# Exploring Carl de Marcken's Word Segmentation

Brian Hempel and Yuxi Chen

June 8, 2016

## Intro

Give a coding scheme and a particular lexion, theoretically, it's possible to calculate the minimum length encoding of this given input. Part of the encoding will be devoted to the lexicon, the rest to representing the input in terms of the lexicon. The minimum description length (MDL) priciple was introduced by Jorma Rissanen, which tries to describe data using fewer symbols than needed to describe the data literally given a set of training data. As applied in linguistic filed, one can hope that pattern in the lexiocn reflect the underlying mechanisms and parameters of the language that generated the input.

Carl de Marcken, in his PhD thesis, Unsupervised Language Acquisition, simulated children's learning processing of natural language, which maps from complex physical signals to grammar that enable them to generate and interpret new utterances. Mainly, it argues how word/grammar are built by perturbing a composition of existing parameters via statistical and linguistical methodology. Learning are expressed at the computational level is the search for the grammar that minimizes the combined description length of th input and the grammar. In his thesis,both utterances and parameters in the grammar are represented by composing parameters, and he tries to add/delete new grammar based on minimizing description length of lexicon.

## What We Did

our implementation

We focus on re-implementing learning algorithm of Carl de Marcken's Phd thesis. The general idea is to start with simplest lexicon, for every iteration, it refines the parameters of the lexicon to reduce the description length until converage.

The learning process is separated into a stage where stochastic properties are optimized assuming a fixed linguistic structure in the lexicon. The expectation-maximization algorithm can be used to arrive a locally optimal set of probabilities and codelengths for the word in the lexicon. For composition by concatenation, it uses Baum-Welch procedure.

Every iteration has two parts, adding and deleting lexicon. For adding lexicon, assumingly deleting this word/grammar, we look at pairs of words that are composed in the parses of words and the input. So long as the composition operation is associative, a new word can be created from such a pair and substitued in place of it wherever it appears. Hence we calculate the description length difference, if it reaches the condition, then adding it.

Similar estimation can be used for deleting word/grammar. We also use n-gram model to speed up searching and learning process.

how it runs

Our data is from Brown Corpus, which contains 44195 vocabularies. We do some pre-processing. We make every sentence one line as well as deleting all whitespaces. Our tool would run 15 rounds iterations, because accoring to Carl de Marchen's theory, it almostly gets local optimization at this point. In order to speed up processing, we use pypy instead of origin python. Our tool would read input corpus, and generate the newly grammar, segmented sentence as well discoved lexicon.

choices we made

- our grammar is composed of probability and Vertib representation
- we use n-gram model as well as pypy.
- most calculations are based on log so that we can handle really low probability calculations.
- Brown corpus is converted to lower case, so that the learning algorithm does not introduce additional parameters to model capitalized words at the start of sentences.

experiments

Firstly, we measure the model performance via cross-entropy rate as well as the number of words in lexicon.

- Baseline control(no changes)
- Words represented flat in lexicon(words represented using only terminals)
- Words represented flat in lexicon, but with a separate probability model for the letters in a lexicon word
- Words represented flat in lexicon, but with O(1) probability model for the letters in a lexicon word
- Original algorithm, but the cost of a grammar entry artifically changed by -8,+1,+2,+4,+8,+16,+32,+64 bits

Secondly, we measure the precision rate and recall rate, not only bracket-based, also word-based. For more information, we also conduct some experiences about whether the segmented word is too short or too long.Specifically, for each found word with both right and left sides on correct breaks, how many of those words should also have a break in their middle? More specifically, how many should have zero breaks in the middle (the word was found exactly correct)? How many should have one break in the middle but it was missed? How many should have 2 breaks? How many should have 3+ breaks? Similarly, for each word with only right side on correct break, left side on correct break, neither sides on correct breaks.

Thirdly, Single-use words are common in practice, but if the algorithm is too eager to add single-use words to the dictionary it could get really polluted.Hence we also change our tool to require a word to occur 2,3,4 times before adding it to the dictionary, then check the precision rate as well as recall reate.

## Results

what do we consider a word graphs/tables

## Discussion

what do the results mean

## Ideas for Improvement

ideas for improvement