BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **22**, No 4
Sofia • 2022 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2022-0042

B-Morpher: Automated Learning of Morphological Language Characteristics for Inflection and Morphological Analysis

László Kovács, Gábor Szabó

Institute of Information Science, University of Miskolc, Hungary
E-mails: kovacs@iit.uni-miskolc.hu szgabsz91@gmail.com

Abstract**:** *The automated induction of inflection rules is an important research area for computational linguistics. In this paper, we present a novel morphological rule induction model called B-Morpher that can be used for both inflection analysis and morphological analysis. The core element of the engine is a modified Bayes classifier in which class categories correspond to general string transformation rules. Beside the core classification module, the engine contains a neural network module and verification unit to improve classification accuracy. For the evaluation, beside the large Hungarian dataset the tests include smaller non-Hungarian datasets from the SIGMORPHON shared task pools. Our evaluation shows that the efficiency of B-Morpher is comparable with the best results, and it outperforms the state-of-the-art base models for some languages. The proposed system can be characterized by not only high accuracy, but also short training time and small knowledge base size.*

Keywords**:** *Morphology, Machine learning, Rule induction.*

1. Introduction

According to the theory of morphology and computational linguistics, words are built up from morphemes, that are the smallest morphological units with associated meaning [1]. The grammatically correct root form of a word is called the lemma, while the added morphemes that modify its base meaning are called affixes. These affixes can be prefixes (prepended to the word), suffixes (appended to the word) or infixes (inserted in the middle of the word). In morphologically complex languages, affixes may change some of the characters in the root form as well, resulting in for example vowel or consonant gradation. This means that determining the lemma is slightly more difficult. Simply dropping the affixes does not yield the grammatically correct root form, but only the stem. The process of adding affixes to a word is called inflection, while the inverse operation is called lemmatization or stemming, depending on whether the required output is the lemma or the stem respectively. The morphological analysis of a word returns both the lemma and the list of affix types, optionally including the affix boundaries too.

In this paper, we present a novel morphology engine having the following key properties:

- The engine can perform inflection, lemmatization and morphological analysis, and learn the necessary rules of morphologically complex languages such as Hungarian.
- The model learns these rules from a training data set containing (word, lemma, morphosyntactic tags) triples.
- The model considers inflection and lemmatization as generic string transformations that are constructed from simple atomic rewrite rules.
- The process of analysis and inflection is considered as a classification problem.
- The engine contains a composition of different models to provide a better accuracy output.
- The engine uses a fast and self-explainable training model.
- The engine supports efficient incremental training, which means that it is able to learn new patterns at any time without any significant overhead.

2. Related survey

First, we introduce the basic morphological concepts used in this paper.

- Morphology: Morphology is concerned with the study of word forms.
- Morpheme A morpheme is the smallest unit which carries meaning.
- Lemma: The basic, canonical form of a word (example: studies → study).
- Stem: The core word after removing the affixes (example: studies → studi).
- Inflection: To change the role of a word in the sentence without changing the category of the word (example: play → played).
- Derivation: To change the meaning, category of a word by adding new morphemes to the stem. (example: sad → sadness).
- Affix: A morpheme unit to change the role or of the category of the words.
- Prefix, suffix, infix: Affix at the beginning, at the end or in the target word.

The first widely used morphology model for agglutinative languages was the two-level morphology model [2]. In this model, the inflected forms are represented on the following two levels: the surface level contains the written form, while the lexical level contains the morphological structure. The valid lemmas and affix types are stored in a dictionary, and Finite State Transducers (FST) are used for applying the transformations. The model differentiates two elementary phonological rules: context restriction rules (only if) and surface coercion rules (if). From these elementary rules, we can create complex composite rules (e.g., if and only if). One of the first approaches to use automated rule generation for two-level morphology has been proposed in [3]. The main idea behind the algorithm is to identify elementary INSERT, DELETE, REPLACE and NOCHANGE transformation steps in the training words, and merge them together. The goal is to have two-level rules whose context is long enough to uniquely identify the transformation position, but not too long to be overspecified. To acquire optimal two-level rules, a Directed Acyclic Graph (DAG) is used.

The most widely used baseline unsupervised method is the Morfessor engine published by [4]. Morfessor is a language independent word segmentation model using statistical approach to determine the building blocks with highest probabilities. In recent years, we can observe an increased interest in semi-supervised models too. In these approaches, only a small amount of annotated word forms is available for model training, but most of the lexicon contains unannotated words. The goal is to find efficient approaches to maximize the information found in the annotated examples. In [5], the model of Conditional Random Fields (CRF) is used to determine the optimal segmentation. CRF is a discriminative model for sequential tagging and segmentation published by [6]. The proposed methods extend the CRF-based approach to leverage unannotated data in a straightforward and computationally efficient manner via feature set augmentation, utilizing predictions of unsupervised segmentation algorithms.

In [7], a Labelled semi-supervised Morphological Segmentation (LMS) engine is presented, that explicitly models morphotactics. The engine can be used for morphological segmentation, for stemming and for morphological tag classification. Unlike the previous models, it uses a rich, fine-grained label set. The engine is based on a probabilistic model to determine the corresponding labels for the tested words. Also, this method applies the CRF model to determine the winner label assignments. Based on the performed test, the Finnish and Zulu languages have the most complex label system.

