Unsupervised Morphological Segmentation with Recursive Neural Network

Minh-Thang Luong – CS224N/CS229 - Final Project Report

1. Introduction

Recent works have been successful in applying Recursive Neural Network (RNN) architectures to predict hierarchical tree structures of scene images and natural language sentences (Socher et al., 2010; 2011). In this project, we focus on the natural language modality and explore how RNNs could address the morphological segmentation problem.

Motivated by (Socher et al., 2010; 2011)'s work in syntactic parsing of natural language sentences, where the input is a sequence of words, our goal is to learn similar hierarchical parse trees but for words instead, treating each character as a unit. By recursively grouping characters together, we aim to achieve unsupervised learning of not only the shallow morphological segmentation, i.e. breaking words into morphemes, but also the deep structure of word formations.

Unlike them, we explore learning the segmentation task in an unsupervised manner. Two novel types of information, lexical and structural, are proposed to incorporate into the RNN that helps boost performance.

The report is organized as follow. Section 2 formulates the RNN architecture, while details of the back-propagation process are given Section 4. Unsupervised learning is described in Section 3, followed by our discussion in Section 5 about incorporating lexical and structural information into the RNN. Experimental setup and results are given in Section 6. We suggest future work in Section 7, and conclude in Section 8.

2. Recursive Neural Network (RNN)

2.1. Representation

Let V be an ordered set of all characters in a language, which is parameterized by the matrix $\boldsymbol{W^c}$ of size $d \times |V|$. Specifically, the ith character is in d-dimensional space, represented by the i^{th} column of $\boldsymbol{W^c}$.

2.2. Structure Prediction Formulation

Suppose the RNN parameters have been learned (details in Section 3), we discuss how the most probable parse tree for a word could be derived from the RNN.

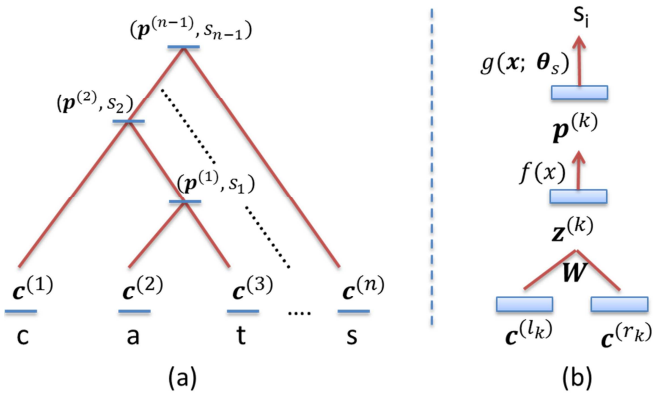

Figure 1. **Recursive Neural Network architecture: (a)** – The recursive process of constructing $(n-1)$ new nodes $[\boldsymbol{p}^{(1)}, \ldots, \boldsymbol{p}^{(n-1)}]$ from the original character nodes $[\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(n)}]$ of a word, e.g. "cats". **(b)** – The merging process of combining children nodes $[\boldsymbol{c}^{(l_k)}; \boldsymbol{c}^{(r_k)}]$ into a parent node $\boldsymbol{p}^{(k)}$ with a local decision score s_k.

Let x be a word of length n, i.e. $x = \overline{c_1 \ldots c_n}$, and a parse tree y corresponds to an ordered set of (n-1) local decisions in the form of merging triplets, $[\boldsymbol{c}^{(l_k)}\boldsymbol{c}^{(r_k)} \rightarrow \boldsymbol{p}^{(k)}]$, where l_k and r_k are indices of the left and right children respectively. The parent node $\boldsymbol{p}^{(k)}$ is also considered as $\boldsymbol{c}^{(n+k)}$. At the end of the merging process, the RNN tree will have a total of $2n-1$ nodes, with $[\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(n)}]$ being the original nodes (Figure 1). The score for each local decision is denoted as s_k and computed as:

