

Experiments on Deep Morphological Inflection

Akhilesh Sudhakar, Rajesh Kumar Mundotiya, and Anil Kumar Singh

IIT (BHU), Varanasi, India
<http://iitbhu.ac.in>

Abstract. Morphological inflection (MI) is the task of generating a target word form based on a source word and a set of target morphological tags. We present different language-agnostic systems for MI and report results on datasets of three different sizes: low, medium and high. All these systems are deep neural network based. We implement and describe a lower baseline, and show that our systems improve on this baseline, as well as meet the state-of-art. One significant contribution through this work is studying the different neural architectures that perform best on different dataset sizes as well as on different languages. Another contribution is exploring the use of phonological features of the language in addition to characters, as well as pre-training of word embeddings. We also implement a hybrid system which combines rules learnt from string alignments along with deep learning. The significance of our work lies in the fact that the systems presented can be used for any language (we present results on 52 languages we experimented with) and in our analysis of how linguistic properties of each language has a strong bearing on the design of the neural architecture used for that language.

Keywords: Morphology, Deep Learning, Inflection

1 Introduction

Inflection is the process of transforming a root word (lemma) to a target word form which has a specific set of grammatical features. Common grammatical features include gender, number, tense, aspect etc. The ability to generate an inflected word form from a given lemma (root) is highly beneficial in many NLP tasks. It can be crucial in downstream tasks like topic modeling, machine translation and question answering. Most of these tasks suffer from data sparsity. This is especially true in the case of morphologically rich languages. For instance, in machine translation using parallel corpora, there is always scope for performance improvement provided more words and sentences were available along with their translations in the corpora. Statistically speaking, even then, commonly used words and their inflected forms would repeat and dominate in the corpus. In order to deal with the problem of data sparsity in these tasks, morphological analysis and subsequent morphological inflection could be used. For each word, the root could be extracted using a morphological analyzer. This root will then be translated to the destination language. Morphological inflection will then be applied on the root word (now in the destination language) to obtain the original

word but in the destination language. In this way, the actual machine translation system would work on a far smaller subset of words, which consists of just the root words, thereby reducing sparsity. Similarly, morphological inflection can be used in other tasks too ([1], [2], [3]).

An inflected form of a word is generally obtained by the use of affixes (prefixes, infixes, suffixes, simulfixes etc.) which attach morphemes to the beginning, in the middle of or to the end of the root word. For instance, the word *writing* is the present continuous inflected form of the lemma *write*.

We present language-agnostic deep neural systems for morphological inflection. *All the neural networks in these systems condition the target inflected word on the root word and the target features of the inflected word.* The models use sequence-to-sequence modeling of characters in the root word, and generate each character of the target word using the sequence information of characters in the root word as well as the target features. A single neural network uses all the training examples for a language and learns a set of weights for all grammatical features and their tags. This is done by using character-level encodings of root words and binary vector encodings of tags. It is common knowledge that the surface structure of a word and its morphological properties are highly related. These neural systems are built using character-to-character sequence models in order to exploit the surface structure of the root word. We also describe and present a baseline system that does not make use of deep neural networks and show improvements over this baseline.

The current work is an extension of our existing work, presented as part of our participation in the SIGMORPHON shared task at CoNLL 2017 [4]. This work is significantly different from our shared task submission in a number of ways- we build a baseline model, present detailed analysis of hyper-parameter selection, present our findings on using pre-trained vector embeddings, build a hybrid model as well as show the incorporation of phonological features, all of which have not been done in previous work.

There has been considerable debate about the use of deep learning in NLP for low-resource languages. A lot of questions have been raised about the ability of deep learning to work well with low-sized datasets. In order to test the performance of our systems, we present deep learning models trained on datasets of different-sizes (low, medium and high). Our models exceed the baselines on all of these different sizes. However it is also obvious that reducing the size of the training does negatively impact the performance of deep learning systems. In order to overcome this shortcoming, we also present a hybrid model based on extracting rules from string alignments, apart from our neural model. As the size of training data reduces, *we show that the hybrid model shows a growing gain in accuracy over the stand alone neural model.* Pre-training of word vectors also shows considerable performance enhancements.

