Training Strategies for Neural Multilingual Morphological Inflection

Anonymous ACL-IJCNLP submission

Abstract

1 Introduction

Morphological inflection is the task of transforming a lemma to its inflected form given a set of grammatical features. [Transition to task...] The task requires a model to recognize which morphological processes should be applied to the word, such as affixing, deletion or reduplication, [Typological diversity ...] [Research problems ...]

In this paper we explore different training schemas for neural networks, in a multilingual model for morphological inflection in 37 different languages.

We are particularly interested in how far we can get using a simple LSTM sequence-to-sequence model with attention, augmented with training procedures informed by human heuristics.

In particular, we consider the following training augmentations:

- Curriculum learning: In this module we augment the order in which the examples are presented to the model
- Multi-task learning: We consider the formal operations required to transform a lemma into its inflected form
- Label smooothing: When calculating the loss, some wrong predictions are better than others. We augment the loss function to consider selecting characters from the target language as "better" than selecting characters from other languages.
- Scheduled sampling: During training, we use a probability distribution to determine whether to present the system with the gold character or the previously generated character.

2 Related work

3 Data

The data released cover 37 languages of varying typology and geneology, [stuff...]. The data for the diffrent languges vary gratly in size, presenting a unique challange to multilingual systems.

For the low-resourced languages we use **hallucinated data**. We follow [REF] and consider the parts of a lemma overlapping with the inflected form as candidates for replacement.

4 Method

In this section the multilingual model and training strategies used are presented. We employ a single model with shared parmeters across all languages.

4.1 Model

We employ a encoder-decoder architecture with attention. First we use a LSTM to obtain a contextual representation for each word in the lemma. We encode the tags using a self-attention module, as the order or the tags does not matter.

The encoder has three parts, a LSTM generator and two attention modules, one attending to the lemma and one attending to the tags. For the lemma attention we use a content-based attention module [REF] as it helps facilitate the copy mechanism. [Cosim attention details...] However, only using cosim attention was shown to have pitfalls, as it mainly focused on the most similar characters in the lemma it ignored contextual cues relevant for the generation.

To remedy this, we combine the cosine attention with additive attention as follows, where superscript cos indicate cosine attention, add additive attention and k the key:

$$
\begin{aligned}
a^{add} &= k^{\top}\tanh(W_a h + W_b h)\\
a^{add} &= \text{softmax}(a^{add})\\
att^{add} &= \sum_{t=1}^{T} a_t^{add} h_t^{add}\\
a^{cos} &= \text{softabs}(cos(k,h))\\
att^{cos} &= \sum_{t=1}^{T} a_t^{cos} h_t^{cos}\\
att &= W[att^{cos}; att^{add}]
\end{aligned}
$$

We employ additive attention for the tags. In each step we pass the concatenation of the character embedding obtained from the previous step, the lemma attention and the tag attention to the decoder.

$$\text{input}_t = [e_{t-1}; att_{char}; att_{tag}]$$

4.2 Multi-task learning

We consider the fact that strings may be transformed using operations as an additional resource when generating inflections [REF]. We break down the task of applying levenshtein-distance operations into two sub-tasks: lemma-reductions, where we predict the copy and deletion operations and lemma-additions, where we predict the copy and addition operations.

For each sub-task we predict the operation based on the hidden states generated by our neural network. In the case of lemma-reductions we predict the operation on the hidden-states of the encoded lemma. For lemma-additions, we predict the operations on the generated characters from the decoder.

4.3 Curriculum Learning

We use the easy-first curriculum strategy [REF] to sort the data after each epoch. For all examples in the batch we sort them according to the loss, in ascending order such that the easy (low loss) occur before the difficult examples (high loss).

For the first epoch, when we dont have any loss for the examples, we sort the dataset according to the ratio of copy levenshtein-operations to other operations. We found that this strategy performs the best among set of initial strategies [1]

[1]We experimented with: fewest addition-operations, least-grammatical-features, and random.

4.4 Scheduled Sampling

It has been shown that models trained with teacher-forcing may suffer at inference time, due to not generating new characters given the model output, but rather given gold characters. To address this issue we use scheduled sampling [REF].

We implement a simple schedule for calculating the probability of using the gold characters or hidden stats by using a global sample-probability variable which is updated at each epoch. Each epoch we decrease the initial probability of 100% by 4%. We sample the probability of selecting the gold for each example every time-step.

4.5 Training

We use KL-divergence loss for the characters and cross-entropy loss for both the lemma-reduction and lemma-addition tasks. Our final loss function consists of the character generation loss, the lemma-reduction and the lemma-addition losses summed.

Language-wise Label smoothing We use language-wise label smoothing to calculate the loss. This means that we remove α probability from the correct character and distribute α uniformly across the characters belonging to the words language. A thing to consider is that we calculate the language membership of characters from the training set only, so we don't want to let α be to large, as some characters in the development and test set may not be accounted for.

Learning rate decay with a Curriculum The outputs will be sorted by the diffiulty in the previous epoch, to further follow the easy-first learning idea we employ decaying learning rate. This has the effect that a model update its parameters *more* on the easy examples and less on difficult examples. The idea is that the morphological processes involved in more difficult words can be discovered from the operations involved in the easier examples.

The hyperparameters used for training are presented in Table X below.

5 Results

5.1 Ablation Study

6 Discussion

7 Conclusions

HYPERPARAMETER	VALUE
Batch Size	256
Embedding dim	128
Hidden dim	256
Initial LR	0.001
Min LR	0.0000001
Smoothing-α	2.5%

Table 1: Hyperparameters.

LANGUAGE	ACCURACY

Table 2: Acc