Hebrew Syllabification

Albert Shalumov
The Open University of Israel
Raanana, Israel
shalumov.albert@gmail.com

Kobi Bodek
The Open University of Israel
Raanana, Israel
kobibodek@gmail.com

Abstract

In this paper we explore several approaches to the task of Hebrew syllabification and phonetic pronunciation. We use classic methods (generative and discriminative) as well as Recurrent Neural Network (RNN) for sequence to sequence mapping. We compare and analyze the results, as well as suggest future research directions. The code is available at: https://github.com/albert-shalumov/nlp_proj

1 Introduction

Division to syllables always played a crucial role in learning to speak a language. People utilize it when pronouncing an unknown or difficult words (i.e. pseudopseudohypoparathyroidism) or learning to speak a new language. Machines require it in ”text-to-speech” software as part of Grapheme-to-Phoneme (G2P) module.
In Indo-European languages, such as English, the task mostly involves placing a syllable separator at the right position in a word. In Hebrew, however, vowel prediction is required, since modern text in Hebrew is written in deficient spelling (e.g. without vowels). The task is further complicated by frequent words having the same morphemes but different meaning and pronunciation depending on context of the sentence.
In Hebrew the number of syllables in a word is equal to the number of vowels. There are 2 types of syllables: open and closed. In an open syllable, the air flow at the end of the syllable continues, while in a closed syllable it is a stopping point. A syllable must include a single vowel and can have 1 to 3 consonants. Special case is for emphasis letters; they are treated as double letter, the first one is grouped at the end of the previous syllable while the second one is the opening letter for the next syllable.

2 Related Work

A number of studies and comparisons between automatic syllabification methods (Adsett and Marchand, 2009; Bartlett et al., 2009; Rogova et al., 2013; Ornan and Leket-Mor, 2016; Singh et al., 2016) were conducted. In case of Indo-European languages, such as English, German and Dutch, impressive results were achieved (Bartlett et al., 2009). For the Hebrew language, Ornan and Leket-Mor (Ornan and Leket-Mor, 2016) provided heuristics for romanization (e.g. transliteration in latin script). Their work starts with words written in full spelling, while modern Hebrew is written mostly in deficient spelling (e.g. without vowels). The most common method used is Conditional Random Fields (CRF), however Maximum Entropy Markov Models (MEMM) and heuristics were explored (Singh et al., 2016).

3 Method and Model

3.1 Method

3.1.1 General

Let $\mathcal{X}, \mathcal{Y}$ denote the model input and output. System architecture described in Figure 1.

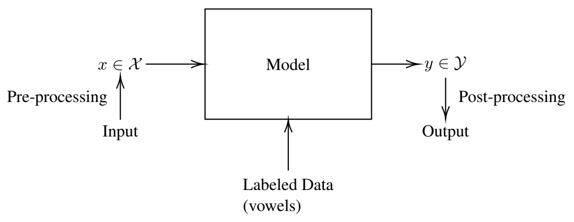

Figure 1: System diagram

Input is given in Hebrew and the output is an ordered set of the voweled word, voweled with syllable separation and romanized syllabled word.

Input	Model Input	Model Output	Output
התבררה	htbrrh	i*aea*	hit*barerah*
			hit*-ba-re-rah*
			it-ba-re-ra

Table 1: Input/Output example

Pre-processing includes conversion into transliterated format suggested by Ornan and Mor (Ornan and Leket-Mor, 2016), word extraction and model specific pre-processing. Post-processing includes syllabification and romanization. An example of inputs and outputs provided in Table 1.

We approach vowel prediction as a sequence-to-sequence task, where for each consonant we predict a vowel following it: {*e,u,i,o,a,**[1]} [2]. The division into syllables and romanization tasks described in the following sections.

In order to evaluate the quality of the results we use standard scores for multi-class classification - Accuracy, Fscore$_{\mu,M}$ (Sokolova and Lapalme, 2009) with $\beta = 1$. For the syllabification and romanization results, as well as vowel prediction we employ edit distance as a quality measure, more specifically Levenshtein distance.

