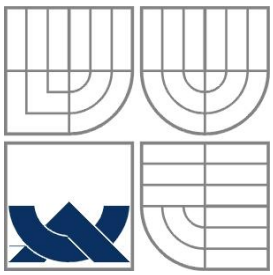
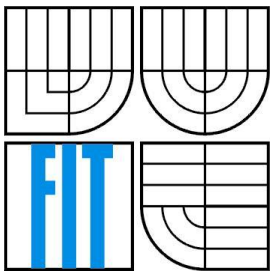


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODELOVÁNÍ JAZYKA V ROZPOZNÁVÁNÍ ČEŠTINY

LANGUAGE MODELING FOR SPEECH RECOGNITION IN CZECH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ MIKOLOV

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2007

Zadání diplomové práce

Řešitel: **Tomáš Mikolov, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Modelování jazyka v rozpoznávání češtiny**
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte literaturu věnovanou jazykovému modelování s důrazem na jazyky s bohatou morfologií

2. Navrhněte a implementujte systém pro jazykové modelování češtiny, který bude možné použít pro automatické rozpoznávání přednášek

3. Porovnejte výsledky realizovaného systému s dostupným řešením na dodaných testovacích datech.

Literatura:

1. Joshua Goodman. 2000. The State of the Art in Language Modeling. A tutorial presented at North American ACL, 2000, Seattle.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.,** UPGM FIT VUT
Datum zadání: 28. 2. 2007
Datum odevzdání: 22. 5. 2007

Abstrakt

Tato práce se zabývá problematikou jazykových modelů v oblasti automatického přepisu mluvené řeči. V teoretické části jsou rozebrány současně používané metody pro pokročilé jazykové modelování založené na statistickém přístupu - modely založené na třídách, na faktorech a na neuronových sítích. Následně je popsána implementace jazykového modelu založeného na dvou neuronových sítích. V závěru práce jsou uvedeny výsledky dosažené na Pražském a Brněnském mluveném korpusu (cca 1 170 000 slov) - redukce perplexity o zhruba 20%. Výsledky dosažené při reskórování N-best listů ukazují zlepšení při rozpoznávání spontánní řeči o více než 1%. V závěru práce jsou uvedeny možnosti využití práce, její možná rozšíření a také jsou uvedeny hlavní nevýhody současně používaných přístupů pro statistické jazykové modelování.

Klíčová slova

jazykový model, čeština, n-gramové statistiky, neuronové sítě, rozpoznávání řeči, umělá inteligence

Abstract

This work concerns the problematic of language modeling in automatic speech recognition. Currently widely used techniques for advanced language modeling based on statistical approach are described in the first part of work - class based language models, factored language models and neural network based language models. In the next section, implementation of neural network based language model is described. Results obtained on "Pražský mluvený korpus" and "Brněnský mluvený korpus" corpora (1 170 000 words) are reported, with perplexity reduction around 20%. Also, results obtained after rescoring N-best lists with spontaneous speech are reported, with absolute improvement in accuracy by more than 1%. In the conclusion, possible uses of the work are mentioned, along with possible extensions in the future. Finally, main weaknesses of current statistical language modeling techniques are described.

Keywords

language modeling, Czech language, n-gram statistics, neural networks, speech recognition, artificial intelligence

Citation

Mikolov Tomáš: Language Modeling for Speech Recognition in Czech, Brno, 2007, Master's thesis, Brno University of Technology.

Language Modeling for Speech Recognition in Czech

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. RNDr. Pavla Smrže, Ph.D.
Další informace mi poskytli Doc. Dr. Ing. Jan Černocký, Ing. Ondřej Glembek, Ing. František Grézl a Ing. Lukáš Burget, Ph.D.

.....
Tomáš Mikolov
20. května 2007

Poděkování

Děkuji všem členům Speech@FIT, kteří mi poskytli mnoho důležitých informací, bez nichž by vypracování této práce nebylo možné.

Contents

1 Introduction

1.1 Motivation

The goal of this work is to improve current language models of Czech used at Brno University of Technology. In this chapter, I will clarify my motivation in this area.

It is well known that computers were built to obtain artificial intelligence. First, it was supposed that most human work can be done by intelligent machines. In 1950s it was predicted that in fifty years, computers will become more intelligent than humans. Many scientists claimed that they can do things like language translation or machine vision in a few months. None of them has succeeded.

After this decade of optimism, many others tried to prove that intelligent machines can't ever be built. For example, Searle with his well known Chinese room experiment tries to prove that a computer that passes Turing test needs not to understand natural language. There is also a plenty of formal proofs that rely on words like consciousness, mind or understanding, "proving" that artificial intelligence can't be built using Turing machines. There were also even more amusing arguments - for example, that computer can't be surprised, frightened or fall in love. Much more about history of AI can be found in [8].

From my point of view, there is no magic about intelligence. It can be shown that if one believes in laws of physics, then as the whole brain can be simulated using a computer, there is no reason why computers can't be intelligent. But effectivity of such solution is another question.

Since I was always interested in artificial intelligence, it was natural that I was wondering what the intelligence really is. To my disappointment, many scientists in this field have no need to define what intelligence is. Answers like "a thing can be called intelligent if it acts somewhat reasonable" or "we all feel what intelligence is, but no one can say it" are maybe funny, but during my studies, there was nothing more.

After some thinking, I decided that intelligence is based on the ability to make predictions – it is easily understood when thinking about behavior. We buy food because we know that it will be useful

when we'll be hungry. Most of our behavior is not useful in the time of execution, but will prove to be useful in the future. The key part of all this are the predictions – we don't know what the future shall be, but we can predict it using our knowledge. Similar conclusions, and much more, can be found in [8].

It can be shown that intelligence = ability to make predictions = compression [9]. For example, we have the following sentence:
THE ROSES ARE *
Our goal is to determine the last word of this sentence. It can be expected that this word could be RED or NICE or FLOWERS. It is not probable that the last word would be BLACK or ISLAND.
It can be said that we are able to predict the last word using our knowledge. This ability is in fact intelligence. There is no known algorithm to make computers as good in making predictions as humans are (well known Shannon experiments with entropy of English text), but to some degree, they are successful.

