Restoring Hebrew Diacritics Without a Dictionary

Anonymous ACL-IJCNLP submission

Abstract

We demonstrate that it is feasible to diacritize Hebrew script without any human-curated resources other than plain diacritized text. We present NAKDIMON, a two-layer character-level LSTM, that performs on par with much more complicated curation-dependent systems, across a diverse array of modern Hebrew sources.

1 Introduction

The vast majority of modern Hebrew texts are written in a letter-only version of the Hebrew script, one which omits the diacritics present in the full diacritized, or *dotted* variant.[1] Since most vowels are encoded via diacritics, the pronunciation of words in the text is left underspecified, and a considerable mass of tokens becomes ambiguous. This ambiguity forces readers and learners to infer the intended reading using syntactic and semantic context, as well as common sense (Bentin and Frost, 1987; Abu-Rabia, 2001). In NLP systems, recovering such signals is difficult, and indeed their performance on Hebrew tasks is adversely affected by the presence of undotted text (Shacham and Wintner, 2007; Goldberg and Elhadad, 2010; Tsarfaty et al., 2019). As an example, the sentence in Table 1 (a) will be resolved by a typical reader as (b) in most reasonable contexts, knowing that the word "softly" may characterize landings. In contrast, an automatic system processing Hebrew text may not be as sensitive to this kind of grammatical knowledge and instead interpret the undotted token as the more frequent word in (c), harming downstream performance.

One possible way to overcome this problem is by adding diacritics to undotted text, or *dotting*, implemented using data-driven algorithms trained on dotted text. Obtaining such data is not trivial, even given correct pronunciation: the standard Tiberian diacritic system contains several sets of identically-vocalized forms, so while most Hebrew speakers easily read dotted text, they are unable to produce it. Moreover, the process of manually adding diacritics in either handwritten script or through digital input devices is mechanically cumbersome. Thus, the overwhelming majority of modern Hebrew text is undotted, and manually dotting it requires expertise. The resulting scarcity of available dotted text in modern Hebrew contrasts with Biblical and Rabbinical texts which, while dotted, manifest a very different language register. This state of affairs allows individuals and companies to offer dotting as paid services, either by experts or automatically, e.g. the Morfix engine by Melingo.[2]

Existing computational approaches to dotting are manifested as complex, multi-resourced systems which perform morphological analysis on the undotted text and look undotted words up in handcrafted dictionaries as part of the dotting process. Dicta's Nakdan (Shmidman et al., 2020), the cur-

	המטוס נחת ברכות
(a)	hamatos naxat ????
	'The plane landed (unspecified)'
	הַמָּטוֹס נָחַת בְּרַכּוּת
(b)	hamatos naxat b-rakut
	'The plane landed softly'
	הַמָּטוֹס נָחַת בְּרָכוֹת
(c)	hamatos naxat braxot
	'The plane landed congratulations'

Table 1: An example of an undotted Hebrew text (a) (written right to left) which can be interpreted in at least two different ways (b,c), dotted and pronounced differently, but only (b) makes grammatical sense.

[1] Also known as *pointed* text, or via the Hebrew term for the diacritic marks, *nikkud/niqqud*.

[2] https://nakdan.morfix.co.il/

rent state-of-the-art, applies such methods in addition to applying multiple neural networks over different levels of the text, requiring manual annotation not only for dotting but also for morphology. Among the resources it uses are a diacritized corpus of 3M tokens and a POS-tagged corpus of 300K tokens. Training the model takes several weeks.[3]

In this work, we set out to simplify the dotting task as much as possible to standard modules. Our system, NAKDIMON, accepts the undotted character sequence as its input, consults no external resources or lexical components, and produces diacritics for each character, resulting in dotted text whose quality is comparable to that of the commercial Morfix, on both character-level and word-level accuracy. To our knowledge, this is the first attempt at a "light" model for Hebrew dotting since early HMM-based systems (Kontorovich, 2001; Gal, 2002). In experiments over existing and novel datasets, we show that our system is particularly useful in the main use case of modern dotting, which is to convey the desired pronunciation to a reader, and that the errors it makes should be more easily detectable by non-professionals than Dicta's.[4]

2 Dotting as Sequence Labeling

The input to the dotting task consists of a sequence of characters, each of which is assigned at most one of each of three separate diacritic categories: one category for the dot distinguishing *shin* (שׁ) from *sin* (שׂ), two consonants sharing a base character ש; another for the presence of *dagesh/mappiq*, a central dot affecting pronunciation of some consonants, e.g. פּ /p/ from פ /f/, but also present elsewhere; and one for all other diacritic marks, which mostly determine vocalization, e.g. דָ /da/ vs. דֶ /de/. Diacritics of different categories may co-occur on single letters, e.g. שָׁ, or may be absent altogether.

