HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

New methods for statistical natural language modeling

Master's Thesis

Sami Virpioja

Laboratory of Computer and Information Science
Espoo 2005

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ
Tietotekniikan osasto

Tekijä Sami Virpioja	**Päiväys** 8. joulukuuta 2005 **Sivumäärä** 4 + 82
Työn nimi Uusia menetelmiä luonnollisen kielen tilastolliseen mallinnukseen	
Professuuri Informaatiotekniikka	**Koodi** T-61
Työn valvoja Prof. Erkki Oja	
Työn ohjaaja TkT Krista Lagus	

Luonnollisten kielten tilastollista mallinnusta ovat jo pitkään hallinneet niin sanotut N-grammimallit, joissa seuraavan sanan esiintymistä ennustetaan muutaman edellisen sanan perusteella käyttäen suuresta tekstiaineistosta laskettuja suurimman uskottavuuden estimaatteja. Mallien ongelmana ovat parameterien suuri määrä, joka aiheuttaa mallien koon suurta kasvua ja ylioppimista, sekä kattavan opetusaineiston puute, joka estää estimaattien löytämisen kaikille sanoille. Tässä työssä tutkitaan erilaisia ratkaisuja näihin ongelmiin.

Toimivaksi osoittautunut menetelmä sanaston koon rajoittamiseen on käyttää sanojen sijasta ohjaamattomasti opittavia morfeeminkaltaisia yksiköitä. Työssä näytetään, miten kielen esityksen dimensiota pystytään pudottamaan edelleen ohjaamattomasti riippumattomien komponenttien analyysillä. Saatavaa hajautettua numeerista esitystä pystytään käyttämään kielen mallinnuksessa esimerkiksi itseorganisoivan kartan avulla.

Suorempia ratkaisuja N-grammimallien koko-ongelmiin ovat yksiköiden tai niiden sekvenssien ryhmittely, sekä toisaalta posterioritodennäköisyyden maksimoinnin tai pienimmän kuvauspituuden periaatteen hyödyntäminen päätettäessä, kuinka paljon parametreja malliin otetaan. Työssä esitetään eräs ratkaisu sille, miten näitä menetelmiä yhdistämällä voidaan päästä hyvin rajoitetun kokoisiin kielimalleihin.

Avainsanat
luonnollisen kielen tilastollinen mallinnus, riippumattomien komponenttien analyysi, hajautetut esitykset, itseorganisoiva kartta, pienimmän kuvauspituuden periaate

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author Sami Virpioja	**Date** December 8, 2005 **Pages** 4 + 82
Title of thesis New methods for statistical natural language modeling	
Professorship Computer and Information Science	**Code** T-61
Supervisor Prof. Erkki Oja	
Instructor Krista Lagus, D.Sc. (Tech.)	

The area of statistical natural language modeling has been dominated by so called N-gram models for the last two decades. N-gram models predict the next word based on a few preceding words, and utilize maximum likelihood estimates calculated from a large text corpus. Problems related to N-gram models are the huge number of potential parameters, which causes large model sizes and overlearning, as well as the lack of extensive training data, which prevents finding estimates for all words. Different solutions to these problems are studied in this thesis.

The utilization of morpheme-like units found in an unsupervised manner instead of words has been proven to work well for restricting the model lexicon. In this thesis it is shown how the dimensionality of the representation of a language can be further reduced using Independent Component Analysis. The emerging distributed representations for the units can be utilized in language modeling, e.g. with a self-organizing map.

More direct approaches for the size problems of the N-gram models include the clustering of model units or sequences of them, and using maximum a posteriori maximization, or the Minimum Description Length principle, in order to decide how many parameters will be included in the model. This thesis gives one example on how these techniques can be combined to produce models that have a substantially lower number of parameters.

Keywords
statistical language modeling, independent component analysis, distributed representations, self-organizing map, minimum description length principle

Preface

This thesis has been written in the Laboratory of Computer and Information Science (CIS) at Helsinki University of Technology during the years 2004 and 2005. It has been supported by Nokia Research Center and the National Technology Agency of Finland (TEKES) under the project Search for Personal Media Content.

I would like to express my gratitude to my instructor Dr. Krista Lagus for guiding me to the interesting area of language modeling and for the encouragement and advice for this research, Professor Erkki Oja for supervising this thesis, and Mathias Creutz for the huge amount of help given over the years. I would also like to thank Docent Mikko Kurimo for providing feedback on the manuscript, Jaakko Väyrynen for help with the Word ICA experiments, Vesa Siivola for giving his advice and software to use in the language modeling experiments, as well as the whole personnel of the CIS laboratory for providing an inspiring working environment.

Otaniemi, December, 2005

Sami Virpioja

Contents

Notation and abbreviations

non-boldface letter	scalar, scalar function, relation, or random variable
boldface lowercase	vector or vector function
boldface uppercase	matrix
w_i	i:th word of text data
w_{i-n+1}^{i}	word sequence $w_{i-(n-1)} \dots w_i$
m_i	i:th morph of text data
$C(\cdot)$	number of occurrences of input pattern
$N(\cdot)$	number of types of input pattern
$P(X)$	Probability of X
$P(X \mid Y)$	Conditional probability of X given Y
$H(X)$	Entropy of X
$L(A)$	Code length of A
$D(q \,\|\|\, p)$	Kullback-Leibler divergence between the two distributions q and p
$\|G\|$	Size of set G
$\|\|\boldsymbol{x}\|\|$	Euclidean norm of vector $\boldsymbol{x}$

BSS	Blind source separation
ICA	Independent component analysis
KL	Kullback-Leibler (divergence)
KN	Kneser-Ney (interpolation)
LSA	Latent semantic analysis
MAP	Maximum a posteriori (solution)
MDL	Minimum description length
ML	Maximum likelihood (solution)
NMF	Non-negative matrix factorization
PCA	Principal component analysis
SLM	Statistical language modeling
SOM	Self-organizing map
SVD	Singular value decomposition

Chapter 1

Introduction

Statistical language modeling (SLM) is the endeavor for finding models that can estimate which phenomena are the most probable in a natural language. More concretely, the goal is to determine the probability distribution $P(S)$ over text strings S, which are usually sequences of words. [58, 24]

Traditionally natural languages have been studied and modeled by linguists. Hand-crafted, rule-based models or grammars based on human knowledge have several problems. Their creation requires human work and is thus expensive. A complete description of a language in all its richness is never obtained due to linguistic variation, and it can be argued if there is a division into grammatical and non-grammatical sentences. "Grammatical" is not even an adequate level of description: A grammatically correct sentence may not make any sense semantically, and thus may be very improbable in a language. A famous example of a grammatically correct but semantically nonsense sentence[1] is "colorless green ideas sleep furiously", composed by Noam Chomsky [10].

In the last 25 years, statistical methods of language modeling have become a main research direction [58]. The amount of data available to train the statistical models has increased significantly, and computers have become capable of using it. At the same time, the main applications of SLM, i.e. speech recognition, information retrieval and machine translation, have become more important, as they are needed to efficiently exploit the amounts of information available.

Of the three mentioned application problems, the one in which the statistical

[1] Which is, of course, nowadays quite a probable sentence in certain kind of topics due to its fame.

language models are most often evaluated is speech recognition. A speech recognition system consists of two models: An *acoustic model* for determining which strings are most probable when the given speech signal is given, and a *language model* for determining which strings are probable to occur in a text. For example, without the language model it would be quite impossible to distinguish whether the speaker said "for" or "four", as they are pronounced similarly. If we know that the previous word was "instructions", the language model can tell that "for" is much more probable. The part of the system that combines the acoustic model and the language model is traditionally called a *decoder*.

Likewise in machine translation it is quite important to know the most typical ways of saying things in the target language. We could make word-to-word translation without language models, but as the word order varies in different languages, and one word can have several meanings, the result would be quite confusing. We need to have some kind of model for the target language in order to eliminate the wrong translation decisions and make the text fluent.

The fundamental problem in statistical language modeling is referred to using the concept *curse of dimensionality*, or the *sparsity of the data*. Curse of dimensionality, the term first utilized by R. Bellman [3], refers to the exponential growth of a hyper-volume as a function of dimensionality, whereas sparsity of the data means that most of the possible phenomena of the modeled signal are never observed in the data.

Let us consider the joint distribution of a sequence of 10 words in a language that has a vocabulary of 100 000 words.[2] This kind of distribution has $100\,000^{10} - 1 = 10^{50} - 1$ free parameters: The probabilities for each different sequence, except the last one that has a probability of one minus the sum of the rest. The number of parameters such as this cannot be estimated directly.[3]

The sparsity of the data is a direct consequence of the enormous dimensionality. On the other hand, most possible word sequences are never observed, no matter how large text or speech corpora we collect. And no sequence should have zero probability in the model, if we want to be able to use our model for arbitrary text produced by humans. On the other hand, those sequences that are observed are easily estimated to be unrealistically probable, i.e., they are *overlearned*.

[2] In English, a vocabulary of 100 000 word types (including inflected word forms) covers about 99.3% of new text [11].

[3] Or even stored anywhere: 10^{50} is roughly the number of the atoms the earth consists of.

The problem of data sparsity may diminish slowly, as we have more and more text data available in digital form to use for training. However, a model that uses huge amount of training data and thus gives very good estimates, can be of little use in practice, if it is so huge that conventional computers cannot use it due to to the limited size of the available space. Also, if we want to be able to use the model in on-line speech recognition (in most practical uses you do not want to wait hours to get the text corresponding to your speech), it must be so compact that it fits well into the memory of the used device (e.g. personal computer, or even personal digital assistant) and thus is quick to use. The quicker it is to get probability estimates of a text sequence, the more extensive search of possible sequences can be made by the decoder, and thus compact models can also lead indirectly to better recognition results.

The problems of data sparsity and overlearning are not only confined to language modeling, but they are fundamental in many other modeling tasks. However, they make SLM one of the most demanding and interesting tasks. If efficient methods for conquering the curse of dimensionality can be found in one area, they are probably applicable also elsewhere.

1.1 Aim of the thesis

A general goal in this thesis, as in most SLM research, is to find methods for fighting the curse of dimensionality and the sparsity of the language data. In particular, we focus on language models that should be applicable for any natural language. The conventional methods, that are mainly developed and used for English, are not directly suitable to e.g. *agglutinative*[4] and *highly-inflecting*[5] languages. In these kinds of languages, words (in their inflected forms) are not suitable units of the model: The vocabulary becomes too large, and the major part of the possible word forms never occur in the training data. For some languages, such as Finnish, Hungarian and Turkish, this is an issue of high concern, for some not as essential but anyhow important (e.g. all compounding languages including for example German, Swedish and Greek) [26]. Instead of words we use *morpheme*-based models. In linguistics, morphemes are defined as the smallest meaningful units of a language [47].

[4]"An agglutinative language is a language in which words are made up of a linear sequence of distinct morphemes and each component of meaning is represented by its own morpheme." [47]

[5]"Inflection is variation in the form of a word, typically by means of an affix, that expresses a grammatical contrast which is obligatory for the stem's word class in some given grammatical context." [47]

Morpheme-like units can be found in an unsupervised manner from a text corpus, with none or little information on the language concerned.

The starting point of the thesis are numerical representations for the units of language obtained by a data-driven technique called *Independent Component Analysis* [33]. First, we want to make sure that the technique gives meaningful results also when utilized for other language units than words, and is thus applicable to any language. The second aim is to study how well these representations can be utilized in the task of SLM, and what methods seem to be most feasible. This thesis includes experiments for two methods: Classification of the model units based on the representations, and modeling the numerical signal with the *Self-Organizing Map* [36].

Also the size of language models is an issue of concern in this thesis. A common way of reducing the number of parameters in a language model is to use *clustering*. Most clustering techniques in language models concentrate on clustering the units of the model. As we have an already compact set of units, morphemes, it will presumably be more useful to cluster sequences of the units. The aim of the last part of this thesis is to study how well clustering of unit histories and *Maximum a Posteriori* (MAP) estimation can be combined to build models that have a smaller number of parameters than the conventional models.

1.2 Structure of the thesis

The structure of this thesis is the following.

After introducing the topic of the thesis in Chapter 1, some background information on language models is covered in Chapter 2. The basics of statistical language modeling are discussed from the viewpoint of how the models are conventionally built and how they are evaluated using the concepts of information theory. Also the *Minimum Description Length* (MDL) [55] principle is presented.

Chapter 3 presents the language data that is used in experiments carried out in the later chapters. It also describes the algorithm that is used to discover morpheme-like units of the language. The description of the algorithm will also serve as an example for using the MDL principle in statistical modeling. Last, we present the three sets of units that are utilized later in the work.

Chapters 4 – 7 present the methods and experiments that were carried out for this thesis in chronological order. Each chapter first discusses the ideas that

led to the method, then introduces the method that was used, then presents the experiments that were done, and last discusses their implications.

In Chapter 4, the foundation of latent codes for language units is discussed. In the following Chapters, 5 and 6, the two different methods of using latent codes in language modeling are introduced. The first method uses traditional N-gram modeling and the second is motivated by neural networks. In Chapter 7, however, the latent codes are not used, but generation of similar cluster models is sought more explicitly, examining the prediction distributions of morph histories. The approach is based on MAP estimation and MDL style priors.

Finally, in Chapter 8, conclusions regarding all the experiments in the thesis are drawn, and directions for future research are outlined.

Chapter 2

Background on language modeling

In this chapter we describe some basic concepts and methods of statistical language modeling. We start with the so called *N-gram language models* [9, 58, 24], which are the most common way of determining the probability of a text sequence. We also describe several *smoothing techniques* [9] of N-gram models, including the current state-of-the-art smoothing that is used in the N-gram models in our experiments.

The second section covers some information theoretic background on stochastic models and their evaluation. We present the concepts of *communication channel* and *entropy* [59] in the SLM domain, and give the standard evaluation measures of statistical language models, *perplexity* and *cross-entropy* [57]. We also discuss *the Minimum Description Length principle* [55], a way of determining compact models of data, that has inspired some of this work.

2.1 N-gram models

N-gram models are simple but efficient models of language. They assume that the language is constructed by some consistent units (usually words) w_i that occur sequentially, and that the occurrence of a unit is only dependent on the $(n-1)$ previous units. The latter is called the *n-gram assumption* [24] and formulated as follows:

$$P(w_i \,|\, w_1 \dots w_{i-1}) \approx P(w_i \,|\, w_{i-(n-1)} \dots w_{i-1}) \tag{2.1}$$

For example, a broadly used model for English is a trigram (or 3-gram) model of words ($n = 3$). An unigram model ($n = 1$) corresponds to a probability distribution of individual words.

Probability of an n-gram $w_{i-(n-1)} \dots w_i$ can be estimated from the training data. If $C(w_{i-(n-1)} \dots w_i)$ is the number of occurrences for $w_{i-(n-1)} \dots w_i$ in the training corpus, we can approximate as follows [24]:

$$P(w_i \,|\, w_{i-(n-1)} \dots w_{i-1}) \approx \frac{C(w_{i-(n-1)} \dots w_i)}{C(w_{i-(n-1)} \dots w_{i-1})} \tag{2.2}$$

This is called the *maximum likelihood* (ML) estimate of the probability [48]. It chooses such n-gram distributions that maximize the probability that the data was generated by the model, i.e. maximizes the *likelihood* of the data.

A language model that is estimated using direct maximum likelihood estimates as in Eq. 2.2 does not work well. This is mostly due to two reasons.

First, the word sequences that do not exist in the training data will get no probability and will not be recognized by the model. Of course, the same applies also to every individual word that did not exist in the corpus.

Second, when we estimate the probability for a certain rare n-gram using only a few number of occurrences, it is likely that it is badly overestimated. This is easy to understand as follows: Think of a large enough text corpus. If the possible number of different word sequences is very large, all of them cannot occur in the corpus. For those sequences that have a small probability, some do exist in the corpus and some do not. Thus the probability mass of those that do not exist is cumulated to those few that do.

2.1.1 Smoothing, back-off and interpolation

To make up for the always inadequate training data (i.e. sparseness of the data), a number of *smoothing* methods has been developed [9]. The idea of the smoothing is to move probability mass from n-grams that occurred rarely to those that did not occur at all.

One of the oldest smoothing methods is Laplace's law, also referred to as *adding one* [48]. It applies an assumption that every seen or unseen event is already seen once. Let us denote $w_{i-(n-1)} \dots w_i$ more shortly by w_{i-n+1}^i. Thus we would estimate the n-gram probability as

$$P(w_i \,|\, w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^i) + 1}{C(w_{i-n+1}^{i-1}) + V}, \tag{2.3}$$

where V is the size of the model lexicon.

Laplace's law is shown to give too much probability to unseen events [48]. A commonly adopted version is *additive smoothing* (or Lidstone's law), in which we do not add one but some constant $0 < \lambda \leq 1$ to the frequencies [9]:

$$P(w_i \,|\, w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^{i}) + \lambda}{C(w_{i-n+1}^{i-1}) + \lambda V} \tag{2.4}$$

There are also more sophisticated smoothing techniques, the most important ones being *Good-Turing estimate* and *absolute discounting* [9]. The latter, first proposed by [50], subtracts a fixed discount $D \leq 1$ from each nonzero count:

$$P(w_i \,|\, w_{i-n+1}^{i-1}) = \begin{cases} \frac{C(w_{i-n+1}^{i}) - D}{C(w_{i-n+1}^{i-1})} & \text{if } C(w_{i-n+1}^{i}) > D \\ \gamma_{w_{i-n+1}^{i-1}} & \text{otherwise} \end{cases}, \tag{2.5}$$

where $\gamma_{w_{i-n+1}^{i-1}}$ is a scaling factor that makes the conditional distribution sum to one.

Regardless of the smoothing method, we will have some probability mass for unseen events. A simple, but not very efficient way would be to distribute the probability uniformly over unseen events. Instead, smoothing is usually combined either with *back-off* or *interpolation*. Both of them try to estimate the probabilities of those n-grams that did not occur in the training data by the means of estimating lower order n-grams.

Back-off means that we use the distributions of shorter n-grams for those words that did not occur in the longer n-gram. A general equation for this kind of smoothing methods is [9]

$$P_{\text{bo}}(w_i \,|\, w_{i-n+1}^{i-1}) = \begin{cases} \alpha(w_i \,|\, w_{i-n+1}^{i-1}) & \text{if } C(w_{i-n+1}^{i}) > 0 \\ \gamma_{w_{i-n+1}^{i-1}} P_{\text{bo}}(w_i \,|\, w_{i-n+2}^{i-1}) & \text{if } C(w_{i-n+1}^{i}) = 0 \end{cases}, \tag{2.6}$$

where $\alpha(w_i \,|\, w_{i-n+1}^{i-1})$ is the smoothed ML estimate and $\gamma_{w_{i-n+1}^{i-1}}$ is a scaling factor for the distribution.