Simple prediction example:
THE ROSES ARE RET
THE ROSES ARE RED
Assuming that the computer has a lot of unambiguous data, it is easy to assign a higher probability of being said/written to the second sentence. We can simply build a vocabulary from the training data and for each sentence containing unknown word assign a low probability.

More complicated example:
THE ROSES ARE MOON
THE ROSES ARE RED
Since all words are known, another technique must be used. For example, we may count how many times each sentence in the training data occurred. It can be expected that the second one occurred more often and thus it should have higher probability.

Even more complicated example:
FIFTY SEVEN AND THIRTY ONE IS EIGHTY EIGHT
FIFTY SEVEN AND THIRTY ONE IS TWENTY TWO
What now? Of course, one may think that by using the rule from the second example, the problem should be solved. But what if none of these sentences appeared in our training data? This is called sparse data problem and it is considered to be a big problem. There exist some solutions to overcome it, but their effectivity is questionable. It can be shown that humans understand novel situations and are able to make correct predictions about future, while computers with classic algorithms are unable

to do this. Some believe the problem is that the amount of training data for computers is too small. But from the previous example, it is clear that the problem is not data sparsity. The problem is that the computer does not understand the data, while humans do.

Speaking about understanding, it is not easy to define it. It is often hard even to tell if a human understands something, or only pretends to. But if we use again the probability – we can compare two systems, and the one that gives better probability on testing data should be the one that has more 'understanding'. But even this approach has some drawbacks – calculating exact probabilities is not easy (since all probabilities must sum to one, it is needed to evaluate all possibilities) and the system giving worse prediction estimates may be the one with more understanding, simply because it does not make a good smoothing (more of this in chapter 2). Probably the most reliable way to determine quality of a language model is to use it in a speech recognizer (or similar system like OCR) and determine WER (word error rate).

The most successful algorithms for language modeling and data compression are based on so called n-grams (the exact definition will be in chapter 2). The idea behind n-grams is simple – to estimate a probability of next symbol in a certain context, we can use only the most recent history. The longer the history is, the better probability estimates we should get, but since the amount of training data is never infinite, the estimates for longer contexts are less reliable. To solve this, many smoothing algorithms were designed.

N-gram statistics itself is very simple, yet it is still used by many as the baseline for comparison. Due to its simplicity, large amounts of data can be processed very quickly and that is probably the reason why this approach is very hard to beat using more precise algorithms. On the other hand, it is clear that N-gram approach itself doesn't lead to AI. It is unable to make good predictions in novel situations. However, n-grams may be closer to real brains then it seems, and this can be the true reason why they are so successful. In the real brains, however, predictions are made at multiple levels using a sophisticated hierarchical neural network (much more about this in [8]).

Another interesting point of view while thinking about intelligence is by using information theory [11]. If we consider a transmitter that sends messages to a receiver, we can intuitively suppose that information that is expected by the receiver has less information content than a surprising one. For example, by sending a message "THE CAPITAL OF FRANCE IS PARIS" to an average European, we do not transmit much information, since the receiver already knows this. On the other hand, speaking to this person about African cities would have more information content. There is a direct connection between information theory and AI: intelligent machine should be able to learn efficiently regularities in the incoming signals, so it should be able to minimize received information content.

This means that intelligent machine should maximize its prediction capability by using the simplest possible solutions. This is quite the same conclusion as the ideas behind Minimum description length (MDL) and Kolmogorov complexity [12].

1.2 Czech language

This work primarily concentrates on creating language model that will be useful for speech recognition in Czech. There are several differences between Czech and English that make classic N-gram approach impractical.

Czech language is inflective, so every word may occur in many forms. This increases the size of vocabulary about three times [3], in comparison to an English vocabulary computed on the same amount of text. This is quite a huge problem, since it increases the data sparsity problem. Another problem with vocabulary size is that commonly used tools for ASR (automatic speech recognition) like HTK are unable to handle more than 65 000 words in vocabulary.

A solution here may be word division into smaller morphological parts - prefix, stem and suffix. Factored language models and neural network based language models are able to handle this type of information.

Important difference between English and Czech is a particularly free word order in Czech sentences. This results again in increased data sparsity. A solution here may be neural network based language models, since they do not use explicit temporal back off.

2 Statistical language modeling

2.1 N-gram statistics approach

The aim of statistical language modeling is to compute probability of some utterance. Language model is built using training data, parameters are tuned using heldout data and the model performance is evaluated on testing data. The division ratio between training, heldout and testing data is usually something like 90: 9 : 1, but according to [2] it is only important to have 100 – 1000 words per parameter for heldout data and for testing data probably a few thousand of words.
Language model is viewed as a probability distribution $P(w_i \mid w_1, w_2, \dots, w_{i-1})$, where $w_1, \dots, w_{i-1}$ is history.

$$P(w_i \mid w_1, w_2, \dots, w_{i-1}) = C(w_1, w_2, \dots, w_i) / C(w_1, w_2, \dots, w_{i-1}) \tag{2.1}$$

where $C(w_1, w_2, \dots, w_i)$ denotes the number of occurrences of $w_1, w_2, \dots, w_i$.

Such probability estimation can't be computed directly, since with the increasing length of context we get less reliable estimations (as described in chapter 1). This problem is typically solved by using only a shorter context, with length of N-1. Such an approach is called N-gram language model (or, in data compression, PPM – prediction by partial match).

Probability of word w_i using N-gram LM is computed as

$$P(w_i \mid w_1, w_2, \dots, w_{i-1}) = C(w_{i-n+1}, \dots, w_i) / C(w_{i-n+1}, \dots, w_{i-1}) \tag{2.2}$$

For N=1 the LM is called unigram, for N=2 bigram and for N=3 trigram. Usually, N is between 1 and 4.