Full script Hebrew script written without intention of dotting typically employs a compensatory variant known colloquially as full script (*ktiv male*, כתיב מלא), which adds instances of the letters י and ו in some places where they can aid pronunciation, but are incompatible with the rules for dotted script. In our formulation of dotting as a sequence tagging problem, and in collecting our test set from raw text, these added letters may conflict with the dotting standard. For the sake of input integrity, and unlike some other systems, we opt not to remove these characters, but instead employ a dotting policy consistent with full script.

New test set Shmidman et al. (2020) provide a benchmark dataset for dotting modern Hebrew documents. However, it is relatively small and non-diverse: all 22 documents in the dataset originate in a single source, Hebrew Wikipedia articles. Therefore, we created a new test set[5] from a larger variety of texts, including high-quality Wikipedia articles and edited news stories, as well as user-generated blog posts. This set consists of ten documents from each of eleven sources (5x Dicta's test set), and totals 20,476 word tokens, roughly 3.5x Dicta's.

3 Character-LSTM Dotter

NAKDIMON embeds the input characters and passes them through a two-layer Bi-LSTM (Hochreiter and Schmidhuber, 1997). The LSTM output is fed into a single linear layer, which then feeds three linear layers, one for each diacritic category. Each character then receives a prediction for each category independently, and decoding is performed greedily with no additional search techniques.[6] In training, we sum the cross-entropy loss from all categories. Trivial decisions, such as the label for the *shin/sin* diacritic for any non-ש letter, are masked.

Training corpora Dotted modern Hebrew text is scarce, since speakers usually read and write undotted text, with the occasional diacritic for disambiguation when context does not suffice. As we are unaware of legally-obtainable dotted modern corpora, we use a combination of dotted pre-modern texts and semi-automatically dotted modern sources to train NAKDIMON. The PRE-MODERN portion is obtained from two main sources: A combination of late pre-modern text from Project Ben-Yehuda, mostly texts from the late 19th century and the early 20th century;[7] and rabbinical texts from the medieval period, the most important of which is Mishneh Torah.[8] This portion contains roughly 2.6 million Hebrew tokens,

[3] Private communication.

[4] The system is a available at www.nakdimon.org

[5] Provided as supplementary material.

[6] We experimented with a CRF layer in preliminary phases and did not find a substantial benefit.

[7] Obtained from the Ben-Yehuda Project https://benyehuda.org/.

[8] Obtained from Project Mamre https://www.mechon-mamre.org/.

	Genre	Sources	# Docs	# Tokens
	Wiki	Dicta test set	22	5,862
	News	Yanshuf	78	11,323
†	Literary	Books, forums	129	73,770
*	Official	gov.il	2	7,619
*	News / Mag	Online outlets	84	55,467
*	User-gen.	Blogs, forums	58	57,672
*	Wiki	he.wikipedia	40	62,723
	Total		413	274,436

Table 2: Data sources for our MODERN Hebrew training set. Rows marked with * were automatically dotted via the Dicta API and corrected manually. Rows with † were dotted at low quality, requiring manual correction. The rest were available with professional dotting.

most of which are dotted, with a varying level of accuracy, varying dotting styles, and varying degree of similarity to modern Hebrew.

The MODERN portion contains manually collected text in modern Hebrew, mostly from undotted sources, which we dot using Dicta and follow up by manually fixing errors, either using Dicta's API or via automated scripts which catch common mistakes. We use the same technique and style for dotting this corpus as we do for our test corpus (§2), but the documents were collected in different ways. We made an effort to collect a diverse set of sources: news, opinion columns, paragraphs from books, short stories, blog posts and forums expressing different domains and voices, Wikipedia articles, governmental publications, and more. As our dotting guidelines aim for readability and faithfulness to modern writing conventions, we were able to create a corpus large enough with limited expertise in dotting. Our modern corpus contains roughly 0.3 million Hebrew tokens, and is much more consistent and similar to the expectation of a native Hebrew speaker than the PRE-MODERN corpus. The sources and statistics of this dataset are presented in Table 2.