F a r u q u i et al. [8] published a model of inflection generation as sequence-to-sequence transducer using a neural network engine. The model transforms its input to a sequence of output characters representing the inflected form. The training set of the model contains pairs of lemma and inflected forms. To improve the supervised model, unlabelled data are added to the training set. The experiments show that the model achieves better or comparable results to the state-of-the-art methods in the benchmark inflection generation tasks.

Many state-of-art morphological models are gathered by SIGMORPHON (Special Interest Group on computational MORphology and PHONology). The training and test data are provided on-line, and the tasks can be solved using any technique. The best models are published in [9-11].

K a n n and S c h u t z e [12] developed a Morphological Encoder and Decoder (MED) engine using neural encoder-decoder models together with special encoding of the input and output as symbol sequences. The model showed superior performance in the SIGMORPHON competitions. The proposed engine is an extension of the network architecture proposed by B a h d a n a u, K y u n g h y u n and Y o s h u a [13] for machine translation, which is a special kind of the Recurrent Neural Network (RNN) encoder-decoder model. The encoder module consists of a Gated RNN Unit (GRU) that reads an input sequence of vectors and encodes it into a fixed length context vector. The decoder uses the context vector to predict the output using conditional probability based on current input, current context and current hidden state values. The attention-based version of this model allows different vectors for each step by automatic learning of an alignment model.

The UF 2017 method proposed in [14] models the morphological reinflection problem using an encoder-decoder architecture. For an input word, every character is encoded through a Bi-directional GRU network. Another GRU network is deployed as a decoder to generate the inflection. The UTNII 2017 model, published in [15] is based on the seq2seq model, and with its configuration, it was the second best of 2017 in the high-resource scenarios. The Hamburg 2018 model published by Schroder [16] introduces the concept of patches that act as string transducer actions. The resulting model is a language-agnostic network model that aims to reduce the number of learned edit operations by introducing equivalence of classes over graphical features of individual characters. The IITBHU 2018 model published in [17] uses a Pointer-Generator Network (PGN) to mitigate the problem of copying many characters between word forms. The lemma and the morphosyntactic tags are encoded by two separate encoders. Compared to other similar performing systems, this model is trained end-to-end, does not require data augmentation techniques, and uses soft attention over hard monotonic attention, making the resulting system more flexible. The MSU 2018 model [18] aimed to improve the accuracy in medium and low-resource scenarios by explicitly equipping the decoder with the information from the character-based language model, however the advantage was not clear.

The main goal of our investigation was to analyse a novel approach with integration of pattern matching and machine learning modules. The motivation is based on the next facts:

- The direct pattern matching based methods provide a more explainable solution than the other machine learning algorithms.
- The concept of locality is a general accepted principle in many knowledge domains.
- The integration of different methods can improve the efficiency of the inference systems.
- Testing in which situations can be the pattern matching approach with Bayes classifier is competitive with the current complex neural network architectures.

3. Architecture overview of the proposed morphological engine

The surface layer of the morphology is usually represented with general string transformation models. Using this approach, both morphological analysis and inflection can be investigated as a classification problem. In the case of morphological analysis, category labels correspond to inflection categories or lemmas, while in the case of inflection, word transformation rules are the related category labels. In the field of classification models in ML, the most widely used approach is the application of neural networks. Neural networks have many advantageous properties, they provide excellent results on complex problem domains, most of the dominating classification engines in Natural Language Processing (NLP) use an engine based on NN. Beside the benefits, we can mention some properties where neural network models are not so powerful:

– descriptive explanatory power (it is hard to explain the reasoning inside the network),

– relatively long training time for model construction,
– higher costs in the case of incremental learning.

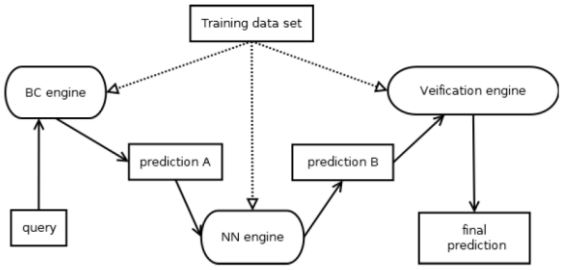


Fig. 1. Architecture overview of the proposed B-Morpher engine

The goal of our paper is to investigate an alternative classification method, which is an extension of the traditional Bayes Classification (BC) approach. As the BC model requires only a thin model, then:

– the training phase is usually cheaper,
– the decision is case-based, thus the elementary decision steps can be more easily verified,
– it provides good results in handling of outlier cases.

Considering the inflection generation task, we use the following interpretation:

- Category label. The string transformation rule describing some replacement operations within the words;
- Attribute. Context substrings of the transformation section plus the current morphosyntactic description.

In the case of morphological analysis, the mapping is slightly different:

- Category label. A pair containing morphosyntactic description and the transformation rule;
- Attribute. Context substrings of the transformation section.

In the proposed engine, an extension of this BC approach is implemented. The key features can be summarized in the following points:

1. The output of the BC module is a weight vector containing the calculated weight values for all categories.

2. In the case of morphological analysis, the engine performs a fine-tuning of the resulting weight vector using a NN classifier. The reason for this step was that our experience has shown that chaining BC and NN classifiers could provide better accuracy than the BC or the NN alone. The key point in the NN component is that the input vector is an estimation of the BC classifier and not the item feature vector. In the case of inflection generation, the use of NN engine provides only marginal improvement in accuracy.

3. An additional verification module containing both ML and rule-based units is applied to measure the validity and credibility of the candidate categories. The verification engine performs among others probability-based ranking where the priority value depends on the conditional probability of the relevant letters.