Interpolation means that we always interpolate between the shorter and longer n-gram distributions. The general equation of the n-gram interpolation can be written as

$$P_{\text{interp}}(w_i \,|\, w_{i-n+1}^{i-1}) = \alpha(w_i \,|\, w_{i-n+1}^{i-1}) + \gamma_{w_{i-n+1}^{i-1}} P_{\text{interp}}(w_i \,|\, w_{i-n+2}^{i-1}). \tag{2.7}$$

Again, $\alpha(w_i \,|\, w_{i-n+1}^{i-1})$ is the smoothed n-gram distribution and $\gamma_{w_{i-n+1}^{i-1}}$ is a scaling factor.

The current state-of-the-art smoothing technique is *modified Kneser-Ney interpolation* [35, 9]. The actual smoothing used in Kneser-Ney interpolation is absolute discounting. The novelty of the method lies on how the lower order probabilities are constructed. Instead of being proportional to the number of occurrences of the lower order $(n-1)$-gram $C(w_{i-n+2}^{i})$, the probability is set to be proportional to the number of different words that it follows,

$$N(\bullet w_{i-n+2}^{i}) = |\{w_{i-n+1} : C(w_{i-n+1}^{i}) > 0\}|. \tag{2.8}$$

If the number of different words is high, we assume that it will be probable for the word to occur also with word types that we have not seen before. Note that by $C(x)$ we denote the number of *occurrences* of an item x, and by $N(\bullet x)$ the number of *context types* in which x occurs. Similarly we will define

$$N(\bullet w_{i-n+2}^{i-1}\bullet) = \sum_{w_j} N(\bullet w_{i-n+2}^{i-1} w_j) = |\{(w_{i-n+1}, w_i) : C(w_{i-n+1}^{i}) > 0\}|. \tag{2.9}$$

Using this notation, the Kneser-Ney back-off probability is

$$P_{\mathrm{KN}}(w_i \,|\, w_{i-n+2}^{i-1}) = \frac{N(\bullet w_{i-n+2}^{i})}{N(\bullet w_{i-n+2}^{i-1}\bullet)}. \tag{2.10}$$

The modified Kneser-Ney interpolation [9] differs from the original Kneser-Ney smoothing [35] in that in it the discount parameter D is optimized separately in the cases where the number of occurrences $C(w_{i-n+1}^{i})$ is one, two, and three or more. The modified KN interpolation has been shown to outperform other smoothing methods especially when high-order n-grams are used [24].

2.2 Information theory and language models

The field of information theory originates from Claude Shannon's work in the 1940s [59]. He worked with the problem of maximizing the amount of information that can be transmitted over a noisy communication channel, e.g. a phone line. He derived theoretical limits for both the data compression and the transmission rate.

In the information theoretic approach we will think of a sequence of words as information that is sent over a communication channel. Figure 2.1 illustrates the approach. First there is an information source which generates language

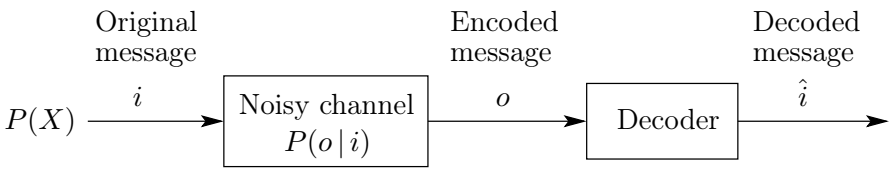

Figure 2.1: Noisy channel model in SLM.

using some stochastic model. We denote that as $P(X)$. Then we have a noisy channel that can modify the information i generated by $P(X)$. The output of the channel is o, the information modified according to the *channel probability* $P(o \mid i)$. In order to *decode* the message, we want to find the most likely i given the *encoded* message o [48]:

$$\hat{i} = \arg\max_i P(i \mid o) = \arg\max_i \frac{P(i)P(o \mid i)}{P(o)} = \arg\max_i P(i)P(o \mid i). \quad (2.11)$$

The first equality follows from the Bayes' theorem, and the next from the fact that $P(o)$ is constant when we choose i.

As an example, in speech recognition we could think that the original message i was what the speaker wanted to say, and the noisy channel includes all his speech organs that encode that message into speech, sounds of the environment that mix to the speech, distortion produced by the microphone, and so on. In order to decode the message, we must have a model for both the original message ($P(i)$) and the channel ($P(i \mid o)$). The former is a language model, and the latter is an acoustic model.

2.2.1 Information and entropy

Let X be a random variable that varies over a discrete set of symbols $\mathcal{X}$. The probability distribution of X is denoted by $P(X)$. The *entropy* [59] of the variable is

$$H(X) = -\sum_{x \in \mathcal{X}} P(x) \log P(x). \quad (2.12)$$

Entropy measures the amount of information in a random variable; in consequence it is sometimes also called *self-information*. The amount of information is the average length of the message needed to transmit an outcome of the variable: An optimal (or least *redundant*) code sends each x in $L(x) = \log_2 \frac{1}{P(x)}$ bits, and any other coding has a longer mean length [59]. Thus entropy can be seen as the expectation of the optimal code lengths.

Normally the base of the logarithm in entropy is also 2, and entropy thus measured in bits.

We can see that $H(X) \geq 0$ always. Moreover, $H(X) = 0$ in the case when $P(x) > 0$ for only one x and zero for the others.[1] Thus, if the X is determinate, it contains no information: Then we already know the exact content of a message generated by $P(X)$.

On the other hand, the entropy is highest when the distribution $P(X)$ is uniform [59]. Then $H(X) = -\sum_V \frac{1}{V} \log \frac{1}{V} = -\log \frac{1}{V}$, where V is the number of the symbols. In this case a message generated by $P(X)$ cannot be encoded very efficiently, i.e., it contains much information.

If we have two distributions for the same variable, $P_1(X)$ and $P_2(X)$, we can calculate the *relative entropy* of the distributions [48]. It is also called *Kullback-Leibler divergence* and defined as

$$D(P_1 \,||\, P_2) = \sum_{x \in \mathcal{X}} P_1(x) \log \frac{P_1(x)}{P_2(x)}, \tag{2.13}$$

where $0 \log \frac{0}{p} = 0$ and $p \log \frac{p}{0} = \infty$. It measures how different the two distributions are, and corresponds to how many bits are wasted if we encode events from distribution P_1 with a code based on P_2.

2.2.2 Cross-entropy and perplexity

Entropy is a property of a random variable, and to calculate it, we need the correct probability distribution of the variable. In statistical language modeling, we can think that a text corpus is generated by such a distribution, where the set of discrete symbols is the vocabulary of the language. However, we do not know the correct distribution of the language, $P(X)$, but try to build a model $P_M(X)$ that is as close to it as possible.

How to measure the closeness of $P(X)$ and $P_M(X)$? If we knew $P(X)$, we could for example calculate the relative entropy (Eq. 2.13) of $P_M(X)$ with regard to $P(X)$. We do not, so our measure must be based on a new data sample $D = \{D_1, D_2, \ldots, D_n\}$ generated by $P(X)$. In practice, what we can do is to calculate the likelihood of the D given the model $P_M(X)$, that is

$$\text{Likelihood}(D \,|\, M) = \prod_{i=1}^{n} P_M(D_i). \tag{2.14}$$

[1] In order to be able to use Eq. 2.12 also with zero probabilities, it is typical to define that $0 \log 0 = 0$.

A more feasible measure, *average log likelihood*, is calculated by taking the logarithm of the product and normalizing with the number of sample points:

$$\text{Average-Log-Likelihood}(D \mid M) = \frac{1}{n} \sum_{i=1}^{n} \log P_M(D_i). \tag{2.15}$$

The complement of the average log likelihood can be viewed as an empirical estimate of a measure

$$H_M(X) = -\sum_{x \in \mathcal{X}} P(x) \log P_M(x), \tag{2.16}$$

which is called *cross-entropy* of the distribution $P(X)$ with regard to the model distribution $P_M(X)$ [58]. Cross-entropy can be interpreted as the average number of bits needed to encode messages generated by $P(X)$ using the compression algorithm associated with the model M. It is worth to note that cross-entropy measures both the complexity of the language and the performance of the model for the language. Later, when we speak of cross-entropy (or just entropy) of a language model, we mean its empirical estimate based on some fixed evaluation data.

Traditionally the performance of the language models is reported in *perplexity*, which is closely related to the (empirical) cross-entropy:

$$Perp_M(D) = (\prod_{i=1}^{n} P_M(D_i))^{-\frac{1}{n}} = 2^{H_M(D)}. \tag{2.17}$$

Perplexity can be interpreted as the geometric average of the branching factor of the language according to the model. It has been suggested that cross-entropy reflects the word-error rates of a speech recognition system more closely than perplexity [24], and thus we have decided to use cross-entropy instead.

One should note that the cross-entropy values for a language model depend on several matters: Model, language, evaluation data, and the units that the language model is based on. In order to compare models that use different units, it is possible to normalize the cross-entropies. If we divide the sum of logarithms of probabilities of data points by the number of words in the evaluation data W_D, instead of the number of data points n, we get the word based cross-entropy estimate

$$H_M^W(D) = \frac{1}{W_D} \sum_{i=1}^{n} \log P_M(D_i) = \frac{1}{W_D} \log P_M(D). \tag{2.18}$$

2.2.3 Minimum description length principle

Information theory gives a measure of the information on an initially assumed distribution $P(X)$. However, in practical problems we seldom know the distribution that we are working with. Instead we have some observed data generated by $P(X)$, and in statistical modeling the goal is to discover the regularities in the data. The Minimum Description Length (MDL) principle [55] is one method of *inductive inference*, i.e. inferring general rules from examples [1], presented in the terms of data compression.

Today there are many versions of MDL, and also methods closely related to it but called with different names. The common idea, in addition to the utilization of the idea of data compression, is the tendency towards *objectivity*. The version of MDL that depends least on prior belief is called *Ideal MDL* [64]. It is based on *Kolmogorov complexity*, also called *algorithmic complexity*. In Ideal MDL the goal is to find the shortest possible Turing machine, i.e. a computer program, that prints the desired data sequence, and then halts. The length of the program code (in bits) is its Kolmogorov complexity d, and the probability of the data is $(\frac{1}{2})^d = 2^{-d}$ accordingly.

Ideal MDL is of little practical use, since it has been shown that it is impossible to design an algorithm that computes the Kolmogorov complexity of a given data set [64]. In practice the coding scheme must be selected to be something less general than a programming language. The most widely adopted way to do that is probably the *two-part coding scheme*, introduced in [55]. When we are later referring to the MDL principle, we are actually using the two-part coding scheme.

In the two part coding scheme, we first select a model class that can describe data given the model parameters θ. The goal is to describe and send a set of data, x, that is assumed to be generated by a model of the decided class, with the minimum number of bits possible. As the receiver does not know the parameters that we choose, also they must be sent. Denote $L(\theta)$ as the description length needed to encode the parameters, and $L(x \,|\, \theta)$ as the description length needed to encode the data when the parameters are known. Thus we need to minimize the *total code length* $L(x, \theta)$:

$$L(x, \theta) = L(\theta) + L(x \,|\, \theta) \tag{2.19}$$

The more accurately we describe the parameters, the more $L(\theta)$ increases, but at the same time $L(x \,|\, \theta)$ decreases. Thus the chosen θ must make a compromise between the two parts.

In statistical modeling, the model class includes a probability distribution

$P(X \mid \theta)$, and a distribution for the parameters, $P(\theta)$. Using the result of optimal coding by Shannon [59], we know that the minimum code length for data x is $\log \frac{1}{P(x \mid \theta)}$, and similarly the minimum code length for the parameters is $\log \frac{1}{P(\theta)}$. The total minimum description length is accordingly

$$\min_{\theta} \{ -\log P(x \mid \theta) - \log P(\theta) \}, \tag{2.20}$$

assuming that the optimal codes can be used. The maximum likelihood estimates used in N-gram models maximize the likelihood of the data given the model, $P(x \mid \theta)$, and thus minimize only its description length, excluding the length of the model. We will discuss the problems of that approach later in this thesis, in Section 7.1. Before that, we will see one example application of the MDL principle in language modeling in the next chapter.

Chapter 3

Dataset and segmentations

In this chapter we present the language data that we have used in our experiments, and the method we have used to produce a compact set of language units for the data. The method is called *Morfessor* [12, 15, 16]. Morfessor finds a compact set of word fragments that can be used to compose all the possible words. These fragments often resemble morphemes, the smallest meaningful units of which the words are composed. The smaller number of basic units leads to constructing efficient N-gram models for languages where the number of possible word forms is enormous. We are using such a language in our experiments, and consequently we shall take advantage of the Morfessor approach.

3.1 Language data and preprocessing

All the language modeling experiments done in this work use Finnish text data. As mentioned in Section 1.1, Finnish is one of the most problematical languages for conventional models due to its rich morphology[1]. That is exactly one of the reasons why we are using it: If we can construct a model that is suitable for Finnish, it should work also for "easier" languages such as English.

Our data consisted of books, magazines and newspapers from the Finnish IT Center of Science (CSC) and short newswires from the Finnish National News Agency (STT). A 16 million word subset of that set of corpora, including parts from all the data sources, was used in every phase of the training: Training the morpheme segmentation, determining the latent codes in chap-

[1] I.e. words tend to have a rich *internal structure* [47].

ters 4 – 6, and training the language models in chapters 5 – 7. A different subset of the same data, total 50 000 words, was used for calculating cross-entropy for the estimated models. Most of the data were newswires from STT (about 50%) and newspapers from CSC (about 40%).

As preprocessing, we removed punctuation, numbers and other special characters, as well as case distinctions, from both subsets. Neither did we include any sentence, chapter, or other breaks, but concatenated the all parts of the subsets into continuous text corpora.

3.2 Morpheme segmentation

The traditional way of constructing statistical language models has been to estimate the probabilities of word sequences. This is adequate if the language concerned is such that possible inflections of the words are quite restricted and if compound words are rare. As this applies to English, the language on which the SLM research has concentrated, most new language models still have words as the basic units of the model.

We want to build language models for Finnish, which is one typical example of a highly-inflecting language as mentioned in Sec. 1.1. In Finnish, a word stem can have several suffixes and even some prefix. In addition, two or even more words including affixes can form one compound word. E.g. "ikkunansirpaleet" (shards of window), consists of two stems both followed by a suffix: `ikkuna + n + sirpalee + t`. The first suffix denotes possessive form of "ikkuna" (window), the second plural form of "sirpale" (shard). Using the stems and affixes as the units of a language model instead of the words can clearly reduce the lexicon size.

This kind of word segmentation can be seen as segmentation to morphemes, the smallest meaningful units of a language. Segmentation to grammatically correct morphemes requires a lot of human work, so some research has been carried out on unsupervised learning of morphology. Result of such an algorithm may not be such morphemes that linguistics use, but suitable for many automatic tasks. In linguistics a *morph* means a (phonetic) realization of a morpheme [47], so later on we shall refer to the found morphemic segments as morphs.

3.2.1 The Morfessor algorithm

The morpheme segmentation used in this work is based on an algorithm by Creutz and Lagus [13]. It makes no assumptions on how the words are formed, but just tries to find a compact representation for the text data using the Minimum Description Length (MDL) principle. In [15] the model was formulated using (equivalent) *maximum a posteriori* (MAP) estimation, and the model named "Morfessor Baseline". Later the algorithm was improved by adding stem, prefix and affix categories, and the new version was named "Morfessor Categories" [14, 12].

Next we shortly describe the Morfessor Baseline algorithm in the MDL framework. A more detailed presentation can be found in [13] or [26], or slightly modified, using the MAP framework, in [15].

The task is to find a segmentation for language from a corpus of raw text, x, in an unsupervised manner. The parameters of the segmentation, θ, consist of a *morph lexicon*, i.e. a set of unique morphs, where each morph is a string of characters. In addition, each morph has a probability of occurrence. In the MDL framework, we can imagine that we have a sending and a receiving party, and the sender tries to transmit the corpus x to the receiver using the shortest possible code. That includes sending the segmentation model with $L(\theta)$ bits and the corpus coded using the model with $L(x \mid \theta)$ bits. Thus we want to find such θ that the sum of the lengths is minimized:

$$\arg\min_{\theta} L(x, \theta) = \arg\min_{\theta} \{L(x \mid \theta) + L(\theta)\} \tag{3.1}$$

The coding length of the model parameters include coding of the morph lexicon, i.e. a text string and a probability for each morph. As the probability is a maximum likelihood estimate based on the training corpus, we do not need to use floating point numbers. Assume that we have N morph tokens in the corpus, and they are of M types. We first send N and then for each morph its number of occurrences, f_{μ_i}. A positive integer C can be encoded using the following number of bits [55]:

$$L(C) \approx \log c + \log C + \log\log C + \log\log\log C + \ldots, \tag{3.2}$$

where the sum includes all positive iterates and c is a constant, about 2.865.

We can get an even more compact code if we do not send each f_{μ_i} separately. As $\sum_i f_{\mu_i} = N$, and there are $\binom{N-1}{M-1}$ ways of choosing M positive integers that sum up to N, so we only need to send *which* one of the possible combinations is used. That requires $\log \binom{N-1}{M-1}$ bits.

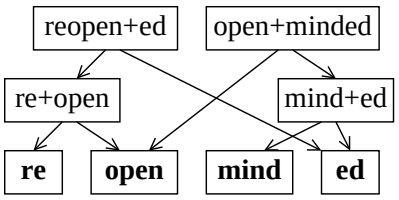

Figure 3.1: Hypothetical splitting trees for two English words. [15, 26]

Then we need to code the morph strings. Assume that the letters are drawn independently of each other from a probability distribution that is known both to the sender and the receiver. The letters, plus a boundary character that is used to distinguish where one morph ends and the next begins, form an alphabet. Each character of the alphabet, α, has an unique code. Optimal codes, also known to the both parties, are derived from the probabilities, so that $L(\alpha) = -\log P(\alpha)$. The code length of the model is thus

$$L(\theta) = \Big(\sum_{j=1}^{M} \sum_{k=1}^{\text{length}(\mu_j)} -\log P(\alpha_{jk}) \Big) + L(N) + \log \binom{N-1}{M-1}, \tag{3.3}$$

where α_{jk} is the kth character of the jth morph in the lexicon.