4 Experiments

In order to report compatible findings with those in previous work, we present results on both the Dicta test, adapted to full script, and on our new test set. We report the following metrics: **decision accuracy (DEC)**, which is computed over the entire set of individual possible decisions: *dagesh/mappiq* for letters that allow it, *sin/shin* dot for the letter שׂ, and all other diacritics for letters that allow them; **character accuracy (CHA)** is the portion of characters in the text that end up in their intended final form (which may be the result of two or three decisions, e.g. *dagesh* + vowel); **word accuracy (WOR)** is the portion of words with no diacritization mistakes; and **vocalization accuracy (VOC)** is the portion of words where any dotting errors do not cause incorrect pronunciation.[9]

We train NAKDIMON over PRE-MODERN followed by five epochs over the MODERN corpus. We pre-process the input by removing all but Hebrew characters, spaces and punctuation; digits are converted to a dedicated symbol, as are Latin characters. We strip all existing diacritic marks. We split the documents into chunks of at most 80 characters bounded at whitespace, ignoring sentence boundaries. We optimize using Adam (Kingma and Ba, 2014), and implement a cyclical learning rate schedule (Smith, 2017) which we found to be more useful than a constant learning rate. Based on tuning experiments (§4.3), we set the character embedding dimension and the LSTM's hidden dimension to $d = h = 400$, and apply a dropout rate of 0.1.

Following Shmidman et al. (2020), we compare against Dicta, Snopi,[10] and Morfix (Kamir et al., 2002).

4.1 Dicta Test Set

We present results for the Dicta test set in the left half of Table 3. In order to provide fair comparison and to preempt overfitting on this test data, we ran this test in a preliminary setup on a variant of NAKDIMON which was not tuned or otherwise unfairly trained. This system, which we call NAKDIMON$_0$, differs from our final variant in two main aspects: it is not trained on the Dicta portion of our training corpus (§3); and it employs a residual connection between the two character Bi-LSTM layers. Testing on the Dicta test set required some minimal evaluation adaptations resulting from encoding constraints,[11] and so we copy the results reported in the original paper as well as our replication.

We see that the untuned NAKDIMON$_0$ performs on par with the proprietary Morfix, which uses word-level dictionary data, and does not perform far below the complex, labor-intensive Dicta on the character level.

[9] These are: the *sin/shin* dot, vowel distinctions across the a/e/i/o/u/null sets, and *dagesh* in the פ/כ/ב characters. We do not distinguish between *kamatz gadol/kamatz katan*, and *schwa* is assumed to always be null.

[10] http://www.nakdan.com/Nakdan.aspx

[11] For example, we do not distinguish between *kamatz katan* and *kamatz gadol*.

System	Dicta – reported		Dicta – reproduced				New test set (§2)			
	CHA	WOR	DEC	CHA	WOR	VOC	DEC	CHA	WOR	VOC
Snopi	78.96	66.41	87.81	79.92	66.57	70.35	91.14	85.45	74.73	77.17
Morfix	90.32	80.90	94.91	91.29	82.24	86.48	97.25	95.35	89.43	91.64
Dicta	**95.12**	**88.23**	**97.48**	**95.67**	**89.25**	**91.03**	**98.94**	**98.23**	**95.83**	**95.93**
NAKDIMON$_0$			95.78	92.59	79.00	83.01	97.05	94.87	85.70	88.30
NAKDIMON							97.37	95.41	87.21	89.32

Table 3: Document-level macro % accuracy on the test set from Shmidman et al. (2020) and on our new test set.

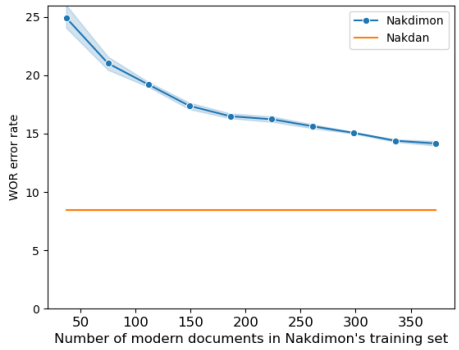

Figure 1: WOR error rate on validation set as a function of training set size vs. SOTA (Dicta), over five runs. Other metrics show similar trends.

4.2 New Test Set

We provide results on the new test set (§2) in the right half of Table 3. The improvement of our tuned NAKDIMON over NAKDIMON$_0$ is clear. NAKDIMON outperforms Morfix on character-level metrics but not on word-level metrics, mostly due to the fact that Morfix ignores certain words altogether, incurring errors on multiple characters. We note the substantial improvement our model achieves on the VOC metric compared to the WOR metric: 16.5% of word-level errors are attributable to vocalization-agnostic dotting, compared to only 2.4% for Dicta and 9.7% for Snopi (but 20.9% for Morfix). Considering that the central use case for dotting modern Hebrew text is to facilitate pronunciation to learners and for reading, and that undotted homograph ambiguity typically comes with pronunciation differences, we believe this measure to be no less important than WOR.