The overview of the architecture of the implemented engine is given in Fig. 1. Another key property of the proposed model is that it uses fusional approach, which means that the set of morphosyntactic descriptions contains not only the elementary morphosyntactic description units, but every composed description found in any training example is a separate unit in the category set. This approach enables a more flexible and more general view as the set of supported languages covers both agglutinative and fusional languages.

4. Formal morphology model

Before defining the formal morphology model of the B-Morpher engine, let us introduce some common notations: $\{x_i\}_{i=1}^{n}$ will denote an unordered set of *n* items. Another common formalism for defining indices is $i \in [1, n]$ which means that the index *i* will run through the integer numbers in the closed interval of 1, …, *n*.

Let Σ be an alphabet containing arbitrary characters. Strings of length *n* are denoted by $\Sigma^n = \{s | s = s_1, s_2, \dots, s_n \in \sum\}$. The length of *s* is denoted by |*s*|. The set of all strings is denoted by $\Sigma^* = \cup_{\{i=0\}}^{\{\infty\}} \Sigma^i$. The set of words is denoted by $W = \{w_i\} \subset \Sigma^*$. Some of these words are lemmas, meaning that they represent the grammatically correct root form of base concepts. The set of lemmas is a subset of the word set $W' = \{w'_i\} \subset W$.

The set of affix types (or morphosyntactic descriptions) is denoted by $T = \{t_i\}$. The set indicates grammatical transformations of words. Applying an affix type on an input word will change its base meaning and transform its surface form by prepending (prefix), appending (suffix) or inserting (infix) additional characters to the word. Each affix type is associated with a set of transformation rules (denoted by $R = \{r\}$ that describe how we can produce the inflected forms of the input words according to the given affix type.

The basis of the B-Morpher model is the so-called transformation engine submodule whose responsibility is to learn the transformation rules of the affix types. These rules model morphological transformations as string transformations, and are generated from a word pair set, extracted by B-Morpher from the original training data.

4.1. Transformation rules

For processing the words in the training set, we introduce an extended alphabet that will be used internally to denote the word-start and word-end positions: $ will mark the start of the word, while # will mark the end of the word. These are special characters; they do not belong to the original Σ alphabet. The extended alphabet will be denoted by $\Sigma^{\#} = \Sigma \cup \{\$, \#\}$. Let us also define a new operator on the domain of words: $\mu(w) = w^{\mathrm{e}} = \$ + w + \#$. The inverse operator drops these characters: $\mu^{-1}(w^{\mathrm{e}}) = w$. The set of extended words is denoted by W^{e}. The goal of this phase is to align variant and invariant segments of the input words and store the changing variant segments in the rule base. For that, we first create an extended training word pair set, where the words are extended with the start and the end symbols. Then we split each word pair into matching segments, where each segment has either two

identical matching substrings of the two words, or two different substrings. A segment is called variant if they are different, otherwise it is called invariant. In a valid segment decomposition, variant and invariant segments are alternating. To select the best possible segment decomposition for each word pair, we choose the best matching invariant segments having maximal fitness value. The fitness value of a segment ψ is inversely proportional with the index difference of the two substrings and proportional with their lengths. This formula encodes that the best segment is the one with the longest substrings that are near to each other's position. After choosing the best invariant segment, we can recursively continue the segment selection algorithm on the remaining parts, until they are short enough to be identified as variant segments.

4.2. Example

For the Hungarian training word pair (dob, ledobott) which means (throw, threw down) in English, we first extend the words with the special characters: ($dob#, $ledobott#). Algorithm yields this segment decomposition: $(\psi_1^1 = \$, \psi_2^1 = \$), (\psi_1^2 = \text{dob}, \psi_2^2 = \text{dob}), \ (\psi_1^3 = \#, \psi_2^3 = \text{ott}\#)$, where the middle segment is invariant, while the others are variant segments.

From the variant segments we can deduce a set of atomic rewrite rules: $R = \{(\alpha, \sigma, \tau, \omega)\}$, where α is the prefix, σ is the changing substring, τ is the replacement and ω is the suffix. The rule context that must be searched in the input words later during inflection is $(r) = \alpha + \sigma + \omega$. We can see that this rule model can describe prefix, infix and suffix rules as well. Let us take a variant segment $\psi_1^i \rightarrow \psi_2^i$. The first rule that we generate is called a core atomic rule $r = (\alpha_{ic}, \sigma_{ic}, \tau_{ic}, \omega_{ic})$, where $\sigma_{ic} = \psi_1^i$, $|\alpha_{ic}| = 0$, $\tau_{ic} = \psi_2^i$ and $|\omega_{ic}| = 0$, meaning that the prefix and suffix parts are empty. The other atomic rules are generated by extending this core atomic rule with one character at a time on the left and right sides, symmetrically.

To make the generated atomic rules unambiguous, we must make sure that only those rules are retained whose contexts appear only once in the base form of the word. This means that the retained rules will always yield the original in form given the base form.

4.3. BC module

In the core Bayes Classification Method, the prediction is based on the following model:

$$c_w = \text{argmax}_c \left\{ P(c) \prod_i P(a_i \,|c) \right\},$$

where *c* is category label and *a* is attribute.

In our proposal, the core element of the classification engine is based on the following formula:

$$c_w = \text{argmax}_c \left\{ \max_i \{ w(c, a_i) \} \right\},$$

where *w* is weight value based on the training set.