$$s_k = g\left(\boldsymbol{p}^{(k)}; \boldsymbol{\theta}_s\right) \quad (1)$$

$$\boldsymbol{p}^{(k)} = f(\boldsymbol{z}^{(k)}) \quad (2)$$

$$\boldsymbol{z}^{(k)} = \boldsymbol{W}[\boldsymbol{c}^{(l_k)}; \boldsymbol{c}^{(r_k)}; 1] \quad (3)$$

where $\boldsymbol{\theta}_s$ and $\boldsymbol{W}$ are part of the RNN parameters. $\boldsymbol{W}$ is of size $d \times (2d+1)$, in which d is the dimension of character vectors. We use *tanh* as our activation function f. For flexibility, we abstractly denote the score function as g which takes in parameters $\boldsymbol{\theta}_s$.

The score s(x, y) of a word x and a parse tree y is

simply the score sum of all the local decisions:

$$s(x, y) = \sum_{k=1}^{n-1} s_k \tag{4}$$

Lastly, given $T(x)$ be the set of candidate parsed trees for a word x, its RNN score is $\max_{y \in T(x)} s(x, y)$.

Note: Follow (Socher et al., 2010; 2011), we use $g(\boldsymbol{x}; \boldsymbol{\theta}_s) = (\boldsymbol{\theta}_s)^\top \boldsymbol{x}$.

3. Unsupervised Learning

3.1. Training Examples

Our input is a set of m distinct words $X = \{x^{(1)}, \ldots, x^{(m)}\}$ in a language, which we will treat as *positive* training examples.

To employ unsupervised learning, *negative* examples are artificially constructed by corrupting the input words. Given an input word $x^{(i)}$, one way to corrupt it is to randomly select a position and replace the character at that position with a newly random one. Other options could be to change multiple characters at a time, or scramble the characters of $x^{(i)}$, or both. We opt to use the former option of corrupting one character at a time to create a controlled setting that could teach the RNN gradually. In fact, initial experiments show that the system learns poorly when multiple mutations take place at the same time.

3.2. Subgradient Methods

From Section 2, our parameters include $\boldsymbol{\theta} = \{\boldsymbol{W^c}, \boldsymbol{\theta}_s, \boldsymbol{W}\}$. Follow (Collobert & Weston, 2008), the cost function is designed so that the RNN will be optimized towards giving higher scores for correct word forms while penalizing corrupted ones. Specifically, our ranking-type cost attempts to boost the score of each positive example $x^{(i)}$ up to a margin α (set to 0.1 experimentally) towards its negative example $\overline{x}^{(i)}$:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \max\left(0, \alpha - s(x^{(i)}) + s(\overline{x}^{(i)})\right) \tag{5}$$

Due to hinge loss, the objective function J is not differentiable. Hence, subgradient method (Ratliff et al., 2007) is employed instead of gradient-ascent ones:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{m} -\frac{\partial s\left(x^{(i)}\right)}{\partial \boldsymbol{\theta}} + \frac{\partial s(\overline{x}^{(i)})}{\partial \boldsymbol{\theta}} \tag{6}$$

For a word x, to compute $\frac{\partial}{\partial \boldsymbol{\theta}} s(x)$, we first decode x to find the best tree y^*. We have, from Eq. (4), $\frac{\partial}{\partial \boldsymbol{\theta}} s(x) = \sum_{k=1}^{n-1} \frac{\partial}{\partial \boldsymbol{\theta}} s_k$, where the gradients for each local decision are addressed in Section 4.

Note: we approximate the best parse tree y^* by performing a greedy search as in (Socher et al., 2010), which iteratively finds a pair of adjacent nodes with the highest score and combine them to yield a new set of adjacent nodes for the next iteration. Context-aware greedy search was experimented as well; however, without any further information about the word structure or the morpheme distribution, the search continues cluelessly, resulting in parameters that just do not correlate with the segmentation task objective. We further justify these in the later sections.

