Neural diacritization

1 Introduction

Many writing systems, both ancient and modern, systematically omit key information about the pronunciation—and ultimately, the meaning—of words. Such writing systems are sometimes referred to as **defective**. In Arabic, for example, consonants and long vowels are written, but except for pedagogical and religious materials, there is no way to indicate the presence of quality of short vowels. Thus the word اليمن [ʔal.ja.man] 'Yemen' is written with five consonant characters ⟨'-l-y-m-n⟩ but no vowels. Similarly, in Latin, vowel length is contrastive—e.g., *malus* 'bad' vs. *mālus* 'apple tree'—but both words are written ⟨MALVS⟩ in Classical texts. Closely related issues arise in the writing systems used for Hebrew, Hindi, French, and Punjabi, and Spanish, among others [1–3]. Predicting or "filling in" the short vowels is not difficult for highly-literate readers of Arabic, nor is predicting vowel length difficult for sophisticated scholars of Latin, but defective scripts are quite difficult to read for young children and second language learners. Furthermore, defective scripts are problematic for many speech and language processing technologies: they make mapping from spelling to pronunciation—essential for speech technologies like automatic speech recognition [4, 5]—much more challenging, and they adversely impact the performance of text processing and understanding systems [6, 7]. Their use of defective scripts have been cited as a major reason that speech and language processing results in Arabic and Hebrew lag behind those of similarly-resourced languages [8, 9].

Diacritizers use machine learning to map defective scripts by inserting **diacritics** that are not normally written, producing what is known as **plene** ('full') text. Examples of defective and plene Arabic and Latin are shown in Table 1 and Table 2, respectively. Diacritizers must incorporate local semantic information and deal with extensive ambiguity; in some extreme cases, there may be more than a dozen different ways to diacriticize an Arabic word [10], making this task particularly challenging.

Diacritization, particularly of Arabic and Hebrew, has been studied for several decades, but prior work—with a few exceptions—uses machine learning techniques such as decision trees or linear classifiers [1, 2, 11, 12], methods that are largely outclassed by the advent of modern neural networks. In this proposal we will implement and compare two neural network approaches to diacritization (section 2). We select Modern Standard Arabic and Classical Latin for the proposed study for several reasons. First, the two languages have rather different diacritic systems: Arabic diacritics attach to consonantal characters, whereas Latin diacritics attach exclusively to vocalic characters (section 3). Secondly, there is plentiful data available for diacritization experiments in these two languages (section 4). Third, these languages suggest intriguing future applications for the proposed technologies (section 5).

2 Models

We propose to implement and evaluate two neural models for diacritization.

(a)	والحقوق الكرامة في متساوين أحرارًا الناس جميع يولد
(b)	وَٱلْحُقُوقِ ٱلْكَرَامَةِ فِي مُتَسَاوِينَ أَحْرَارًا ٱلنَّاسِ جَمِيعُ يُولَدُ
(c)	‘All people are born free and equal in dignity and rights’

Table 1: Sample Modern Standard Arabic text (from the Universal Declaration of Human Rights); (a): defective Arabic text; (b): plene Arabic text; (c): English gloss.

(a)	…TROIAE QVI PRIMVS AB ORIS ITALIAM FATO PROFVGVS LAVINIAQVE VENIT LITORA…
(b)	*…trojae quī prīmus ab ōris ītaliam fātō profugus lāvīniaque vēnit lītora…*
(c)	‘…first driven by fate from the coast of Troy to Italy and the Lavinian coast…’

Table 2: Sample Classical Latin text (Verg., *Aen.* 1.1–3); (a): text as it is appears in the oldest extant manuscripts; (b): diacriticized text from Pharr’s Vergil reader [13]; (c): English gloss.

Neural lexicalized diacritization Diacritization can be treated as a token tagging problem such that the tagger output can be used to map defective tokens to contextually-appropriate plene tokens. Such methods are termed **lexical** when they make use of a morphological analyzer with diacritic information, or a diacriticized lexical list, to generate candidate plene tokens. In one common lexical approach, an analyzer or lexicon is used to generate a lattice of candidate diacritizations of the token sequence, then the best path is selected using a series of tagger and/or ranker models [7, 14, 15]. We propose a simple alternative which represents a more straightforward approach to lexicalized diacritization. Straka, Straková, and Hajič [16] use BERT [17] neural taggers to predict part-of-speech (POS) tags, fine-grained morphological feature bundles, and finally lemmas. At each step, later predictions are conditioned on earlier predictions. The resulting model represents the state of the art in sentence-level morphological analysis [18]. To adapt this model for diacritization, we will simply add a fourth prediction step, in which BERT embeddings, the predicted POS tags, morphological features, and lemmas are together used to predict the plene tokens subject to the constraints of a lemmatized lexicon.