Depending on the task type, the category c is either the affix type (morphosyntactic description) or the transformation rule. Attribute a_i corresponds to a context substring (pattern), thus context substrings will be used as attributes to determine the winner category. Weight values denote the relevance of the pattern. When constructing the weight function our main considerations were the followings:

- Rules whose context matches the input word are the relevant rules:
- A rule with a longer matching substring in the input word is better than a rule with a shorter matching substring.
- We should differentiate rules that have similar fitness values using their frequencies, i.e., the number of word pairs in the training set they apply to.

In the proposed model, the following formula is implemented:

$$w(c,a) = f(w_s, \frac{|s|}{|q|} w''(c,s)),$$

where:

q is query item;

a is a matching substring pattern, i.e., $s(q)$ is met;

w_s is weight value of s, based on the position of s in q;

$w''()$ is weight value of s in the training set;

$f()$ is monotone increasing function.

4.4. Verification module

In the verification unit, one of the steps is to calculate the conditional probabilities of given vowel sequences in the context area of the affix transformation:

$$p(v_1, \dots, v_m|c),$$

where $v_i$denotes vowel elements from the context part of the investigated word and c denotes the corresponding category label. The weight values for the candidate categories are updated using the following formula:

$$c_w = \text{argmax}_c \left\{ f(p(v_1, \dots, v_m|c) \max_i \{w(c,a_i)\} \right\}.$$

In the classification process, the weight values depend on the length and position of the common matching substrings. One key issue of this approach is the case when the training set does not contain matching samples. This can happen in case of small or unbalanced training sets. To manage this problem, the proposed verification engine contains a Nearest Neighbor Searching (NNS) module, too. The NNS module performs a similarity-based search in the training set to find the most similar examples. This kind of search differs from exact search in that for a given query word q, the NNS should retrieve a set of candidate words.

In our model, we use the edit distance function to measure the similarity of words. The complexity of NNS algorithms is in general much higher than that of exact search. Based on the literature [19] there are two main techniques to reduce computation costs: a) filtering candidate words from the dictionary, and b) using a search tree to locate the neighboring elements. Concerning the search tree, M-tree or VP-tree [20] are the dominating techniques.

In our system, we have developed a novel method based on the combination of the dynamic programming and prefix tree approaches. According to our experiments,

this method significantly dominates the baseline VP-tree method. The NNS module generates a search tree corresponding to a prefix tree of the words, where nodes are assigned to characters. The proposed NNS algorithm performs a modified A* search algorithm where the cost value is equal to the sum of the previous editing transformation costs plus the upper limit cost of the expected future steps. The value of the second component depends on the length of the word segment that is not processed yet.

4.5. Transducer module

The first module in the engine that performs morphological analysis uses a BC classification method with the following representation formalism:

- Input items are triplets (w_0, t, w_1), where w_0 is a lemma, *t* is the inflection class and w_1 denotes the inflected word.
- Having (w_0, w_1), we can generate the corresponding transformation rule *r*, as string transformation function, where $r(w_0) = w_1$.
- Pattern attributes of the words are given by $w = w(a_1, \ldots, a_m)$, where a_i is a substring in *w*.

Every transformation rule r may belong to one or more inflection classes. During the morphological analysis, the goal is to determine the winner inflection rule having the largest probability. Instead of using real probability values, we introduce a weighting model. The model is based on the following assumptions:

- The longer the pattern attribute a_i is for a training word *w* in sample *s*, the larger the training weight of a_i is for the rule *r*.
- The longer the pattern attribute a_i is for a query word *q*, the larger the testing weight of a_i is for the rule *r*.
- The global training weight of a_i is calculated with the maximum aggregator.
- In the calculation, the weight of a pattern substring depends on the total length of the query word as well as on the length of the training word.

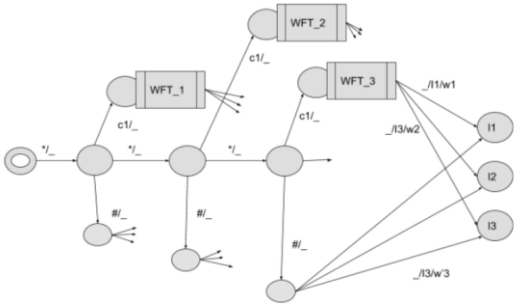


Fig. 2. Schema of the WFST-BC transducer

The operation of the module can be given with a Weighted Finite State Transducer (WFST) having the structure depicted in Fig. 2. Considering the standard alphabet of the words, the upper limit for the total number of nodes can be approximated with the following formula:

$$\sum_{i\in[1,\ldots,L]} \min\{C^i, N\} \leq LN,$$

where N is the number of training classes, C is the number of characters in the alphabet, L is maximum length of the words.

Considering the any-character cases (which are used to process the remaining parts of the words), the upper limit can be given with

$$L^2N.$$

As each standard or any-character node may be the last node of the word, the upper approximation for the termination nodes is also

$$L^2N.$$

Having T inflection classes and N training items, the corresponding WFST has:

- $O(LN)$ standard character nodes,
- $O(L^2N)$ any-character nodes (*),
- $O(L^2N)$ word termination nodes,
- $O(T)$ are rule nodes.

Thus, the corresponding transducer graph has $O(L^2N)$ nodes. We remark that the given WFST structure is suitable for prefix-based prediction. A similar WFST can be constructed for the postfix-based generation using back propagation traversing, too.