When the parameters of the segmentation model are known, we can encode the corpus using it. Each morph μ_i is represented by a code of $\log P(\mu_i)$ bits, i.e. using the optimal coding. Each word in the corpus can be rewritten as a sequence of morphs, so the code length of the corpus is

$$L(x \,|\, \theta) = \sum_{j=1}^{W} \sum_{k=1}^{n_j} -\log P(\mu_{jk} \,|\, \theta), \tag{3.4}$$

where j runs over the W words in the corpus and k over n_j morphs in each word.

The search algorithm of the Morfessor Baseline utilizes a greedy search. In the initial stage each word in the corpus is a morph of its own. The algorithm uses a data structure where each distinct word form in the corpus has its own binary splitting tree. The leaf nodes of the tree represent the morphs that are present in the current morph lexicon. An example of such trees is shown in Figure 3.1. Only the leaf nodes contribute to the code length of the model, whereas higher-level nodes are used only in the search.

The algorithm tries to split words in random order into two parts, and calculates the resulting total code length $L(x, \theta)$ for each possible split. The one

that yields the lowest code length is selected, presuming that it is lower than the model with no split. In case of a split, the algorithm recursively tries to split the two new parts. After all words have been processed once, they are shuffled by random and the splitting is sought for each word again. The algorithm stops when the total code length does not decrease significantly from one epoch to the next.

3.2.2 Utilizing Morfessor in n-gram modeling

The whole process of estimating language models for morphs when using Morfessor is shown in Figure 3.2. Note that instead of the corpus as a whole we give to the algorithm only the distinct word forms of the corpus. This results in smaller morph lexicon, since the data part of the code length is also smaller, and the algorithm needs to find a balance between them. The ability to find good morphs still remains, as they occur many times in different combinations with other morphs to form the words.

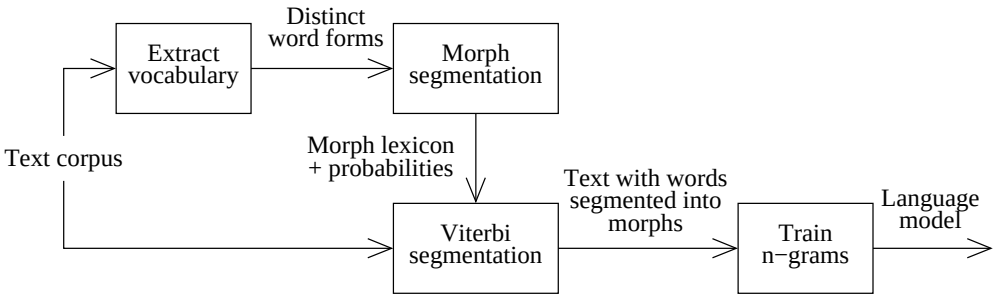

Figure 3.2: The steps in the process of estimating a language model based on statistical morphs from a text corpus. [26]

When the morph lexicon and probabilities are determined, the *Viterbi algorithm* [20] can be used to segment any text of the same language to morphs. The Viterbi algorithm finds efficiently the most probable segmentations for the words.

When the text is segmented into morphs, we can use *morph n-grams* instead of word n-grams in statistical models. In order to distinguish the word boundaries, we usually add into the lexicon a special "morph" that denotes a word break.

N-gram language models based on the statistical morphs produced by Morfessor have been compared to traditional word-based models and models based

on grammatical morphs. The prediction performance of N-gram models for Finnish has improved compared to words [26]. In speech recognition system for Turkish [25] and Finnish [26], statistical morphs are shown to reduce word error rate when compared to either words or grammatical morphs. Thus there should be no reason for staying with words based models if models of large or unlimited vocabulary are needed.

3.3 Morph lexicons

As discussed in Sec. 1.1, we segmented the text into morpheme-like units. The segmentations for the words were obtained using the baseline algorithm of the Morfessor software [15], introduced originally in [13]. It produces a segmentation model from the training data, and the model can be used to segment new corpora to a set of induced morphs. (See Sec. 3.2 or referred papers for the details of the algorithm.)

Using the basic settings for the Morfessor, the training data was segmented to nearly 24 million morphs of 89 368 types. From now on, this segmentation model and morph lexicon is referred to as *MorphSetA*.

For more complex models we also wanted a segmentation with a smaller lexicon. That was obtained by training the segmentation with a corpus vocabulary where the least frequent words (less than 20 occurrences) were filtered out. This way we got a lexicon of 10 528 of morphs, referred to as *MorphSetB*.

We had also a segmentation to *grammatical morphs*, fragments that are linguistically correct realizations of morphemes. It was obtained with *Hutmegs* (Helsinki University of Technology Morphological Evaluation Gold Standard) [17, 16] package, based on hand-made lexicon and rule set. However, this kind of method cannot analyze words that are unknown to it (e.g. many foreign names and misspelled words), so language models based on it will have a limited vocabulary. Grammatical morphs were used only in the first experiments of finding latent features for morphs, presented in Chapter 4.

Table 3.1 shows the segmentations of a Finnish sentence taken from the evaluation corpus. Compared to a grammatical segmentation, both *MorphSetA* and *MorphSetB* make mistakes. The former tends to keep some common inflected forms in one piece (e.g. "maailmassa", which means *in the world*), while the latter makes some excess segmentation to more rare words. For example, "amerikasta" (from America) has been segmented to a single "a", "meri" (sea), and "kasta" (imperative of *dunk*). In addition, both statistical

Table 3.1: An example phrase from the evaluation corpus segmented using different models. The phrase, *"Etelä-Amerikasta on tullut luonnon monimuotoisuuden symboli koko maailmassa"*, translates to *"South America has become a symbol of the diversity of the nature in the whole world"*. Word breaks are marked with the symbol #.

Model	Segmentation
MorphSetA	etelä # amerika sta # on # tullut # luonnon # monimuoto isuuden # symboli # koko # maailmassa #
MorphSetB	etelä # a meri kasta # on # tullut # luonnon # moni muoto isuuden # symboli # koko # maailma ssa #
Grammatical	etelä # amerika sta # on # tul lut # luonno n # moni muotois uude n # symboli # koko # maailma ssa #

segmentations kept very common inflected forms such as "tullut" (perfect form of *become*) and "luonnon" (possessive of *nature*) intact.

Chapter 4

Distributed latent features for morphs

In this chapter we discuss a method of finding *distributed latent representations* for words or their parts. The features are latent in the way that their values are not obtained from the word itself but from their usage in the language. That the features are distributed means that the representation is not just a single symbolic unit, but arises from combined activity of several units.

The motivation for looking for such features comes from the research connected to *Latent Semantic Analysis* (LSA) [19, 44]. It was first introduced as a tool for information retrieval, where it is known as *Latent Semantic Indexing* (LSI). The underlying idea is that the contexts where a given word does and does not appear in, provide information on determining the similarly of words and sets of words. In LSA, *singular value decomposition* (SVD) is used to decompose a co-occurrence matrix of words and documents to a product of three other matrices, which project both documents and words to a number of latent topics.

Latent Semantic Analysis has proved to work in many tasks that require some conceptual knowledge of a language. In addition to information retrieval, these include synonym tests [45, 63], semantic priming [44], emulation of expert essay evaluations [45] — and statistical language modeling [2].

A well-known problem in LSA is that the latent presentations found by it often do not have any meaningful interpretations for humans: They are more like mixtures of features. Recent research has shown that at least in some cases, *Independent Component Analysis* (ICA) [33] performs better than LSA. In a research by Väyrynen and Honkela [65], ICA and SVD (i.e.

LSA) were compared in the task of finding linguistic categories (such as verb, noun, adjective, comparative, past tense) for English words. The conclusion was that the ICA based features corresponded to the human intuitions much more closely than the SVD based features in both visual inspection and systematic comparison.

If traditional linguistic features and categories (such as part-of-speech tags) can be found in an unsupervised manner, as illustrated by [65], it may notably help speech recognition of several languages. There is evidence that combination of category-based and word-based models remarkably helps speech recognition [52], and automatically derived categories often work better than part-of-speech categories tagged by hand [51].

How does LSA, and possibly ICA, help with the curse of dimensionality? If we think of a large vocabulary of words, including inflected and compound forms, our linguistic intuition clearly tells that there is a lot of structure in the relationship among words. That none of the structure is utilized, results in a large number of parameters. *Dimension reduction* methods such as LSA can find the structure, and map the words to a space that has much lower dimensionality [58].

We can also consider the feasibility of distributed features from a quite practical point of view. If words or other units of a language are considered only as strings, i.e. sequences of letters, in many tasks the actual strings are irrelevant apart from the fact that they separate different words from each other. Two words that consist of nearly equal letters can mean totally different things ("scream", "cream") and synonyms might not have any resemblance to each other ("holiday", "vacation").

The problem of *homographs*, words that have the same spelling but different meaning [47], remains whether we use distributed features or stay with the strings. If we do not have the means of separating the meanings, the latent feature must be such that the feature of a word can include several, possibly independent, meanings.

Of course, the irrelevance of the strings is not true when we do tasks like finding allomorphs (complementary morphs that manifest the same morpheme in its different morphological or phonological environments [47]) or consider inflections of words. In most statistical language modeling these issues are anyway neglected, and the vocabulary of the model is but a very large set of unrelated entries: Words could as well be replaced by some other arbitrary identification strings or numbers. From this perspective, changing to any representation that gives more relevant information on the similarity of the units could be useful.

4.1 Introduction to Independent Component Analysis

Independent Component Analysis (ICA) is a data-driven signal-processing technique that attempts to find the latent components that generated an observed data set. It has been used to separate mixed signals or find hidden factors for example in financial, biomedical, telecommunications and image data. [33]

Recently ICA has also been applied to natural language data to find linguistic representations for words or morphemes of a language [28, 41]. The representations were found to code some syntactic categories, thematic roles or semantic properties of the used units. In earlier research, ICA has been used in finding topics in text documents or Internet chats [38, 5].

Independent Component Analysis is one type of method for *blind source separation* (BSS) problem. In BSS, we know that our data matrix $\boldsymbol{X}$ is generated by some unknown sources $\boldsymbol{S}$ that are somehow mixed. Usually it is assumed that the mixing is linear, i.e. $\boldsymbol{X}$ is a product of a mixing matrix $\boldsymbol{A}$ and the source matrix $\boldsymbol{S}$. The problem is blind source separation when we know neither $\boldsymbol{S}$ nor $\boldsymbol{A}$. However, if we want to be able to do anything at all, some assumptions of the unknown matrices must be made.

In ICA it is assumed that the sources in $\boldsymbol{S}$ are *statistically independent* of each other. This may not be the real case, but the assumption, whether entirely correct or not, leads to efficiently estimated solutions and often also to meaningful and useful results. The independence is usually gained by maximizing the nongaussianity of the distributions. There are also other methods for the task (e.g. maximum likelihood estimation or minimization of mutual information), but maximizing is both simple and intuitive [33].

The maximization of the nongaussianity often leads to super-Gaussian, or *sparse*, distributions [33, 29]. A sparse distribution is characterized by the fact that it takes very small (absolute) values and very large values more often than a Gaussian distribution. In the extreme case this means that a variable is always either "active" (has some large value) or "passive" (zero), and never something in between. These kind of variables are easy for humans to interpret, and there is some evidence that the brain processes information similarly at least in the visual cortex [21].

4.2 The Word ICA method

The Word ICA method by Honkela et al. [27, 28, 29] creates automatically syntactic and semantic features based on analyzing words in contexts.

The Word ICA papers used English text as the extraction data. A set of common words were selected for which the features are created. These are called *focus words*. Another set of words, *context words*, were used to calculate context information for the focus words. The information was collected into a context matrix $\boldsymbol{C}$, in which c_{ij} denotes the number of occurrences where i:th focus word was followed by jth context word with no words between.

The word-context matrix $\boldsymbol{C}$ is fed to the FastICA algorithm [32], in such a way that each column was considered one data point, and each row one random variable. Before that, a logarithm of the number of occurrences (increased by one to keep the zeros unmodified) was taken in order to reduce the effect of the most common words, and each row normalized to unit length.

The FastICA algorithm is divided to three consecutive steps. First, the dimension of the data is reduced using *Principal Components Analysis* (PCA). PCA is a classic technique for data analysis, feature extraction and data compression, based on finding largest eigenvalues of correlation matrix. After the potential dimension reduction, the data is *whitened*, i.e. variances of the variables are normalized. Last step is the actual ICA *rotation*, that finds the most independent components. A good description of the ICA and related methods is found in the book by Hyvärinen et al. [33].

The result of ICA is two matrices, the mixing matrix $\boldsymbol{A}$ and the source matrix $\boldsymbol{S}$. In this case, the features of the words are found in the columns of $\boldsymbol{A}$. In the Word ICA papers [27, 28, 29] the dimension of the feature vector was reduced to 10. The found features were both syntactic and semantic, such as "adjective", "plural form of noun" or "being a science or scientific discipline".

In more recent research of Word ICA, there are slight differences in the method [65]. Instead of a word-context matrix, a context-word matrix is collected. Thus, the column number of the matrix corresponds to the focus word, and row number to the context word. This is analogous e.g. to the temporal versus spatial domain analysis with fMRI data, where both produce similar results [54].

The context-word matrix, $\boldsymbol{X}$, is assumed to be generated by independent features of the focus words: $\boldsymbol{X} = \boldsymbol{AS}$, where the features are the column vectors of the matrix $\boldsymbol{S}$ and $\boldsymbol{A}$ is the mixing matrix. For a single word w_j,

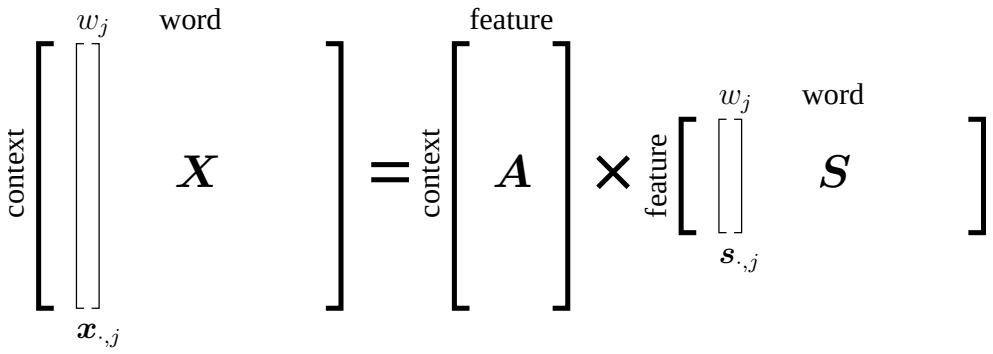

Figure 4.1: Illustration on using Independent Component Analysis to find latent features for words. Context vectors of the words are collected to the columns of $\boldsymbol{X}$, and the features emerge in columns of $\boldsymbol{S}$.

the context vector $\boldsymbol{x}_{\cdot,j}$ is thus generated as follows:

$$\boldsymbol{x}_{\cdot,j} = \boldsymbol{A}\boldsymbol{s}_{\cdot,j} = \sum_{i=1}^{n} \boldsymbol{a}_i s_{i,j} \tag{4.1}$$

Figure 4.1 illustrates the factorization of $\boldsymbol{X}$ into the product of $\boldsymbol{A}$ and $\boldsymbol{S}$. Note that we call the columns of the matrix $\boldsymbol{S}$ features, and rows of the matrix $\boldsymbol{S}$ *components*. Component number refers to the row number of the corresponding component. The component i of the word j, i.e., value of the component number i of the feature of the word j, refers to the element $s_{i,j}$.

4.3 Experiments

We had three goals in our experiments: First, to see that the Word ICA method is applicable to the case where we study smaller segments than words. I.e., instead of English words we derive the latent features for morphs of Finnish text. Second, to study what kind of latent features are found in this case, and whether we can find meaningful interpretations for them. Third, to study how the features and their interpretations change if we use longer contexts in the input data.

We have made two sets of experiments to achieve these goals. The first experiments were done with grammatical morphs given by Hutmegs (see Sec. 3.3 and [17]). The results of the those experiments are described also in [41]. After that we moved to use the statistical morphs found by Morfessor

(see Sec. 3.2 and [15]). In the second experiments we also use longer contexts than just the following morph in the context-word matrix $\boldsymbol{X}$.

Analogously to the words in Word ICA, we call the morphs for which we collect the context information and derive the latent features as *focus morphs*, and those morphs that counted as context as *context morphs*.

The latent features can be interpreted by studying the columns of the mixing matrix $\boldsymbol{A}$. When we have a single-morph context, each row of the matrix correspond to one context morph. Thus we examine which context morphs had the most effect on the component, i.e. had the largest values in the corresponding column of the mixing matrix $\boldsymbol{A}$.

When the context is longer than one word, interpretation of the components becomes a little more burdensome, but is otherwise similar: Each context morph can occur in several positions, and each combination of a position and a morph corresponds to one row of the matrix $\boldsymbol{A}$. Thus we must examine which of these combinations make the component active. Intuitively, a typical combination might be that the previous morph is word break and the next morph is a locative suffix (active for e.g. places), or that the previous morph is a word break and the one preceeding it is an adjective suffix (active for noun stems), or that the previous morph is something else than a word break, and the next morph is either a word break or a suffix (active for suffixes).

In order to help the interpretation, we could of course see which focus morphs were the most active ones for each component. Activities of the focus morphs are found in the matrix $\boldsymbol{S}$.

4.3.1 Context data and parameters

We used the text corpus described in Sec. 3.1 to collect the statistical data for the morphs. We chose a large set of morphs which we used as focus morphs, and another, smaller set that was used as context morphs.

The choosing criteria for adding a morph in the focus or context morphs was the frequency of the morph in the corpus. We had two thresholds for the two sets: F-limit is the threshold for in how many contexts a morph must occur in order to be chosen as a context feature, and M-limit threshold for how many times a morph must occur in order to be chosen as a focus morph. One special "morph" was added to represent word breaks. Morphs that did not exceed the F-limit were additionally used as a cumulative "rare morph" feature.

We had also parameters for which positions we collect the context morphs

from: The simplest possibility is to use the immediate position on either the left side (CL) or the right side (CR) of the focus morph. Then our feature vector has a corresponding value for each context morph. To use more of the data, we used the further positions on either side and concatenated the result vectors (e.g. in CR3 we use three nearest positions on the right side of the focus morph) and as well concatenate the two sides (e.g. in CB2 we use the two nearest positions on both sides of the focus morph, so that the total length of the vector is four times the number of context morphs).

As mentioned, in the first experiment we used grammatical morphs (see Sec. 3.3 for details). The data was the same as described in Sec. 3.1, but here we used a larger subset of the data, total 30 million words. Our context was only the following morph (CR). The F-limit was 320 and the M-limit 604, resulting in 506 context morphs and 3759 focus morphs. Using this input data, we obtained 50 first independent components and interpreted them by visual inspection.