3.1.2 Syllabification

We propose the following method for separating voweled word into syllables:

1. Find first actual vowel (not '*') and skip it
2. For every subsequent vowel: place syllable separator before the associated consonant[3]

For example: hit*barerah* $\rightarrow$ hit*-**b**a-**r**e-**r**ah*

This method generates correct number of syllables. One possible issue is the case of emphasis letter (dagesh), which should be treated as a double letter - one instance assigned to the end of the previous syllable and one to the beginning of the next. Since our main purpose it to provide pronunciation cues, we can ignore this case.

3.1.3 Romanization

First, we convert consonant literals in Ornan encoding to their phonetic equivalents: w$\rightarrow$v, ḥ$\rightarrow$h, ṭ$\rightarrow$t, ç$\rightarrow$ts, q$\rightarrow$k, š$\rightarrow$sh.

Next, we deduplicate "y" (yod) and "v" (vav) letters when for pronunciation one is enough (i.e. da-**yay***n $\rightarrow$ da-yan).

At this stage we remove "aleph", "heh" and "ayn" because their pronunciation is determined by the associated vowel (i.e. to-pa-a). Now we substitute the letters "bet", "kaf", "peh", "shin" that can have different pronunciations, with "b"/"v", "h"/"k", "p"/"f", "s"/"sh" depending on frequency of appearance (MLE).

3.2 Workflow

We initially divide the annotated data into two parts: 10% for final evaluation and 90% for training. During training and experiments phase we further divide the training data into training and validation: 90%, 10% respectively. We use the validation data as a stand-in for test data to find the best features and hyper-parameters. During the testing phase we use the entire training set to build the model and use the test set to evaluate its score. The exception to this, is the training of neural networks where we divide 95:5 and use validation set to determine over-fitting as well as for accuracy measure.

3.3 Models

The first task in the proposed pipeline is vowel prediction. The vowel selected for each consonant is one with the highest probability, more formally: let $\hat{y_i} \in \{e, u, i, o, a, *\}, x_i$ a consonant: $\hat{y_i} = \underset{y_i \in \{e,u,i,o,a,*\}}{argmax} \; P(y_i|x_1 x_2 \ldots x_n)$, where n is word length and i is the letter position in a word.

Since all the models are well known, we'll list modifications and hyper-parameters only, except in case of RNN, where we'll include architecture and character embedding methods as well.

3.3.1 Hidden Markov Model (HMM)

For this model the hyper-parameters are ngram size and smoothing techniques: Maximum Likelihood Estimate (MLE), Laplace, Good-Turing, Add-δ.

3.3.2 Maximum Entropy Markov Model (MEMM)

For this model the hyper-parameters are word features: Is first char, Is last char, Char. position, Char. value[4], Previous char. value, Next char.

[1] * signifies absence of a vowel

[2] Ornan and Mor's transliterated format doesn't contain those vowels letters, therefore adding them will not cause ambiguity

[3] Consonant directly before the vowel

[4] Refers to ordinal value of the character - $1 \ldots \#consonants$

value, First char. value, Last char. value, Second char. value, Second to last char. value, Word length]

3.3.3 Conditional Random Fields (CRF)

We test different features on both word level and sentence level. For both we use word features discovered in MEMM exploration and for sentence - we test additional features, such as: word position in a sentence, neighbouring words features.

3.3.4 Recurrent Neural Network (RNN)

We use a basic RNN encoder architecture (Figure 2), with various consonants embedding and cross-entropy loss for vowel classification.