Concerning the infix inflections, we can construct a compound transducer depicted in Fig. 3. Thus, the BC engine can be represented as a weighted finite state rational transducer. The output of the module is the weight vector of the inflection classes $\langle w_1, \ldots, w_m \rangle$. Thus, considering the computational complexity of the proposed system, we can show that it is a special case of weighted rational transducers and it is in general not equivalent with probabilistic transducers (or Turing machines).

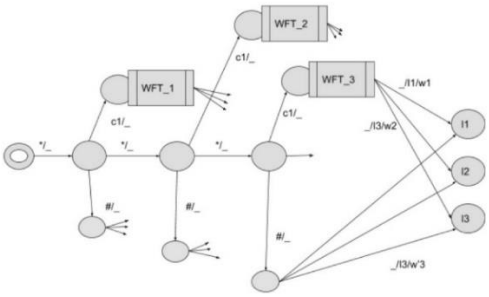


Fig. 3. Schema of the WFST-BC transducer for infix transformation

5. Experiment results

For the experimental tests, we have used two main categories of training data sets. For the case of large data sets, a training data set for Hungarian was generated where the inflection cases were collected from on-line documents and the related morphological annotations were constructed automatically using a high quality existing morphological analyser program. For generation of the training and evaluation data we used Hunmorph published in [21] and Morphdb.hu [22]. We collected input word set from the National Szechenyi Library in an automated way, and Hunmorph determined the morphosyntactic tags and lemmas. For the transformation engines, we needed to generate word pairs, too. These word pairs were

extracted from the output of Hunmorph. The number of word candidates was 13,345,903 from which Hunmorph recognized 4,423,882 items. After additional data filtering, the resulting training set contained 2,147,703 clean training triplets. The training and evaluation data was generated randomly from this data set, published on Github (**https://github.com/szgabsz91/morpher-data**).

5.1. Efficiency comparison of BC and NN engines

In the first experiments, we have compared the efficiency of the proposed BC engine with the efficiency of the standard three-layered back-propagation network. For the tests, we have selected the task to predict the POS property of the words. For input, we have used the large Hungarian training set (N_0 = 2,147,703 items), and we have tested the following variants:

- the engine contains only NN-unit,
- the engine contains only BC-unit,
- majority voting with BC-units,
- majority voting with NN-units,
- BC unit with NN-based refinement unit.

The experimental results are summarized in Table 1. We can add that there is a big execution cost difference between the BC and NN modes in the case of large data sets. Considering $N = 0.9N_0$, the training time for NN was 130 min, while 6 min for BC unit.

Table 1. Comparison of different model architectures

Size of training data set	Method	Accuracy (%)
N=0.90×N_0	NN	86
	BC	96
N=0.01×N_0	NN	82
	NC	87
N=0.05×N_0	NN	86
	BC	94
	Majority voting (3 NN units)	87
	Majority voting (3 BC units)	96
	BC+NN refinement	97

As the experimental results show, the BC approach dominates the selected base NN version, but it is worth using majority voting and performing a NN-based refinement step. These additional processing elements can increase the accuracy level by 2-3 %.

To analyse the execution cost efficiency of the proposed NNS unit, we compared the method with the naive brute-force method and with the VP-tree method. As the resulting data given in Table 2 shows, the proposed engine significantly dominates the other methods and it provides an acceptable speed to be used in the prototype system. The time values in Table 2 are given in seconds. It should be noted that this module will only be used if no exact matching can be found for the query word.

In Table 2, column N denotes the size of the dataset, and column min$D$ denotes the distance to the nearest element, column vpt-bt symbolizes the index construction

time for the VP-Tree Algorithm, column vpt-qr is for the query time. The last two columns denote the same data for the proposed NNS Algorithm. As it can be seen, the query cost decreases if an exact matching element exists, i.e., there is no need to perform search in a wider area.

Table 2. Comparison of execution costs in NNS

N	min*D*	Naive	vpt-bt	vpt-qt	p-bt	p-qt
100,000	2	5.9	126	1.9	1.2	0.001
500,000	2	29.3	776	4.6	5.4	0.005
500,000	1	29.1	773	0.21	5.1	0.0005
500,000	0	28.3	763	0.001	5.8	0.0002
1,000,000	1	59.5	1245	0.38	7.4	0.0004

5.2. Experiments with the large Hungarian training set

In these tests of the B-Morpher system, the data set was split into two disjoint parts: one for training and the other for tests. The size of the test part is 3000 examples. In the experiments, the accuracy of inflection generation was investigated using training sets of different sizes. The test results are presented in Table 3. The second column shows the number of affix types found in the training set. In the test, the prediction engine yields three candidates as output. Column acc1 denotes the accuracy level considering only the first candidate in the output. Column acc2 is for the case when the first two candidates were considered. Column acc3 denotes the case when all three output items are tested for matching. One can see that the method's efficiency tends to be 100 % in limit, as every item in the training set will be recognized properly.