In the second experiment we used the 16 million word subset of the corpus, and statistical morphs in *MorphSetA* (again, see Sections 3.1 and 3.3). As context we had the two nearest morphs at the both sides of the focus morph (CB2), and the F-limit was 5600, resulting in 250 context morphs. The M-limit was zero, i.e. we calculated the latent features for each of the 89 368 morphs. The number of components for visual inspection was chosen to be 80.

4.3.2 Results

In the first experiment we analyzed 50 components for grammatical morphs, using immediate right context. Using the inspection scheme described before, 16 of the components seemed particular to verbs, 12 to inflected forms of nouns, 5 to locations and persons, 4 to adjective stems. Some components where harder to interpret, and some were not specific enough to indicate just one syntactic category. Figure 4.2 shows the feature vectors of three common nouns. More examples are found in the paper [41].

The results from the first experiment was that the latent features found for morphs were indeed quite meaningful and easy to interpret, and resembled those that were found for English words by Honkela et al. Thus the Word ICA method seems to be suitable for finding latent distributed representations for many kinds of units of any language.

In the second experiment we studied how the features formed when the context was longer than the immediate right (or left) position. Instead of denot-

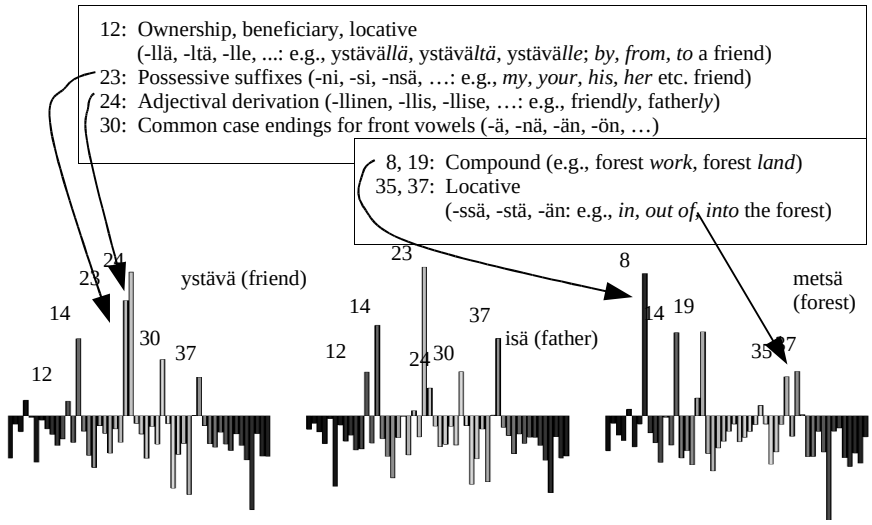

Figure 4.2: The ICA features for three sample morphemes, ystävä (friend), isä (father) and metsä (forest), along with some interpretations. Component 12 appears to code change and maintenance of the state ownership and 23 the owner. 24 codes for adjectival role; note for "friend" this component is strong (friendly), for "father" weaker (fatherly) and for "forest" nonexistent (*forestly). [41]

ing which kind of morphs follow (or precede) the morph, values in context-feature matrix ($\boldsymbol{A}$) form *patterns* of context morphs. Nine context patterns, randomly chosen from total 80 components, are described in Table 4.1. The patterns vary by which context morph positions they are affected: Some may use only preceeding morphs, some following morphs, some both. Some patterns mix similar suffixes that have different meaning (e.g. in component 20, "si" is both possessive suffix for nouns and tense suffix of verbs), but most patterns result in a very homogeneous set of active morphs.

Major part of the components seem to be largely affected by the following (right) context. Of the 80 components, just 2–3 are active on suffix morphs. This is quite natural. The left context of a suffix usually varies much, and the right context is mostly word breaks and other suffixes. The most distinct suffix component, 51, is mainly affected by the right context. It collects nearly all of the suffixes, which is seen if we sort the morphs by their activity of the component 51: The 15 most active morphs — *nut, sti, ttiin, taan, tti, isesti, ivat, vat, vät, itten, nsa, än, ssa, neet, tään* — are all clearly suffixes. Any other component that inspected the right context for word breaks would be quite dependent on the component 51, and thus ICA does not come up

with another suffix component.

Figure 4.3 shows some actual component values of morphs. The shown components are the same that are described in Table 4.1. Note that activity of a feature may show either in large positive or large negative value. Besides, in order to make sense which value is large and which is not, the values of the different components of the same feature vector should not be compared to each other, but the values of the same component for different morphs. Variance of each component is one, but the mean values vary (as well as the higher order statistics).

It is moderately easy to see why the selected morphs have high activity in those components that they do: Morph "vähintä" has a large value in component 58, as "vähintään" (at least) is a very common word. "Valitti" and "arveli" have large values in component 8, as they often occur in patterns such as "Räikkö nen <w> arveli" (Räikkönen thought) or "hän <w> valitti" (he complained). "Veto" occurs often in possessive form (component 20). "Vetosi" (your bet) is also a past tense of "vedota" (plead). Also "arveli" has some activity in component 20, which is due to the conditional form "arvelisi" (would suppose). "Saari" and "tikka" are nouns that are base forms for Finnish family names Saarinen and Tikkanen. "Tikkailla", inflected form of "tikkaat", means "on the ladder" and thus also component 44 has some activity in the latter morph. "Vanho" and "kuolle" have both plural stems that can be followed by plural locative suffixes "ista" (component 18), "ille" and "illa" (component 44). "Kuolle" has also a high activity in component 38, as "kuolleet" (the dead) and "kuolleen" (of the dead) are common forms. "Yleis" inflects similarly to "yleiset" (general) and "yleisen" (of the general).

As these examples show, using a context larger than the following or preceding morph results in finding longer context patterns, instead of activities of individual context morphs. The patterns tell in which kind of contexts the active morphs occur. E.g., morphs that refer to actions made by humans often occur so that the previous word is a personal pronoun or a name of a person (component 8 in Table 4.1). We also see that the features are still distributed: Different, independent meanings or uses of the morphs are coded in different components.

4.4 Discussion

Our experiments show that the Word ICA method is straight-forward to utilize in order to find latent features also for morphs. The features are

quite equally easy to interpret whether we use English words, grammatical morphemes of Finnish, or statistical morphemes for Finnish. Thus, it seems probable that the method gives sensible results for any language, using a suitable set of language units.

These sensible, easily interpreted features are the major advantage of the ICA method compared to Latent Semantic Analysis. The latent concepts found by LSA rarely have a meaningful interpretation for humans [27]. It is promising to see that the interpretations for the ICA components do not become much harder even when we have units not so clear to our intuitions as words.

The Word ICA method has still a couple of unanswered questions. Whether it is useful to utilize context on both sides of the focus morph is unclear and probably depends on the application. Right context seemed to be of more use than left context: Major part of the both-side context patterns used the following morphs more clearly than the preceeding ones.

Neither we still have a systematic way of determining an optimal number of ICA components. In the first experiments, where only one context position was used, we concluded that 20 components seemed too few, while 50 was still not excessive. For 5-gram contexts (two positions to both sides of the focus morph) 80 components is not too many. At least there seemed not to be any "noisy" components. Instead it seems that the more components we get, the more detailed the corresponding context patterns become. Of course, at some point they must become *too* detailed, even so that one component will model one individual type of context.

Table 4.1: Some of the 80 context patterns formed by ICA when context was two nearest positions to both sides. <w> means word break and _ is the position of the focus morph. Parentheses and vertical lines inside denote a number of alternative morphs in the position. In the third column there are some example morphs that have a high activity in the corresponding component.

No.	Context pattern	Examples of focus morphs
7	<w> _ in <w>	*vieti* (take away, "vietiin" = was taken), *turv* (safety, "turvin" = by means of), *vasto* (anti, "vastoin" = contrary to)
8	(nen | hän) <w> _ <w>	*keskeytti* (aborted), *kuvaili* (depicted) *muisteli* (recalled)
18	<w> _ ista <w>	*tavall* (common, "tavallista" = usual), *vanho* (old, "vanhoista" = of the old), *kuolle* (die, "kuolleista" = of the dead)
20	_ (si | ni | nsa) <w>	*painal* (push, "painalsi" = pushed), *valmen* (coach, "valmensi" = coached), *surma* (death, "surmasi" = slew)
26	<w> _ (nen | sen) <w>	*suho, korho, räikkö* (common Finnish names, e.g. "Räikkösen" = Räikkönen's)
36	_ (n | <w> | a | t | ...)	*muisto* (memory), *iso* (large), *varjo* (shadow)
38	_ (et | en | ten | ...) <w>	*keskinäis* (mutual), *yksittäis* (one-off), *eläkeläis* (retired)
44	_ (ille | illa | ...) <w>	*ulottuv* (extend, "ulottuvilla" = within reach), *vanho* (old, "vanhoille" = to the old), *entis* (past, "entisille" = to the former)
58	_ (än | ssä | nsä | ...) <w>	*selvittämä* (settle, "selvittämänsä" = settled by her), *viemä* (bring, "viemään" = to bring), *käymä* (visit, "käymässä" = visiting)

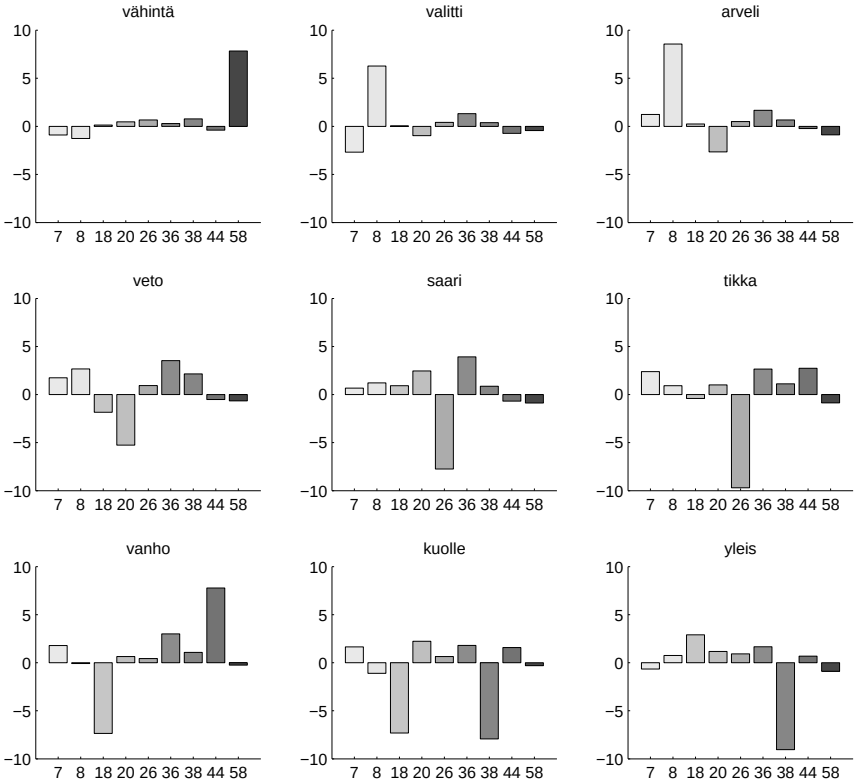

Figure 4.3: Nine selected ICA components of nine moderately common morphs: vähintä (inflected form of *least*), valitti (complained), arveli (supposed), veto (a *bet* or a *pull*), saari (island), tikka (*woodpecker*, *dart*, or a stem form of *ladder*), vanho (plural form of *old*), kuolle (inflected form of *dead*) and yleis (a prefix translating to *general*). The context patterns of the same nine components were studied in Table 4.1.

Chapter 5

N-gram model for classes derived from ICA features

In Chapter 4 we made experiments with Word ICA, an unsupervised method for extracting latent features for words. As the method resembles Latent Semantic Analysis, which is shown to work well in statistical language modeling [2], we interested ourselves in finding methods to use also our ICA-based method in SLM. Moreover, the Word ICA method is also shown to give more intuitive features than LSA [65], resembling linguistic categories. These results were supported also by our own experiments. In addition, we saw that the method works well on statistical morphs, and thus fits to our tendency towards modeling methods that work for any language.

How to use the distributed latent features in language modeling? As we have a feature vector for each unit of the language, one simple way would be to use it as a distance criterion between the units. And when we can measure distances, we can also do clustering.

As mentioned in Chapter 4, combining category-based and word-based models has been successfully applied to N-gram language models and especially speech recognition [24, 52]. Using categories instead of individual word forms in N-gram models significantly reduces the number of parameters in the model, and thus helps with the data sparsity problem. Encouraged by these results, our first try was to use the latent features for clustering of the morphs.

Thus, the purpose of this study is to examine whether the latent feature vectors can be utilized for clustering the morphs into equivalence classes in order to reduce the size of the N-gram model. In addition, we are interested on how the length of the context used in deriving the features, and also the length of the feature vector itself, affect the results.

5.1 Related work

When constructing a class-based model for a language, we actually confront several, in some cases separate, questions. One is naturally how to find the clusters for the units of the model. Others discuss how to build models that can best make use of the classes.

The basic class-based N-gram model was presented in [7], often referred as *IBM model* or *clustering*. In later research, some improvements have been proposed. Goodman [24] proposes a generalization of IBM clustering, *full IBM clustering*, that utilizes clusters of previous words together with the predicted cluster to predict the next word. He also explores some more variants, where word-based and cluster-based estimates are combined. He concludes that the IBM clustering works consistently very well. Full IBM clustering works clearly better with large training corpora, but has problems with a smaller training corpus.

A traditional way of finding clusters for words is to maximize the average mutual information of adjacent clusters [7], which leads to *symmetric clustering*. In *asymmetric clustering* [22], different clusters are found for the cases where word is in the history (*conditional cluster*), and the case when the word is predicted (*predicted cluster*). This is usually done by minimizing the perplexity of the training data for bigrams of the form $P(w_i \mid W_{i-1})$ (conditional clusters) and $P(w_{i-1} \mid W_i)$ (predicted clusters), where W_j denotes the cluster of word w_j. The asymmetric cluster model by [22] worked significantly better than the basic cluster model.

Of course, the language modeling based on LSA is also closely related to this work. Bellegarda [2] uses LSA to find latent space of words and documents. When predicting, *pseudodocument* of the text seen so far is used to find those words that have occurred in the similar documents. The LSA information is combined with standard N-gram modeling, and the hybrid model gives perplexity reduction of 20% compared to 3-gram model alone. However, the analysis of word-document co-occurrences is not in the scope of modeling short span dependencies, and neither we are aware of other work utilizing LSA in modeling n-grams of restricted length.

5.2 Method

In SLM, we wanted a model that gives a probability distribution for a sequence of text. The probability can be split into a product, where we multi-

ply the probabilities of the unit w_i of the sequence given the preceding units $w_1, \ldots, w_{i-1}$. From Sec. 2.1 we have the n-gram assumption

$$P(w_i \,|\, w_1^{i-1}) = P(w_i \,|\, w_{i-n+1}^{i-1}). \tag{5.1}$$

This puts the histories that have the same $(n-1)$ last words into the same equivalence class.

In a similar manner we can make more equivalence classes. If we have a set of W clusters so that each w_i belongs to one cluster, denoted by W_i, we can have the equivalence

$$P(w_i \,|\, w_1^{i-1}) = P(w_i \,|\, W_{i-n+1}^{i-1}). \tag{5.2}$$

Now the model has theoretical maximum of $V \cdot W^{n-1}$ parameters instead of V^n, where V is the size of the vocabulary. More, is we first predict the next cluster instead of next unit, and assume that the next unit depends only on its cluster, we get the equation

$$P(w_i \,|\, w_1^{i-1}) = P(w_i \,|\, W_i) \cdot P(W_i \,|\, W_{i-n+1}^{i-1}). \tag{5.3}$$

Now the maximum number of parameters is $V + W^n$. This type of class-based N-gram model is the IBM model mentioned in previous section.

In order to apply the latent features for the IBM model, we need to get the clusters from the feature vectors of the morphs. In the next two subsections we first describe how the clusters were achieved and then how the N-gram model was estimated using them.

5.2.1 Deriving binary codes from the ICA features

There are many quite simple and fast methods to do clustering, e.g. hierarchical clustering or K-means algorithm. For each of them, we need to make at least two decisions: What is the metric that is used to measure the distances between the units, and how many clusters do we want. As we have already made quite a many similar decisions when calculating the latent features, it would be very convenient if the number of clusters would be implicitly based on how accurate features do we have.

Remember from Section 4.1 that the components of the features (i.e. rows of matrix $\boldsymbol{S}$ in Fig. 4.1) are sparse. In the extreme case this meant that the component is either active or passive for each unit. If we assume this, we can discretize the components to binary vectors, so that each item of the feature vector of a morph is either one (active) or zero (passive). The

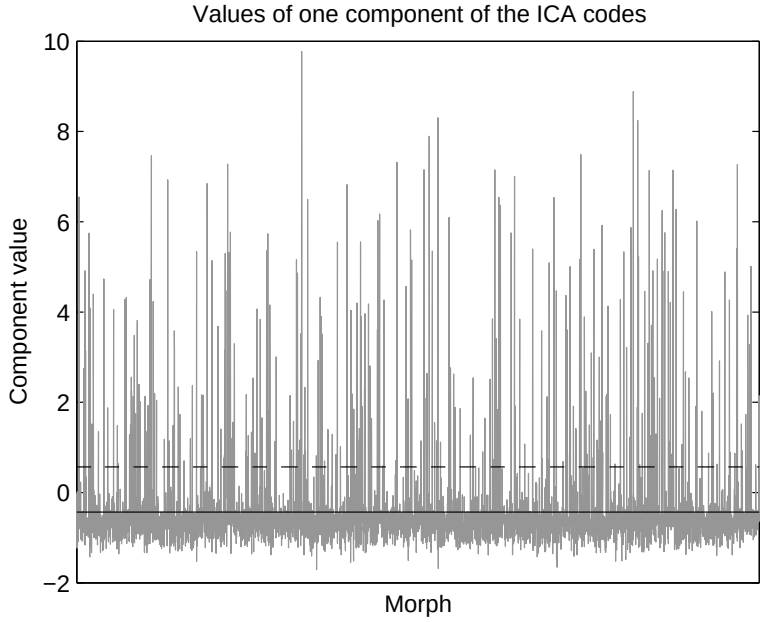

Figure 5.1: Example of one component of the features produced by FastICA. Solid black line is the component mean. Dashed black line is the mean plus variance, which is used as a boundary value for the binarization.

binarization leads to a case where those morphs that have similar activities of the components will get identical vectors. The less components we have, the less there are different binary vectors available, and thus the number of non-identical vectors of the morphs will also be smaller.