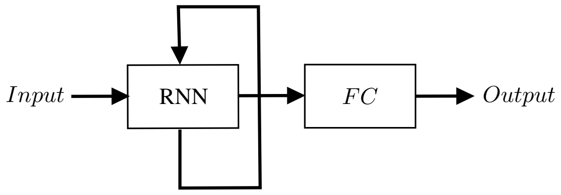

Figure 2: RNN setup

3.3.5 Consonant embedding

We test 3 different embedding methods:

1. One-hot encoding
   Let n_c be the number of consonants and $c \in \{1, \ldots, n_c\}$ the ordinal value of a consonant. One-hot encoding is a mapping:
   $f : \{1, \ldots, n_c\} \to \mathbb{R}^{n_c}$
   $f(k)_i = \begin{cases} 1 & i = k \\ 0 & otherwise \end{cases}$

2. Multi-dimensional scaling (MDS)
   We consider the following proposition: for vowel prediction, it's beneficial for embedded consonants to be "similar" to their associated vowel[5] probability distribution. One option is to represent them directly as the distribution in $\mathbb{R}^{n_v}$, where n_v number of vowels. This method is too limiting - it doesn't allow for use of higher or lower dimensions and it limits the L_2 norm to 1. Instead we use multi-dimensional scaling, preserving the distance between the consonant distributions for embedding as well. We use Earth Mover distance (or Wasserstein distance) to calculate distance between probability distributions. After calculating the $n_c \times n_c$ matrix of distances we can perform MDS and find a lower dimension representation.

3. Neural network encoder-decoder
   We use a simple, fully connected encoder-decoder with dual output (Figure 3). The loss comprised of cross-entropy loss on $\hat{x}$ (to reconstruct consonant correctly) and embedding distance.
   $D_e = \sum_{j=1}^{n_c} \sum_{i=1}^{n_c} \| \|x_{ld}^{(i)} - x_{ld}^{(j)}\| - EMD(x^{(i)}, x^{(j)}) \|$
   In simple words minimizing euclidean distance between embedding and Earth Mover Distance between the same consonants.

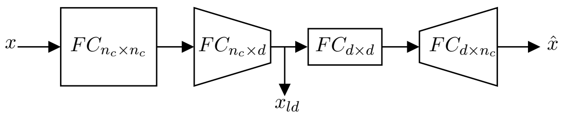

Figure 3: Embedding network architecture
Notation: x:one-hot consonant, x_{ld}: embedding, $\hat{x}$: consonant reconstruction

4 Data

We use HaAretz corpus[6] provided by the Knowledge Center for Processing Hebrew (MILA) (Itai and Wintner, 2008). We convert the original data into the following format:

- # sent_id = sentence index starting with 1
- # text = sentence in hebrew
- For each word in the sentence:
  Leading "#" if a word contains a symbol or a digit - will be skipped during processing.
  The following fields placed in the same line, seperated by spaces:
  word index starting with 1, word in hebrew, word in Ornan transliteration, word in Ornan transliteration [7].

After this stage, the annotation is performed. We extract each sentence block into to two separate files - one for training one for testing. We fill in the vowels, syllabification and romanization, resulting in the following data format:

- # sent_id = sentence index starting with 1
- # text = sentence in hebrew

[5]Vowel directly after the consonant

[6]Contains HaAretz news articles from 1990-1991

[7]This second word is a placeholder for annotator

- For each word in the sentence:
  Leading # if a word contains a symbol or a digit - will be skipped during processing.
  The following fields placed in the same line, seperated by spaces:
  word index starting with 1, word in hebrew, word in Ornan transliteration, word in Ornan transliteration with vowels and syllables marked, romanized word with syllables marked

5 Experiments

In the following sections we list the performed experiments and the most significant results [8]

5.1 Hidden Markov Model

We test the Cartesian product of ngram=$\{1 \dots 7\}$, smoothing=$\{MLE, Laplace, GoodTuring, Add-\delta : \delta \in \{0.1, 0.2, \dots 0.9\}\}$. See Table 2.