4. Back-propagation Through Structure

To compute the gradients for each local decision score s_k, backpropagation through structure (BTS) (Goller & Küchler, 1996) is employed.

4.1. "Error-term" Derivation

As an intermediate step, we define the "error terms" $\boldsymbol{\delta}_k^{(h)} = \frac{\partial}{\partial \boldsymbol{z}^{(h)}} s_k$, related by a recursive formula[1]:

$$\boldsymbol{\delta}_k^{(h)} = \begin{cases} f'\left(\boldsymbol{z}^{(h)}\right) \circ ((\boldsymbol{W}^{(1)})^\top \boldsymbol{\delta}_k^{(p(h))}) & \text{if left split} \\ f'\left(\boldsymbol{z}^{(h)}\right) \circ ((\boldsymbol{W}^{(2)})^\top \boldsymbol{\delta}_k^{(p(h))}) & \text{if right split} \end{cases} \tag{7}$$

for $h \prec k$, where $\boldsymbol{W} = [\boldsymbol{W}^{(1)} \boldsymbol{W}^{(2)} \boldsymbol{b}]$. At the base case:

$$\boldsymbol{\delta}_k^{(k)} = f'\left(\boldsymbol{z}^{(k)}\right) \circ \boldsymbol{\theta}_s \tag{8}$$

4.2. BTS Gradient Formulae

The gradients of the model parameters, $\boldsymbol{\theta} = \{\boldsymbol{W^c}, \boldsymbol{\theta}_s, \boldsymbol{W}\}$ with respect to s_k could be derived as:

$$\frac{\partial s_k}{\partial \boldsymbol{W}} = \boldsymbol{\delta}_k^{(k)} [\boldsymbol{c}^{(l_k)}; \boldsymbol{c}^{(r_k)}; 1]^\top \tag{9}$$

$$\frac{\partial s_k}{\partial \boldsymbol{\theta_s}} = \boldsymbol{p}^{(k)} \tag{10}$$

For $\boldsymbol{W^c}$, let $\boldsymbol{w}^c$ be the column vector representing for a character c. Let $\{\boldsymbol{p}^{(i_1)}, \ldots, \boldsymbol{p}^{(i_h)}\}$ be all the nodes under the subtree rooted at $\boldsymbol{p}^{(k)}$ (inclusive), which are arranged in the order added by the RNN, Using the chain rule technique for ordered derivatives suggested in (Werbos, 1990), we have:

$$\frac{\partial}{\partial \boldsymbol{w}^c} s_k = \sum_{j=1}^{h} \frac{\partial^*}{\partial \boldsymbol{w}^c} \boldsymbol{z}^{(i_j)} \cdot \frac{\partial}{\partial \boldsymbol{z}^{(i_j)}} s_k = \sum_{j=1}^{h} \frac{\partial^*}{\partial \boldsymbol{w}^c} \boldsymbol{z}^{(i_j)} . \boldsymbol{\delta}_k^{(i_j)}$$

[1] $h \preceq k$ indicates that the node $\boldsymbol{p}^{(h)}$ is part of the tree rooted at $\boldsymbol{p}^{(k)}$. $\boldsymbol{p}^{(h)}$ is a child of the node $\boldsymbol{p}^{(p(h))}$.

Len	Type	Token	"Morpheme" subsequences							
2	400	9418	es	231	ing	128	tion	43	ation	31
3	2232	8123	in	222	ion	54	atio	31	tions	14
4	4167	6829	er	191	ati	51	ness	24	ating	12
5	4425	5550	re	168	ers	48	ting	22	iness	8
6	3809	4316	ed	158	ate	45	ling	16	ities	7
7	2927	3168	ng	150	ess	45	ions	15	ement	7
8	2060	2179	at	145	tio	44	ator	14	house	7
9	1343	1399	ti	141	ies	38	ally	13	alize	7
10	805	829	te	134	ent	36	ates	12	ously	6

Table 1. **Training Data Statistics & Examples:** (left) – the type and token counts of "morpheme" subsequences of lengths 2 to 10 characters; (right) the top frequent "morpheme" subsequences of lengths 2 to 5 with their token counts.