2 Terminology Used

To maintain a consistent terminology, we use the term ‘root’ for the root word, and ‘target’ for the inflected form of the root word. We use the term ‘feature’ to represent grammatical properties of the inflected word, like, ‘category’, ‘gender’, ‘number’, ‘person’, ‘case’, ‘TAM’, ‘suffix’ etc. We use the term ‘tag’ to represent each of the values taken by a feature for a particular word. For instance, ‘M (male)’, ‘F (female)’ and ‘N (neuter)’ are possible tags for the feature ‘gender’.

3 Background

Prior to neural network based approaches to morphological inflection, most systems used a 3-step approach to solve the problem: 1) String alignment between the lemma and the target (morphologically transformed form), 2) Rule extraction from spans of the aligned strings and 3) Rule application to previously unseen lemmas to transform them. [5] and [6, 7] used the above approaches, with each of them using different string alignment algorithms and different models to extract rules from these alignment tables. However, in these kinds of systems, the types of rules to be generated must be specified, which should also be engineered to take into account language-specific transformational behavior. [8] proposed a neural network based system which abstracts away the above steps by modeling the problem as one of generating a character sequence, character-by-character. Akin to machine translation systems, this system uses an encoder-decoder LSTM model as proposed by [9]. This model takes into account the fact that the target and the root word are similar, except for the parts that have been changed due to inflection, by feeding the root word directly to the decoder as well. A separate neural net is trained for every language. The results reported by [8] are state-of-art and we match state-of-art on many languages and exceed on some.

Apart from sharing some similarities with [8], our work is unique in many ways. Firstly, the authors in [8] work on a smaller subset of languages and hence report only a limited set of results. Secondly, we detail a hybrid approach to supplement the deep model as well as propose different neural architectures for different sizes of data and for different languages within a particular dataset size too. This has been done taking into account the specific morpho-linguistic properties of these languages. This naturally gives better performance than a single model for all languages. On the other hand, they train a separate neural network for each target feature, whereas we train a single model (specific to a given language and dataset size) on each language. The advantage of doing so is that the model needs lesser data to learn from, because this kind of a model shares weights learned across all the features. This works well because word inflection itself is inherently a mechanism in which the patterns of transformation between a root and a target based on a particular feature’s tag’s contribution are similar to those based on other features’ tags. Training on separate features would not only make the model lose out on making use of this information but would also require more training data to learn from, since sufficient examples

need to be present emphasizing the contribution of each of the features. Our work is also novel in presenting the correlation of a language’s linguistic properties (based on their occurrence in the phylogenetic tree of languages and based on their morphological complexity) along with the results obtained and the neural architectures designed for each of them. To the best of our knowledge, this is also the only work that additionally makes use of phonological features for morphological inflection.

4 System Description

We present a baseline system as well as two systems for morphological inflection. Each system is tested on all the three dataset sizes. The difference between these two systems is that while the first uses the same neural architecture for all languages (for a particular size of dataset), the second system uses different neural architectures for different languages in order to make best of language-specific morphological information. We present the results of these two systems separately for two reasons. Firstly, the first system can be used to make comparisons across languages because it uses the same configurations for all languages. The first system is more simplistic than the second and provides a higher baseline than the baseline implemented using rule extractions. Secondly, the two systems can be compared with the baseline as well as with each other in order to observe language-dependent variations in performance, which could be insightful for further explorations.

4.1 Broad Basis for Neural Architecture

All the models are deep neural networks that take the root word and the target tag set as inputs. They then generate the predicted target inflected word as output. *Broadly, these systems treat a word as a sequence of characters and perform sequence-to-sequence treatment of characters in the root word. This sequence modeling allows the model to capture information about other characters in the context of a particular character. This in turn, provides the model with information on how affixes must be introduced into a root word to inflect it.* Moreover, we also observe that the only differences between the root and the target words are those characters that have been added, substituted or deleted due to the inflections. The rest of the characters remain the same in both. Concatenating the character embedding vectors directly with the sequence vector generated by sequence modeling of the characters helps the neural network to take advantage of this fact.

4.2 Common Aspects Across Models

In all the models in both the systems, certain structural and hyper-parametrical features remain the same. The characters in the root word are represented using character indices, while the morphological features of the target word are

represented using binary vectors. Each character index of the root word is then embedded as a character embedding of dimension 64, to form the root word embedding, i.e., the root word embedding is a 2-dimensional vector, with each row corresponding to a character position in the word, and the columns of the row corresponding to the embedding of that particular character. If an encoder is used, it is bidirectional and the input word embeddings feed into it. The output of the encoder (if any), concatenated with the root word embedding, feeds into the decoder. All recurrent units have hidden layer dimensions of 256. Over the decoder layer is a softmax layer that is used to predict the character that must occur at each character position of the target word. In order to maintain a constant word length, we use paddings of ‘0’ characters. All models use categorical cross-entropy as the loss function and the Adam optimizer [10].