Experiment description	F_{μ}	F_{M}	Accuracy
ngram=3,$Add-\delta : 0.3$	**0.62**	**0.49**	**0.87**
ngram=3,MLE	0.55	0.47	0.85
ngram=2, Laplace	0.61	0.45	0.87
ngram=2,$Add-\delta : 0.7$	0.59	0.42	0.86

Table 2: HMM experiments results

5.2 Maximum Entropy Markov Model

We test all combinations from the feature set. . See Table 3. Number of tested combinations: $\sum_{i=1}^{n=11} \binom{n}{i}$ = 2047

Experiment description	F_{μ}	F_{M}	Accuracy
Current char. value	0.484	0.248	0.828
Next char. value	0.601	0.434	0.867
Char. position	0.534	0.19	0.845
Is first char., Is last char., Current char. value, Previous char. value, Next char. value, First char. value, Last char. value, Second char. value, Second to last char. value	**0.781**	**0.714**	**0.927**

Table 3: MEMM experiments results

5.3 Conditional Random Fields

We test features extracted from the current word, its neighbours and sentence features. See Table 4.

5.4 Recurrent Neural Network

We test the Cartesian product of number RNN layers=$\{1, 2, 3, 4, 5, 6\}$,

[8]Full results are provided on project repository

Experiment description	F_{μ}	F_{M}	Accuracy
Current word only, Is first char., Is last char., Char. position, Current char. value, Previous char. value, Next char. value, First char. value, Last char. value, Second char. value, Second to last char. value	**0.797**	**0.729**	**0.932**
Current word only, Word position in a sentence Is first char., Is last char., Char. position, Current char. value, Previous char. value, Next char. value, First char. value, Last char. value, Second char. value	0.793	0.713	0.931

Table 4: CRF experiments results

size of hidden layer=$\{32, 64, 128\}$, embedding=$\{$1-hot, MDS, NN$\}$. See Table 5. We use AdamW optimizer for training (Loshchilov and Hutter, 2017), with starting learning rate 0.01 and learning rate reduction on plateau. We use early stopping to avoid over-fitting. See Figure 4.

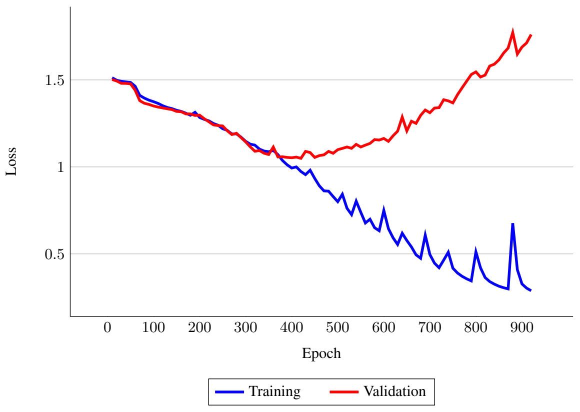

Figure 4: RNN loss

Experiment description	F_{μ}	F_{M}	Accuracy
Number of layers=3, RNN hidden dimension=32, Embedding=MDS	**0.706**	**0.431**	**0.902**
Number of layers=3, RNN hidden dimension=32, Embedding=NN	0.685	0.405	0.895
Number of layers=4, RNN hidden dimension=32, Embedding=MDS	0.686	0.432	0.895

Table 5: RNN experiments results

5.5 Random seed influence

In addition to determining the best hyper-parameters we've tested the best configuration with random seed $\in \{0, \dots, 9\}$ to determine the influence of data order. We provide median, minimum and maximum for each random seed (Figure 5).

6 Evaluation

We select the best model configuration from each model family and rebuild the model on the entirety[9] of the training data as described in section

[9]RNN excepted

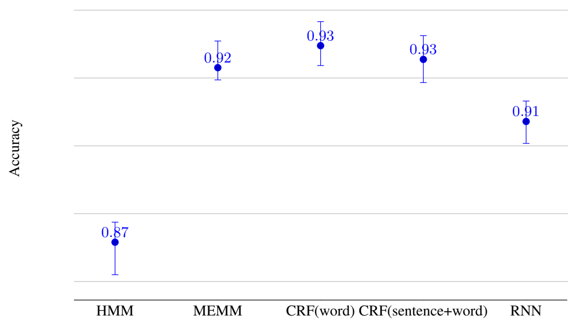

Figure 5: Best model accuracy range

3.2. Afterwards we apply post-processing to receive the remaining output. The results summarized in Table 6 and Figure 6.

