedings of HLT-NAACL*.

Nizar Habash, Abdelhadi Soudi, and Timothy Buckwalter. 2007. On Arabic Transliteration. In Abdelhadi Soudi, Günter Neumann, and Antal Van den Bosch, editors, *Arabic Computational Morphology: Knowledge-based and Empirical Methods*, pages 15–22. Springer.

Nizar Habash, Owen Rambow, and Ryan Roth. 2009. MADA+TOKAN: A Toolkit for Arabic Tokenization, Diacritization, Morphological Disambiguation, POS Tagging, Stemming and Lemmatization. In *Proceedings of the Second International Conference on Arabic Language Resources and Tools*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780.

Mohamed Maamouri, Ann Bies, and Seth Kulick. 2006. Diacritization: A challenge to Arabic treebank annotation and parsing. In *Proceedings of the British Computer Society Arabic NLP/MT Conference*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of Workshop at ICLR*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of NIPS*.

M.A.A. Rashwan, M.A.S.A.A. Al-Badrashiny, M. Attia, S.M. Abdou, and A. Rafea. 2011. A Stochastic Arabic Diacritizer Based on a Hybrid of Factorized and Unfactorized Textual Features. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(1):166–175, Jan.

Felix Weninger, Johannes Bergmann, and Björn Schuller. 2015. Introducing CURRENNT: The Munich Open-Source CUDA RecurREnt Neural Network Toolkit. *JMLR*, 16:547–551.

Imed Zitouni and Ruhi Sarikaya. 2009. Arabic Diacritic Restoration Approach Based on Maximum Entropy Models. *Comput. Speech Lang.*, 23(3).

Imed Zitouni, Jeffrey S. Sorensen, and Ruhi Sarikaya. 2006. Maximum Entropy Based Restoration of Arabic Diacritics. In *Proceedings of COLING and ACL*.