Dataset The models were trained on three differently-sized datasets. The low-sized datasets had around 100 training samples, the medium-sized datasets had around 1000 training samples and the high-sized datasets had around 10000 samples for most languages. Datasets were provided for a total of 52 languages. Further details about this dataset, where it was obtained from, and the actual data itself, can be found at the CoNLL-2017 website¹.

Hyper-parameters In order to tune the hyper-parameters, we compute a categorical cross entropy error on a held-out validation set. As this tuning would contaminate the validation set and lead to biased estimates, we hold out a separate test set to calculate the model’s performance. The reported values are calculated on this test set. Our model has the following hyper-parameters:

1. **Initial learning rate:** We identify this hyper-parameter as the most influential in our model. By this, it is meant that small variations in the initial learning rate led to appreciable changes in accuracies. Given that in our choice of task (inflection), even accuracy changes in the magnitude of 1% can be considered appreciable, we find that the initial learning rate has substantial significance. We use an adaptive learning rate schedule, as suggested by [11]. After experimenting with multiple initial learning rates, we arrive at an initial learning rate of 0.5.
2. **Minibatch size:** While we found that the effect of this hyper-parameter was not so pronounced on accuracy, changes in this hyper-parameter reflected upon the computational time taken during the training phase. Considering that we used a GPU for our experiments, we set this size to 32 to take advantage of matrix computational speedups. We also note that setting this value to anything below 15 does not show any speedups.
3. **Training epochs:** While our final model uses early stopping to avoid overfitting, we observe that using early stopping masks the effect of the other hyper-parameters. In order to study these more carefully, we initially let our

¹ <https://github.com/sigmorphon/conll2017>

model run all the 10,000 training epochs completely. However, as we have described in the later section on ablation studies, we arrive at a patience value of 10 epochs. Further elaborations have been done in the relevant section.

4. **Momentum:** We find that setting the momentum to 1 gives good results and hence we did not experiment much with this hyper-parameter. This decision was also guided by the intuition that our datasets are not very large and that generally, mostly only unsupervised settings tend to make substantial gains from choosing momentum values with care.
5. **Hidden units:** Dimensions of embedded layer and hidden layers of recurrent units had to be arrived at. We did not observe a progressive trend in increases these dimensions but did observe that the model did marginally better when setting these values to powers of 2. We also note that increasing these above the values chosen (64 for embedding layers and 256 for recurrent layers) did not add to any performance improvements but neither did it degrade performance. However, in the case where we pre-trained word vectors (described in later sections), we observe that increasing the word embedding dimensions to 300 gave better results. We believe that this might be because the word vectors were trained on a Wikipedia dump corpus that naturally has more information than the dataset, which just contains individual words. In order to account for this excess information being passed (as compared to vectors that weren't pre-trained), the model perhaps requires more dimensions.

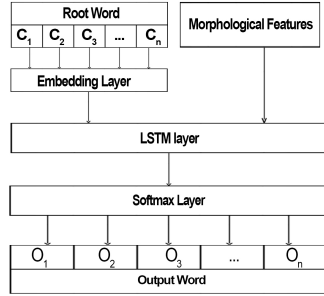


Fig. 1: C_1, \dots, C_n represent characters of the root word while O_1, \dots, O_n represent characters of the output word

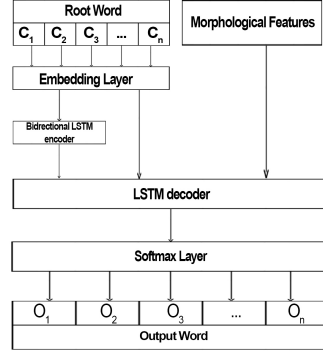


Fig. 2: C_1, \dots, C_n represent characters of the root word while O_1, \dots, O_n represent characters of the output word

Low-sized Dataset For training the model on the low-sized dataset, we did not use any encoder and we used a simple LSTM with a single layer as the recurrent unit (Figure 1).

Medium-sized Dataset For training the model on the medium-sized dataset, we used a bidirectional LSTM as the encoder and a simple LSTM with a single layer as the decoder (Figure 2).