Neural character tagger Alternatively, diacritization can be treated as a task in which each eligible character in the defective text is tagged with its diacritics. Such methods are termed **non-lexical** or **character-based** in that they do not depend on an analyzer or lexicon and therefore may outperform lexical methods on out-of-vocabulary words. Following prior work on Hebrew [3], we will use a multi-layer character-based bidirectional LSTM and a greedy decoder to tag each eligible character with the appropriate diacritics.

3 Modeling Details

Arabic Modern Standard Arabic is written with an **abjad**, a consonantal alphabet which omits short vowels. In plene Arabic, four diacritics known as *ḥarakāt* indicate which short vowel follows the diacriticized consonant. The three *tanwīn* diacritics are used only at the end of words to indicate the presence of a word-final short vowel followed by an [n]. The *sukūn* indicates that a consonant is not followed by a vowel. The *shaddah* indicates that a consonant is geminate (i.e., doubled). Some types of diacritic (e.g., *shaddah* and *ḥarakat*) may co-occur on a single conso-

nant. Following Gershuni and Pinter [3], in neural character model, we will predict vowel-related diacritics—*ḥarakāt*, *tanwīn*, and the *sukūn*—and the *shaddah* gemination diacritic using shared encoder layers but two separate softmax layers; this reduces a 14-class multinomial classification problem to two simpler, partly-independent classification tasks. We will also mask forbidden diacritizations, such as the prediction of *tanwīn* in non-final position, during decoding.

Latin Classic Latin has phonemic contrasts between short and long monophthongs and between the high vowels [i, u] and glides [j, w] [19], though these distinctions are not indicated in the standard defective orthography used until the 20th century. In Latin pedagogy, *macrons*, horizontal bars written above the vowel, are used to distinguish long monophthongs from short monophthongs. Some pedagogical texts also place a *diaeresis* over certain short monophthongs in hiatus to indicate that they are to be read as a separate vowel rather than as part of a diphthong. In addition to these two conventions, following Winge [7] we will also use *j* and *v*, respectively, to distinguish glides from high vowels. This is not a matter of diacritization under the strictest definition, though it provides essential information for the study of scansion. In total, there are between two and four possible ways—plain, macron, diaeresis, glide—to diacritize a Latin vowel character, depending on the vowel and its position; unlike Arabic, none of these diacritics co-occur. Once again, we will use masking to prevent the model from entertaining forbidden diacritizations, such as diaeresis when not preceded by another vowel, during decoding.

4 Data

For both languages we will make use of pre-existing resources for diacritization, allowing us to compare our results to those of previously published work.

Arabic Following Belinkov and Glass [10] and Zalmout and Habash [15], we will use the Penn Arabic Treebank [20], which includes nearly 700,000 tokens of plene text from Arabic-language newswire services. This resource is proprietary, but the Graduate Center already holds a license.

Latin Following Winge [7], we will use freely-available Classical Latin texts from the Latin Dependency Treebank and PROIEL [21, 22]. We will also make use of a novel corpus of 30,000 words of diacriticized Latin poetry [13] digitized by the PI. These three resources together comprise over 140,000 tokens of plene text.

5 Significance and Outcomes

To our knowledge, the proposed work represents the first attempt to apply neural diacritization methods to Latin, and the first to apply a purely character-level neural diacritization to Arabic, and we hypothesize that the results will represent a new state of the art for diacritization in the two targeted languages. Once experiments are complete, an article will be submitted to the journal *Transactions of the Association for Computational Linguistics*, and all code will be released under a permissive open-source license.

Results from this study will be used as pilot data to support an application for a NSF-BSF Joint Funding Research Grant in collaboration with Prof. Yuval Pinter (Ben-Gurion University of the Negev); the proposed grant will expand the evaluation to Modern Hebrew and several other languages, and experiment with additional neural network techniques.

The Arabic diacritizer will be used to pilot diacritization of North African dialects of Arabic text using data derived from **Arabizi**, a widely used but non-standardized romanization of Arabic dialects [23]. Despite the lack of convention, short vowels are usually written in Arabizi, allowing us to mine data to train North African Arabic diacritizers simply by mapping Arabizi (e.g., *sa7el* 'coast') to diacriticized (سَاحِل) and undiacritized (ساحل) Arabic text [24].

