A System for Diacritizing Four Varieties of Arabic

Hamdy Mubarak **Ahmed Abdelali** **Kareem Darwish** **Mohamed Eldesouki**
Younes Samih **Hassan Sajjad**
{hmubarak,aabdelali}@qf.org.qa
Qatar Computing Research Institute, HBKU Research Complex, Doha 5825, Qatar

Abstract

Short vowels, aka diacritics, are more often omitted when writing different varieties of Arabic including Modern Standard Arabic (MSA), Classical Arabic (CA), and Dialectal Arabic (DA). However, diacritics are required to properly pronounce words, which makes diacritic restoration (a.k.a. diacritization) essential for language learning and text-to-speech applications. In this paper, we present a system for diacritizing MSA, CA, and two varieties of DA, namely Moroccan and Tunisian. The system uses a character level sequence-to-sequence deep learning model that requires no feature engineering and beats all previous SOTA systems for all the Arabic varieties that we test on.

1 Introduction

Most varieties of Arabic are typically written without short vowels, aka diacritics, and readers need to recover such diacritics to properly pronounce words. Modern Standard Arabic (MSA) and Classical Arabic (CA) use two types of diacritics, namely: core-word diacritics, which specify lexical selection, and case endings, which generally indicate syntactic role. Conversely, Arabic Dialects mostly use core-word diacritics and usually use a silence diacritic (sukun) for case-endings. For example, given the present tense MSA and CA verb "yfhm"[1] can be diacritized as "yafoham" (meaning: "he understands") or "yufah∼im" ("he explains"). Both can accept dammah (u), fatHa (a), or sukun (o) as grammatical case endings according to surrounding context. The equivalent version in some Arabic dialects is "byfhm", which can be diacritized as "biyifohamo" ("he understands") or "biyofah∼imo" ("he explains"). This highlights the complexity of the task of recovering the diacritics, a prerequisite for Language Learning (Asadi, 2017) and Text to Speech (Sherif, 2018) among other applications.

In this paper, we present a system that employs a character-based sequence-to-sequence model (seq2seq) (Britz et al., 2017; Cho et al., 2014; Kuchaiev et al., 2018) for diacritizing four different varieties of Arabic. We use the approach described by Mubarak et al. (2019), which they applied to MSA only, to build a system that effectively diacritizes MSA, CA, and and two varieties of Dialectal Arabic (DA), namely Moroccan (MA) and Tunisian (TN). Our system beats all previously reported SOTA results for the aforementioned varieties of Arabic. The underlying approach treats diacritic recovery as a translation problem, where a sequential encoder and a sequential decoder are employed with undiacritized characters as input and diacritized characters as output. The system is composed of four main componenets, namely: 1) a web application that efficiently handles concurrent user diacritization requests; 2) a text tokenization and cleaning module based on Farasa (Abdelali et al., 2016), a SOTA Arabic NLP toolkit; 3) Arabic variety identifier based on a fastText (Joulin et al., 2016), a deep learning classification toolkit, to properly ascertain the appropriate diacritization model; and 4) a Neural Machine Translation (NMT) based architecture, based on OpenNMT (Klein et al., 2017), to translate sequences of undiacritized characters to diacritized sequences.

The contributions in this paper are:

- We deploy a web-based system that diacritizes four varieties of Arabic (MSA, CA, DA-MA, and DA-TN) with appropriate RESTful API.
- We employ one architecture to effectively diacritize the different varieties. The model re-

[1] Buckwalter Arabic transliteration scheme is used throughout the paper.

Proceedings of the 2019 EMNLP and the 9th IJCNLP (System Demonstrations), pages 217–222
Hong Kong, China, November 3 – 7, 2019. ©2019 Association for Computational Linguistics

quires no feature engineering or external resources such as segmenters or POS taggers. Our system surpasses SOTA results for all varieties of Arabic that we test on.

- We created large training and test datasets for CA that are highly consistent. We plan to make the test set, which is composed of 5,000 sentences (400k words) publicly available.