The values returned by FastICA are in a region on both sides of zero. Large values in the other, significant side correspond to the activity of the component, and values in the other side passivity. When we look at the values of one component, there is some noise near zero (with noise mean usually in the passive side) and high spikes on the significant side. Figure 5.1 shows an example for values of a component for 5 000 morphs.

A simple solution for determining the significant direction is to find the direction of the distribution's distortion. The significant side can be set to the positive direction according to that. There are several methods for calculating the distortion, and the results did not seem to vary much. We decided to calculate the dispersion of the values for both sides of the component median: Larger dispersion indicates the significant side.

Last, we need a boundary for the activity. We found no mathematical basis for the boundary value, and by visual inspection we chose to use component mean plus variance. Due to the whitening of the signals, the latter is always one.

After the binarization we have 2^C possible binary vectors (later referred as *codes*) for the morphs, where C is the number of the independent components. However, as the coding is sparse, only a fraction of those are actually used. At least there are less utilized code types than there are morphs which we have taken as focus morphs. The utilized code types, i.e. the *states* in which an unknown morph can be, will be our clusters.

5.2.2 Constructing class-based models from the binary codes

As mentioned, our class-based model is a simple extension to the standard N-gram language model, similar to one proposed by Brown et al. [7]. The model is estimated as follows:

Let there be morphs m_i that have binary codes with values b_k^i (for each morph $i = 1, \ldots, M$ and component $k = 1, \ldots, C$). We construct an nth order Markov Model for them. The model has Q states, each corresponding to unique vector $\boldsymbol{q}^t = [q_1^t \ldots q_C^t]$, where each q_i is either one or zero. Thus there could be total 2^C states. However, only the ones for which there exists one or several morphs with an equal code are used. As in the IBM model in Eq. 5.3, we have the transition probabilities between the states (i.e. clusters), $P(\boldsymbol{q}^t \,|\, \boldsymbol{q}^{t-n} \ldots \boldsymbol{q}^{t-1})$, and the emission probabilities from states (clusters) to morphs, $P(m_i \,|\, \boldsymbol{q}^t)$.

Emissions are estimated from the training corpus as maximum likelihood (ML) estimates. A state may produce only those morphs that have equal active components ($b_k^i = q_k^t \ \forall k$), the most common morph being the most probable.

Respectively we must estimate the transition probabilities. It can be done as in standard N-gram models. As a smoothing method we used modified Kneser-Ney interpolation, which is a state-of-the-art smoothing technique producing good estimates also for high-order n-grams [9].

5.3 Experiments

We used *MorphSetA* (details in Sec. 3.3) for the experiments. Thus we had a lexicon of 89 368 morphs, and both the training and evaluation corpus segmented to them. The latent features were calculated for all of the morphs, using training corpus as described in 4.3. After that we clustered the morphs using the binarized codes, and last estimated class-based N-gram models from the training data. The estimation of the class n-gram probabilities was carried out with SRI toolkit [62], utilizing modified Kneser-Ney interpolation. As a baseline language model we had a morph-based N-gram model using modified Kneser-Ney interpolation, estimated also with the SRI toolkit.

We calculated cross-entropy of the models over the test data. As we are comparing models that use the same set of units, we did not calculate the normalized word-based entropies. Thus the cross-entropies reported later are not comparable with e.g. ones counted with word-based models.[1]

In addition to the entropies we can consider sizes of the models. In standard ARPA format N-gram models are presented one n-gram per line, so the number of lines is one measure that is not affected by the naming of the classes. In our class models there are less units, so the length is smaller with short n-grams; on the other hand with longer n-grams more class combinations may be found.

In Figure 5.2 there are entropy against size curves for three class-based N-gram models and the baseline model. We see that the class-based N-gram models get very near to the standard N-gram models, regardless of the n-gram length, as both the context length used in extracting the ICA components and the length of the resulting code increases.

The properties of the estimated models are presented in Table 5.1 in more details. The various parameters of the models are presented in the table as follows: In the N-gram model, Q is the number of transition classes in the model. Context definitions and F- and M-limits defined earlier in section 4.3.1 are concatenated to present the feature extraction parameters (e.g. CR2 F1080 M1 means that context positions are two nearest on the right side, feature morphs must have occurred 1 080 times and all units are taken to focus morphs). The length of the counted context feature vector was about

[1]The word-based entropy of our baseline 5-gram model was 14.11 bits. In [26], same kind of morph-based language models got similar entropies for harder evaluation data (a book), and a little below 13 bits for easier (broadcast news). However, there they used over twice as much training data and different preprocessing for the text.

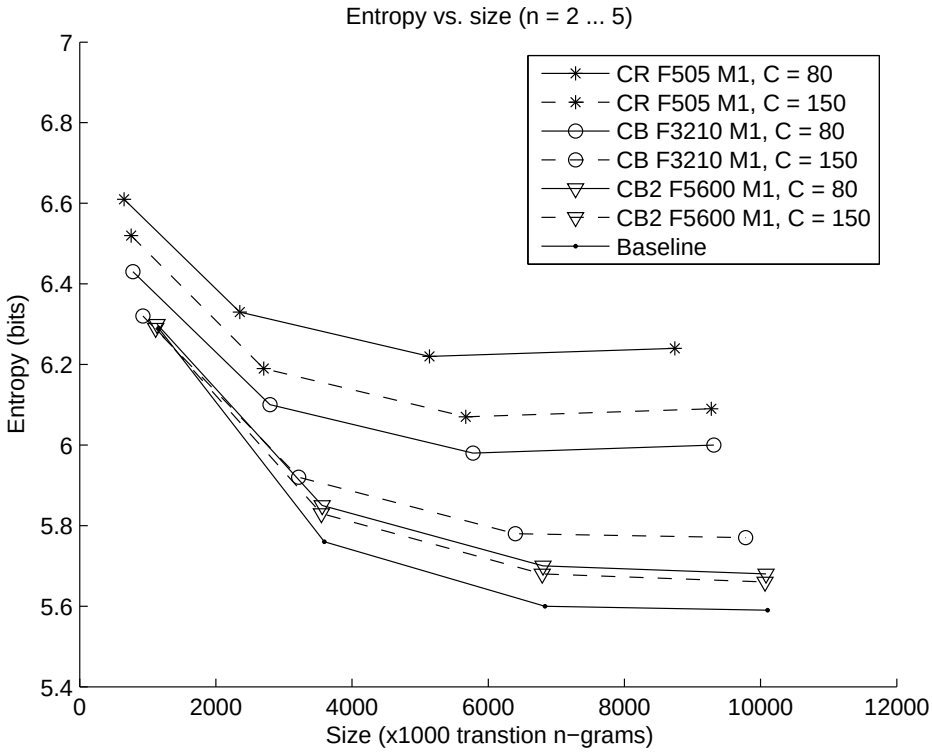

Figure 5.2: The entropies and model sizes for some N-gram models of order 2–5. Baseline is a standard N-gram model. For explanation for the parameters of the class based models, see text.

1 000 for all parameter settings.[2] C is the number of extracted independent components. Finally, n is the maximum n-gram length used in model.

From Table 5.1 we can see the reason for the proximity of the baseline N-gram models and those class-based models in which more context information was used in the clustering. Upper in the table there are figures for models which had less context information for the extraction of the latent features. The first one, CR F505 M1, has only context information on the nearest morph on the right. That resulted in about 26 000 and 33 000 clusters after the binarization, for component numbers 80 and 150, respectively. As we move down, the context information increases, and at the same time the number of classes, Q, increases close to the total number of morphs, 89 368.

[2]I.e., for CR F505 M1, there were 1 000 context morphs, but for CB2 F5600 M1 only quarter of that, 250, for they were examined in four different positions.

All common morphs start to get unique binary vectors and thus their own classes. If each morph had its own class, the model would totally converge with the corresponding morph-based model.

We also tried to interpolate our class-based models with standard N-gram models to see if they would help. For interpolation coefficients small enough (0.1 and below for the class-based model) some marginal tweak could be gained, but the decrease was below one percent in perplexity and thus indistinguishable in entropy. This confirms further that the models are close to each other.

To answer the questions presented at the beginning of this chapter, we have seen that the there are no problems in using the latent features in clustering. In addition, we have one special clustering method, i.e. binarization, that is suitable only for sparse codes such as the ICA components. From Table 5.1 we can see that both increasing the context used in deriving the latent features and increasing the number of components give more fined-tuned features and result in better predictions for the model. However, the five most accurate settings, CR2 with 150 components, CB2 with 80 and 150 components and CB3 with 80 and 150 components, give results that are very near to each other. The main reason for this is probably that the number of clusters starts to reach its limit, the number of morphs.

5.4 Discussion

Experiments considered in this chapter were quite preliminary. The presented class-based model uses similar methods as standard N-gram models; thus it is natural that the results are close to each other. We lose some accuracy due to the clustering of the morphs, but also save a little space.

Recent research on several different class-based N-gram models [22] includes a similar class-based model as ours, the IBM model. The results of that model compared to traditional word trigram models, with clustering based on maximizing the average mutual information, were also similar: Perplexity vs. size was worse in the class model, except for the smallest models. Thus the reason of our quite poor results might be more influenced by the model than the clustering method.

The same research gives best results for an asymmetric cluster model, which includes two different clustering: One for the predicted words (predicted clusters), and one for the words in the history (conditional clusters). We could use the ICA based clusters also for that model, by calculating two

different latent features for each morph. A natural idea would be to calculate predicted clusters based on the left context data, and conditional clusters one based on the right context data.

However, a more essential result from this chapter is that when we have already a compact set of language units (morphs), no noteworthy unit clusters seemed to emerge from distributed features based on analysis of the co-occurrence distributions. I.e., when we (1) use a feature vector that is long enough so that essential information can be included, (2) do not restrict the input matrix to include information only on the occurrences of the nearest one or two morphs, and (3) consider only the morphs for which we have a good amount of context information, each morph will have a cluster of its own. Thus, if we trust that the PCA[3] compress the contextual information well, reducing only noise, it seems that there is nothing to cluster. Of course, that does not mean that the features would not be of use. It only means that hard clusters based on the full set of components are not useful.

If we do not reduce the number of model parameters by clustering of the morph lexicon, is there something else to cluster? The answer is, of course, yes. In the next chapters we start to develop methods for clustering *sequences* of morphs.

[3]Remember from Sec. 4.2 that the dimension reduction part of the method is based on PCA.

Table 5.1: Details of the estimated N-gram models. As the context information increases, number of clusters (Q) in class-based models approaches the number of the morphs, and entropies come close to those of the morph-based N-gram model.

Morph-based n-gram model

	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$
$Q = 89\,368$	8.35	6.29	5.76	5.60	5.59

Class-based n-gram models

CR F505 M1

	$n=2$	$n=3$	$n=4$	$n=5$
$C = 80, Q = 25\,976$	6.61	6.33	6.22	6.24
$C = 150, Q = 33\,444$	6.52	6.19	6.07	6.09

CB F3210 M1

	$n=2$	$n=3$	$n=4$	$n=5$
$C = 80, Q = 29\,987$	6.43	6.10	5.98	6.00
$C = 150, Q = 42\,084$	6.32	5.92	5.78	5.77

CR2 F1080 M1

	$n=2$	$n=3$	$n=4$	$n=5$
$C = 80, Q = 47\,137$	6.36	5.94	5.79	5.88
$C = 150, Q = 56\,648$	6.30	5.85	5.70	5.68

CB2 F5600 M1

	$n=2$	$n=3$	$n=4$	$n=5$
$C = 80, Q = 71\,747$	6.31	5.86	5.71	5.69
$C = 150, Q = 76\,446$	6.29	5.83	5.68	5.66

CB3 F7910 M1

	$n=2$	$n=3$	$n=4$	$n=5$
$C = 80, Q = 81\,920$	6.30	5.85	5.70	5.68
$C = 150, Q = 82\,298$	6.29	5.83	5.68	5.66

Chapter 6

Modeling n-grams with Self-Organizing Map

In Chapter 4 we binarized the latent features found by ICA to a discrete space. To some extent the binarization can be grounded by the sparseness of the components: Sparse distribution has little values that are "in between". However, it is clear that some information is lost in the process (i.e. how strongly one component is active for a certain morph). And on the other hand, most natural data is not nominal, and (written) language is one of the rare exceptions. If we can project language data to a real-valued space, we have many general algorithms and methods developed for other kinds of natural data available. The latent feature vectors are one possibility for this.

Consider a morph sequence (n-gram) $(m_{i-n+1}, \ldots, m_i)$. If we have a C-dimensional vector $\boldsymbol{s}_j$ for each morph m_j, and we replace each morph by its vector, we will have a C-dimensional time series of length n. If we could learn how to predict the vector of m_i using the concatenated history vector $(\boldsymbol{s}_{i-n+1}, \ldots, \boldsymbol{s}_{i-1})$, we would have a model for the language.

A problem like this is called *multivariate time series prediction*. Prediction methods are usually based on assumption of some local smoothness properties [4]. This means that small changes in the previous signal values results only in small changes in the prediction. Neural networks and Gaussian mixture models are some common tools used for the task.

In this chapter we use a neural network algorithm called *Self-Organizing Map* [36] to predict morphs using the feature vectors of the preceding morphs. The Self-Organizing Map has been effectively used for a wide range of applications, including some problems of natural language modeling. We will see that our method can be though as a clustering of the morph histories to a

set of prototypes, which for their part predict the next morph.

The application of self-organizing maps in language modeling is also interesting from the view of neuroscience. There are evidence on representations resembling these artificial neural maps in areas of the brain cortex that process sensory perceptions [36]. We may well assume that similar learning principles are used within the whole brain, thus also in the areas related to language processing [40]. Modeling data with models resembling those by which it is generated should have its benefits.

6.1 Introduction to Self-Organizing Map

The Self-Organizing Map (SOM, Kohonen map) is a type of neural network that is widely adopted for several reasons: It is computationally efficient, easy to understand, and it creates visualizations of the data that are clear to humans. In this section we describe the basic SOM algorithm according to the book *Self-Organizing Maps* by Kohonen [36]. Refer to it for an extensive depiction of the algorithm and its applications.

A Self-Organizing Map is usually constructed as a two dimensional one layer network. Each neuron of the network, or *map unit*, is connected to adjacent map units thus forming its neighborhood. The map is adapted to multidimensional data using competitive learning and co-operation of the neighborhoods.

Let $\boldsymbol{x}$ be a data point in d-dimensional space; $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T$. Every map unit (neuron) i has a corresponding d-dimensional weight vector $\boldsymbol{w}_i = [w_{i1}, w_{i2}, \ldots, w_{id}]^T$, $i = 1, 2, \ldots, l$. In competitive learning we choose so called winner neuron or *best matching unit* (BMU) for each data sample $\boldsymbol{x}$:

$$c = \arg\min_{i=1,\ldots,l} ||\boldsymbol{x} - \boldsymbol{w}_i|| \tag{6.1}$$

After choosing the winner neuron, it and its neighborhood is moved towards the data point in such way that the further a map unit is from the BMU, the less it is moved. The *neighborhood function* is usually set to be Gaussian:

$$h_{ci} = \exp(-\frac{||r_c - r_i||^2}{2\sigma^2}), \tag{6.2}$$

where r_c and r_i are the location vector of map units c and i in the map array.

To stabilize the adaptation of the map, a learning-rate factor $0 < \alpha(t) < 1$ is needed. It is reduced after each update round t. Also the neighborhood

parameter σ^2 can be changed during the learning. For example, first one may use a large value so that the map quickly moves to the dense areas of the data cloud, and next a small value so that the map spreads out better.

Using these definitions, the update rule for the map units is the following:

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \alpha(t)h_{ci}(t)(\boldsymbol{x} - \boldsymbol{w}_i(t)), \quad i = 1, 2, \ldots, l \tag{6.3}$$

The training can be done in batch mode, which means that the changes are first calculated for all samples of the training data (one epoch) and the weights are updated after it.

6.2 A SOM-based N-gram model

The basic Self-Organizing Map algorithm was not developed for nominal (symbolic) data, but there are several methods to project the symbols to numerical values [37], e.g. by Latent Semantic Analysis [19] or random projection. As described in the beginning of this chapter, we will start with the idea of predicting language signal that is formed by distributed latent features of the morphs.

Using latent features presented in Chapter 4, each morph has its own C-dimensional vector $\boldsymbol{s}_j$. A morph n-gram $(m_{i-n+1}, \ldots, m_i)$ can thus be mapped into matrix $\boldsymbol{S}^{(i-n+1,i)} = \boldsymbol{S}^{(i;n)} = (\boldsymbol{s}_{i-n+1}, \ldots, \boldsymbol{s}_i)$, where columns are different latent components and rows follow time, i.e. positions of the morphs in the sequence. These matrices can be collected from the training corpus by first extracting the different n-grams and then replacing each morph by its numeric representation.

The SOM algorithm uses only $[d \times 1]$ dimensional data points, i.e. vectors. If we make the decision that each value of the matrix $\boldsymbol{S}^{(i;n)}$ is initially as important as the others, we can reshape the matrix into concatenated vector $\boldsymbol{u}^{(i;n)}$, where $u^{(i;n)}_{C\times j+i} = \boldsymbol{S}^{(i;n)}_{ij}$. Each vector $\boldsymbol{u}^{(i;n)}$ (for a fixed n and varying i) is a data point for training the SOM. If size of the map is $M_x \times M_y$, it forms the same number of prototype vectors for the n-grams. We will denote the prototypes (map units) by c_k, $k = 1, 2, \ldots, M_x \times M_y$, each associated with a d-dimensional weight vector $\boldsymbol{w}_k$, where $d = nC$. Based on training procedure described in previous section, they will try to move into the places where the data points are dense. Self-Organizing Maps trained with this kind of vectors made by concatenating vectors of successive words are usually called *semantic* or *contextual maps* [36].

After the SOM is trained, we can directly use it for predicting the feature

vector of the next morph when we know the feature vectors of the history. We can find a set of best matching map units (BMUs) for the history, in the subspace of $(n-1) \times C$ first dimensions. Then each of the BMUs would give us one prediction for the feature of the next morph.

If we want to use the information contained in longer n-grams, but either give less weight to further morphs or just reduce the dimension to less than $n \times C$, we can calculate average of the k first morph features and use that vector for the first C dimensions. Of course the vector of the last morph (the one to predict) should be left as it is, as well as the previous one (as it is the most important for the prediction).

Let us say that we find K BMUs for the vector $\boldsymbol{u}$ of a seen morph history h, each corresponding to one n-gram prototype c_k. Next we need prediction distributions for the prototypes. Two ways of estimating them were tested. After the estimation, we have a full generative model

$$P(m \mid h) = \sum_{k \in \mathrm{BMUs}(\boldsymbol{u},K)} P(m \mid c_k) P(c_k \mid h), \tag{6.4}$$

where $P(c_k \mid h)$ can be simply $1/K$ for all $k \in \mathrm{BMUs}(\boldsymbol{u}, K)$.