4.3 Development Experiments

While developing NAKDIMON, we performed several evaluations over a held-out validation set of 40 documents with 27,681 tokens, on which Dicta performs at 91.56% WOR accuracy. We show in Figure 1 the favorable effect of training NAKDIMON over an increasing amount of MODERN text (§3), which closes more than half of the error gap from Dicta. We tried to further improve NAKDIMON by initializing its parameters from a language model trained to predict masked characters in a large undotted Wikipedia corpus (440MB of text, 30% masks), but this setup provided only a negligible 0.07% improvement.

Error analysis A look at 20 cases which Dicta got right but NAKDIMON got wrong, and 20 of the opposite condition, reveals clear yet expected patterns: Dicta's errors mostly constitute selection of the wrong *in-vocabulary* word in a context, or a wrong inflection of a verb. NAKDIMON tends to create unreadable vocalization sequences, as it does not consult a dictionary and is decoded greedily. These types of errors are more friendly to the typical use cases of a dotting system, as they are likely to stand out to a reader. Other recurring error types include named entities and errors at sentence boundaries, which likely stem from lack of context.

5 Related Work

As noted in the introduction, works on diacritizing Hebrew are not common and all include word-level features. In Arabic, diacritization serves a comparable purpose to that in Hebrew but not exclusively: most diacritic marks differentiate letters from each other (which only the sin/shin dot does in Hebrew), while vocalization marks are in a one-to-one relationship with their phonetic realizations. Recent work in neural-based Arabic diacritization includes a dictionary-less system (Belinkov and Glass, 2015) which uses a 3-layer Bi-LSTM with a sliding window of size 5. Similarly, Abandah et al. (2015) also use a Bi-LSTM, but replace the sliding windows with a one-to-one architecture (one diacritic for each letter) and a one-to-many architecture (allowing any number of diacritics per character). The former, which performed best, is similar to this work except for our separation of diacritization categories. Finally, Mubarak et al. (2019)

has tackled Arabic diacritization as a sequence-to-sequence problem, tasking the model with reproducing the characters as well as the marks.

6 Conclusion

Learning directly from plain diacritized text can go a long way, even with relatively limited resources. NAKDIMON demonstrates that a simple architecture for diacritizing Hebrew text as a sequence tagging problem can achieve performance on par with much more complex systems. We also introduce and release a corpus of dotted Hebrew text, as well as a source-balanced test set. In the future, we wish to evaluate the utility of dotting as a feature for downstream tasks, taking advantage of the fact that our simplified model can be easily integrated in an end-to-end Hebrew processing system.

References

Gheith 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)*, 18:183–197.

Salim Abu-Rabia. 2001. The role of vowels in reading semitic scripts: Data from arabic and hebrew. *Reading and Writing*, 14(1-2):39–59.

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. Association for Computational Linguistics.

Shlomo Bentin and Ram Frost. 1987. Processing lexical ambiguity and visual word recognition in a deep orthography. *Memory & Cognition*, 15(1):13–23.

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*, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010. Easy-first dependency parsing of modern Hebrew. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 103–107, Los Angeles, CA, USA. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Dror Kamir, Naama Soreq, and Yoni Neeman. 2002. A comprehensive NLP system for modern standard Arabic and modern Hebrew. In *Proceedings of the ACL-02 Workshop on Computational Approaches to Semitic Languages*, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Leonid Kontorovich. 2001. Problems in semitic nlp: Hebrew vocalization using hmms.

Hamdy Mubarak, Ahmed Abdelali, Hassan Sajjad, Younes Samih, and Kareem Darwish. 2019. Highly effective Arabic diacritization using sequence to sequence modeling. 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)*, pages 2390–2395, Minneapolis, Minnesota. Association for Computational Linguistics.

Danny Shacham and Shuly Wintner. 2007. Morphological disambiguation of Hebrew: A case study in classifier combination. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 439–447, Prague, Czech Republic. Association for Computational Linguistics.

Avi Shmidman, Shaltiel Shmidman, Moshe Koppel, and Yoav Goldberg. 2020. Nakdan: Professional Hebrew diacritizer. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 197–203, Online. Association for Computational Linguistics.

Leslie N Smith. 2017. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE.

Reut Tsarfaty, Shoval Sadde, Stav Klein, and Amit Seker. 2019. 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 (EMNLP-IJCNLP): System Demonstrations*, pages 259–264, Hong Kong, China. Association for Computational Linguistics.