As it can be seen, the above formula shall assign zero probability to n-grams that have not occurred in the training data. This is quite a problem, since the overall probability of an utterance is computed as

$$P(W) = \prod_{i=1}^{K} P(w_i \mid w_1, w_2, ..., w_{i-1})$$

(2.3)

so if some probability estimate is zero, then the overall probability is zero too. To prevent this, many smoothing techniques have been developed.

2.2 Smoothing

2.2.1 Add one smoothing

This technique, also called Laplace smoothing, is probably the easiest one. To avoid zero counts, we simply add one to every count. The probability is then computed as

$$\mathrm{P}(\mathrm{w_i} \mid \mathrm{w_1}, \mathrm{w_2}, \ldots, \mathrm{w_{i\text{-}1}}) = \frac{\mathrm{C}(\mathrm{w_{i\text{-}n+1}}, \ldots, \mathrm{w_i}) + 1}{\mathrm{C}(\mathrm{w_{i\text{-}n+1}}, \ldots, \mathrm{w_{i\text{-}1}}) + \mathrm{V}}$$

(2.4)

where V denotes vocabulary size.

This solves the zero probability problem, but usually too much probability mass is redistributed among unknown n-grams, from which many are highly unlikely.

2.2.2 Add lambda smoothing

Instead of adding a constant, we can assume that adding a certain value λ that will be determined on heldout data should be more useful:

$$\mathrm{P}(\mathrm{w_i} \mid \mathrm{w_1}, \mathrm{w_2}, \ldots, \mathrm{w_{i\text{-}1}}) = \frac{\mathrm{C}(\mathrm{w_{i\text{-}n+1}}, \ldots, \mathrm{w_i}) + \lambda}{\mathrm{C}(\mathrm{w_{i\text{-}n+1}}, \ldots, \mathrm{w_{i\text{-}1}}) + \lambda \mathrm{V}}$$

(2.5)

In this case, the probability mass reserved for unknown n-grams is much more optimal than in the previous case. However, this technique doesn't take into account the fact that unknown n-grams have different probabilities, which can be already estimated using training data.

2.2.3 Simple interpolation

Uses different n-gram statistics to obtain better estimates. For example, unigram statistics are reliable but context independent, while trigram is very noisy. So the mixture of both, and even bigram, should result in better estimates than by using only one of these alone.

$$P(z \mid xy) = \lambda \frac{C(xyz)}{C(xy)} + \mu \frac{C(yz)}{C(y)} + (1 - \lambda - \mu) \frac{C(z)}{C(\bullet)} \tag{2.6}$$

λ and μ parameters can be estimated on heldout data, $C(\bullet)$ means vocabulary size.

Of course, the most reliable estimates should have high interpolation parameter values – and their value does not depend only on order of n-gram, but even on the reliability of estimate given by the number of occurrences. So the idea of interpolation of multiple n-gram statistics is extended in Jelinek-Mercer deleted interpolation using buckets (more about this in [2]).

2.2.4 Good-Turing discounting

This technique tries to estimate the probability mass that should be assigned to unknown n-grams using known frequency of n-grams that occurred only once. The saved probability mass is uniformly redistributed among unknown n-grams. These ideas are extended in Katz smoothing.

2.2.5 Kneser-Ney discounting

Kneser-Ney smoothing exploits the fact that although some unigrams are frequent, they appear almost always after some symbol - for example, "Francisco" appears usually after "San". So if the language model is backing off to unigram, "Francisco" should not have such a big unigram probability.
Modified Kneser-Ney smoothing is considered to be the state of the art in current smoothing techniques [2] and should be used as a baseline for experiments, since it is implemented in SRI LM toolkit [13] and thus can be easily computed.

2.2.6 Comparison of different smoothing methods

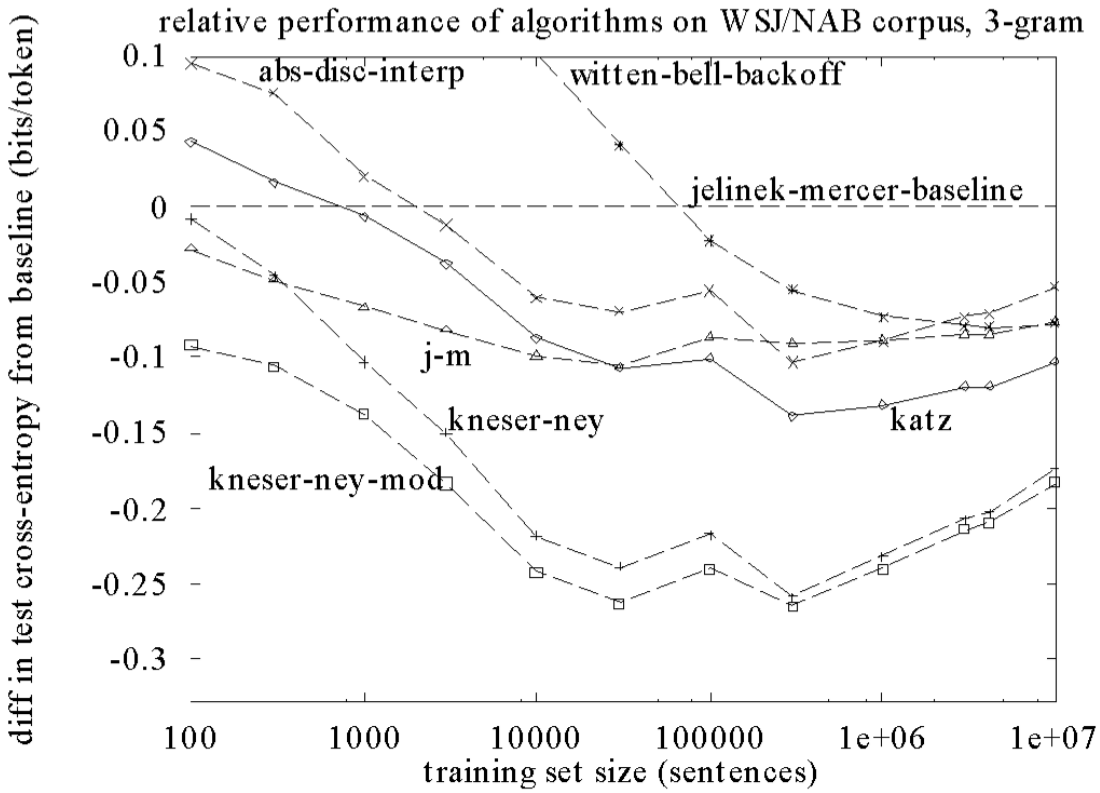


Figure 1: Comparison of different smoothing methods from [2] shows the best performance of Modified Kneser-Ney smoothing

As it can be seen, many different techniques were developed to solve the zero probability assignment problem. More about them and a good comparison can be found in [2].

