

Morphological Analyzer (Using Deep learning)

-Gaurav Sinha

What is a Morphological Analyzer?

Studying the internal structure
of Words

Tasks

Previously, Morphology learning was based on a semi-supervised setting, where a small set of linguistic gold standard analyses is available. Morfessor Baseline, is a method for unsupervised morphological segmentation. Morfessor Baseline shows linguistic segmentations can be exploited by adding them into the data likelihood function and optimizing separate weights for unlabeled and labeled data.

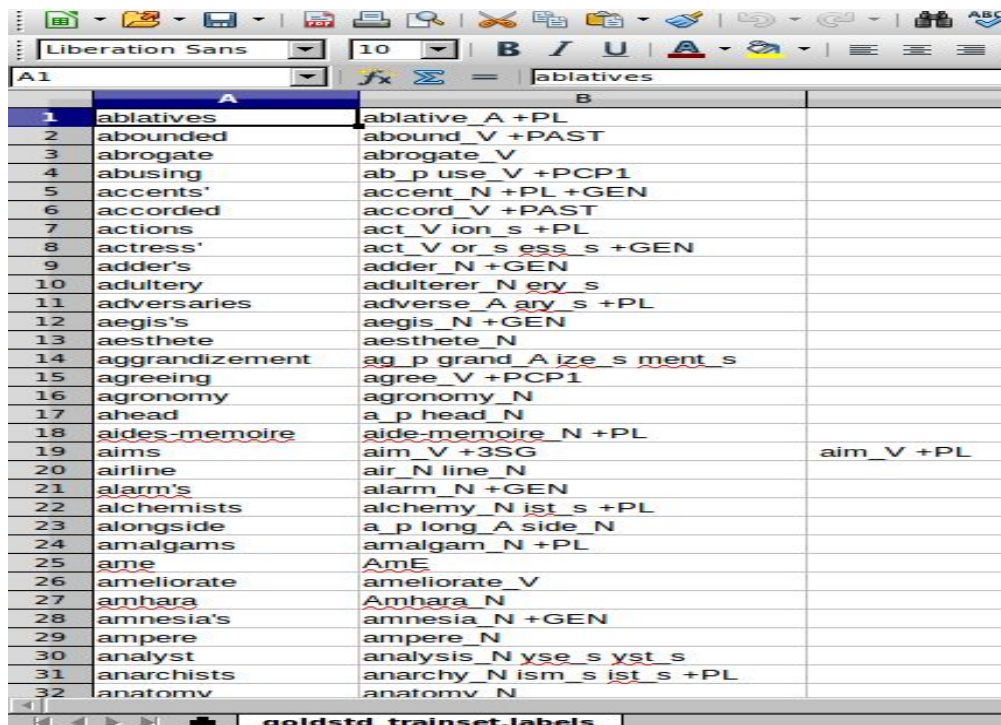
The work deals with Morphological Analysis in an unsupervised setting using state-of-the-art LSTMs

Related Work

- There is surprisingly little work that consider improving the unsupervised models of morphology with small amounts of annotated data. In the related tasks that deal with sequential labeling(word segmentation, POS tagging, shallow parsing, named-entity recognition), semi-supervised learning is more common.
- In semi-supervised learning, the learning system has access to both labeled and unlabeled data. Typically, the labeled data set is too small for supervised methods to be effective, but there is a large amount of unlabeled data available
- Generative Models , Descriptive Models have been used in Morpho Challenges (2005-2010).
- Character-based representations have been explored in other NLP tasks; for instance, dos Santos and Zadrozny (2014) and dos Santos and Guimarães (2015) learned character-level neural representations for POS tagging and named entity recognition, getting a large error reduction in both tasks.

Dataset

The English gold standards data are based on the [CELEX data base](#).