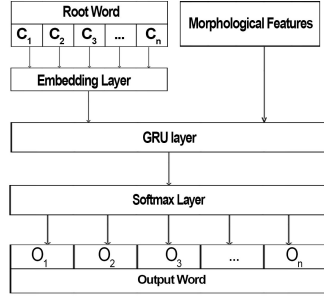


Fig. 3: C_1, \dots, C_n represent characters of the root word while O_1, \dots, O_n represent characters of the output word

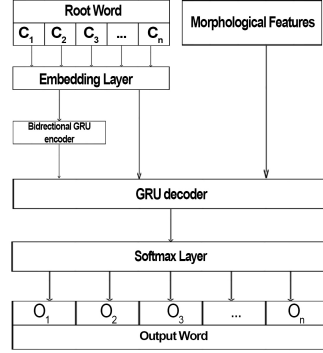


Fig. 4: C_1, \dots, C_n represent characters of the root word while O_1, \dots, O_n represent characters of the output word

High-sized Dataset For training the model on the high-sized dataset, we used a bidirectional LSTM as the encoder and a simple LSTM with a single layer as the decoder (Figure 2).

4.3 System 2

Low-sized Dataset For training the model on the low-sized dataset, we did not use any encoder and, instead, we used a simple GRU, as reported by [12], with a single layer as the recurrent unit (as shown in Figure 3).

Medium-sized Dataset For medium-sized dataset, we used different model configurations for different languages. Four different kinds of configurations were used:

1. Bidirectional LSTM as the encoder and a simple LSTM with a single layer as the decoder (Figure 2)

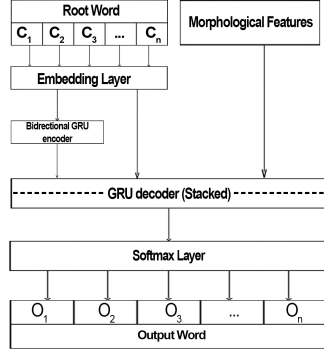


Fig. 5: C_1, \dots, C_n represent characters of the root word while O_1, \dots, O_n represent characters of the output word

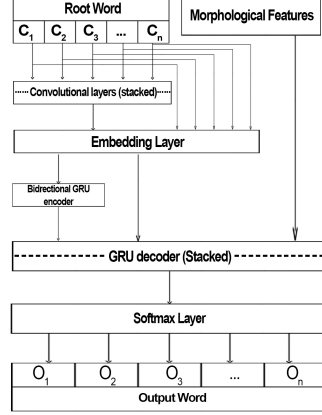


Fig. 6: C_1, \dots, C_n represent characters of the root word while O_1, \dots, O_n represent characters of the output word

2. Bidirectional GRU as the encoder and a simple GRU with a single layer as the decoder (Figure 4)
3. No encoder and a simple GRU with a single layer as the recurrent unit (Figure 3)
4. Bidirectional GRU as the encoder and a deep GRU (two GRUs stacked one above the other) as the decoder (Figure 5)

High-sized Dataset For training the model on the high-sized dataset, we used a bidirectional GRU as the encoder and a deep GRU with 2 layers as the decoder (Figure 6). Additionally, we use a 2 layer deep Convolutional Neural Network (CNN) to convolve the input word and embeddings. Each convolutional layer uses 64 convolutional filters, each having a width of 5.

4.4 Baseline System

In order to compare our method with a traditional string alignment based method, we use a baseline model as suggested by [13]. However we have made certain modifications to the rule application phase of this baseline model, deviating from the method suggested. This has been done in order to set a more relaxed baseline for low-sized data, given that we are using deep learning techniques, and also to reduce computation time. This model learns a set of rules for each unique combination of target features. For instance, as an example suggested by the authors of the above cited shared task paper, let us assume a training example with root word ‘schielen’, target word ‘geschielt’ and has a target feature set

'V.PTCP, PST'. The root and target word are aligned as (based on Levenshtein distance):

```
--schielen
geschielt_
```

Next prefix, stem and suffix are extracted from this alignment as follows:

Prefix	Stem	Suffix
--	schiele	n
ge	schielt	-

Based on these pairings, two kinds of rules are extracted: prefix rules and suffix rules. The prefix rule extracted here is to insert the characters 'ge' at the start of the root word. The suffix rules extracted here are to replace the last letter 'e' in the stem of the root word, with the letter 't', and to delete the suffix 'n', of the root word. These 3 rules are now added to other such extracted rules corresponding to the target feature set 'V.PTCP, PST'. Now if a previously unseen test example with root word 'kaufen' and target feature set 'V.PTCP, PST' is encountered, the model searches for a longest match between possible suffixes of 'kaufen' ('n', 'en', 'fen', ...) and a root word suffix that exists in the saved rules for the feature set 'V.PTCP, PST'. For instance, if this longest match is 'en' and is part of the rule that replaces 'en' with 't', then the predicted target word for 'kaufen' is 'kauft'.