When comparing different smoothing techniques, or different language models, it is useful to do so with another measure than plain probability. Since the probability value itself is usually very small, logarithmic probability is used instead. But the comparison is difficult even with log probability, since it is affected by the size of testing data. So a size-independent measure is used, called perplexity. It is defined as

$$PPL = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i \mid w_{1\ldots i-1})}} \tag{2.7}$$

Perplexity can be viewed as the size of a vocabulary of an equivalent uniform language model, or as an average branching factor. Its value depends directly on the quality of the language model and the

vocabulary size. However, as mentioned in chapter 1, better probability estimates may not lead to better language models, although perplexity decreases. So it was observed by many that reduction in perplexity may not lead to reduction in word error rate (WER) in automatic speech recognition (ASR):

$$WER = \frac{i + s + d}{n} \tag{2.8}$$

where n is the count of words, i number of insertions, s substitutions and d deletions between the correct utterance and the one given by ASR system.

Accuracy is then defined as ACC = 1 – WER.

2.3 Advanced approaches for statistical language modeling

Although the plain n-gram approach seems to be very simple, it is considered to be the most successful single technique for language modeling. N-gram statistics are easy to compute even for huge training data, fast to evaluate and their results are very good. However, they are still far from the optimal solution, so more sophisticated techniques were developed.

2.3.1 Class based language models

Probably the most natural extension is by defining classes of words. In the simple case, where each word belongs exactly to one class, the model is called deterministic class based language model. For example, similar words, like days of a week, should belong to the same class. This reduces sparse data problem, since the algorithm uses probability estimates from all similar words. It is clear that this approach is most successful for small amounts of training data, where there is not enough information for classic n-grams.

Another approach is to use statistical classes, where for each word and each class is defined probability of the word membership to that class. The problem is that known algorithms for estimating word membership to every class are computationally very expensive.

Improvement over n-gram baseline can be achieved by interpolation with classic n-grams, since class based LMs lose some information.

2.3.2 Factored language models

In the classical language modeling, each word is translated as an index to dictionary, which is formed during training phase. This means that every sense of nearness between words based on written form is lost and must be reobtained using class based LMs or neural network based LMs. In a highly inflectional language, like Czech, it is better to not lose this information. Much better approach would be to divide words into smaller parts - factors - and compute statistics using them. Every word is then seen as a vector of *k* factors - $w_i = \{f_i^1, f_i^2, ..., f_i^k\}$. Factors can be anything - prefix, stem and suffix of words, class membership and so on. This approach is quite interesting in some aspects, since it allows us to compute statistics using words that were not present in the training data (OOVs, out of vocabulary). For example, we may have 10 occurrences of Czech word "Fourierova" in our training data, but no occurrence of "Fourierovy". If we use factored LMs with word division into prefix, stem and suffix, we should obtain useful estimates - while without this approach, the new word would be treated as OOV and no meaningful estimates for the next word would be possible to be computed.
Factored language modeling is still quite a new technique, promising interesting results in the future. It would be best used with another technique that allows us to compute statistics from many factors - for example, using Generalized parallel back-off [14] or neural networks working in continuous space [15].

2.3.3 Approaches based on neural networks

The use of neural networks in the field of artificial intelligence is a common task. However, for language modeling, this approach is quite new. The major reason is high amount of computational power needed by these algorithms. But it has been reported that the biggest improvement over classic n-grams using a single technique was obtained by using neural networks [2]. This is probably because a well trained neural network with hidden layer of adequate size can in theory perform any computable function. So, with a long context on input, the neural network may work as a mixture of many other techniques – n-grams, skipping n-grams, class based models and others.

But there are two big problems – first was already mentioned, the high requirements for computational power needed by these algorithms. The second is the training algorithm – although it is known that there exists an optimal solution if we use a hidden layer (with adequate size), the question is how to obtain this solution. This means finding global optimum in a large search space.

The basic algorithm used for training neural networks with hidden layers is a backpropagation algorithm. It is a gradient algorithm, so the obtained solution is only locally optimal.