	A	B	
1	ablative	ablative_A +PL	
2	abounded	abound_V +PAST	
3	abrogate	abrogate_V	
4	abusing	ab_p use_V +PCP1	
5	accents'	accent_N +PL +GEN	
6	accorded	accord_V +PAST	
7	actions	act_V ion_s +PL	
8	actress'	act_V or_s ess_s +GEN	
9	adder's	adder_N +GEN	
10	adultery	adulterer_N ery_s	
11	adversaries	adverse_A ary_s +PL	
12	aegis's	aegis_N +GEN	
13	aesthete	aesthete_N	
14	aggrandizement	ag_p grand_A ize_s ment_s	
15	agreeing	agree_V +PCP1	
16	agronomy	agronomy_N	
17	ahead	a_p head_N	
18	aides-memoire	aide-memoire_N +PL	
19	aims	aim_V +3SG	aim_V +PL
20	airline	air_N line_N	
21	alarm's	alarm_N +GEN	
22	alchemists	alchemy_N ist_s +PL	
23	alongside	a_p long_A side_N	
24	amalgams	amalgam_N +PL	
25	ame	AmE	
26	ameliorate	ameliorate_V	
27	amhara	Amhara_N	
28	amnesia's	amnesia_N +GEN	
29	ampere	ampere_N	
30	analyst	analysis_N yse_s yst_s	
31	anarchists	anarchy_N ism_s ist_s +PL	
32	anatomy	anatomy_N	

goldstd trainset.labels

Methodology Used for Semi-supervised & Unsupervised Learning

Extending of the Morfessor Baseline method to the semi-supervised setting. Morfessor Baseline is based on a generative probabilistic model. It is a method for modeling concatenative morphology, where the morphs—i.e., the surface forms of morphemes—of a word are its non-overlapping segments.

The model parameters θ encode a morph lexicon, which includes the properties of the morphs, such as their string representations. Each morph m in the lexicon has a probability of occurring in a word, $P(M = m \mid \theta)$.

During model learning, θ is optimized to maximize the posterior probability:

$$\theta^{\text{MAP}} = \arg \max (P(\theta \mid D_w))$$

- Likelihood:

- The latent variable of the model, $Z = (Z_1, \dots, Z_{|D_W|})$, contains the analyses of the words in the training data D_W . The conditional likelihood is :

$$\begin{aligned}
 P(D_W | Z = z, \theta) \\
 &= \prod_{j=1}^{|D_W|} P(W = w_j | Z = z, \theta) \\
 &= \prod_{j=1}^{|D_W|} \prod_{i=1}^{|z_j|} P(M = m_{ji} | \theta),
 \end{aligned}$$

where m_{ij} is the i :th morph in word w_j .

- Priors:

- Morfessor applies Maximum A Posteriori (MAP) estimation, so priors for the model parameters need to be defined. The parameters θ of the model are:
 - Morph type Count , size of morph lexicon.
 - Morph token Count
 - Morph Strings

MDL-inspired and non-informative priors have been preferred. When using such priors, morph type count and morph token counts can be neglected when optimizing the model. The morph string prior is based on length distribution $P(L)$ and distribution $P(C)$ of characters over the character set Σ , both assumed to be known:

$$P(\sigma_i) = P(L = |\sigma_i|) \prod_{j=1}^{|\sigma_i|} P(C = \sigma_{ij})$$

- **Segmenting new words:**

- After training the model, a Viterbi-like algorithm can be applied to find the optimal segmentation of each word.

- **Deep learning:**

- It presents extensions to a continuous state dependency parsing method that makes it applicable to morphologically rich languages. Starting with a high-performance transition-based parser that uses long short-term memory (LSTM) recurrent neural networks to learn representations of the parser state, we replace lookup-based word representations with representations constructed from the orthographic representations of the words, also using LSTMs. This allows statistical sharing across word forms that are similar on the surface.

Experiment

- At the heart of natural language parsing is the challenge of representing the “state” of an algorithm—what parts of a parse have been built and what parts of the input string are not yet accounted for -as it incrementally constructs a parse.
- Continuous-state parsers have been proposed, in which the state is embedded as a vector. The idea of continuous state parsing a step further by making the word embeddings that are used to construct the parse state sensitive to the morphology of the words.
- Building an LSTM dependency parser:
 - Like most transition-based parsers, Parser can be understood as the sequential manipulation of three data structures: a buffer B initialized with the sequence of words to be parsed, a stack S containing partially-built parses, and a list.A of actions previously taken by the parser.

At each time step t , a transition action is applied that alters these data structures by pushing or popping words from the stack and the buffer, the operations are listed in Figure 1.