Model	F_μ	F_M	Accuracy
HMM	0.62	0.497	0.873
MEMM	0.779	0.734	0.927
CRF	**0.803**	**0.763**	**0.934**
RNN	0.479	0.329	0.826

Table 6: Testing results

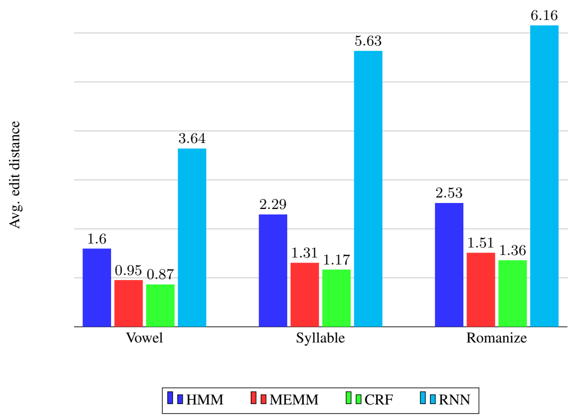

Figure 6: Edit distance.
Smaller is better

7 Discussion

First, we establish a probabilistic baseline for vowel prediction task. If we assume the vowels i.i.d. with equal probability, the accuracy is $\frac{1}{6}$. Let $|v_{avg}|$ denote the average number of vowels in a word, then the average edit distance is the expected value: $\sum_{|v_{avg}|}(0 \cdot \frac{1}{6} + 1 \cdot \frac{5}{6}) = |v_{avg}| \cdot \frac{5}{6} = 3.725$. Another way is to use vowel appearance statistics and use MLE. Since the normalized frequency of most likely vowel is 0.44, the average edit distance calculated as $\sum_{|v_{avg}|}(0 \cdot 0.44 + 1 \cdot 0.56) = 2.49$.
We can see from the results summaries that all the models, with the exception of RNN, are better than chance. RNN performance, from all indicators is quite poor. CRF model provides best accuracy and Fscore, although it's similar to MEMM performance.
The degradation in edit distance through post-processing stages almost the same, depending on vowel prediction accuracy: $vowels \xrightarrow{\times 1.4} syllables \xrightarrow{\times 1.15} roman.$
As we can see in Table 6 less frequent vowels contribute more to overall error (since F_M treats all classes equally, while F_μ favors larger classes). This problem is more apparent from analysis of confusion matrix (Figure 7). We can clearly observe that for "e","u","o" the error is frequent. This can be the product of inconsistent annotation or insufficient training data as well as class weighting issue.
In Table 7 we provide several examples of system results as well as ground truth. In the first 3 words we see that there is a perfect match between predicted and ground truth, except for romanization stage, where consonants with dual pronunciation were predicted incorrectly[10]. In the next 3 words we start from incorrect vowel prediction that ripples through the rest of the process. This effect emphasizes the importance of correct vowel prediction and difficulty of recovery down the stream. Word 5 provides an interesting case: the predicted word doesn't match the ground truth, but the prediction may be a correct word in different context.