5 System Enhancements

5.1 Word vectors and pre-training

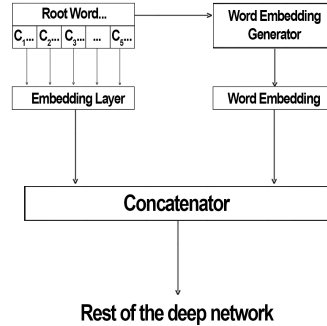


Fig. 7: Incorporation of word vectors

It is not practical to assume that deep learning would give any decent results on datasets of sizes 100 or 1000. Without pre-training, the word vectors are initialized randomly and the onus is completely on the model to train the word

vectors. While such training of word vectors entirely in a task-specific manner is known to give good results on supervised learning tasks, the model does a poor job of determining appropriate embeddings with even moderately small datasets. Our analysis shows that the currently obtained results for low and medium sized datasets can be improved by a good margin if pre-trained word embeddings were used. Due to time constraints, we were unable to perform pre-training of word vectors on all the languages that we had datasets for. However, we ran experiments with pre-trained vectors on the datasets for English. Depending on model configurations, we obtained accuracy improvements between 8-10% for low-sized datasets, between 5-7% on medium-sized datasets, and between 0-2% for high sized datasets. We used the model suggested in [14] for the same. One of the main reasons for this performance boost is that the pre-trained word vectors capture syntactic and morphological information from short neighbouring windows. This also means that word vectors can directly capture analogical relations to do with morphology. An example given in [14] is “dance is to dancing as fly is to __?”. Since the task of morphological inflection is concerned with grammatical constructs such as verb tenses, forms of adjectives etc., it is not surprising that enhanced accuracy is observed using pre-training, as this enables the model to draw analogies between training data and previously unseen data. The decrease in accuracy improvement with increase in dataset size is not surprising, as a larger dataset would have already facilitated the generation of better task-specific word embeddings by the model with lesser contribution from pre-trained vectors. Further we also note that languages like Turkish, Czech, Finnish etc. have a lot more to gain from pre-training as compared to English, since they are much more inflectional morphologically complex than English. For instance, in Archi, a single verb lemma can give rise to 1,502,839 distinct forms. The model uses word vectors, incorporated by concatenation with the character vectors, as shown in Figure 7.

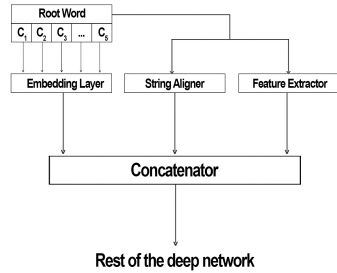


Fig. 8: Hybrid model

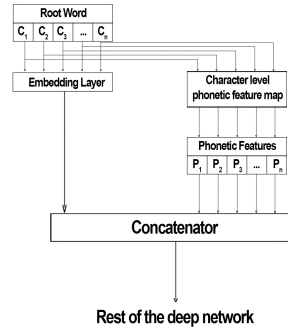


Fig. 9: Incorporation of phonological features

5.2 Hybrid models

As we have already been highlighted, deep learning as a standalone model is inadequate to handle low-sized data scenarios. This issue becomes important to address as one would ideally like to leverage the advantages that a deep learning model has to offer even in the cases of low-resource languages. Pre-training will only help these languages to some extent, as it too requires a large corpus to learn embeddings from. We explored the use of a hybrid model (Figure 8) in the case of low-sized and medium-sized datasets on the English dataset. As mentioned earlier, the conceptual framework for morphological inflection consists of 1) *String alignment between lemma and target*, 2) *Rule extraction from spans of these aligned strings* and 3) *Application of these rule to previously unseen lemmas*. The hybrid system employs traditional methods to address steps 1 and 2, and then feeds the outputs of these traditional methods as inputs to the deep network. Specifically, we borrow from the work of [5] which learns morphological paradigms, by comparing edit operations between the target and the lemma. As an extension to this work, we propose extracting edit rules between the two strings using string alignment algorithms proposed in [5], obtaining a set of features from each lemma-target pair and concatenating the edit rules as well as the features along with the original inputs to the deep net.