Here ∂^* indicates "simple" derivatives that ignore the fact that $\boldsymbol{z}^{(i_j)}$ depends on $\boldsymbol{z}^{(i_1)}, \ldots, \boldsymbol{z}^{(i_{j-1})}$:

$$\frac{\partial^* \boldsymbol{z}^{(i_j)}}{\partial \boldsymbol{w}^c} = \begin{cases} (\boldsymbol{W}^{(1)})^\top & c = c^{(l_{i_j})} \neq c^{(r_{i_j})} \\ (\boldsymbol{W}^{(2)})^\top & c = c^{(r_{i_j})} \neq c^{(l_{i_j})} \\ \left((\boldsymbol{W}^{(1)})^\top + (\boldsymbol{W}^{(2)})^\top\right) & c = c^{(l_{i_j})} = c^{(r_{i_j})} \\ \mathbf{0} & \text{Otherwise} \end{cases} \tag{11}$$

5. Morpheme Prior and Word Structure Information

5.1. Character Subsequence Distribution

The RNN could potentially learn the distributional representation of characters through $\boldsymbol{W}^{\boldsymbol{c}}$, and their compositional patterns by means of the parameters $\boldsymbol{W}$. However, it is not clear how the cost function could drive the model towards optimizing the morphological segmentation task. Especially in the context of unsupervised learning, without labeled data, the model needs some form of prior information about what could possibly be a morpheme unit and what may not, to help it bootstrap the learning process. Hence, we incorporate the distribution of morpheme subsequences into the model objective.

Specifically, we first collect counts of all character subsequences of each word across the entire dataset. After that, ML estimates conditioned on the subsequence length, in terms of characters, are derived, which gives us the prior probabilities $\text{lex}(m)$ of each "morpheme" subsequence m. Such lexical information is incorporated into model by modifying Eq. (1) to become:

$$s_k = g\left(\boldsymbol{p}^{(k)}; \boldsymbol{\theta}_s\right) + \beta\, \text{lex}(\boldsymbol{p}^{(k)})$$

where β is the lexical weight, which we set to 10 experimentally. Table 1 gives the training set statistics on the type and token counts of "morpheme" subsequences with some examples of the top frequent ones.

5.2. Minimum-Height Parse Trees

Linguistically, words are constructed by building up from minimal meaning-bearing units, which are morphemes. However, the RNN has no clue about the underlying structure of a word, which we could think of as a hidden layer of morpheme labels. As a result, it could end up in a wrong left-skewed parse tree as in Figure 2(a). To alleviate this problem, we enforce a minimum-height constraint on each node. Specifically, recall that a binary tree of n nodes will have a minimum height of $([log_2(n-1)]+1)$. Hence, for each subtree rooted at a parent node p, spanning over n leaf nodes and possessing a height of h, the structural score of that parent node will be $\text{struct}(p) = ([log_2(n-1)]+1-h)$.

We incorporate such structural scores into the RNN model similar to the addition of lexical information in the previous section. Note, however, that only nodes in the parse trees of the positive training examples will include structural scores. As these scores are non-positive, such structural constraints are expected to prevent the RNN from considering unbalanced trees for "good" words. Eq. (1) for the positive training examples is extended to:

$$s_k = g\left(\boldsymbol{p}^{(k)}; \boldsymbol{\theta}_s\right) + \beta\, \text{lex}(\boldsymbol{p}^{(k)}) + \gamma\, \text{struct}(\boldsymbol{p}^{(k)})$$

where γ is the structural weight and set to 0.1 experimentally. While ad-hoc in nature, this method, to some extent, has helped alleviated the problem. Figure 2(b) shows an example in which our model could find a correct tree with a more balanced structure.