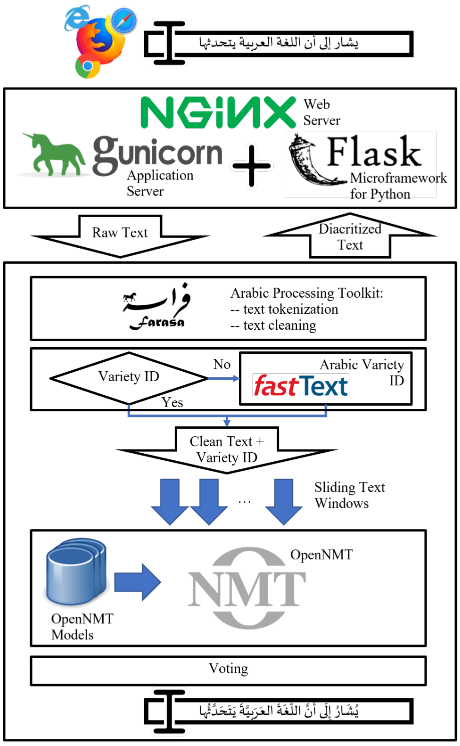


Figure 1: System architecture

2 System Architecture

2.1 Web Application

Figure 1 showcases the component of our demo. In a web browser, the user submits a sentence to diacritize. An instance of NGiNX web server receives the user request and routes it to the underlying application server. The application server is composed of two parts, namely “Flask” and “GUnicorn” (a.k.a. Green Unicorn). Flask is a web framework that deploys Python-based web application. Due to Flask’s load restriction, where it does not handle concurrent requests and lacks security, we use GUnicorn, which is a web server gateway interface, as an interface between NGINX and Flask to handle concurrent requests and to properly handle security issues. Gunicorn converts NGINX requests into Python objects, which are usable by the Flask frameworks. We make the diacritizer available as a web application and a web service that are freely accessible at: https://bit.ly/2IdFRVE.

2.2 Arabic Preprocessing

Flask deploys our diacritization application, which is composed of multiple components. The first component is Farasa segmenter (Abdelali et al., 2016), which is a SOTA publicly available Arabic processing toolkit. From the Farasa utilities, we use tokenization and text cleaning. Tokenization processes the text stream character by character to produce individual tokens composed of letters and numbers. Normalization maps Hindi numbers to Arabic numbers, converts non-Arabic extended Arabic script letters, such as those from Farsi or Urdu, to the closest Arabic letters, and removes all non-Arabic letters and numbers.

2.3 Arabic Variety Identification

The user may explicitly specify the variety of Arabic that they have entered. Available options include MSA, CA, DA-MA, and DA-TN. If the user does not specify the variety of Arabic, we employ a variety ID classifier. For this, we use a character-based deep-leaning classifier using fastText (Joulin et al., 2016) with character segments ranging between 3 and 6 grams, a learning rate of 0.05, and 50 training epochs. we opted to use characters for classification instead of words because Arabic is a complex language with a rich morphology, and many prefixes and suffixes can be attached to words. Also for dialectal Arabic, words can be written in many different accepted ways due to the lack of a standard orthography.

2.4 Diacritization

For diacritization, Mubarak et al. (2019) show the effectiveness of using Neural Machine Translation (NMT) framework to properly diacritize MSA while recovering both core-word diacritics and case-endings jointly and without the need for any feature engineering. We use their approach to train diacritizers for MSA, and we extend their work to train diacritizers for CA, DA-MA, and DA-TN. The method is composed of three component:
The first component produces overlapping **sliding window** sequences of n words. Diacritization requires that word and character orderings are preserved. Thus, the NMT model needs to be constrained to avoid word and character re-ordering,

insertion, and deletions. As in Mubarak et al. (2019), we enforce these constraints using a sliding window strategy, where the model is trained and tested on consecutive text windows of fixed length of 7 words.

The second component is an **NMT model**, which translates undiacritized sequences of characters to diacritized sequences. We use the OpenNMT implementation. Formally, given a word sequence w_i in source sentence S_{src}, we want to map it to the diacritized word w'_i in target sentence S_{trg} such that:

$$\begin{aligned} S_{src} &= w_0, w_1, w_2, ..., w_n \\ S_{trg} &= w'_0, w'_1, w'_2, ..., w'_n \end{aligned} \qquad (1)$$

and:

$$\begin{aligned} &w_i = c_0, c_1, .., c_m \;\; c \in \{C, L\}. \\ &\quad C: Arabic\ characters \\ &w'_i = c_0v_0, c_1v_1, .., c_mv_m \;\; v \in \{V, \emptyset\} \\ &\quad V: Arabic\ diacritics \end{aligned} \qquad (2)$$

The third is a **voting component**. Since a word may appear in multiple consecutive windows, we get multiple diacritized versions of every word. This allows us to use voting to select the most common diacritized form for a word. In case of a tie, we favor the window in which the word appears exactly in the middle. Table 1 provides an example for a three words sentence “ktb Alwld Aldrs” (the boy wrote the lesson) with a 3 word sliding window.