1. **String alignment edits:** Rules in this step consist of no-operation, insert, delete and replace. For instance, if the lemma is 'write' and the target is 'had written', one possible set of rules extracted that would have led to each character position of the target string would be as follows: i) insert('h') ii) insert('a') iii) insert('d') iv) insert(' ') v) no-operation vi) no-operation vii) no-operation viii) no-operation ix) insert('t') x) no-operation xi) insert('n') Each of 'no-operation', 'insert', 'delete' and 'replace' are mapped to integers 0-3 respectively (rule types), and a specific rule is encoded as a 3-tuple [rule type, lemma character, target character]_i. The lemma character and target character are the same for no-operations, the lemma character is an empty character for insert, the target character is an empty character for delete and the target and lemma characters are non-empty for replace rule types.
2. **Word features:** In addition to string alignment rules, we also input word specific features for each character, to the neural network. These are as follows: i) Bi-gram and tri-gram contexts of characters in both lemma and target word ii) Distances of characters in the target word from beginning and end of lemma iii) Distances of characters in lemma from beginning and end of target word iv) Characters' index in both lemma and target word (i) and (iv) are especially necessary for low-size dataset configurations, which cannot handle the complexity of an encoder-decoder layer. (ii) and (iv) are required by both medium and low-sized datasets, as these deal with relative with mutual character positions, which the neural network does not handle on its own.

On the low-size dataset, incorporating string alignment rules and word features gave an accuracy improvement of around 1.5% across different configurations. On

the medium-sized dataset, the resulting accuracy improvement was around 0.85% across different configurations. The model for the high-sized dataset did not show any improvements. Figure 8 shows the architecture of this hybrid model.

5.3 Phonological features

It is well known that the phonology of a language interacts closely with its morphology. We used this insight to use phonological or articulatory features as additional inputs to the neural networks. These features are mainly the standard articulatory features such as type (vowel or consonant), place and manner etc. We also use some orthographic features. The details of these features are given in [15]. Each character has a set of 16 phonological features, each having categorical values. The total number of categorical values across all 16 phonological features is 57. The phonological features of each character were encoded in a binary vector of dimension 57, with each position indicating a particular {feature,value} tuple. In order to explore the effect of incorporating character-level phonological features, we ran 2 sets of ablations on the Hindi dataset. In the first experiment, we replaced the characters with phonological features, i.e., each character of a word would be represented by a 57-length binary vector. Doing this gave us an accuracy drop of around 3.4% on the low-sized dataset, 3.2% on the medium-sized dataset and 1.6% on the high-sized dataset. In our second ablation, we concatenated the character-level phonological features along with the character-level embeddings, before passing it to further layers. Doing this improved accuracy on low-sized data by 2.35% and medium-sized data by 0.6% but did not have an effect on high-sized data. Figure 9 shows how the phonological features have been incorporated into the model.

6 Results and Evaluation

Language	B(A)	B(L)	S-1(A)	S-1(L)	S-2(A)	S-2(L)
Norwegian-Bokmal	69	0.489	52.6	0.71	62.7	0.55
Danish	59.8	0.669	46.1	0.95	49.8	0.87
Urdu	30.3	4.201	31.2	2.48	43.7	1.63
Hindi	31	3.798	33.4	2.34	40.8	2.02
Swedish	54.3	0.884	40.6	1.08	39.4	1.09

Table 1: Accuracies for top-5 languages for low data.

6.1 Results on Test Set

The evaluation results were obtained using the evaluation script and the test set provided by the shared task organizers. Baseline accuracies were also obtained

Language	B(A)	B(L)	S-1(A)	S-1(L)	S-2(A)	S-2(L)
Quechua	66.2	1.706	93	0.28	93	0.28
Bengali	71	0.44	91	0.19	91	0.19
Portuguese	78	0.103	86	0.21	89.6	0.16
Urdu	81.1	0.287	88	0.47	88	0.47
Georgian	73	0.225	87.7	0.32	87.7	0.32

Table 2: Accuracies for top-5 languages for medium data.