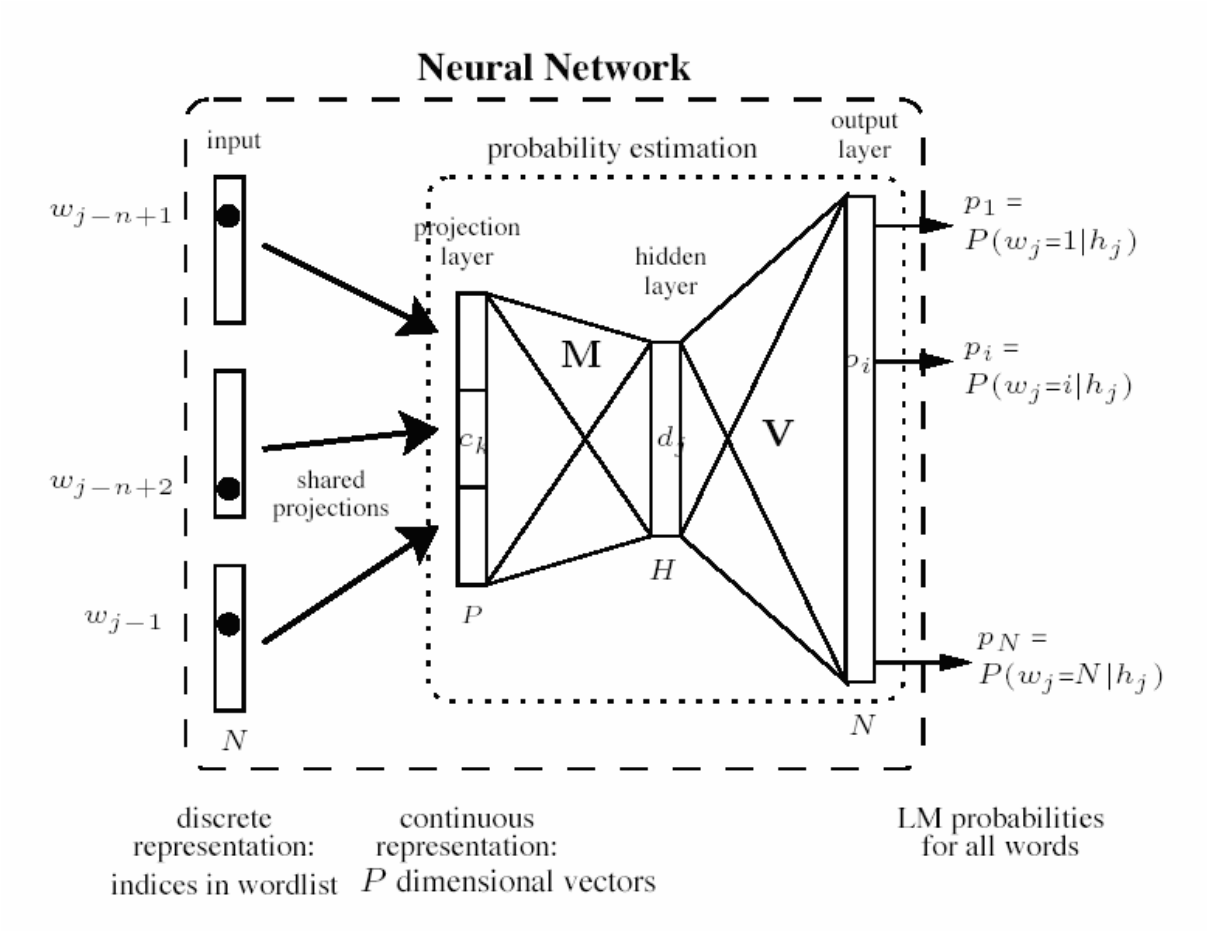


Figure 2: Architecture of the neural network language model used by [1]. *hj* denotes the context *wj-n+1, ..., wj-1*. *P* is the size of one projection and *H* and *N* is the size of the hidden and output layer respectively

At figure 2 is a typical architecture of a neural network for language modeling used by [1, 7]. In the input layer, words are coded as „one of N“ (input has the same size as vocabulary, at the word position is 1, elsewhere are 0). The history length is usually 4-8, so with a vocabulary size of 50, 000 words the input layer size would be too big.

To overcome this, words are projected onto much smaller space. In practice, each word in history is „translated“ to a vector of size 50 – 100. This step is very important: not only the number of synapses in the whole network dramatically decreases, but translating each word into a continuous space enables the network further exploit nearness of similar words. This compression loses some

information, and is somewhat similar to statistical classes in its behavior - two words that are used exactly the same will occupy the same position in the space.

Size of the hidden layer is typically 500 – 1000 units. The output layer has the same size as vocabulary. For a given context, output layer can be seen as a probability distribution for next word (softmax function is used to ensure that the probability sums to 1). More about this architecture can be found in [7].

3 Previous work in Czech language modeling

Previous works concentrating on creation of language models for Czech (at least, those known to me) are trying to solve problems with high OOV rate (Out Of Vocabulary words, those not found in the training data). As was mentioned in chapter 1, Czech is an inflective language and one word may have a lot of different written forms. This increases the size of vocabulary and makes classic language modeling less practical (and in case of some languages even nearly impossible).

In [4], inclusion of unigrams of rare words was investigated. This resulted in absolute 3% decrease in WER and increased the language model size only slightly. Another 2% improvement was achieved after lattice rescoring, again motivated by high OOV rate. Little difference between OOVs and IVs (In Vocabulary words) on the phoneme level was used to obtain this improvement. Third, lemmas were used as recognition units, but without much success.

In [3], author is at first describing simple experiments like different smoothing techniques comparison and determination of language scaling factor (used in ASR to combine language model probabilities with acoustic probabilities). The same idea as in the previous work to use lemmas to reduce OOV rate was investigated. The results are that using lemma together with word based LMs is slightly beneficial (0.2% absolute improvement). Next, tags were used to obtain classes instead of lemmas (1.6% absolute improvement over baseline trigram) and morphemes (2% better than baseline).

At Brno University of Technology, current language models for Czech were created by Ondřej Glembek and Ilya Oparin. For lecture transcription, the best results were achieved by interpolating data from several sources. Data sparsity problem was partially "solved" by using large amount of training data. Statistical class based models for Czech were created, but without much success, because with huge training data size, the beneficial effect of classes seems to be lesser.

3.1 Available data

Statistical language modeling techniques are very data hungry, since the more data we have, the better models we are able to train. A size of typical training data corpus may be easily few gigabytes. Despite this huge size, one cannot expect that a model trained on such an amount of text will be able

to handle all possible words that a person may use in spontaneous speech. For example, we cannot expect any corpus to contain all surnames, names of cities etc.

At Brno University of Technology, there are several data sources usable for training language models. General corpus, consisting of various articles, provides about 4.6 GB of text data (829 million words). However, for lecture recognition, spontaneous data are more desirable. For this purposes are available *Pražský mluvený korpus* (PMK, 686 000 words) and *Brněnský mluvný korpus* (BMK, 484 000 words). Note that sizes of spoken corpora are very small, compared to general corpus - in this case, we must at least try to mine the most data from available sources that is possible.

Probably the most valuable data are manually transcribed lectures, since they contain the target data - spontaneous speech with use of rare technical words. However, it is very time consuming to obtain such data and so the amount of text is very little (a few thousand words).