Source	Target
<s>_<s>_k t b	<s>_<s>_ka ta ba
<s>_k t b _A l w l d	<s>_ka ta ba _A lo wa la du
k t b _A l w l d _A l d r s	ka ta ba _A lo wa la du _ A l d∼a ro sa
A l w l d _A l d r s _<e>	A lo wa la du _A l d∼a ro sa _<e>
A l d r s _<e>_<e>	A l d∼a ro sa _<e>_<e>

Table 1: Example sentence: “ktb Alwld Aldrs” with context window size of 3. Symbols “<s>” and “<e>” are added to mark start and end of the sentence.

3 Data and Training

For **MSA**, we used a diacritized corpus of 4.5m words for training (Darwish et al., 2017; Mubarak et al., 2019). This corpus covers different genres such as politics, economy, religion, sports, society, etc. And for testing, we used the freely available WikiNews corpus (Darwish et al., 2017) which contains 18.3k words and covers multiple genres.

For **CA**, we obtained a classical diacritized corpus of 65m words from a publisher. We used 5k random sentences (400k words) for testing, and we used the remaining words for training. We are making the test set available at: https://bit.ly/2KuOvkN.

For **DA**, we used the corpora described in (Darwish et al., 2018), which is composed of two diacritized translations of the New Testament into Moroccan (DA-MA) and Tunisian (DA-TN). These corpora contain 166k and 157k words respectively. For each dialect, we split the diacritized corpora into 70/10/20 for training/validation/testing splits respectively. Our splits exactly match those of Abdelali et al. (2018). We used 5-fold cross validation.

Table 3 lists the details of the training and test sets including the unique diacritized and undiacritized tokens and the percentage of OOVs in the test set that don’t appear in the training set. For MSA and CA, we randomly used 10% of the training set for validation and the rest for training.

3.1 Training

For variety identification, given the diacritization training sets, we trained the classifier using 7,000 random sentences from each corpus and we used 1,000 sentences for testing. As the testing sequences increase in length (more input words), the accuracy of the classifier increases as in Table 2.

Input length	5	10	15	20
Accuracy	93.8	98.5	99.0	99.1

Table 2: Arabic variety identification per input length

When building the diacritization models, we used the OpenNMT-tf implementation for training with the hyperparameters suggested in the OpenNMT website[2]. We used two RNN layers of size 500 each and embeddings of size 300. We ran 1M training epochs for each system, which took on average 8 to 24 hours per system.

4 Evaluation and Analysis

For evaluation, we report on the results of evaluating the models as well as the performance of the deployed system.

4.1 System Results

Table 4 summarizes the results per dataset and compares our system to other SOTA systems on

[2] https://github.com/OpenNMT/OpenNMT-tf

		Train		Test		
	Word	Total	Uniq	Total	Uniq	OOV%
MSA	Diac.	4.5m	333k	18.3k	7.9k	5.0
	Undiac.		209k		6.8k	3.3
CA	Diac.	65.6m	489k	409k	39k	3.6
	Undiac.		254k		29k	2.3
DA-MA	Diac.	151k	16.3k	15.4k	4.1k	19.5
	Undiac.		15.9k		4.0k	19.0
DA-TN	Diac.	142k	17.2k	15.3k	4.4k	21.5
	Undiac.		16.6k		4.3k	20.7

Table 3: Number of words in training and test data for MSA, CA, and DA

identical test sets. WER is computed at word-level, and hence the whole word is counted as an error if a single character therein receives an incorrect diacritic. Since we did not have access to systems that are specially tuned for CA, we compared our system to Farasa (Darwish et al., 2017), which was tuned for MSA. As the results clearly show, using the NMT model at character level consistently produced better results than all SOTA systems. This confirms the previous conclusions of (Mubarak et al., 2019) about the superiority of using a character based seq2seq model for diacritization. While previously published results using DNN BiLSTM approaches have improved the results over other machine learning approaches; NMT invariably reduced the errors further – between 25% to 60%.