Language	B(A)	B(L)	S-1(A)	S-1(L)	S-2(A)	S-2(L)
Basque	6	3.32	100.0	0	100.0	0.0
Welsh	67	0.45	99.4	0.01	100.0	0.0
Hindi	94	0.075	99.3	0.02	100.0	0.0
Portuguese	97.4	0.034	98.5	0.03	100.0	0.0
Persian	77.6	0.567	98.9	0.02	99.9	0.01

Table 3: Accuracies for top-5 languages for high data.

from the baseline model provided. The best five baseline accuracies and Levenshtein distances, accuracies and Levenshtein distances for the first submission and accuracies and Levenshtein distances for the second submission can be found in Table 1, Table 2 and Table 3 for each of the three dataset sizes: low, medium and high respectively. In these tables, 'B' stands for Baseline, 'S-1' stands for Submission-1 and 'S-2' stands for Submission-2. 'A' stands for accuracy and 'L' stands for Levenshtein distance.

We have placed the complete set of accuracies and Levenshtein distances for all languages in a Google drive folder², sorted by accuracies. The main observation from these tables is that *languages belonging to the same language family tend to get similar similar results by our system, which is intuitively valid (although there are many exceptions)*. For example, Romance and Slavic languages tend to occur together in these tables. However, it is not evident from these tables that morphologically more complex languages should be harder to learn, which seems to be counter-intuitive. For example, Turkish is above French. This may be because of hyper-parameters or configurations selected for different languages (which were different, in an attempt to maximize accuracy on the development data).

Figures 10 to 12 show the correlation between accuracy and Levenshtein distance for all three sizes of datasets for submission-1.

The results for the other systems in the shared task that we submitted to, and complete details about the shared task can be found in [13]. The reader may refer to these results to see a comparison of how our system performed.

² <https://goo.gl/eW46CC>

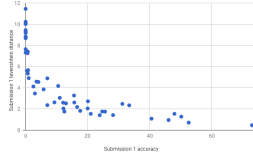


Fig. 10: Accuracy vs. Levenshtein Distance for low data (submission-1)

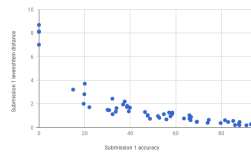


Fig. 11: Accuracy vs. Levenshtein Distance for medium data (submission-1)

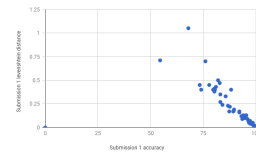


Fig. 12: Accuracy vs. Levenshtein Distance for high data (submission-1)

6.2 Ablation Studies

Early Stop Patience We observed that for low-sized datasets, both the models (LSTM as well as GRU based) required that at least 10 epochs be run before early stop, every time no progress is detected on the validation set. Setting this patience to less than 5, resulted in near 0 accuracies for most languages and printing of nonsensical target words. For medium-sized datasets, this patience value can be set to around 6-8 while for high-sized datasets, it can be set to around 3-4. However, in order to ensure best results, we set our patience value to 10 across all models, training sizes and languages in the final system.

External Feature Categories In last year’s version of the shared task [16], the morphological features in the dataset were annotated along with the category of each feature. For instance, a sample training feature set from last year is: ‘pos=N,def=DEF,case=NOM/ACC/ GEN, num=SG’. This year, however, the category of each feature was not provided, i.e., the same example above would appear in this year’s format as: ‘N,DEF, NOM/ACC/GEN,SG’. Our studies show that while it is conceptually true that the presence of feature categories means exploring a shorter search space, the absence of them does not make a difference to the accuracies obtained for high and medium sized datasets. In the case of low-sized datasets, marginally better accuracies (around 0.5-1%) were obtained when the categories were incorporated into the dataset (this was done manually). However, this might also be the effect of random initialization of parameters.

Choice of Recurrent Unit Simple Recurrent Neural Networks (RNNs) performed the poorest on all sizes of datasets. For low-sized datasets, in almost all cases, using a GRU gave better results than using an LSTM. On an average, the accuracy increased by 2.33% when shifting from LSTM to GRU as the choice of recurrent unit.

In the case of medium-sized datasets, 8 out of 52 languages performed better with an LSTM than a GRU, while the rest showed better performance with a GRU.

Convolutional Layers We also ran experiments using convolutional layers, in which the root word was convolved and the convolution was concatenated along with the root word and passed to the encoder layer (if any). The rest of the network structure remained the same. For low-sized and medium-sized datasets, adding convolutional layers resulted in the accuracy dropping to near 0. For high-sized datasets, we were unable to finish running the experiments on all languages due to lack of time. However for the few languages on which we performed convolutional ablation studies, it did seem to improve accuracy by around 1.5% on an average.