Table 3. Efficiency of B-Morpher for inflection generation for Hungarian

Training data size	Number of affix types	acc1	acc2	acc3
20,000 (1 %)	349	83.0	91.0	93.7
100,000 (5 %)	489	90.0	96.0	97.3
200,000 (10 %)	578	93.0	97.5	98.5
400,000 (20 %)	688	95.3	98.4	99.0
600,000 (30 %)	771	96.3	99.0	99.4
800,000 (40 %)	805	96.6	99.1	99.4
1,000,000 (50 %)	871	97.3	99.3	99.5
2,000,000 (100 %)	1043	99.4	100.0	100.0

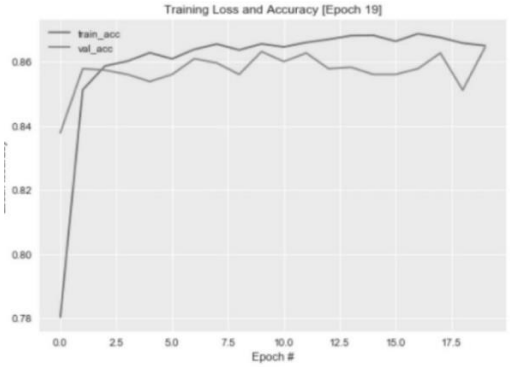


Fig. 4. Learning curve of the NN engine ($N = 0.05 \times N_0$)

In these experiments, the proposed model was compared with available baseline models submitted as part of previous SIGMORPHON tasks: Helsinki (2016), UF and UTNII (2017), Hamburg, IITBHU and MSU (2018). Since the evaluation of these methods were tested only on CPU, we do not include time comparisons. To evaluate the B-Morpher model using the Hungarian language, the volumes of training data were gradually increased to see how the model scales. We used 10,000, 20,000,…,100,000 training items. During comparison, 10,000 random input words were used and the query words were disjoint with the training item set. The knowledge base size is compared with Hunmorph.

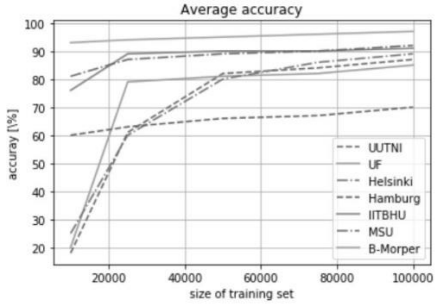
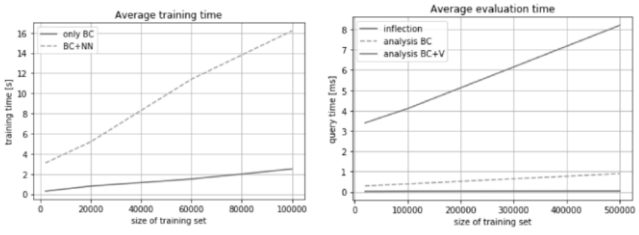


Fig. 5. Average accuracy vs training size

The accuracy analysis of the models can be seen in Fig. 5, using previously unseen words. As we can see, B-Morpher has the highest accuracy (about 97.3 percent using 100,000 training items), both for inflection and morphological analysis. This means that B-Morpher's generalization is exceptional, since it can inflect and analyse previously unseen words correctly in nearly 98 % of the cases. IITBHU and MSU are very close, too, but the Hamburg model reaches only about 70 %. These results were achieved using our generated data. Comparing these with the originally published results of the baseline models, it can be seen that for the Hungarian language, the best models were CLUZH and LMU in 2017 with about 86 %, and the UZH model in 2018 reaching about 87 %. Fig. 6a shows the average training time of the B-Morpher model in seconds. Using 100,000 training items, the training phase ends in about 2.6 s in average. Fig. 6b displays the average inflection and morphological analysis times in milliseconds.



(a) Average training time (b) Average evaluation time

Fig. 6. Average training, inflection and analysis times vs. training set size

Considering inflection, the execution times increase about linearly from 0.03 ms to 0.05 ms. For the task of morphological analysis, using only the base BC engine,

query times increase from 0.4 ms to 1 ms. In this case, the accuracy level is about 2-3 % weaker than the accuracy of the extended engine. These query response times (about 1000 words per 1 s) are weaker than the leading industrial analysers (about 10000 words per 1 s), however our implementation is a prototype system not an optimized code written in some efficient language like C. Including the verification engine, the query times will increase. In this case, the obtained response time values are between 3.3 ms and 8 ms.

Considering the file size of the exported knowledge bases, the values can be seen in Table 4. These files can be loaded quickly, skipping the training time. As the results show, MSU and UF produce smaller files, and B-Morpher provides a database of average size.

Table 4. The database size of the exported knowledge bases

Model	File size (MB)
B-Morpher	16.1
Hunmorph-Ocamorph	22.7
Helsinki 2016	58.3
UF 2017	4.5
UTNII 2017	92.4
IITBHU 2018	8.3
MSU 2018	1.5

5.3. Experiments with the SIGMORPHON data sets

First, the Task1 dataset from the SIGMORPHON 2016 shared tasks about inflection learning were analysed. The Hungarian training set contains 16,219 training examples. In the SIGMORPHON 2016 competition, the winner was the LMUMED engine using RNN methods. The winner accuracy level for the Hungarian was 99.3 %. The other participating systems could achieve significantly lower results below 97 %.

The B-Morpher system could achieve the following accuracy levels on related training and testing data sets for Hungarian. For the training phase, only the provided Task1 train dataset was used. The value of acc1 denotes the accuracy level considering only the first candidate in the output. Value acc2 is for the case when we consider the first two candidates and acc3 denotes the case when all three output items are tested for matching:

- acc1=92.0 %,
- acc2=97.3 %,
- acc3=98.6 %.

These numbers are below the winner accuracy but still they are slightly better that of the other participating systems. This result shows that although B-Morpher prefers larger datasets, it is competitive also for smaller datasets. It can provide similar efficiency as the other NN based systems, and it is significantly better than the non-neural network systems, which have an accuracy, level 10 % below the NN-based systems.