6. Experiments and Results

6.1. Data and Evaluation Metrics

We test our model on the English portion of the morphological segmentation data are publicly available at

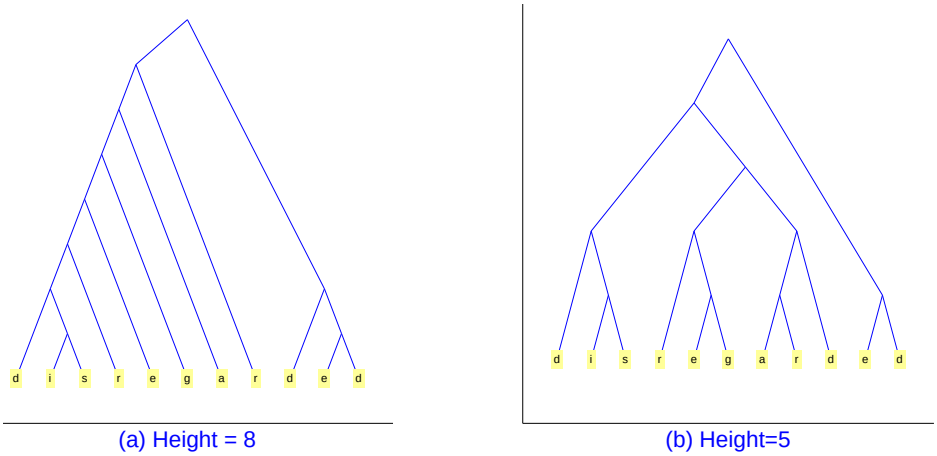

Figure 2. **Parse trees of the word "disregarded"**, shown are: (a) a wrong left-skewed parse tree of height 8 and (b) a correct parsed tree of height 5.

the *Morpho Challenge 2010 - Semi-supervised and Unsupervised Analysis* website. There are a total of 1686 words with gold-standard segmentation, and those do not contain punctuation, e.g., quotes or hyphens, are retained, which leaves us with a dataset of 1295 words. We train and test on the same set, but no segmentation information is used during training.

Without label information, we need to approximately evaluate how good or bad a parse tree is, with respect to the true segmentation of a word, e.g., "dis regard ed". To do that, a simple metric is designed to look at indices of the word span that each node in a tree covers, and compare them with the true segmentation spans. In our example, the true segmentation results in spans [1-3], [4-9], and [10-11], while those of the tree in Figure 2(b) are [1-1], [2-2], [3-3], [2-3], [1-3], etc.

If a span of the correct segmentation appears in the tree spans, a *segment score* of one is given. If all correct spans of a word are in the tree spans, a *word score* of one is awarded. Normalizing the segment and word scores by the total number of correct spans and words respectively gives the *segmentation* and *word* accuracy.

6.2. Results

Our final segmentation results are presented in Table 2 in terms of word and segment accuracy. We start with a baseline where a greedy search strategy is used without any other information, which gives 23.38% and 37.76% in word and segment accuracy respectively.

At this point, one might wonder, by means of our metrics, how a random system would score. To make it fair, we give some explicit figures here: there are 262 words without any segmentation out of 1295 words, and 387 segments with a single character out of 2733 correct segmentations. These will give a random system a score of $262/1295 = 20.23\%$ in word accuracy and $(262 + 387)/2733 = 23.75\%$ in segment accuracy.

	Word	**Segment**
Greedy	23.38%	37.76%
Greedy+lex	27.08%	48.65%
Greedy+lex+struct	30.48%	53.25%
Context+lex+struct	**37.19%**	**56.18%**

Table 2. **Experimental results**: shown are the word and segment accuracy of (a) the base system using greedy search, (b) adding lexical information, (c) constraining on the word structure, and (d) greedy, context-aware search with lexical and structural information.

Gradually adding the lexical information and constraining on the word structure consistently improve the system performance, with a gain of 7.10% and 15.49% in absolute word and segment accuracy respectively compared to the baseline. Context-aware greedy search further boosts the results by another 6.71% in absolute word accuracy and 2.93% in absolute segment accuracy, achieving the best performance of 37.19% and 56.18% in word and segment accuracy.