	Setup	WER%
MSA	**Our System**	**4.5**
	Microsoft ATKS (Said et al., 2013)	12.3
	Farasa (Darwish et al., 2017)	12.8
	RDI (Rashwan et al., 2015)	16.0
	MADAMIRA (Pasha et al., 2014)	19.0
	MIT (Belinkov and Glass, 2015)	30.5
CA	**Our System**	**3.7**
	Farasa (Darwish et al., 2017)	12.8
DA-MA	**Our System**	**1.4**
	Bi-LSTM DNN (Abdelali et al., 2018)	2.7
	CRF (Darwish et al., 2018)	2.9
DA-TN	**Our System**	**2.5**
	Bi-LSTM DNN (Abdelali et al., 2018)	3.6
	CRF (Darwish et al., 2018)	3.8

Table 4: Results and comparison of full diacritization systems.

4.2 System Performance

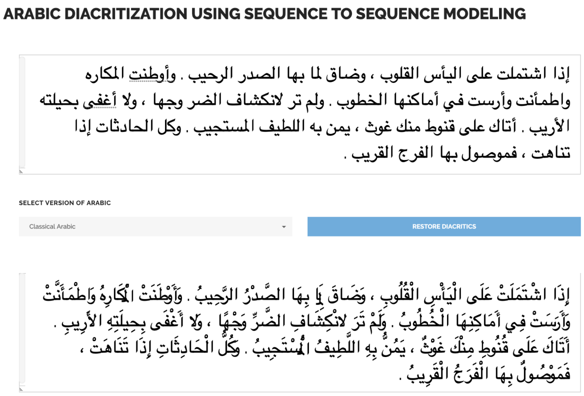
The system is running on a Microsoft Azure virtual machine with 4 CPU cores and 16 gigabytes of memory. The system does not require a GPU for decoding. We configured the application server to handle 100 concurrent requests, with each request containing a text sentence. In our testing, the system is able to process 10,000 requests in 14.9 seconds with zero failures. In other words, the server is able to handle 672 requests per second, with each request finishing in 149 milliseconds on average. Implementing memory mapped files to minimize disk access would further speed up the processing of requests and enhance the performance of the system.

4.3 Error Analysis

For **MSA**: we randomly selected 100 word-core and 100 case-ending errors to ascertain the most common error types. For case-endings, the top 4 error types were: long-distance dependency (e.g. coordination or verb subj/obj), which is an artifact of using limited context – 24% of errors; confusion between different syntactic functions (e.g. N N vs. N ADJ or V Subj vs. V Obj) – 22%; wrong selection of morphological analysis (e.g. present tense vs. past tense) – 20%; and named entities (NEs) – 16%. For long distance dependencies, increasing context size may help in some case, but may introduce additional errors. Perhaps combining multiple context sizes may help. As for word-cores, the top 4 errors were: incorrect selection for ambiguous words, where most of these errors were related to active vs. passive voice – 60%; NEs – 32%; borrowed words – 4%; and words with multiple valid diacritized words – 4%.

For **CA**: We randomly selected 100 errors and the top 4 error types, which summed up to 85% of errors, were: wrong diacritized form selection – 36% of errors (e.g. "Almalik" (the king) vs. "Almalak" (the angel)); long-distance dependency (e.g. coordination or verb subj/obj as in "jA' Alnby mlk", (an angel came to the prophet) where the object preceded the subject) – 32%; confusion between different syntactic functions (e.g. N N vs. N ADJ) – 9%; confusion between different suffixes or prefixes, (e.g. "katabta" (you wrote) vs. "katabat" (she wrote)) – 8%. We also found that in 7% of the differences, the diacritizations of both the reference and system output were in fact correct (e.g. "jinAzp" and "janAzp" (funeral)).

For **DA**: We manually inspected 100 random errors. The bulk of these errors (71%) came from confusing the vowels a, i, and u, with sukun or sukun-shaddah. This is normal due to the high fre-



ARABIC DIACRITIZATION USING SEQUENCE TO SEQUENCE MODELING

إذا اشتملت على اليأس القلوب ، وضاق لما بها الصدر الرحيب . وأوطنت المكاره
واطمأنت وأرست في أماكنها الخطوب . ولم تر لانكشاف الضر وجها ، ولا أغفى بحيلته
الأريب . أتاك على قنوط منك غوث ، يمن به اللطيف المستجيب . وكل الحادثات إذا
تناهت ، فموصول بها الفرج القريب .