A problem domain area where B-Morpher has advantage is the domain of incremental training processes. The update of the classifier model is a relatively simple and low cost operation. If it is allowed to add incorrectly predicted words to

the training set on an incremental way, we get drastically improving accuracy result. The second group of experiments relate to the SIGMOPRHON 2019 Task 2 shared task, where the task was to perform morphological analysis. The training data was given in the form of sentences. The sentence form can be used to get additional context level information in the classification process. With B-Morpher only word level analysis was performed since B-Morpher is a word level analyser. For this task, the best result was 95.05 % accuracy level. The corresponding accuracy values of the B-Morpher system on the Task 2 for Hungarian are the followings:

- acc1=91.6 %,
- acc2=95.0 %,
- acc3=95.7 %.

Based on these result values, we can say that B-Morpher provides good efficiency, namely the acc2 and acc3 accuracy values are above the winner 95.0 % accuracy.

The test for morphological analysis was extended also for some other languages in order to investigate the language dependency of the proposed engine. In the selection of the test languages, a key aspect was to involve languages from different language families; therefore, the test was extended to the following languages:

- Lithuanian,
- Turkish,
- Finnish,
- Polish,
- Slovak,
- English,
- Spanish,
- Basque,
- Indonesian,
- Irish,
- Breton.

The accuracy for the selected languages is summarized in Table 5. In the table, the second column denotes the number of detected affix types in the training data sets. The column entitled baseline denotes the baseline accuracy values given in the shared task call. Column best denotes the best accuracy result achieved at the shared task competition. The last column denotes the difference between the winner accuracy and our acc3 values. Based on this accuracy difference, we can categorize the languages into four groups.

Group A: The B-Morpher can provide very good results, acc3 value is better than the winner accuracy:

- Hungarian,
- Lithuanian,
- Breton.

Group B: The acc3 accuracy value of B-Morpher is satisfactory, not far from the best values:

- English,

– Spanish,
– Basque.

Group C: The acc3 accuracy values are slightly lower than the winner accuracy at the shared task competition:

– Finnish,
– Irish.

Group D: The acc3 accuracy values are significantly lower than the winner accuracy at the shared task competition:

– Turkish,
– Polish,
– Slovak,
– Indonesian.

In this categorization, languages with a higher number of possible affixes are in Group D. One reason for the weak performance of B-Morpher for these languages can be that B-Morpher works mainly at surface level and the training sets at the competition are relatively sparse, there are no examples for all cases in the test set.

One important experience from the performed tests is that we can see very significant accuracy differences for the different languages. Although a deeper analysis requires more investigations in the future, we can find some reasons for this variety.

- The proposed method is strong for the agglutinative languages where well defined, relatively stable morphemes are used to denote different grammatical roles.
- In the case of fusional languages, there are more flexibility in the surface form, more training data would be needed to discover the related statistical rules. For example, the inflection sample in Polish

– byt’ ⇒ je (V; SG; PRS; NEG; IND; IPFV; FIN; 3).
– is a hard case for a surface level grammar induction engine.
– In many languages, the surface level is not enough to detect the grammatical roles. A good example is the following example in Indonesian:

dimekarkan ⇒ dimekarkan (PASS; V; SG),

where there are no surface level changes between the lemma and inflected form.

Table 5. Comparison of the accuracy for different languages (SIGMORPHON 19)

Language	affix_cnt	Baseline	Best	acc-1	acc-2	acc-3	diff
HU_Szeged	332	65.9	95.0	91.7	95.0	95.7	+0.7
LI_HSE	336	41.4	80.1	74	82	83.4	+3.3
TR_PUD	502	66.3	87.6	71	79	81.5	–6.1
EN_GUM	73	79.6	97.5	90.1	96.0	97.1	–0.4
SP_Ancora	172	84.3	98.8	95	98.2	98.8	0.0
PL_SZ	716	63.1	95.1	73	83	86.5	–8.5
SK_SNK	829	64.0	95.4	76	85	89	–6.4
FI_FTB	659	72.9	96.9	87	92.5	93.9	–3.0
BA_BDT	854	67.7	92.3	81	90	92.2	–0.1
IN_GSD	128	71.7	92.5	78	86	88	–4.5
IR_IDT	162	67.7	86.4	74	82	83.6	–2.8
BR_KEB	86	76.5	91.1	87	91	93.4	+2.3

5. Conclusion

In this paper, a novel morphological model called B-Morpher was presented that can learn complex inflection rules. B-Morpher can be used for both inflection analysis and morphological analysis. The core element of the engine is a modified Bayes classifier whose output is sent to an additional NN engine and an ML-based verification unit in order to increase the output accuracy. In the classification, class categories are given by simple string transformation rules or by affix types. The proposed model can be trained incrementally. Our evaluation shows that B-Morpher can achieve in general a good, but for some languages an excellent accuracy in both tasks. It outperforms state-of-art morphological models including those submitted to SIGMORPHON. B-Morpher's accuracy reaches here near 99 %. Considering a training set of 100,000 examples and using our Python and Java implementations, the average training time is 3 s, the average inflection time is 0.05 ms and the average morphological analysis time is between 0.5-4.0 ms depending on the required accuracy level. The average size of the knowledge base is 16 MB for the simple engine model, which is a common size among the different models. The proposed B-Morpher model provides good efficiency in incremental training mode and it uses a very fast training process. The generated model also presents a self-explainable knowledge representation format.