Figure 6.1 illustrates the training procedure. Steps (1) and (2) are done first, then the emission distributions $P(m \mid c_k)$ are estimated either by method (3a) or (3b). The first method is simply to calculate distances from prototypes to morphs in the last C dimensions of the SOM space (denoted by superscript $[d-C+1, d]$), as those correspond to the last morph of the n-gram. The nearer the morph is, the more probable it will be. We chose the inverse Euclidean distances to be proportional to the probabilities, i.e.

$$P(m_i \mid c_k) = \frac{||\boldsymbol{s}_i - \boldsymbol{w}_k^{[d-C+1,d]}||^{-1}}{\sum_j ||\boldsymbol{s}_j - \boldsymbol{w}_k^{[d-C+1,d]}||^{-1}}, \tag{6.5}$$

We tried also a more explicit estimation of emission distributions. After the SOM was trained and positions of the prototypes determined, we went through the training n-grams again, and estimated the emission distributions for the prototypes directly as smoothed ML estimates. Smoothing was done using one of the simplest methods, additive smoothing [9]. Thus the estimate was

$$P(m_i \mid c_k) = \frac{C(m_i, c_k) + \lambda}{C(c_k) + M\lambda}, \tag{6.6}$$

where $C(c_k)$ is the number of times c_k was chosen for BMU for the histories of the training n-grams, $C(m_i, c_k)$ is the number of times c_k was chosen for

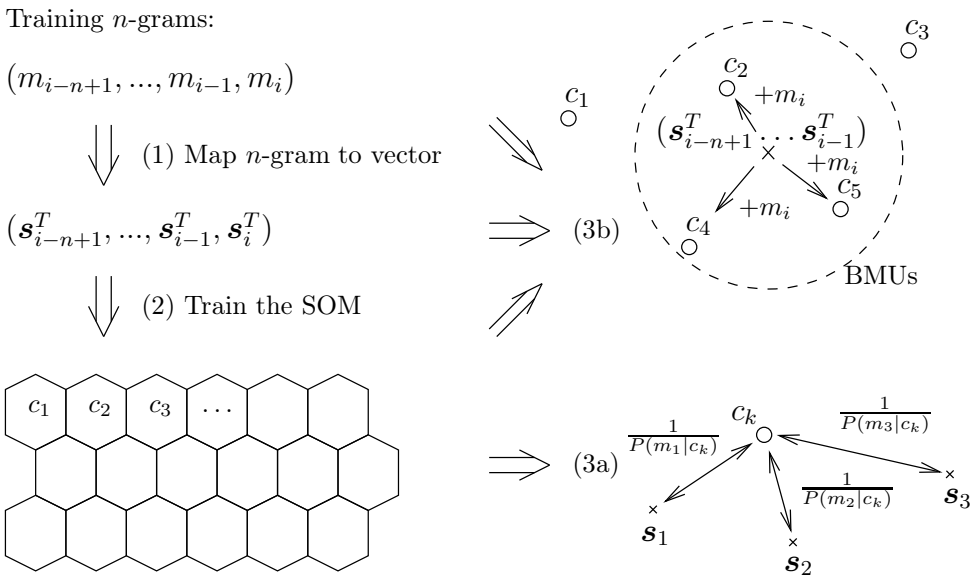

Figure 6.1: Building the SOM model. Training n-grams are mapped to vectors using the distributed features for morphs. Then the map is trained. Last, the emission distributions from map units is either estimated using distances from units to single morph vectors, or trained explicitly using the training data.

BMU when the last morph of the training n-gram was m_i, M is the number of the morphs, and λ is a parameter that tells how much weight is given to the prior belief that all morphs are of equal probability.

6.3 Experiments

We did a set of experiments in order to find out whether time series prediction of the morph features, and especially with the Self-Organized Map applied to the task, could be used to model n-grams, and how well this kind of simple approach could make it compared to standard N-gram models.

We used the reduced set of morphs, *MorphSetB*, for the experiments. Thus the lexicon consisted of 10 528 morphs. (See Chapter 3 for details of the corpora and segmentations.) Again, we calculated modified Kneser-Ney interpolated N-gram models for comparison, and calculated cross-entropies of

all the models on the evaluation corpus.[1]

There are quite a number of parameters that must be set when training the language model, e.g. n-gram length, number of components in morph features, map size, training parameters of the map, and number of BMUs to use. As training and evaluation of the full models is not fast, we did not want to try the effect of every parameter, but made reasonable decisions for those we could, and kept to them. For morph feature vectors we used the ICA extraction setting CB2 F11255 M1 (see Sec. 4.3.1) and 80 components. Tested map sizes were 100×60 and 120×75. Number of the BMUs was mainly kept to 20, but also 10 and 50 were tried. For model order, 2-, 3- and 4-grams were tested (4-grams also with averaged first two feature vectors).

The details of the results are presented in Table 6.1. Results of the standard N-gram model trained with the same units and data is presented first. In SOM-based models, emission column means which of the two methods presented in the previous section was used for emissions from BMUs to morphs. The last model was trained with 4-grams so that the vectors of the first two morphs were averaged.

The SOM-based models that used distance based emission distribution for map units (Eq. 6.5) gave quite rough estimates. The evaluation results of a 4-gram model, morph entropy of 7.34 bits, were slightly better than standard morph unigram model (entropy 7.96 bits) but worse than bigram (5.62 bits). SOM trained with bigrams managed to the entropy of 7.59 bits. Linear interpolation of the SOM-based and standard N-gram models did not work any better.

We got somewhat better cross-entropy from the models where the emission distributions were estimated directly from the training data as smoothed ML estimates (with smoothing parameter $\lambda = 0.01$). The best result was 6.00 bits for a 4-gram model where vectors of the first two morphs were combined to one average vector. However, if any of the SOM models with ML estimates were interpolated with the N-gram model of the same order, no benefit was gained. Entropy started to increase as soon as coefficient of the SOM model was set above zero.

[1] Again, we use the unnormalized (i.e. morph-based) entropies. This time the word-based cross-entropy for the baseline 5-gram model was 13.98 bits, i.e. little less than what was obtained with *MorphSetA* used in Chapter 5.

Table 6.1: Entropies of the SOM-based models. The first emission method (dist.) gives results that are not much better than a standard unigram model. The second method (MLE) gives somewhat better results, but increasing the n-gram length over 3 starts to degrade the entropy. Averaging the values of the furthest components prevents the degrading (last model in the table).

Kneser-Ney interpolated N-gram model

n = 1	n = 2	n = 3	n = 4	n = 5
7.96	5.62	4.97	4.65	4.63

SOM-based models

n	Emission	Map size	K	Entropy
2	Dist.	100×60	10	7.66
2	Dist.	100×60	20	7.63
2	Dist.	100×60	50	7.59
4	Dist.	120×75	10	7.38
4	Dist.	120×75	20	7.34
4	Dist.	120×75	50	7.38
2	MLE	100×60	20	6.13
3	MLE	100×60	20	6.11
4	MLE	120×75	20	6.18
4 (av)	MLE	100×60	20	6.00

6.4 Discussion

The presented method for SOM-based n-gram modeling did not work very well. It gave reasonable results when ML estimation was used in emissions, but then the estimates were near those given by standard N-gram models, and interpolation was of no use.

However, neural networks should have something to give to the language modeling problem. N-gram models are really "dumb" models compared to what human does when dealing with language: Definitely humans do not save lots and lots of probabilities of sequences of text fragments in their memory. Meanwhile in particular the neural network algorithms have taken ideas from how the brain works.

A highly relevant paper for neural network language models is one by Bengio et al. [4]. They managed to build a 5-gram neural model that worked

better than conventional state-of-the-art 5-gram models. They used a multi-layer network which had the distributed features of previous words as input vector and probability for each possible word for output vector. The multi-layer perceptron network and the distributed representations were learned simultaneously. They reported that computational requirements for training the network grow only in a linear fashion relative to the size of the model. Regardless of that, the training phase took over 3 weeks using 40 CPUs.

Another experiment in neural network language modeling is done by Xu and Rudnicky [68]. They did not use any distributed coding for the words, but the inputs and outputs had both one connection for one vocabulary item. To make the task computationally feasible, they restricted the experiment to small vocabulary (1200 words and word classes) and bigram modeling. Also they reported somewhat better perplexity for the neural network model than for Kneser-Ney interpolated bigram model.

As seen from these two examples, a major problem in using neural networks in language modeling is in the computational complexity of the learning algorithms. Compared to the multi-layer perceptrons that were used in the research mentioned above, Self-Organizing Map is very fast to learn even when large datasets are used. It has been successfully used in language technology for text retrieval and mining (e.g. the WEBSOM method [37, 42]), but apparently not much in statistical language modeling. One exception is the research by Kurimo and Lagus [39, 43], where Self-Organizing Map was successfully used to adapt conventional N-gram model to different topics. Our experiment shows that SOM can be used also directly to model the n-grams, but the conventional algorithm seems to be too restricted. Some ideas could probably be adopted from the SOM extensions used in time series prediction.

Chapter 7

N-gram models based on morph history clustering

In Chapter 6 we tried to cluster morph histories encoded as latent features obtained using ICA, using Self-Organizing Map to the task. One major problem in the approach was that two histories that precede same kinds of morphs may not look alike at all, when examining the morphs of which they are compounded of.

For example, consider English phrases "regardless of" and "in spite of". Their meaning is nearly the same, and in most cases one could be replaced by another. However, if we replace each morph by a distributed latent feature, "regardless" and "spite" have probably somewhat different vectors. At least they should have, as they mean different things.

There might well be some histories that would be worth to cluster. Benefits of the successful clustering should be both better estimated parameters (due to their reduced number) and a smaller size of the model. However, thinking of the former examples, the clustering method should be insensitive to the history lengths.

Thus, if we combine the features of the fragments of the history, we should somehow be able to normalize the result. Taking an average of the fragments is not a good idea, as the result will then contain information which will actually predict the events inside or even before the history (depending on how the features are obtained). Separating that from the relevant information is not simple. In our morph-based models, it will be easier if the clusters are not produced by examining the individual morphs in the history, but by examining the morphs that are *followed* by it.

In addition to clustering n-gram histories to equivalence classes, we use *Maximum A Posteriori* (MAP) estimation in order to make decisions on which histories are put into the same cluster. Like the MDL principle presented in 2.2.3, also the MAP framework can be used to produce models that are compact in size.

In the following section we will introduce the background on the MAP estimation, and see how it is related to MDL. After that we discuss a few works on language models that resemble our model either due to the usage of MDL principle or similar history clustering. Then we present the two models that we experimented with, and finally discuss the results.

7.1 Compactness using MAP estimation

Let us think of how N-gram models are usually built. From the training corpus we collect all different n-grams and their frequencies. Probability for the n-gram $w_{i-(n-1)}...w_i$ is estimated by the following ratio of occurrences:

$$\frac{C(w_{i-(n-1)}...w_i)}{C(w_{i-(n-1)}...w_{i-1})}, \tag{7.1}$$

which is then smoothed, giving some probability mass to unseen events as well. Those n-grams that do not occur in the training data will have a zero probability and back-off weight one.

If we try to reduce both entropy and the size of a model, it might not be a good idea to add every found n-gram to the model with its own parameters. We might have a history $w_{i-(n-1)}...w_{i-1}$ that has the same prediction distribution as a bigram history w_{i-1}, or an n-gram history that has the same prediction history as some other n-gram. In the first case, n-gram could well be dropped out of the model. In the second case, we could make a model in which the two histories would have the same prediction distribution and estimate it using the occurrences of the both.

In a bit more general level, traditional N-gram models are based on maximum likelihood (ML) estimates (such as Eq. 7.1), i.e., they try to find a model G_{ML} that maximizes the likelihood of the data O:

$$G_{\mathrm{ML}} = \arg\max_G P(O \mid G) \tag{7.2}$$

If the data is coded by the model G_{ML}, we know from the information theory that the minimum number of bits needed in the coding is

$$L(O \mid G_{\mathrm{ML}}) = -\log_2 P(O \mid G_{\mathrm{ML}}). \tag{7.3}$$

Thus, if we maximize the likelihood $P(O \mid G)$ we minimize the coding length of the data given the model. But this is not what we really want. If the model is flexible enough, it will overlearn the training data, and not generalize it.[1] An extreme example is that our model G could just include all the data O, declare that $P(O \mid G) = 1$ if and only if O is the training data, and thus predict it perfectly. But then every other data O' that differs from O would have a zero probability.

Instead of maximizing $P(O \mid G)$, what we are interested in, is to find the model G that is the most probable when we know the data O that it has generated. Using the Bayes rule

$$P(G \mid O) = \frac{P(G)P(O \mid G)}{P(O)}, \tag{7.4}$$

we get

$$G_{\mathrm{MAP}} = \arg\max_G P(G \mid O) = \arg\max_G P(G)P(O \mid G), \tag{7.5}$$

as the prior probability $P(O)$ is not affected by the G. This estimate is called *Maximum A Posteriori* (MAP) estimate [23].

With a suitable prior probability $P(G)$, MAP estimation has a direct connection to the Minimum Description Length (MDL) principle described in Sec. 2.2.3. This is easiest to see by taking logarithm of the MAP product:

$$\begin{aligned} G_{\mathrm{MAP}} &= \arg\max_G P(G)P(O \mid G) \\ &= \arg\min_G [-\log_2 P(G)P(O \mid G)] \\ &= \arg\min_G [-\log_2 P(G) - \log_2 P(O \mid G)] \\ &= \arg\min_G [L(G) + L(O \mid G)], \end{aligned} \tag{7.6}$$

where L is the length of the optimal coding. Now we see that we must not code only the data, but also the the model. In our language model example, it is not anymore worthwhile to include all the n-grams of the training data in the model, but only those that are most useful in the prediction of the data. A good introduction to the connection between Bayesian and MDL frameworks is found in Stanley F. Chen's PhD thesis [8].

[1] N-gram models, even if based on strict ML estimates, do not totally overfit, because of the n-gram assumption they cannot express full histories of a training corpus that is longer than n. However, they do overlearn quite a lot, as discussed in Sec. 2.1.

7.2 Related work

Ristad and Thomas [56] utilize the Minimum Description Length principle as a model selection criterion in their non-monotonic extension model. Their application is *character*-based language modeling: Model predicts the following character given the previous characters. The model is a Markov model and thus resembles N-gram models, but there are some differences. The model does not include all the n-grams in the training data, but utilizes greedy heuristics to add the most profitable extensions first. Extension is a relation between histories and predicted characters, and for each existing extension there is a corresponding probability value. They use a smoothing technique that follows PPM data compression scheme by Moffat [49] and resembles the better known *Witten-Bell smoothing* [9].

A word-based N-gram model utilizing MDL in model estimation is studied by Siivola and Pellom [60, 61]. They build the model incrementally, starting with a unigram model and adding new n-grams in sets. The new n-grams are accepted if the change is total code length is negative. They also apply interpolated Kneser-Ney smoothing for the distributions. The reported size vs. entropy results were better than for baseline N-gram models. The growing model also outperformed *entropy pruned* models. Entropy pruning is a method where one starts with a full N-gram model and drops out those n-grams that affect least the cross-entropy for some trimming data.

We will use the model presented in [61] (referred later as *growing n-gram model*) to compare the results with our own models.[2]

To our knowledge, there is not much research done on models where n-gram histories would be clustered into equivalence classes in a similar manner that we will do. Goodman [24] mentions the idea in his section that concerns clustering, but solely states that there are many difficult issues to solve in it. One exception is the work by Xu and Jelinek [66], where Random Forests (randomly grown Decision Trees) were used to construct a language model. They also applied a smoothing similar to interpolated Kneser-Ney smoothing to the model. The results were promising: Their model outperformed traditional KN-smoothed N-gram model in both perplexity (i.e. cross-entropy) and word error rate (percentage of incorrectly recognized words in speech recognition).

[2]Thanks to Mr. Siivola for giving the related software and instructions for using it.

7.3 An N-gram model for clustered histories

The models presented in this chapter utilize hard clustering of n-gram histories. As touched in the beginning of this chapter, a natural way of clustering is to consider the prediction distributions of the histories, and to put those histories that have similar ones into the same equivalence classes. Each equivalence class, or cluster, will then share the same distribution over the predicted words.

We start with collecting all morph n-grams (for a selected n) and their frequencies from the training data. Using those, we can calculate a ML prediction distribution for each $(n-1)$-gram (i.e. n-gram history) as in Eq. 7.1. Let us mark the predicted morph as $m_i\,(i = 1, \ldots, M)$ and each history type as $h_j\,(j = 1, \ldots, H)$. We try to find a set of clusters of histories $c_k\,(k = 1, \ldots, C)$ so that each history belongs to one cluster. In addition, we assume that the next morph depends only on the cluster of the history, not the history itself. When predicting the next morph m for the known history h, we get:

$$\begin{aligned} P(m \,|\, h) &= \sum_k P(m, c_k \,|\, h) \\ &= \sum_k P(c_k \,|\, h) P(m \,|\, c_k, h) \\ &= \sum_k P(c_k \,|\, h) P(m \,|\, c_k) \\ &= P(m \,|\, c(h)), \end{aligned} \tag{7.7}$$

where $c(h)$ is the cluster of the history h. For this kind of model to work in practice, those histories that belong to the same cluster should have as similar prediction distributions as possible.

The similarity of the distributions can be measured by several methods. We use the *information radius*, which is based on Kullback-Leibler divergence of distributions. KL-divergence of the distribution p from the distribution q is

$$D(p \,||\, q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{7.8}$$

and it measures how much information is lost if distribution q is assumed when the correct distribution is p. Information radius between the distributions p and q is

$$\mathrm{IRad}(p \,||\, q) = D(p \,||\, \frac{p+q}{2}) + D(q \,||\, \frac{p+q}{2}). \tag{7.9}$$

We can see that it is symmetric regarding the distributions, and corresponds to how much information is lost if an average distribution is used instead of both distributions.

If the observed full history h is not known in the model, we can try to drop the first morphs away until we find histories of the model that have the same last morphs. Then we can calculate the distribution by averaging over distributions of those histories. If we save the counts of the history types in the model, we can weight the average accordingly.

Using ML estimates for emission distributions $P(m \mid c_k)$ is of course a bad idea, because we need a probability above zero for all events. Thus we will smooth the distributions using additive smoothing as in Eq. 6.6.