إِذَا اشْتَمَلَتْ عَلَى الْيَأْسِ الْقُلُوبِ ، وَضَاقَ لِمَا بِهَا الصَّدْرُ الرَّحِيبُ . وَأَوْطَنَتْ الْمَكَارِهُ وَاطْمَأَنَّتْ
وَأَرَسَتْ فِي أَمَاكِنِهَا الْخُطُوبُ . وَلَمْ تَرَ لانْكِشَافِ الضَّرِّ وَجْهًا ، وَلا أَغْفَى بِحِيلَتِهِ الأَرِيبِ .
أَتَاكَ عَلَى قُنُوطِ مِنْكَ غَوْثٌ ، يَمُنُّ بِهِ اللَّطِيفُ الْمُسْتَجِيبُ . وَكُلُّ الْحَادِثَاتِ إِذَا تَنَاهَتْ ،
فَمَوْصُولٌ بِهَا الْفَرَجُ الْقَرِيبُ .

Figure 2: Screenshot for the demo site https://bit.ly/2IdFRVE

quency of sukkun in both Moroccan and Tunisian dialects compared to other varieties of Arabic (Abdelali et al., 2018). In about 8% of the cases, which involved NEs, the gold reference was not consistently diacritized, making it difficult for the system to learn properly. In an additional 5%, the system hallucinated producing output that is completely untethered from the source material (e.g. “$ayaTino” (the devil) vs. “$iTnino” (non-word)). These type of errors are likely an artifact of the seq2seq model. We plan to handle these errors in future by aligning input and output words and make sure that output letters are exactly the same as input, and by handling special characters that sometimes appear in DA. The remaining errors (16%) were due to erroneous selections, where for example, the system confused between feminine and masculine pronouns (e.g. “noti” (you: Fem.) vs. “nota” (you: masc.)).

5 Related Work

While the bulk of Arabic diacritic recovery research was devoted to MSA diacritization, work on CA and DA is still scarce. This can be attributed to many factors, primarily the absence of large standard resources.

In early experiments on **CA** diacritization by Gal (2002) and Elshafei et al. (2006), they used the Qur’anic text, composed of 18,623 diacritized words, and a Hidden Markov Model. Gal (2002) reported a word error rate (WER) of 14%, and Elshafei et al. (2006) reported a character error rate of 2.5%. As for **DA**, a number of systems designed either solely for diacritization or that supports the diacritization among other functionalities. Linguistic Data Consortium (LDC) CallHome corpus, containing 160K words worth of transcripts of informal Egyptian Arabic was among the earlier resources used for DA. Employed technologies varied from manually crafted rules (Vergyri and Kirchhoff, 2004), finite state transducer and support vector machine (Habash et al., 2012; Khalifa et al., 2017; Jarrar et al., 2017), Conditional Random Fields (Darwish et al., 2018), and Deep Neural Networks (Abdelali et al., 2018). While there is no standard dataset for evaluation, recent reported performance on Moroccan and Tunisian was a WER of 2.7% and 3.6% respectively (Abdelali et al., 2018).

Similarly, for **MSA**, LDC Arabic Treebank (Part 2)[3] and its successor Part 3 (ATB3) v 1.0 and 3.2 were used with a myriad of technologies (Habash and Rambow, 2007; Pasha et al., 2014; Abandah et al., 2015) that combines SVM classifier and Recurrent Neural Networks. Abandah et al. (2015) reported a WER of 9.07% using a neural architecture consisting of two Recurrent Neural network layers with 250 nodes each. Darwish et al. (2017) used a corpus of 4.5m fully diacritized words to train an SVM classifier that achieved a 12.76% WER.

In sum, three resources were explored for diacritics recovery, namely:
– For CA: Qura’nic text with 18k words.
– For DA: LDC CallHome with 160k words; the Moroccan and Tunisian Bibles with 166k and 157k words respectively.
– For MSA: LDC Arabic Treebank with 340k words (v3.2); and a proprietary corpus of 4.5m words (Darwish et al., 2017).

6 Conclusion

In this paper, we introduced a system for diacritizing four different varieties of Arabic, namely MSA, CA, DA-TN, and DA-MA. The system employs a character based seq2seq model without the need for any feature engineering while beating other SOTA systems. The system is deployed as a web application with corresponding RESTful API. Our WER results respectively for MSA, CA, DA-MA, and DA-Tunisian are: 4.5%, 3.7%, 1.4%, and 2.5%. We plan to extend the system by integrating diacritization models for other dialects and to make the system more robust to handle different kinds of input texts with special characters, which are prevalent in tweets.

[3] https://catalog.ldc.upenn.edu/LDC2004T02

References

Gheith A. Abandah, Alex Graves, Balkees Al-Shagoor, Alaa Arabiyat, Fuad Jamour, and Majid Al-Taee. 2015. Automatic diacritization of arabic text using recurrent neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(2):183–197.