4 Goals of this work

This work primarily focuses on improving accuracy in the task of automatic speech recognition of lectures in Czech language. First, I am going to create a language model that will be able to better model Czech language than the classic n-gram approach. For comparison, I will be using perplexity. Then I shall use this LM to rescore N-best lists generated with baseline back-off n-gram LM.

The problems with ASR of lectures were already mentioned - mainly the use of words that were seen rarely or not at all in the training data. Since the speech is spontaneous and most of the available training data are just transcribed newspaper articles, there is a slight mismatch between training data and the speech we want to obtain.

I am going to use methods non-specific for Czech language only, so that this work may be usable for other languages, and even for other areas than language modeling. I shall compare my results with those obtained from SRI LM toolkit, which is freely available [6].

Language modeling is quite a difficult area of research, since optimal language models demand existence of artificial intelligence. In the conclusion, I am going to point out main weaknesses of current approaches.

5 Syllable-based language models

It is natural to notice first that problems with Czech language models using n-grams based on words are caused by inflection of words. My first idea to overcome this problem was to use syllable based language model instead of the word based one. It was supposed that this approach should reduce the number of OOVs, because many OOVs can be divided into known syllables.

But since ASR generates lattices using classic n-grams with a little (65, 000 words) vocabulary, rescoring with syllable based LM would be probably of no use, since there are no OOVs in those lattices. So another point of using syllables is to improve probability estimates for rare words, which are mostly inflected form of words with more reliable probability estimates. Experimental results of syllable based models and their comparison to word based models is summarized in table 1.

	Size	*Probability/1000*	*OOV rate (%)*
word based LM	10	-482	5.35
word based LM	40	-489	2.41
word based LM	80	-483	1.72
word based LM	250	-470	0.72
syllable based LM	10	-589	0.09
syllable based LM	40	-553	0.04
syllable based LM	80	-539	0.02
syllable based LM	250	-516	0.01

Table 1: Comparison of word based and syllable based n-gram language models. Size denotes training data size in megabytes (in case of syllable based LMs, it is the size of training data before word division is performed), probability denotes overall log probability of testing data obtained from SRI LM toolkit. Word based LMs are 3-gram, syllable based are 5-gram. Testing data size is 1MB.

As was expected, syllable based LMs are much better at handling OOVs, resulting in much lesser OOV rate. Their overall performance is significantly worse, but since SRI LM toolkit assigns zero log probability to OOVs, probability alone can't be used for comparison. Perplexity is not mentioned at all, since it uses count of words in testing data, which is different in both models (since syllable based model uses words divided into syllables). Vocabulary sizes are 385,481 for word LM and 21,776 for syllable LM when trained on 80MB.

It can be expected that for large training data sizes, the both models would be performing very similarly. There is no gain in using syllables, because true relations between syllables are longer range than normal n-gram approach is able to handle. So the syllable n-gram model collapses to just a word n-gram model and is not able to use more incoming information, as was expected at first.

Syllables are used just to divide the words and are not real syllables from linguistic point of view. The algorithm for dividing words is based on simple rules. Example of such division:

```
Original: ZIMNÍ OLYMPIJSKÉ HRY JAKO NEJLEPŠÍ V HISTORII

Divided: ZI MNÍ x OLY MPI JSKÉ x HRY x JA KO x NE JLE PŠÍ x V x HI STO RI I x
```

('x' is used as a special word, so that the transformation does not lose any data and is reversible)

Assume now that we have a rare word "FOURIEROVY" and a more common word "FOURIEROVA". We know that bigram "FOURIEROVA TRANSFORMACE" is quite often; our task is to assign part of this high probability to "FOURIEROVY TRANSFORMACE", since the rare word written form is very similar to the more common word. If we divide words into syllables, the history will change from "FOURIEROVY" to "FOU RI E RO VY". We have good estimates for history "FOU RI E RO VA"; however, with language model with temporal back-off, we will end with the same results as when using words, since the "good" statistics of "FOURIEROVA" will not be used at all - "VY" differs from "VA". This can be partially solved by using skip n-grams, or better with class based language models or neural networks.

Because plain n-gram statistics is unable to use more information provided by syllables, other modeling approaches were investigated. The most powerful seems to be the neural networks, because of their ability to perform any function.

6 Neural networks in language modeling

Problems with neural networks in language modeling have been already mentioned in chapter 2. Their need for computational power limits their use, but since they offer significant improvement in WER over classic n-grams, it can be expected that in the future their use may arise.

6.1 Architecture

My first experiments with neural networks were therefore made on a small amount of data. I have used similar architecture as the one mentioned in chapter 2, but instead of using one neural network with two hidden layers, I have decided to use two networks, both with one hidden layer. The first neural network learns to project words from vocabulary into a continuous space while learning bigram language model. The second neural net is used to learn LM based on longer history, using the word projections computed by the first network.