Stack_t	Buffer_t	Action	Stack_{t+1}	Buffer_{t+1}	Dependency
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-RIGHT(r)	$(g_r(\mathbf{u}, \mathbf{v}), u), S$	B	$u \xrightarrow{r} v$
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	REDUCE-LEFT(r)	$(g_r(\mathbf{v}, \mathbf{u}), v), S$	B	$u \xleftarrow{r} v$
S	$(\mathbf{u}, u), B$	SHIFT	$(\mathbf{u}, u), S$	B	—
$(\mathbf{u}, u), (\mathbf{v}, v), S$	B	SWAP	$(\mathbf{u}, u), S$	$(\mathbf{v}, v), B$	—

Figure 1: Parser transitions indicating the action applied to the stack and buffer and the resulting stack and buffer states. Bold symbols indicate (learned) embeddings of words and relations, script symbols indicate the corresponding words and relations. Dyer et al. (2015) used the SHIFT and REDUCE operations in their continuous-state parser; we add SWAP.

Along with the discrete transitions above, the parser calculates a vector representation of the states of B , S , and A ; at time step t these are denoted by \mathbf{b}_t , \mathbf{s}_t and \mathbf{a}_t , respectively. The total parser state at t is given by

$$\mathbf{p}_t = \max \{0, W[\mathbf{s}_t; \mathbf{b}_t; \mathbf{a}_t] + \mathbf{d}\}$$

where the matrix W and the vector \mathbf{d} are learned parameters.

Stack LSTM

RNNs are functions that read a sequence of vectors incrementally; at time step t the vector \mathbf{x}_t is read in and the hidden state \mathbf{h}_t computed using \mathbf{x}_t and the previous hidden state \mathbf{h}_{t-1} . In principle, this allows retaining information from time steps in the distant past, but the nonlinear “squashing” functions applied in the calculation of each \mathbf{h}_t result in a decay of the error signal used in training with backpropagation.

Past work explains the computation within an LSTM through the metaphors of deciding how much of the current input to pass into memory (\mathbf{i}_t) or forget (\mathbf{f}_t). We refer interested readers to the original papers and present only the recursive equations updating the memory cell \mathbf{c}_t and hidden state \mathbf{h}_t given \mathbf{x}_t , the previous hidden state \mathbf{h}_{t-1} , and the memory cell \mathbf{c}_{t-1} :

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i)$$

$$\mathbf{f}_t = \mathbf{1} - \mathbf{i}_t$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} +$$

$$\mathbf{i}_t \odot \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

where σ is the component-wise logistic sigmoid function and \odot is the component-wise (Hadamard) product. Parameters are all represented using \mathbf{W} and \mathbf{b} .

Predicting Parser Decisions:

The parser uses a probabilistic model of parser decisions at each time step t . Letting $A(S, B)$ denote the set of allowed transitions given the stack S and buffer B (i.e., those where preconditions are met; see Figure 1), the probability of action $z \in A(S, B)$ defined using a log-linear distribution

$$p(z \mid \mathbf{p}_t) = \frac{\exp(\mathbf{g}_z^\top \mathbf{p}_t + q_z)}{\sum_{z' \in A(S, B)} \exp(\mathbf{g}_{z'}^\top \mathbf{p}_t + q_{z'})}$$

(where \mathbf{g}_z and q_z are parameters associated with each action type z)

Baseline: standard word embeddings: Parser generates a word representation for each input token by concatenating two vectors: a vector representation for each word type (w) and a representation (t) of the POS tag of the token (if it is used), provided as auxiliary input to the parser. 2 A linear map (V) is applied to the resulting vector and passed through a component-wise ReLU:

$$x = \max \{0, V[w; t] + b\}$$

This mapping can be shown schematically as in Figure 2.

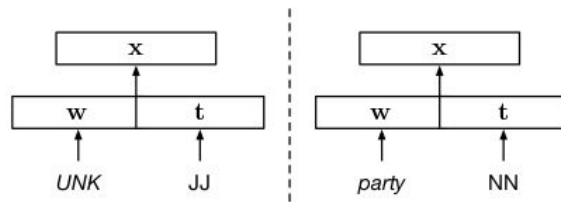


Figure 2: Baseline model word embeddings for an in-vocabulary word that is tagged with POS tag NN (right) and an out-of-vocabulary word with POS tag JJ (left).