References

1. Bauer, L. Introducing Linguistic Morphology. Edinburgh, Edinburgh University Press, 2003.
2. Koskenniemi, K. Two-Level Morphology: A General Computational Model for Wordform Recognition and Production. Department of General Linguistics, University of Helsinki, 1983.
3. Theron, P., I. Cloete. Automatic Acquisition of Two-Level Morphological Rules. – In: Proc. of 6th Conference on Applied Natural Language Processing, 1997, pp. 103-110.
4. Creutz, M., K. Lagus. Unsupervised Models for Morpheme Segmentation and Morphology Learning. – ACM Transactions on Speech and Language Processing (TSLP), Vol. **4**, 2007, No 3, p. 34.
5. Ruokolainen, T., O. Kohonen, S. Virpioja. Painless Semi-Supervised Morphological Segmentation Using Conditional Random_ELDS. – In: Proc. of 14th Conference of the European Chapter of the Association for Computational Linguistics, 2014, pp 84-89.
6. Lafferty, J., A. Mc Callum, F. Pereira. Conditional Random_ELDS: Probabilistic Models for Segmenting and Labeling Sequence Data. – In: Proc. of 11th International Conference on Machine Learning, 2002, pp. 282-289.
7. Cotterell, R., T. Muller, A. Fraser, H. Schutze. Labeled Morphological Segmentation with Semi-Markov Models. – In: Proc. of 9th Conference on Computational Natural Language Learning, 2015, pp. 164-174.
8. Faruqui, M., Y. Tsvetkov, G. Neubig, C. Dyer. Morphological Injection Generation Using Character Sequence to Sequence Learning. – arXiv preprint arXiv:1512.06110, 2015.
9. Cotterell, R., C. Kirov, J. Sylak-Glassman, D. Yarowsky, J. Eisner, M. Hulden. The SIGMORPHON 2016 Shared Task – Morphological Reinjection. – In: Proc. of 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, 2016, pp. 10-22.
10. Cotterell, R., C. Kirov, J. Sylak-Glassman, G. Walther, E. Vylomova, P. Xia, M. Faruqui, S. Kubler, D. Yarowsky, J. Eisner et al. CoNLL-SIGMORPHON 2017 Shared Task: Universal Morphological Reinjection in 52 Languages. – arXiv preprint arXiv:1706.09031, 2017.

11. C o t t e r e l l, R., C. K i r o v, J. S y l a k-G l a s s m a n, G. W a l t h e r, E. V y l o m o v a, A. D. Mc C a r t h y, K. K a n n, S. M i e l k e, G. N i c o l a i, M. S i l f v e r b e r g et al. The CoNLL{SIGMORPHON 2018 Shared Task: Universal Morphological Reinjection. – arXiv preprint arXiv:1810.07125, 2018.
12. K a n n, K., H. S c h u t z e. Unlabeled Data for Morphological Generation with Character-Based Sequence-to-Sequence Models. – arXiv preprint arXiv:1705.06106, 2017.
13. B a h d a n a u, D., C. K y u n g h y u n, B. Y o s h u a. Neural Machine Translation by Jointly Learning to Align and Translate. – arXiv preprint arXiv:1409.0473, 2014.
14. Z h u, Q., Y. L i, X. L i. Character Sequence-to-Sequence Model with Global Attention for Universal Morphological Reinjection. – In: Proc. of CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinjection, 2017, pp. 85-89.
15. S e n u m a, H., A. A i z a w a. Seq2seq for Morphological Reinjection: When DeepLearning Fails. – In: Proc. of CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinjection, 2017, pp. 100-109.
16. S c h r o d e r, F., M. K a m l o t, G. B i l l i n g, A. K o h n. Finding the Way from ä to a: Sub-Character Morphological Injection for the SIGMORPHON 2018 Shared Task. – arXiv preprint arXiv:1809.05742, 2018.
17. S h a r m a, A., G. K a t r a p a t i, D. M. S h a r m a. IIT (BHU) { IIITH at CoNLLSIGMORPHON 2018 Shared Task on Universal Morphological Reinjection. – In: Proc. of CoNLL SIGMORPHON 2018 Shared Task: Universal Morphological Reinjection, 2018, pp. 105-111.
18. S o r o k i n, A. What Can We Gain from Language Models for Morphological Injection? – In: Proc. of CoNLL SIGMORPHON 2018 Shared Task: Universal Morphological Reinjection, 2018, pp. 99-104.
19. Y u, M., G. L i, D. D e n g, J. F e n g. String Similarity Search and Join: A Survey. – Frontiers of Computer Science, Vol. **10**, 2016, No 3, pp. 399-417.
20. F u, A. W. C., P. M. S. C h a n, Y. L. C h e u n g, Y. S. M o o n. Dynamic vp-Tree Indexing for n-Nearest Neighbor Search Given Pair-Wise Distances. – The VLDB Journal, Vol. **9**, 2000, No 2, pp. 154-173.
21. T r o n, V., A. K o r n a i, G. G y e p e s i, L. N e m e t h, P. H a l a c s y, D. V a r g a. Hunmorph: Open Source Word Analysis. – In: Proc. of Workshop on Software, 2005, pp. 77-85.
22. T r o n, V., P. H a l a c s y, P. R e b r u s, A. R u n g, P. V a j d a, E. S i m o n. Morphdb.hu: Hungarian Lexical Database and Morphological Grammar. – In: Proc. of 50th International Conference on Language Resources and Evaluation (LREC'06), 2006.

Received: 01.04.2022; Second Version: 02.09.2022; Accepted: 16.09.2022