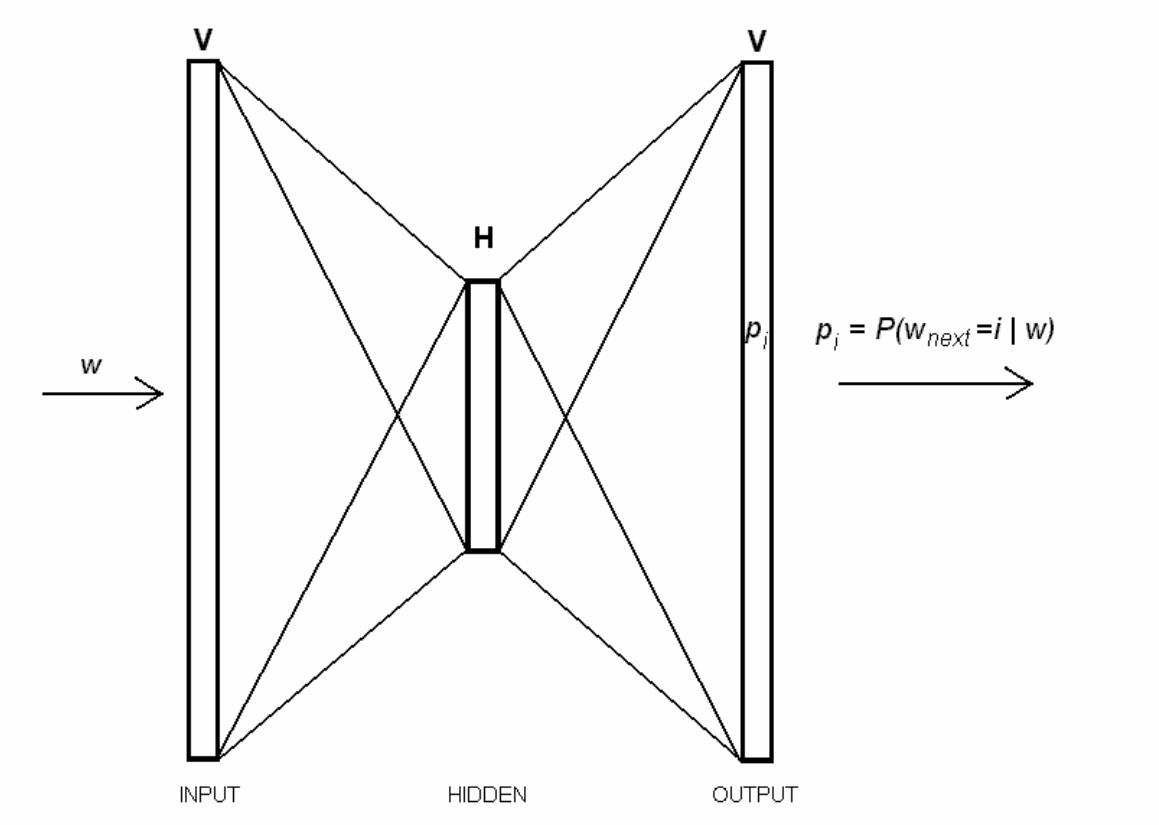


Figure 3: Neural network used for learning word projections into continuous space (bigram NN).

The first neural network has one hidden layer. Input and output layers have the same size as the vocabulary; hidden layer size is typically 10-50 neurons. Activation function in hidden layer is sigmoid, for output layer is used softmax, to ensure that sum of probabilities in the output layer will be 1.

Since only one neuron in the input layer is active in one time, it is not needed to propagate signals to and from whole input layer. The complexity to calculate one probability estimation with this neural network is therefore

$$O = 1 + H + H * V + V \tag{6.1}$$

The memory requirements to store this neural network is

$$O = 2 * V * H \tag{6.2}$$

Using the word projections learned by the bigram neural network, it is possible to train n-gram neural network:

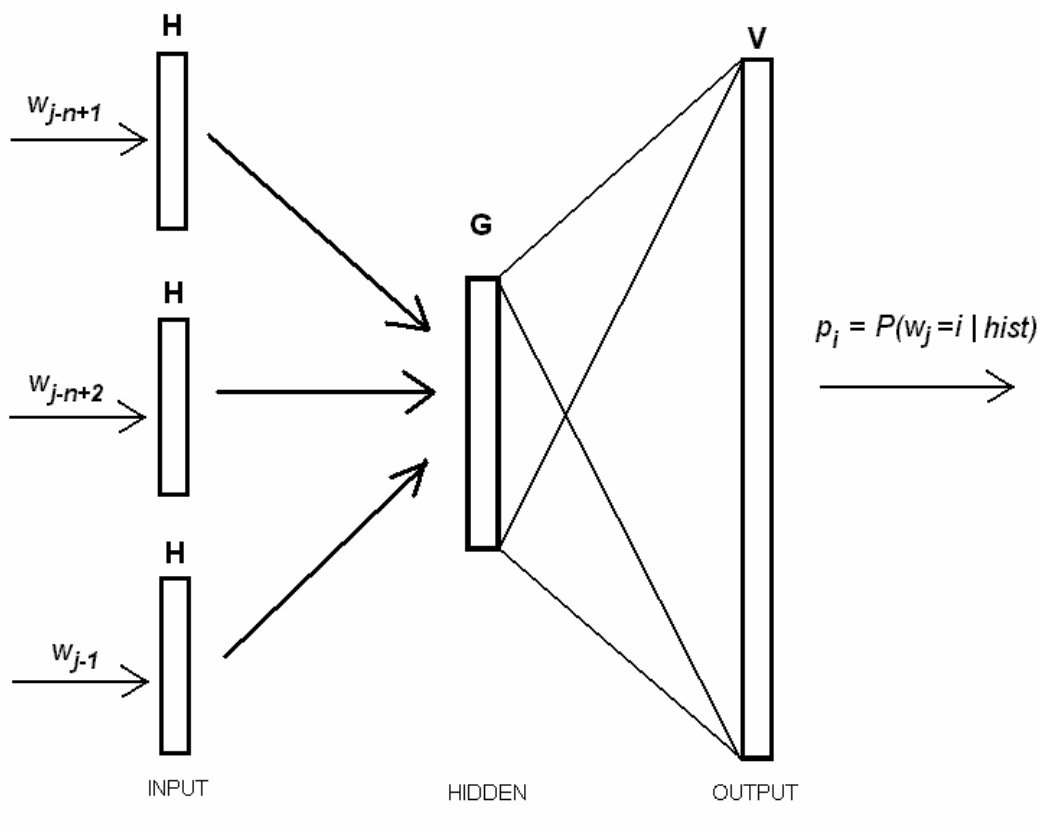


Figure 4: Neural network used for learning n-gram language model, *hist* denotes the context.

The second network has input layer with size (N-1) * H, where N-1 is the length of context (for trigram model, N=3) and H is the size of projections learned by the first network (typically 10-50). Useful size of hidden layer (G) is 20-100 neurons. Output layer size is the same as the size of vocabulary. Activation functions are the same as in the first network (sigmoid, softmax).