Character-Based Embeddings of Words: When the parser initiates the learning process and populates the buffer with all the words from the sentence, it reads the words character by character from left to right and computes a continuous-space vector embedding the character sequence, which is the h vector of the LSTM; we denote it by w .

$$x = \max\{0, V[w;w;t] + b\}$$

The process is shown semantically in figure 3.

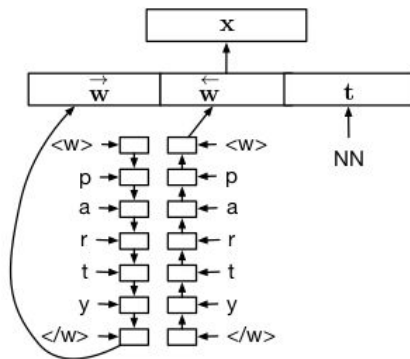


Figure 3: Character-based word embedding of the word party. This representation is used for both in-vocabulary and out-of-vocabulary words.

Configuration: In order to isolate the improvements provided by the LSTM encodings of characters, we run the stack LSTM parser. LSTM Hidden states are of size 100, and using two layers of LSTMs for each stack. Embeddings of the parser actions used in the composition functions have 20 dimensions, and the output embedding size is 20 dimensions.

Tools Used

- Python 3
- Jupyter
- C++
- Keras
- Tensor Flow Backend

Discussion

The method developed as the basis of Morfessor Baseline (at Morpho Challenge 2010) gives the accuracy around 70 % as shown in figure 4.

English			
<i>labeled data</i>	<i>F0.5</i>	<i>F1</i>	<i>F2</i>
0	69.16	61.05	62.70
100	73.23	65.18	68.30
300	72.98	65.63	68.81
1000	71.86	68.29	69.68
3000	74.34	69.13	72.01
10000	76.04	72.85	73.89

Finnish			
<i>labeled data</i>	<i>F0.5</i>	<i>F1</i>	<i>F2</i>
0	56.81	49.07	53.95
100	58.96	52.66	57.01
300	59.33	54.92	57.16
1000	61.75	56.38	58.24
3000	63.72	58.21	58.90
10000	66.58	60.26	57.24

Table 1: The F0.5, F1 and F2 measures for the *semi-supervised + weighting* method.

	Segmented	Labeled
English, $D = 1\,000$		
Precision	69.72%	69.30%
Recall	66.92%	72.21%
F-measure	68.29%	70.72%
English, $D = 10\,000$		
Precision	77.35%	77.07%
Recall	68.85%	77.78%
F-measure	72.86%	77.42%
Finnish, $D = 1\,000$		
Precision	61.03%	58.96%
Recall	52.38%	66.55%
F-measure	56.38%	62.53%
Finnish, $D = 10\,000$		
Precision	69.14%	66.90%
Recall	53.40%	74.08%
F-measure	60.26%	70.31%

Table 3: Results of a simple morph labeling after segmentation with semi-supervised Morfessor.

Figure 4: Results using Semi Supervised Learning

Hypothesis

- The Stack based LSTM model is supposed to outperform the Semi-supervised / Unsupervised Learning Model with an accuracy around 90% as predicted from early stages of development.
- The character based embedding using the word2vector model is also likely to show graphical results of inflections
- The character-based representation for words is notably beneficial for out-of-vocabulary (OOV) words.
- Both labeled and unlabeled attachment scores would have significant improvement.

Conclusion

- In this project ,I have tried to find several interesting findings. First, adding new evidence that character-based representations are useful for NLP tasks. In this project, the focus is to implement model useful for transition-based dependency parsing, since they are capable of capturing morphological information crucial for analyzing syntax.
- The improvements suggested by the character-based representations using bidirectional LSTMs are strong for languages supporting the POS tagging.
- The correct prediction of OOV rates is supposed to be higher.

Requirement & Schedule

- The project requires a detailed study of Recurrent Neural networks and the state of art models using semi-supervised learning and unsupervised learning.
- The expected date to complete the project is September 2017.

Thankyou