Ahmed Abdelali, Mohammed Attia, Younes Samih, Kareem Darwish, and Hamdy Mubarak. 2018. Diacritization of maghrebi arabic sub-dialects. *arXiv preprint arXiv:1810.06619*.

Ahmed Abdelali, Kareem Darwish, Nadir Durrani, and Hamdy Mubarak. 2016. Farasa: A fast and furious segmenter for arabic. In *Proceedings of NAACL-HLT 2016 (Demonstrations)*, pages 11–16. Association for Computational Linguistics.

Ibrahim A. Asadi. 2017. Reading arabic with the diacritics for short vowels: vowelised but not necessarily easy to read. *Writing Systems Research*, 9(2):137–147.

Yonatan Belinkov and James Glass. 2015. Arabic diacritization with recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2281–2285, Lisbon, Portugal.

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive Exploration of Neural Machine Translation Architectures. *ArXiv e-prints*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Kareem Darwish, Ahmed Abdelali, Hamdy Mubarak, Younes Samih, and Mohammed Attia. 2018. Diacritization of moroccan and tunisian arabic dialects: A CRF approach. In *OSACT 3: The 3rd Workshop on Open-Source Arabic Corpora and Processing Tools*, page 62.

Kareem Darwish, Hamdy Mubarak, and Ahmed Abdelali. 2017. Arabic diacritization: Stats, rules, and hacks. In *Proceedings of the Third Arabic Natural Language Processing Workshop*, pages 9–17.

Moustafa Elshafei, Husni Al-Muhtaseb, and Mansour Alghamdi. 2006. Statistical methods for automatic diacritization of arabic text. In *The Saudi 18th National Computer Conference. Riyadh*, volume 18, pages 301–306.

Ya'akov Gal. 2002. An HMM approach to vowel restoration in arabic and hebrew. In *Proceedings of the ACL-02 workshop on Computational approaches to Semitic languages*, pages 1–7. Association for Computational Linguistics.

Nizar Habash, Ramy Eskander, and Abdelati Hawwari. 2012. A morphological analyzer for egyptian arabic. In *SIGMORPHON 2012*, pages 1–9. ACL.

Nizar Habash and Owen Rambow. 2007. Arabic diacritization through full morphological tagging. In *HLT-NAACL*, pages 53–56.

Mustafa Jarrar, Nizar Habash, Faeq Alrimawi, Diyam Akra, and Nasser Zalmout. 2017. Curras: an annotated corpus for the palestinian arabic dialect. *Language Resources and Evaluation*, 51(3):745–775.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Salam Khalifa, Sara Hassan, and Nizar Habash. 2017. A morphological analyzer for gulf arabic verbs. *WANLP 2017*, page 35.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*.

Oleksii Kuchaiev, Boris Ginsburg, Igor Gitman, Vitaly Lavrukhin, Carl Case, and Paulius Micikevicius. 2018. Openseq2seq: extensible toolkit for distributed and mixed precision training of sequence-to-sequence models. *arXiv preprint arXiv:1805.10387*.

Hamdy Mubarak, Ahmed Abdelali, Hassan Sajjad, Younes Samih, and Kareem Darwish. 2019. Highly effective arabic diacritization using sequence to sequence modeling. In *Proceedings of NAACL-HLT 2019*.

Arfath Pasha, Mohamed Al-Badrashiny, Mona Diab, Ahmed El Kholy, Ramy Eskander, Nizar Habash, Manoj Pooleery, Owen Rambow, and Ryan M Roth. 2014. Madamira: A fast, comprehensive tool for morphological analysis and disambiguation of arabic. In *LREC-2014*, Reykjavik, Iceland.

Mohsen Rashwan, Ahmad Al Sallab, M. Raafat, and Ahmed Rafea. 2015. Deep learning framework with confused sub-set resolution architecture for automatic arabic diacritization. In *IEEE Transactions on Audio, Speech, and Language Processing*, pages 505–516.

Ahmed Said, Mohamed El-Sharqwi, Achraf Chalabi, and Eslam Kamal. 2013. A hybrid approach for arabic diacritization. In *Natural Language Processing and Information Systems*, pages 53–64, Berlin, Heidelberg.

Youssef Sherif. 2018. Arabic tacotron text to speech.

Dimitra Vergyri and Katrin Kirchhoff. 2004. Automatic diacritization of arabic for acoustic modeling in speech recognition. In *Proceedings of the workshop on computational approaches to Arabic script-based languages, COLING'04*, pages 66–73, Geneva, Switzerland.