7.3.1 Model cost

In order to get a compact model, we try to find a MAP solution such as in Eq. 7.6. In our set of models G, each has a number of parameters that are the same for a given training data. Those are the set of the morphs (size M) and the histories in the training data (h_j, total H) and their counts ($C(h_j)$, $\sum_j C(h_j) = N$). The varying parameters are number of the clusters (C), mapping from the histories to the clusters, and prediction distributions of the clusters.

When considering the changes in the MAP probability, the static parameters of the model may be skipped. The following are taken into account:

1. **Probability of the model**

   (a) *Number of the clusters*
   If we do not limit the number anyhow, it can be encoded using the following number of bits [55]:

   $$L(C) \approx \log c + \log C + \log\log C + \log\log\log C + \ldots, \quad (7.10)$$

   where the sum includes all positive iterates and c is a constant (about 2.865). The corresponding probability is $P(C) = 2^{-L(C)}$.

   (b) *Clusters of the histories*
   Let us assume that the first history is in cluster c_1. The rest $H - 1$ histories are divided up to C clusters. There are C^{H-1} possible combinations, and if we give an equal probability for each

combination, $P(\text{clustering}) = (C^{H-1})^{-1}$. The corresponding code length is

$$L(\text{clustering}) = -\log P(\text{clustering}) = (H-1)\log C. \tag{7.11}$$

(c) *Frequencies of the clusters*
Frequencies of the histories are our non-variant information, and together with the cluster combination above we can get the cluster frequencies without additional coding bits.

(d) *Prediction distributions*
We know the frequency of each cluster, so for every one we need to code M non-negative integers that sum up to the frequency $f(c)$. Giving an equal probability to each distribution,

$$P(\text{distributions}) = \prod_c \binom{f(c)+M-1}{M-1}^{-1} \tag{7.12}$$

and

$$L(\text{distributions}) = \sum_c \log \binom{f(c)+M-1}{M-1}. \tag{7.13}$$

2. **Probability of the data given the model**
Probability of the data given the model can be calculated as a probability of any sequence of text:

$$P(O \mid G) = \prod_i P(m_i \mid h_i, G) \tag{7.14}$$

and thus

$$L(O \mid G) = \sum_i -\log P(m_i \mid h_i, G). \tag{7.15}$$

Note that we assume that the n-grams from which the model is built are always the same, and thus their histories are not included into the model length. Similarly conventional N-gram models include all the n-grams of the training data. For now we only try to find a good clustering for the n-grams, not to find which n-grams are worth to include into the model.

7.3.2 Search algorithm

The following algorithm was designed in order to find a set of clusters that maximizes the posterior probability:

1. Start with each history in its own cluster. Count prediction distributions for each using ML estimates.

2. Join those clusters that have exactly equal distributions. This is fast as long as the number of non-zero emission probabilities is small. Larger clusters may well be skipped, as it is anyway improbable that there would be equal distributions in those.

3. Sort the clusters by their frequencies. Collect K largest to a list L, so that the largest is the first (list index zero). Initialize a counter $t = 0$.

4. Calculate the information radius between distribution of cluster indicated by $L(0)$ and each other cluster in the list. Choose the cluster with smallest radius (list index j) and calculate the change in posterior probability if it is merged with $L(0)$.
   If the probability increases, go to step 5, otherwise to step 6.

5. Merge the cluster $L(0)$ to the cluster $L(j)$ and remove it from the list. Add the next largest cluster to the list: If K has been increased, add to the beginning of the list, otherwise to the end. If there is no more clusters outside the list, decrease K by one. Set t to zero. Continue from step 4.

6. Move $L(0)$ from the beginning of the list to the end. Increase t by one. If $t < K$, continue from step 4.
   If $t = K$, check if there are more clusters left outside the list. If there are, increase K by one, add the next largest cluster to the beginning of the list and continue from step 4. Otherwise stop.

Of course, there is no guarantee that the algorithm will find a global maximum for the posterior probability. It is greedy in a sense that it tries to merge the history clusters that are most alike, i.e. have the smallest information radius. However, it is not greedy in a way that it would make those linkages first that would increase the posterior probability most. The greediness is indirect for three reasons: An extensive search of the cluster pairs cannot be done, sometimes the largest gain to the probability comes from merging two large clusters even if they are not much alike, and calculating the probability change is slower than calculating the information radius.

The parameter K controls how extensively the clusters are compared in order to find the minimum information radius. Small values make the algorithm faster, and as the size will increase if needed, we could start with the smallest possible value, two.

It may seem strange that the clustering of the histories is started from the ones with the largest frequencies. There is a good reason for this: Merging two small clusters often increases the probability even if they are not at all alike, since the data cost increases only marginally. As K cannot be very large because of the the efficiency reasons, if we started from the smallest clusters, we would start by combining to each other those histories that we know little of.

Instead, when we first compare the two most frequent histories, it is very likely that they are unlike and the merging would decrease the posterior probability as errors in their distributions affect the data probability much. Thus we actually start by increasing the K so much that we start to find some genuinely similar distributions.

7.3.3 Experiments

We constructed morph 2-gram, 3-gram, and 4-gram models using the same training corpus as in previous chapters. The set of morphs was *MorphSetB* as in Chapter 6, and thus the lexicon size was 10 528. They were evaluated by calculating cross-entropy for the 50 000 word corpus in different phases of the training: First before the clustering was started, then couple of times in the middle of the training, and last when the training algorithm was stopped.

The entropy results are in Table 7.1. For each order of cluster model, the first row is for starting phase (no clustering done) and the last row for ending phase (algorithm had stopped). Results are not even near to those of the standard N-gram model (second column of the table) but nevertheless interesting.

The entropy of the bigram model increases the more histories are clustered. The change is largest at the beginning, then evens out, which is natural as the most frequent histories are clustered first (as explained in Sec. 7.3.2).

The trigram model starts with the smallest entropy, 5.70 bits. As the first third of the clusters are gone, entropy has increased to 5.94 bits. After that the entropy starts to decrease, and as the algorithm clusters most distributions it reaches 5.85 bits. The last 10 000 clusters can be merged without any more increment.

The highest tried order, 4-gram, behaves again differently. It starts with a high entropy (more that that of trigram), first increases a bit, but then decreases almost 10% to six bits. Entropy stays at that level until the algorithm starts to cluster the last 1 000 distributions, then increases 2 percent to 6.13

Table 7.1: Entropies of history cluster models. In all models, entropy starts to increase immediately as the clustering is started. In 2-gram model it keeps increasing all the time and reaches asymptotically 6.12 bits. In 3-gram and 4-gram models the entropy starts to decrease soon after more histories are clustered. In the 4-gram model the entropy even comes to a lower level than where it started. However, the best entropies were obtained with the 3-gram model, which implies that the model has major problems with high order n-grams.

n-gram length	Baseline entropy	Number of clusters	Entropy
2	5.62	10 198	5.78
2		9 000	5.95
2		7 000	6.05
2		1 000	6.12
2		269	6.12
3	4.97	168 027	5.70
3		100 000	5.94
3		10 000	5.85
3		1 000	5.85
3		300	5.85
4	4.65	1 257 253	6.62
4		1 000 000	6.65
4		100 000	6.02
4		10 000	6.01
4		1 000	6.01
4		143	6.13

bits.

At intermediate stage of the algorithm, merging of the clusters reduced entropy in all experiments. From that fact we might draw the conclusion that the clustering is worthwhile, but we do not have good enough method for it. Of course, it is also possible that at the beginning the algorithm made some severe mistakes, i.e. put together common histories that had very different emission distributions, and later adding new histories to this mixed cluster evens out the prediction errors.

7.4 Building the model incrementally

The model presented in Section 7.3 has some drawbacks. If we wanted to use longer n-grams, the number of histories that we start with would grow very fast. Building the 4-gram model took a couple of weeks; building a 5-gram model would be many times slower. Another problem is that we cannot back-off or interpolate with smaller N-gram models as we do in standard N-gram models, as the cluster model includes only the histories of the maximum length. Consequently it must also include all of those in order to be able to calculate correct estimates for unseen histories.

Another way of constructing a similar model is to start with shorter n-grams and add to the model only those that increase the posterior probability of the model. First we estimate an unigram model, and then start to add bigrams. For each bigram history we calculate the maximum likelihood prediction distribution. For each history we have three possibilities: Either add it to the model using a new history cluster, add it to an existing history cluster, or leave it out. For each option we estimate the posterior probability of the model, and choose the option that has the best value.

The static parameters of the model are the set of M morphs and the unigram distribution of the morphs. (It is useful to keep an accurate unigram distribution in order to have correct estimates for the situations where we do not know any history, and thus it is kept separately.) The following MDL style priors are used for the varying parameters:

1. *Number of the histories in the model (N)*
   Rissanen's prior:

$$\begin{aligned} L(N) &\approx \log c + \log N + \log\log N + \log\log\log N + \ldots \\ P(N) &= 2^{-L(N)} \end{aligned} \tag{7.16}$$

2. *Morphs of the histories*
   We do not know the history lengths. We could make probabilities for them first, but another possibility is to draw random numbers from $M+1$ choices, one for each morph and one for end of the sequence. Let us assume that $o(h)$ is the length of the history h. Thus

$$P(\text{histories}) = \prod_h (\frac{1}{M+1})^{o(h)+1}. \tag{7.17}$$

3. *Frequencies of the histories*
   If our training corpus had N_0 morphs, each frequency is at the most

that much. An integer between one and N_0 can be drawn from uniform distribution and coded using $\log N_0$ bits.

$$P(\text{history freqs}) = (\frac{1}{N_0})^N \tag{7.18}$$

4. *Number of the clusters (C)*
   Rissanen's prior as in Eq. 7.16.

5. *Clusters of the histories*
   Lets assume that the first history is in cluster c_1. The rest $N-1$ histories are divided up to C clusters. As before,

$$P(\text{clustering}) = \frac{1}{C^{N-1}}. \tag{7.19}$$

6. *Prediction distributions*
   We know the frequency of each cluster, so for every one we need to code M non-negative integers that sum up to the frequency $f(c)$. Giving an equal probability to each distribution,

$$P(\text{distributions}) = \prod_c \binom{f(c)+M-1}{M-1}^{-1}. \tag{7.20}$$

When we use the model, if we encounter a history that is not included in the model, we need to back-off. This time we do not need to calculate back-off distributions by averaging over full-length histories, but only find if the shorter n-gram exists in the model, and if not, back-off more, up to the unigram distribution if needed. Naturally we need also to smooth the cluster emission distributions, and for that we used the additive smoothing as before.

7.4.1 Experiments

We built a morph model using the incremental method described above and calculated cross-entropy for new data. The morpheme segmentation was done using *MorphSetB*, and the corpora were as defined in Chapter 3. We also trained Kneser-Ney interpolated baseline N-gram models and growing n-gram models (see 7.2 and [61]) for comparison.

In order to be able to compare the models both in entropy and size, we need some estimate for the size of the models. In the baseline N-gram model and the growing model, a clear measure of size is the number of n-grams

in the model. For each n-gram, there is at least one associated floating point number, the probability of the last morph of the n-gram given the rest. (There are also the back-off or interpolation weights, but we leave those out from this calculation.) In order to get a comparable number for the model of clustered histories, we sum up the number of histories in the model (each is associated with a integer that indicates its cluster), and for each cluster the number of morphs that have direct probability estimates for emission. Thus, if the model has 500 000 histories in 1 000 clusters, and clusters have estimates for 1 000 morphs on average, the number of parameters would be $500\,000 + 1\,000 \times 1\,000 = 1\,500\,000$.

In Table 7.2 we show the number of histories, clusters and parameters in the model after training with 2-grams and 3-grams, alongside with the cross-entropies. We started with an unigram model with no histories or clusters. When bigrams were added, the model took in almost all of the observed histories, 9 010. (Number of bigram history types in the training data was the same as the number of morph types, 10 528.) They were clustered into 543 clusters, which is about twice as much as in the former model (Table 7.1), but still in the same order of magnitude.

As the order was increased to 3-gram and 551 415 new histories (i.e. 2-gram types) of the training data given to the model, most of them were left out. The number of histories doubled to almost 18 000 and the number of clusters increased to 824. Cross-entropy dropped from 6.01 to 5.71 bits. Both 2-gram and 3-gram entropies are better than the final entropies of the models in Sec. 7.3, so at least we can say that using the shorter n-grams alongside with the full-length n-grams (as done before) is more efficient than keeping all of the full-length n-grams in the model (as done now) in terms of entropy.

Compared to the baseline N-gram model, the cluster model was left behind in entropies. However, the size of the model trained with 2-grams was 41% smaller, and for 3-gram already over 82% smaller. It seems that the growth of the cluster model as the function of training n-grams is very controlled, just as we would like it to be.

Figure 7.1 shows entropy against size curves for the baseline models (n-gram lengths 1 – 3), the model of clustered histories (after training with 1-, 2-, and 3-grams), and the growing N-gram model. There is no reasons to restrict the growing model to some n-gram length, so we have varied the coefficient of model description length in order to get models of various sizes.

We see from Fig. 7.1 that the growing model is superior to the others. It has the advantage that instead of short n-grams that do not weight much in entropy, it can include common long n-grams in the model. The smallest

Table 7.2: Comparison of the evaluation results of the incrementally built history cluster model with baseline N-gram model. Training of the cluster model was started with a unigram model, then bigrams from the training data were added, and last trigrams. Adding of the trigrams doubled the model size and decreased entropy more than 5%. In the baseline N-gram model, the corresponding decrease was 11.5%, but the number of parameters (n-grams) almost multiplied by ten.

n	N-gram model		Clustered model			
	n-grams	Entropy	Histories	Clusters	Parameters	Entropy
1	10 528	7.96	0	0	10 528	7.96
2	551 415	5.62	9 010	543	325 257	6.02
3	4 264 183	4.97	17 980	824	745 868	5.71

growing model (excluding unigram model) was 4-gram, and the largest 6-gram. As the allowed size increases, the baseline model gets closer, and finally there would be a point where the models converge (unlimited n for baseline model and zero model cost for growing model).

The clustered model makes steeper drops than the baseline model, but does not get as low entropies. The behavior of the clustered model for longer n-grams is unclear, as we could not yet calculate longer than a 3-gram model due to the slowness of the algorithm.

7.5 Discussion

The incrementally built cluster model had a very restricted size, but the cross-entropy stayed above the baseline model. One clear explanation for the higher entropy is the worse smoothing method. The relative drop of in entropy when adding 3-grams to the model was 10% for the baseline model and 5% for the clustered model. The different is not so large when compared to how much more the model size increased (670% for the baseline and 129% for the clustered model).

So it seems that to get good prediction results for the n-gram history cluster model, applying the idea behind the Kneser-Ney smoothing would be strongly recommended. We would also need a more efficient way of constructing this kind of models, since our incremental algorithm started to be too slow even for 4-grams.

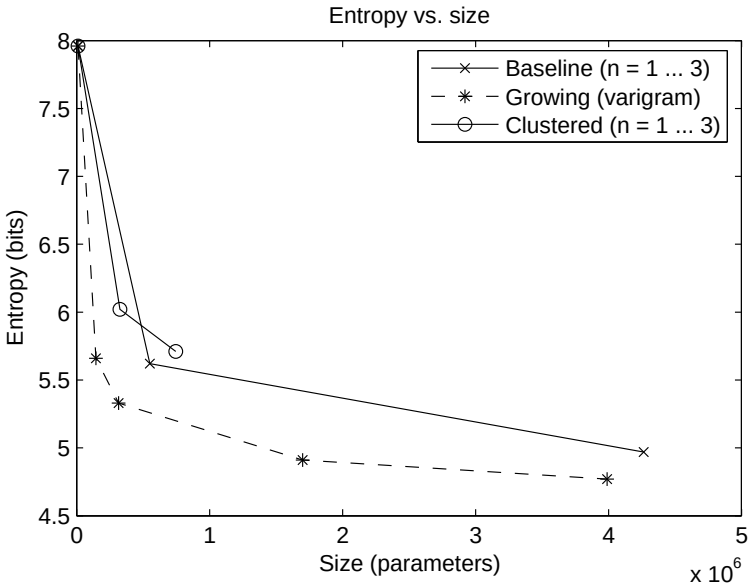

Figure 7.1: Cross-entropies against model sizes. Measurement points of the growing model correspond to different parameter values. For the baseline N-gram model and the clustered model they are maximum n-gram lengths.

The main constraint on the efficiency of the search algorithm is that each history that might be added to the model, must be compared to all existing clusters in order to find the one that has the nearest emission distribution. For every distribution one must go through every morph that it can emit. Thus the vocabulary size of the model affects the efficiency substantially. Using even less than ten thousand morph types might speed up the search, although that means that more history types will be found. Another possibility could be to apply some dimension reduction method (e.g. PCA, ICA) to the distributions, in order to quicken the calculation of distances.

There is one more issue that requires some consideration. In all tested models of clustered histories, the number of clusters at the end of the algorithm was just between one hundred and one thousand. That is a surprisingly small number, even if the n-gram length was not very high. The reason is found in the prior probability of the model.

The prior of prediction distributions of the clusters was such that every possible combination has the same prior probability (Eq. 7.12). It may seem to be as objective a prior as possible. However, it actually favors large clusters. Using Stirling's approximation $n! \approx (n/e)^n \sqrt{2\pi n}$, we can calculate the

derivative of the logarithm of the number of combinations with respect to the size of the cluster:

$$\begin{aligned}
\frac{\partial}{\partial f} L(\text{distrib.}) &= \frac{\partial}{\partial f} \log \binom{f+M-1}{M-1} \\
&= \frac{\partial}{\partial f} \log \frac{(f+M-1)!}{(M-1)!\, f!} \\
&\approx \frac{\partial}{\partial f} \log \frac{((f+M-1)/e)^{f+M-1}\sqrt{2\pi(f+M-1)}}{((M-1)/e)^{M-1}\sqrt{2\pi(M-1)}(f/e)^f\sqrt{2\pi f}} \\
&= \frac{\partial}{\partial f} \log \frac{(f+M-1)^{f+M-1}\sqrt{f+M-1}}{(M-1)^{M-1}\sqrt{M-1} f^f \sqrt{2\pi}\sqrt{f}} \\
&= \Big(1+\log(f+M-1)\Big) + \frac{1}{2}\Big(\frac{1}{f+M-1}\Big) - \Big(1+\log f\Big) \\
&\quad -\frac{1}{2}\Big(\frac{1}{f}\Big) \\
&= \log(f+M-1) - \log(f) - \frac{1}{2}\big(\frac{1}{f} + \frac{1}{f+M-1}\big) \qquad (7.21)
\end{aligned}$$

Now we see that the change in code length reaches asymptotically zero as f increases, and thus the larger the cluster already is, the smaller the increase in code length (i.e. decrease in model probability) will be. In consequence, if the nearest cluster is large enough, the new history is very likely to be added in it. The cost of adding a new cluster is much higher, and those new histories that do have a large cluster near are more likely to be skipped. Large clusters grow larger, and new clusters are created rarely.