Word	Transliteration	Vowels Prediction	Syllabification	Romanization	
למעשה	lmˁšh	lemaˁašeh*	le-ma-ˁa-šeh*	le-ma-a-she	Model
		lemaˁašeh*	le-ma-ˁa-šeh*	le-ma-a-se	Ground Truth
משובצת	mšwbçt	mešuw*beçet*	me-šuw*-be-çet*	me-shu-ve-tset	Model
		mešuw*beçet*	me-šuw*-be-çet*	me-shu-be-tset	Ground Truth
המושבים	hmwšbym	hamow*šabiy*m*	ha-mow*-ša-biy*m*	a-mo-sha-vim	Model
		hamow*šabiy*m*	ha-mow*-ša-biy*m*	a-mo-sha-vim	Ground Truth
משמשים	mšmšym	miš*mašiy*m*	miš*-ma-šiy*m*	mish-ma-shim	Model
		mešam*šiy*m*	me-šam*-šiy*m*	me-sham-shim	Ground Truth
חברות	ḥbrwt	ḥabarow*t*	ḥa-ba-row*t*	ha-va-rot	Model
		ḥeb*row*t*	ḥeb*-row*t*	hev-rot	Ground Truth
ברחבת	brḥbt	beraḥabet*	be-ra-ḥa-bet*	be-ra-ha-vet	Model
		bereḥabat*	be-re-ḥa-bat*	be-re-ha-vat	Ground Truth

Table 7: CRF error analysis

8 Conclusion

In this paper we've presented and analyzed a scheme for syllabification of Hebrew words as

[10]Described in more detail in 3.1.3

	Ground truth					
	e	u	i	o	a	*
e	0.010	0.133	0.051	0.066	0.022	0.010
u	0.051	0.022	0.036	0.275	0.052	0.015
i	0.092	0.111	0.343	0.110	0.100	0.046
o	0.441	0.089	0.161	0.143	0.108	0.037
a	0.246	0.333	0.219	0.209	0.576	0.076
*	0.159	0.311	0.190	0.198	0.141	0.815

Figure 7: CRF confusion matrix

well as their romanization. We've shown that discriminative methods used outperformed generative methods significantly.
The main challenges addressed in this paper:

- Defining annotation scheme for Hebrew syllabification
- Vowel prediction in deficient spelling using various methods
- Defining syllabification and romanization heuristics

The results achieved are valuable mainly as a baseline for future research and can be expanded upon greatly.

9 Suggestions for Future Work

We suggest a three-fold approach:

- **Data**
  As we've seen amount of annotated data has significant impact on some models (especially RNN), therefore we propose to channel a significant effort toward data annotation.
  In addition, the current data-set replaced some letters in words with foreign origin (such as Chernobyl, George, Jacques), therefore a different data-set should be considered.
- **Collaboration with a linguist**
  Collaborating with a linguist on both annotation, and post-processing will ensure both correctness and attention to grammar nuances.
- **Architecture & Model** We suggest the use of more complex models (such as Transformer (Vaswani et al., 2017), Convolutional LSTM (Yu2, 2018)) as well as approaching the task of syllabification as end-to-end prediction task, with fewer post-processing steps.

References

2018. A General-Purpose Tagger with Convolutional Neural Networks. pages 124–129.

Connie R. Adsett and Yannick Marchand. 2009. A comparison of data-driven automatic syllabification methods. In *String Processing and Information Retrieval*, pages 174–181, Berlin, Heidelberg. Springer Berlin Heidelberg.

Susan Bartlett, Grzegorz Kondrak, and Colin Cherry. 2009. On the syllabification of phonemes. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 308–316, Stroudsburg, PA, USA. Association for Computational Linguistics.

Alon Itai and Shuly Wintner. 2008. Language resources for Hebrew. *Language Resources and Evaluation*, 42(1):75–98.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Uzzi Ornan and Rachel Leket-Mor. 2016. Phonemic Conversion as the Ideal Romanization Scheme for Hebrew: Implications for Hebrew Cataloging. *Judaica Librarianship*, 19(1):43–72.

Kseniya Rogova, Kris Demuynck, and Dirk Van Compernolle. 2013. Automatic syllabification using segmental conditional random fields. *Computational Linguistics in the Netherlands Journal*, 3:34–48.

Loitongbam Gyanendro Singh, Lenin Laitonjam, and Sanasam Ranbir Singh. 2016. Automatic syllabification for manipuri language. In *COLING*.

Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45(4):427–437.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December(Nips):5999–6009.