6.3. Discussion

Compared to the performance of state-of-the-art unsupervised methods, in which the best result for English has (Precision, Recall, F-Measure) of (80.77%, 53.76%, 64.55%)[2], our performance is considered modest. However, it is worth noting that we have not made

[2] `http://research.ics.tkk.fi/events/morphochallenge2010/comp1-results.shtml`

use of the full word list of 878K words with frequency information that these unsupervised methods utilize.

At the same time, we only model each character with a 100-dimensional vector, while in principle, it could be scaled up to a much larger value since our vocabulary size is small (only 26 characters). Due to time constraint, the vector dimension is kept small so that we could experiment with different training parameter configurations. In reality, the set of parameter combinations to choose from is enormous, which makes it very tricky to get the RNN to work. As such, it is encouraging to start seeing the model produces sensible analyses though with errors. Sample segmentations of several long words in the dataset are shown in Table 3.

Gold	**RNN**
anthrop olog ical	anthrop olog ical
collect iv iz ation	collecti viz ation
co religion ist s	core ligion ist s
rational iz ation	ration al iz ation
re conciliat ion s	re conc iliat ion s
respons ibil iti es	respons ibilit ies
transmogrific ation	trans mogrifi cation

Table 3. Sample segmentations of several long words in the dataset: shown are the true and automatic ones.

7. A Final Thought - Future Work

While adding lexical and structural information does help limiting the search space during parsing, it does not, however, directly involve in the optimization process. As a result, the RNN sometimes could scale up the parameter values, diminishing the lexical and structural influence. Hence, we suggest for future work an approach that tackles the problem of finding the underlying structure of words.

In (Creutz & Lagus, 2002), a codebook of all morphemes m_i in the data is maintained, and an EM-like algorithm was used to minimized a cost function as the data likelihood: cost(data) = $\sum_{\text{morph tokens } m_i} - \ln p(m_i)$. Given segmented data, $p(m_i)$ could be reestimated using ML estimates.

We mimic by maintaining list of morphemes, which will be iteratively updated, and introduce a logistic layer for the RNN to give each node a probabilistic score of being a morpheme unit or not. Given parameters of the RNN, morpheme labels could be inferred for a tree by labeling nodes with the classification scores sorted in descending order; each subtree will be ignored once its root node has been labeled. For each morpheme-labeled node, a term -ln $p(m)$, where m is the character sequence that node covers, will be added to its score formula in Eq. (1), which essentially incorporates the data likelihood into the RNN cost function.

8. Conclusion

In this project, we have achieved a better understanding of how RNN could be applied and "interact" with the morphological segmentation task, especially in unsupervised context. While the performance is modest, we have demonstrated the effectiveness of using morpheme subsequence distribution and tree height constraint. By making the RNN cost function better correlated with the task objective, these additional information has yielded non-trivial improvements for the task. Capturing the underlying structure of words in a more principle manner is a worthy goal that we plan to pursuit in future work.

Acknowledgement: we thank Richard Socher for providing the code base and feedbacks for the project.

References

Collobert, R. and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 2008.

Creutz, Mathias and Lagus, Krista. Unsupervised discovery of morphemes. In *Workshop on Morphological and Phonological Learning of ACL*, 2002.

Goller, C. and Küchler, A. Learning task-dependent distributed representations by backpropagation through structure. *IEEE Transactions on Neural Networks*, 1:347–352, 1996.

Ratliff, Nathan D., Bagnell, J. Andrew, and Zinkevich, Martin A. (online) subgradient methods for structured prediction, 2007.

Socher, Richard, Manning, Christopher, and Ng, Andrew. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. In *NIPS*2010 Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

Socher, Richard, Lin, Cliff C., Ng, Andrew Y., and Manning, Christopher D. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *ICML*, 2011.

Werbos, P. Back propagation through time: What it does and how to do it. In *Proceedings of the IEEE*, volume 78, pp. 1550–1560, 1990.