Stacking Recurrent Units Deeper models (more than one layer of LSTM/GRU) resulted in drastic accuracy drops for low-sized datasets. For medium-sized datasets, 30 out of 52 languages showed an accuracy improvement upon stacking two GRU layers, while the accuracy drop in the rest 22 was not drastic but appreciable.

7 Future Work

We aim to perform an exhaustive exploration of language-specific enhancements across all languages in future. Further, we also aim to build separate deep neural networks for different parts of speech (POS) as preliminary experiments show that this gives better performance. In this regard and in the general context of the task of morphological inflection too, we believe that better models can be built by developing a thorough understanding of the morphological nature of each language. While it is true that the techniques described in our work can be used for any language, we hope to customize the configurations by supplementing the word inputs with extra heuristical, rule-based and transduction-based information that are specific to each language. Further insights can be developed by clustering the languages based on accuracy and Levenshtein distance values obtained and correlating these clusters with different classifications. Possible classifications could be done based on the morphological types of languages, namely, analytic, synthetic, agglutinative and polysynthetic. This could not only help in predicting language-specific accuracies for other closely related tasks like morphological disambiguation, but could also help in fine tuning models better for these tasks as well as morphological inflection itself.

8 Conclusions

Different configurations of deep neural networks work well for different languages, based on the morphological properties of the language. It is also true in the context of this task, that phonological features aid the model to perform better. However, standalone deep learning may not be the right approach for low-sized data and hybrid approaches like the one proposed by us give better results.

Further, deep learning can be augmented with other transduction, rule-based or knowledge-based methods, to improve on the results we have achieved.

For high-sized data, we achieved accuracies of 100% for some languages. For medium, the highest was 93% and for low, the highest was 69%.

References

1. Minkov, E., Toutanova, K., Suzuki, H.: Generating complex morphology for machine translation. In: ACL. Volume 7. (2007) 128–135
2. Chahuneau, V., Smith, N.A., Dyer, C.: Knowledge-rich morphological priors for bayesian language models, Association for Computational Linguistics (2013)
3. Oflazer, K., Say, B., et al.: Integrating morphology with multi-word expression processing in turkish. In: Proceedings of the Workshop on Multiword Expressions: Integrating Processing, Association for Computational Linguistics (2004) 64–71
4. Sudhakar, A., Singh, A.K.: Experiments on morphological reinflection: Conll-2017 shared task. Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection (2017) 71–78
5. Durrett, G., DeNero, J.: Supervised learning of complete morphological paradigms. In: Proc. of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Atlanta, Georgia (June 2013) 1185–1195
6. Ahlberg, M., Forsberg, M., Hulden, M.: Semi-supervised learning of morphological paradigms and lexicons. In: Proc. of the 14th Conference of the European Chapter of the Association for Computational Linguistics: Language Technology (Computational Linguistics), Gothenburg, Sweden (April 2014) 569–578
7. Ahlberg, M., Forsberg, M., Hulden, M.: Paradigm classification in supervised learning of morphology. In: Proc. of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado (June 2015) 1024–1029
8. Faruqui, M., Tsvetkov, Y., Neubig, G., Dyer, C.: Morphological inflection generation using character sequence to sequence learning. In: Proc. of NAACL. (2016)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8) (November 1997) 1735–1780
10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014)
11. Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. In: Neural networks: Tricks of the trade. Springer (2012) 437–478
12. Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: Proc. of EMNLP 2014. (2014)
13. Cotterell, R., Kirov, C., Sylak-Glassman, J., Walther, G., Vylomova, E., Xia, P., Faruqui, M., Kübler, S., Yarowsky, D., Eisner, J., Hulden, M.: The CoNLL-SIGMORPHON 2017 shared task: Universal morphological reinflection in 52 languages. In: Proceedings of the CoNLL-SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection, Vancouver, Canada, Association for Computational Linguistics (August 2017)
14. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. Volume 14. (2014) 1532–1543

15. Singh, A.K.: Digitizing The Legacy of Indian Languages. ICFAI Books, Hyderabad (2009)
16. Cotterell, R., Kirov, C., Sylak-Glassman, J., Yarowsky, D., Eisner, J., Hulden, M.: The sigmorphon 2016 shared taskmorphological reinflection. In: Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology. (2016) 10–22