Thus it seems that we would need a cleverer prior probability or coding scheme for the emission distributions. If we think how we would actually save the emission frequencies in a computer, a natural scheme would be to first write the number of morphs that have a frequency above zero, and then list those together with their frequencies. Due to the sparseness of the data, most frequencies *should* be zero. An example prior that takes that into account is to first give the number of non-zero frequencies according to some distribution, then select a random permutation of that many morphs, and last give equal probability to all combinations of frequencies, as before, but only for the selected subset of morphs.

Chapter 8

Conclusions and discussion

In this thesis, various ways of modeling short span dependencies of a natural language were studied. All of the language models were based on statistical segmentation of words to morphs, and applied the n-gram assumption, i.e. that the next morph depends only on the $n-1$ previous morphs.

In addition, all the methods, even if introduced from different points of view, applied *clustering* to the problem in order to fight the curse of dimensionality. A general equation for all of the models is

$$P(m_i \,|\, m_{i-n+1}^{i-1}) = \sum_{k \in G(m_{i-n+1}^{i-1})} P(m_i \,|\, c_k) P(c_k \,|\, m_{i-n+1}^{i-1}), \tag{8.1}$$

where m_{i-n+1}^{i-1} is the morph history, m_i predicted morph, $c_k \, (k = 1, \ldots, C)$ is a set of clusters, and G a relation that maps a history to a subset of the clusters.

Clustering leads to a more compact model. In addition to the benefit of saving space and memory, one can hope that the decrease in the number of parameters helps avoiding overlearning. On the other hand, a clustered model is usually less accurate than a conventional N-gram model, that has very reliable estimates for frequent events. The rarest events are very hard both with or without the clustering: If the data is insufficient for a reliable direct estimate, it is probably insufficient also for determining the cluster. So, it is the middle cases that we hope to be able to improve.

In the class-based N-gram model presented in Chapter 5, it was assumed that each morph belongs to one cluster, and that $G(m_{i-n+1}^{i-1})$ equals $G(c_{i-n+1}^{i-1})$ and $P(c_k \,|\, m_{i-n+1}^{i-1})$ equals $P(c_k \,|\, c_{i-n+1}^{i-1})$, i.e. that the selected cluster depends only on the clusters of the morphs in the history. We sought the clustering by

finding latent features for the morphs using a method based on Independent Component Analysis. When examining the cross-entropy versus size of the models, we noticed that the class-based models were somewhat smaller but clearly worse than standard N-gram models. When the length of the latent features was increased, and longer context information was added to the input matrix of ICA, the models got very close to the baseline models, as most morphs were left in their own clusters.

In Chapter 6, our model was based on the Self-Organizing Map. We projected the morphs to latent features obtained by the ICA method. SOM was used to cluster morph histories into map units, and a prediction distribution from map units to morphs was estimated using a couple of ways. In this approach, $G(m_{i-n+1}^{i-1})$ equals $\mathrm{BMUs}(m_{i-n+1}^{i-1}, K)$, where K is the number of best-matching map units to select for a given history, and c_k:s are the map units. This is neither soft nor hard clustering, but something in between: Each history belongs to exactly K clusters, where K is larger than one but much smaller than the total number of clusters. SOM-based models did not reach low cross-entropies, but the main problem might not be in the model itself, but in that the concatenation of the latent features of the morphs does not lead to history features that could be efficiently utilized by the model. We might want to find latent features directly for the morph histories instead of morphs, or cluster n-grams based on just the feature vector of the last morph.

Our last model proposed clustering of morph histories based on their prediction distributions. We utilized hard clustering, in which case G is a many-to-one function and sum of Equation 8.1 reduces to a single product. MAP estimation of the number of the clusters for a 4-gram model suggested that less than a thousand clusters could be enough, instead of the hundreds of thousands of observed histories. A different set of priors for the model might however make a less extravagant result. The full potential of the model of clustered histories is unclear until we have a faster training algorithm and utilize a more trustworthy smoothing technique. It seems to be worth studying more: The proposed kind of clustering is intuitive, reduces the number of parameters clearly, works with an already compact set of model units such as statistical morphs, and maximum a posteriori estimation assures that the frequent events do not lose too much accuracy.

8.1 How to beat N-gram models?

None of the models presented in this work could outperform standard N-gram models. Those that do, usually do it by using information on the global or long-span properties of the text (e.g. LSA models such as [2]), developing models that are more compact but still based on maximum likelihood modeling of n-grams (e.g. the growing model in [61]), or otherwise are computationally much more heavy (e.g. neural network model in [4]).

If we think of language models based strictly on the n-gram assumption (with moderately small n), can we do better than N-gram models? One answer is given by Brill et al. [6], who studied whether *humans* could improve results given by an N-gram model in speech recognition by post-processing. In the experiment, human subjects where given a list of 10 sentences, that were the most probable according to the recognizer, to choose from or edit if needed. On average, human sophistication improved the results clearly enough, even if not relatively very much. They also studied what kind of information was used by the subjects, and concluded that there were many linguistic proficiencies that appear to be solvable also without (interactive) human aid.

Also Rosenfeld [58] emphasizes the importance of injecting human knowledge of language into the models. Furthermore, he proposes two ways of doing this. In interactive modeling, human knowledge and decision making would improve data-driven optimization, and vice versa. Regarding more direct ways for inserting human sophistication, he proposes that linguistic theories should be preferably encoded into the process as Bayesian priors. This way the human knowledge, often overstated, would be applied mostly to phenomena for which there is not sufficient amount of data.

Instead of inserting human made linguistic theories in to the language models, a more aspiring goal would be to find methods for discovering statistical equivalents of such theories from the data in an unsupervised manner. Whether this is possible, and to what extent, is an issue that is under debate in the field of linguistics [40]. The research direction that speaks for an unsupervised learning approach is *cognitive linguistics*, where one major hypothesis is that the knowledge of language emerges from language use [18]. The Word ICA method [27, 28, 29] discussed and utilized in Chapter 4 clearly supports this kind of approach. However, Honkela et al. [29] suggest also that for a more realistic language learning simulation it would be necessary to include other kinds of context, such as visual perceptions, actions and activities associated with linguistic expressions, along with the text.

8.2 Future work

In the discussions of the previous chapters, we have proposed a number of new experiments. Clusters derived from ICA features might work better in class-based N-gram models if a model utilizing asymmetric clustering was used. For a SOM-based model, one might need some SOM variant developed for multivariate time series prediction instead of the basic SOM algorithm, or the features of the histories should be constructed otherwise. For the model for clustered histories, one might need a more efficient search algorithm, more sophisticated priors, and probably utilization of Kneser-Ney smoothing.

In addition, there is one more interesting direction where Equation 8.1 leads us. Clustering histories by successor distributions, as done in Chapter 7, is intuitive, but hard clusters are often too restrictive and do not generalize much. It would be better to leave the last step in Equation 7.7 out (i.e. apply Eq. 8.1 so that G always returns every cluster). This way the prediction distribution of a history h would be a mixture of some latent distributions $P(m \mid c_k)$:

$$P(m_i \mid h_j) = \sum_{k=1}^{C} P(c_k \mid h_j) P(m_i \mid c_k) \tag{8.2}$$

How is it possible to find this kind of latent distributions? By some consideration we can see that this is a blind source separation (BSS) problem.[1] However, now the assumptions of the unknown parameters are such that Independent Component Analysis does not give a desired answer: We want strictly non-negative values, and and columns of source matrix and rows of mixing matrix that sum up to one. Instead we could use a variant of *Non-Negative Matrix Factorization* (NMF) [46], an algorithm that tries to find non-negative solutions for BSS problems. NMF has previously been used e.g. in language model adaptation [53] and document clustering [67].

Sparseness of the signals has been shown to be an important part of processing information in the visual cortex of brains [21, 34]. Sparseness of the signals was also essential in the Word ICA method, resulting in latent features that were more practical than the non-sparse features calculated using only SVD [65]. So it seems promising to apply the endeavor for sparse coding also to the problem of language modeling. Thus a suitable variant of NMF could be something resembling the algorithm derived by Hoyer [30, 31].

[1] Assume that we have estimates for n-gram probabilities collected to a matrix $\boldsymbol{V}$ so that the element v_{ij} is the probability of the morph i given the history j. Denoting $w_{ik} = P(m_i \mid c_k)$ and $h_{kj} = P(c_k \mid h_j)$ we get $\boldsymbol{V} = \sum_k \boldsymbol{w}_{\cdot k} h_{kj} = \boldsymbol{W}\boldsymbol{H}$. (Cf. to Eq. 4.1 and Fig. 4.1.)

Finally, we want to emphasize that most of the experiments of this thesis were preliminary work. The conventional N-gram models and their class-based variants have years of research in their background. Neural networks and data-driven methods such as ICA may have a lot to give to the field of statistical language modeling, but the models that would outperform the N-gram models are also likely to need a lot of serious work. This thesis has hopefully given some insight on where to start.

Bibliography

[1] D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *ACM Computing Surveys*, 15(3):237–269, 1983.

[2] J. R. Bellegarda. Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, 88(8), 2000.

[3] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[5] E. Bingham, J. Kuusisto, and K. Lagus. ICA and SOM in text document analysis. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 361–362, 2002.

[6] E. Brill, R. Florian, J. C. Henderson, and L. Mangu. Beyond n-grams: Can linguistic sophistication improve language modeling? In *Proceedings of COLING/ACL 1998 Conference*, volume I, pages 186–190, Montreal, Canada, 1998.

[7] P. F. Brown, V. J. DellaPietra, P. V. deSouza, J. C. Lai, and R. L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[8] S. F. Chen. *Building Probabilistic Models for Natural Language*. PhD thesis, Harvard University, 1996.

[9] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–393, 1999.

[10] N. Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

[11] R. A. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, and V. Zue. *Survey of the State of the Art in Human Language Technology*. Cambridge University Press, 1996.

[12] M. Creutz and K. Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing*. Accepted for publication.

[13] M. Creutz and K. Lagus. Unsupervised discovery of morphemes. In *Proc. Workshop on Morphological and Phonological Learning of ACL'02*, pages 21–30, Philadelphia, Pennsylvania, USA, 2002.

[14] M. Creutz and K. Lagus. Induction of a simple morphology for highly-inflecting languages. In *Proc. 7th Meeting of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, pages 43–51, Barcelona, July 2004.

[15] M. Creutz and K. Lagus. Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor. Technical Report A81, Publications in Computer and Information Science, Helsinki University of Technology, 2005. `http://www.cis.hut.fi/projects/morpho/`.

[16] M. Creutz, K. Lagus, K. Lindén, and S. Virpioja. Morfessor and Hutmegs: Unsupervised morpheme segmentation for highly-inflecting and compounding languages. In *Proceedings of the Second Baltic Conference on Human Language Technologies*, pages 107–112, 2005.

[17] M. Creutz and K. Lindén. Morpheme segmentation gold standards for Finnish and English. Technical Report A77, Publications in Computer and Information Science, Helsinki University of Technology, 2004.

[18] W. Croft and D. A. Cruse. *Cognitive Linguistics*. Cambridge Textbooks in Linguistics, 2004.

[19] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Hashman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6), 1990.

[20] G. D. Forney. The Viterbi algorithm. In *Proceedings of the IEEE*, volume 61, pages 268–278, 1973.

[21] P. Földiák and M. P. Young. Sparse coding in the primary cortex. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 895–898, Cambridge, Massachusetts, 1995. The MIT Press.

[22] J. Gao, J. T. Goodman, G. Cao, and H. Li. Exploring asymmetric clustering for statistical language modeling. In *Proc. 40th Annual Meeting of the ACL*, pages 183–190, Philadelphia, Pennsylvania, USA, 2002.

[23] J.-L. Gauvain and C.-H. Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE Transactions on Speech and Audio Processing*, 2(2):291–298, 1994.

[24] J. T. Goodman. A bit of progress in language modeling — extended version. Technical Report MSR-TR-2001-72, Microsoft Research, 2001.

[25] K. Hacioglu, B. Pellom, T. Ciloglu, O. Ozturk, M. Kurimo, and M. Creutz. On lexicon creation for Turkish LVCSR. In *Proc. Eurospeech'03*, pages 1165–1168, Geneva, Switzerland, 2003.

[26] T. Hirsimäki, M. Creutz, V. Siivola, M. Kurimo, S. Virpioja, and J. Pylkkönen. Unlimited vocabulary speech recognition with morph language models applied to Finnish. *Computer Speech and Language*. Accepted for publication.

[27] T. Honkela and A. Hyvärinen. Linguistic feature extraction using independent component analysis. In *Proceedings of IJCNN 2004, International Joint Conference on Neural Networks*, pages 279–284, 2004.

[28] T. Honkela, A. Hyvärinen, and J. Väyrynen. Emergence of linguistic representations by independent component analysis. Technical Report Publications in Computer and Information Science, Report A72, Helsinki University of Technology, 2003.

[29] T. Honkela, A. Hyvärinen, and J. Väyrynen. Emergence of linguistic features: Independent component analysis of contexts. In *Ninth Neural Computation and Psychology Workshop: Modeling Language, Cognition and Action*, Plymouth, England, Sep. 8-10 2004.

[30] P. O. Hoyer. Non-negative sparse coding. In *Neural Networks for Signal Processing XII (Proc. IEEE Workshop on Neural Networks for Signal Processing)*, pages 557–565, Martigny, Switzerland, 2002.

[31] P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

[32] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999. `http://www.cis.hut.fi/projects/ica/fastica/`.

[33] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. John Wiley & Sons, 2001.

[34] A. Hyvärinen, P. O. Hoyer, J. Hurri, and M. Gutmann. Statistical models of images and early vision. *Proceedings of the Int. Symposium on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, 2005.

[35] R. Kneser and H. Kney. Improved backing-off for m-gram language modeling. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1:181–184, 1987.

[36] T. Kohonen. *Self-Organizing Maps*. Springer, 1995.

[37] T. Kohonen, S. Kaski, K. Lagus, J. Salojärvi, J. Honkela, V. Paatero, and A. Saarela. Self organization of a massive document collection. *IEEE Transactions on Neural Networks*, 2000.

[38] T. Kolenda, L. K. Hansen, and J. Larsen. Signal detection using ICA: application to chat room topic spotting. In *Proceedings of ICA2001, the Third International Conference on Independent Component Analysis and Signal Separation*, pages 540–545, 2001.

[39] M. Kurimo and K. Lagus. An efficiently focusing large vocabulary language model. In *International Conference on Artificial Neural Networks (ICANN'02)*, pages 1068–1073, Madrid, Spain, 2002.

[40] K. Lagus. Miten hermoverkkomallit selittävät kielen oppimista? In A. M. Korpijaakko-Huuhka, S. Pekkala, and H. Heimo, editors, *Kielen ja kognition suhde. Puheen ja kielen tutkimuksen yhdistyksen julkaisuja 37*, Helsinki, 2005.

[41] K. Lagus, M. Creutz, and S. Virpioja. Latent linguistic codes for morphemes using independent component analysis. In *Ninth Neural Computation and Psychology Workshop: Modeling Language, Cognition and Action*, Plymouth, England, Sep. 8-10 2004.

[42] K. Lagus, S. Kaski, and T. Kohonen. Mining massive document collections by the WEBSOM method. *Information Sciences*, 163(1–3):135–156, 2004.

[43] K. Lagus and M. Kurimo. Language model adaptation in speech recognition using document maps. In *IEEE Workshop on Neural Networks for Signal Processing (NNSP'02)*, pages 627–636, Martigny, Switzerland, 2002.

[44] T. K. Landauer, P. W. Foltz, and D. Laham. Introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

[45] T. K. Landauer, D. Laham, and P. W. Foltz. Learning human-like knowledge by singular value decomposition: A progress report. *In M. I. Jordan, M. J. Kearns and S. A. Solla (Eds.), Advances in Neural Information Processing Systems*, 10:45–51, 1998.

[46] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[47] E. E. Loos, S. Anderson, D. H. Day, P. C. Jordan, and J. D. Wingate, editors. *Glossary of linguistic terms*. SIL International, 2004. `http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/`.

[48] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.

[49] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, 1990.

[50] H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependencies in stochastic language modeling. *Computer Speech and Language*, 8:1–28, 1994.

[51] T. R. Niesler, E. W. D. Whittaker, and P. C. Woodland. Comparison of part-of-speech and automatically derived category-based language models for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 177–180, 1998.

[52] T. R. Niesler and P. C. Woodland. Combination of word-based and category-based language models. In *Proceedings of ICSLP-96*, pages 220–223, 1996.

[53] M. Novak and R. Mammone. Use of non-negative matrix factorization for language model adaptation in a lecture transcription task. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 541–544, Salt Lake City, UT, USA, 2001.

[54] K. Petersen, L. Hansen, T. Kolenda, E. Rostrup, and S. Strother. The independent components in functional neuroimages. In *Proc. Int. Workshop on Independent Component Analysis and Blind Source Separation (ICA2000)*, pages 251–256, 2000.

[55] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*, volume 15. World Scientific Series in Computer Science, Singapore, 1989.

[56] E. S. Ristad and R. G. Thomas. New techniques for context modeling. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 220–227, Cambridge, Massachusetts, 1995.

[57] R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10:187–228, 1996. Longer version: Carnegie Mellon Tech. Rep. CMU-CS-94-138.

[58] R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, 88(8), 2000.

[59] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[60] V. Siivola. Building compact language models incrementally. In *Proceedings of the Second Baltic Conference on Human Language Technologies*, Tallinn, Estonia, 2005.

[61] V. Siivola and B. L. Pellom. Growing an n-gram language model. In *Proceedings of Interspeech*, Lisbon, Portugal, 2005.

[62] A. Stolcke. SRILM – an extensible language modeling toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP)*, pages 901–904, 2002. `http://www.speech.sri.com/projects/srilm/`.

[63] P. D. Turney. Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. *Lecture Notes in Computer Science*, 2167:491–502, 2001.

[64] P. M. B. Vitanyi and M. Li. Minimum description length induction, Bayesianism, and Kolmogorov complexity. *IEEE Transactions on Information Theory*, 46(2):446–464, 2000.

[65] J. Väyrynen and T. Honkela. Comparison of independent component analysis and singular value decomposition in word context analysis. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, 2005.

[66] P. Xu and F. Jelinek. Random forests in language modeling. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[67] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 267–273. ACM Press, 2003.

[68] W. Xu and A. Rudnicky. Can artifical neural networks learn language models? In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, 2000.