The Latin diacritizer will be integrated into a web-based Latin scansion tool. This scansion tool allows students to visualize the rhythm of lines of Latin poetry. Users of this tool will no longer need to painstakingly diacriticize poetic text as a result of this integration.

6 Methods and Evaluation

Systems will be evaluated using per-token and per-character error rate. We will also compute per-token and per-character error rate specifically for out-of-vocabulary tokens (i.e., those not present in the training set) so as to better understand the effect of lexicalization. Model hyperparameters will be tuned random search to minimize error rate on a held-out validation set. Models will be compared statistically using Bayesian hypothesis testing [25], and manual error analyses will be conducted for both languages. All experiments and evaluations will be conducted using a GPU-enabled computing cluster located in the PI's lab. It is anticipated that data collection and software development will be completed by winter 2022, leaving the remaining six months of the grant for experimentation and manuscript preparation.

References

[1] D. Yarowsky. Decision lists for lexical ambiguity resolution: application to accent restoration in Spanish and French. In *32nd Annual Meeting of the Association for Computational Linguistics*. 1994. [2] A. Arora et al. Supervised grapheme-to-phoneme conversion of orthographic schwas in Hindi and Punjabi. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020. [3] E. Gershuni and Y. Pinter. Restoring Hebrew diacritics without a dictionary. ArXiv preprint arXiv:2105.05209. 2021. [4] K. Gorman et al. The SIGMORPHON 2020 shared task on multilingual grapheme-to-phoneme conversion. In *17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. 2020. [5] L. F. Ashby et al. Results of the second SIGMORPHON shared task on multilingual grapheme-to-phoneme conversion. In *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. 2021. [6] A. M. Azmi and R. S. Almajed. A survey of automatic Arabic diacritization techniques. *Natural Language Engineering* 21.3 (2013). [7] J. Winge. Automated annotation of Latin vowel length. Bachelor's thesis. Uppsala University, 2015. [8] N. Y. Habash. *Introduction to Arabic Natural Language Processing*. Morgan & Claypool, 2010. [9] R. Tsarfaty et al. What's wrong with Hebrew NLP? And how to make it right. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on*

Natural Language Processing: System Demonstrations. 2019. [10] Y. Belinkov and J. Glass. Arabic diacritization with recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2015. [11] D. Yarowsky. A comparison of corpus-based techniques for restoring accents in Spanish and French text. In *Natural Language Processing Using Very Large Corpora*. Ed. by S. Armstrong et al. Springer, 1999. [12] J. Francom and M. Hulden. Diacritic error detection and restoration via part-of-speech tags. In *Proceedings of the 6th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*. 2013. [13] C. Pharr. *Vergil's Aeneid Books I–VI*. D.C. Heath and Company, 1964. [14] A. Shmidman et al. Nakdan: professional Hebrew diacritizer. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2020. [15] N. Zalmout and N. Habash. Joint diacritization, lemmatization, normalization, and fine-grained morphological tagging. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020. [16] M. Straka et al. UDPipe at SIGMORPHON 2019: contextualized embeddings, regularization with morphological categories, corpora merging. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*. 2019. [17] J. Devlin et al. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019. [18] A. D. McCarthy et al. The SIGMORPHON 2019 shared task: morphological analysis in context and cross-lingual transfer for inflection. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*. 2919. [19] A. Cser. Diphthongs in the syllable structure of Latin. *Glotta* 75.3–4 (1999). [20] M. Maamouri et al. Arabic Treebank: Part 3 (full corpus) v 2.0 (MPG + Syntactic Analysis). LDC2005T20. 2005. [21] D. Bamman and G. Crane. The ancient Greek and Latin dependency treebanks. In *Language Technology for Cultural Heritage*. Ed. by C. Sporleder et al. Springer, 2011. [22] D. T. Haug and M. Jøhndal. Creating a parallel treebank of the old Indo-European Bible translations. In *Proceedings of the Language Technology for Cultural Heritage Data Workshop*. 2008. [23] D. Seddah et al. Building a user-generated content North-African Arabizi treebank: tackling hell. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020. [24] K. Darwish. Arabizi Detection and Conversion to Arabic. In *Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing*. 2014. [25] P. Szymański and K. Gorman. Is the best better? Bayesian statistical model comparison for natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. 2020.