The computational complexity of one probability estimation using this neural network is

$$O = (N-1) * H * G + G + G * V + V \tag{6.3}$$

The memory requirements are

$$O = (N-1) * H * G + G * V \tag{6.4}$$

6.2 Training algorithm

Classic backpropagation algorithm is used for training of both networks. In the first network, input is coded as 1 of N - on the position of last word in history is 1, everywhere else in the input vector are zeros. The desired output vector uses the same coding – contains 1 on the position of the word that should have been predicted, elsewhere 0. Error vector is then computed as *Error = Desired - Output*. The second network uses word projections to obtain its input, training is the same as in the previous case. Starting learning rate values are 0.05 – 0.2. Both networks learn until no improvement on validation data set is obtained, then the learning rate is halved. After no significant improvement after learning rate division is obtained, the learning process is finished (it takes usually 10-30 epochs to train one network).

[1] suggests using weight decay to prevent overfitting the training data by penalizing big weights in the network. This is done by adding some value to the error vector, based on a sum of weights of incoming synapses to some particular neuron, multiplied by some parameter β that has to be determined experimentally.

For n-gram neural network, another modification of the training phase was investigated. By assuming temporal backoff (words in recent history are more important than words in distant history), vectors that represent positions of words in the continuous space in a distant history were multiplied by some constant lesser than 1 (which has to be again found experimentally). For example, for 6-gram neural network trained on little data, the best results were obtained by multiplying vectors of words in

history (w_{j-5}, w_{j-4}, w_{j-3}, w_{j-2}, w_{j-1}) by constants (0.15, 0.3, 0.7, 0.9, 1.0). It was found that a 6-gram network trained without this modification provides worse results than a 3-gram network, while with the modification the results are getting better with a longer context. This is caused probably by two reasons: the network with long context possibly overfits the data, and may be confused by long history. So this is a mechanism to tell the network, which data are more important. It was also observed that modification of the constants during training may also be useful.

6.3 Implementation

The implementation itself consists of a few simple programs written in C language. On the input, clear text usable for building a language model is expected (one sentence per line, all characters uppercase, only letters + space + end of line symbols). Example of good training data:

```
TŘEBA V TŘEBONI NEBO ČESKÉM KRUMLOVĚ TAM JE TO NÁDHERNÉ
PAKLIŽE TAM MÁ ČLOVĚK ZÁZEMÍ A PRÁCI MŮŽE TO BÝT PŘÍJEMNÉ
ALE V PRAZE JE NAŠTĚSTÍ SPOUSTA MÍST KTERÁ MILUJI
JÁ MOC RÁDA CHODÍM A TAK JDU Z DIVADLA ČASTO PĚŠKY DOMŮ
VEZMU TO NERUDOVKOU PŘES HRAD A POKAŽDÉ JSEM OKOUZLENÁ
ÚPLNĚ PŘEKRÁSNÉ JE TO PRÁVĚ VEČER
V ZIMĚ NEBO V LÉTĚ TO JE JEDNO
```

To control training and prevent data overfitting, validation data with the same format are required (a few hundred words should be enough). The performance is then evaluated on testing data set, again with the same format of data. SRI LM toolkit is used for comparison of results.

If we are interested in evaluation of perplexity improvements of neural networks over back-off n-gram models, typical sequence of required steps is this:

1. Add end of sentence tag (**</s>**) to the data files before each end of line symbol. This is done to ensure compatibility with SRI LM - each sentence is processed independently, assuming no relations between ongoing sentences. In n-gram neural network, history is erased before new sentence is processed, to ensure that the network has no more data than SRI LM uses.

2. Create vocabulary using the training data. Vocabulary consists of all distinct words used in the training data. Vocabulary is sorted, so that the most frequent words are first in the list.

3. Rewrite training, validation and testing data files, so that every word is rewritten as an index to the vocabulary. For words from validation and testing data which are not found in the vocabulary, special OOV word index is generated.

4. Compute bigram neural network on the training data while using validation data to control the training process.

5. Use results from step 4 to establish a file containing positions of all words from vocabulary in the continuous space, as it was learned by the bigram neural network. For example, if the first network used a hidden layer with 30 neurons, each word may be rewritten as a vector in 30-dimensional space

6. Compute n-gram neural network using training and validation data. Every word is translated to the continuous space, as it was computed in the step 5. For details, see figure 3 and 4.

7. Compute language model using SRI LM toolkit. This is usually done by executing

```
ngram-count -text train_text -lm language_model -order 4
 -kndiscount -interpolate
```

It is possible to tune the resulting language model by modifying the order, or by using different smoothing methods. However, 4-gram with modified KN discounting is usually the best choice.

8. Evaluate testing data by n-gram neural network and using LM learned by SRI LM toolkit. This is done by using command

```
ngram -lm language_model -ppl test_text -order 4 -debug 2
```

9. Using results from step 8, compute perplexity of the test data using predictions from neural network and SRI LM toolkit. The currently used interpolation method is simple linear interpolation, with interpolation coefficient γ=0.5 (can be computed using validation data).

Sample output file:

```
Word       NN prob.         SRI prob.       Interpolated prob.
   18   NN:   0.003689     SRI:   0.003684   MIX:   0.003686     LOGP:  -2.4334
  176   NN:   0.004950     SRI:   0.005980   MIX:   0.005465     LOGP:  -4.6958
    1   NN:   0.148345     SRI:   0.208526   MIX:   0.178436     LOGP:  -5.4443
  766   NN:   0.000284     SRI:   0.000126   MIX:   0.000205     LOGP:  -9.1322
  215   NN:   0.000315     SRI:   0.000315   MIX:   0.000315     LOGP: -12.6337
```

```
 7783   NN:   0.000005     SRI:   0.000005   MIX:   0.000005     LOGP: -17.9728
  136   NN:   0.001380     SRI:   0.000574   MIX:   0.000977     LOGP: -20.9830
   52   NN:   0.003587     SRI:   0.003176   MIX:   0.003381     LOGP: -23.4539
    6   NN:   0.183653     SRI:   0.236825   MIX:   0.210239     LOGP: -24.1312
.
.
.
.
   21   NN:   0.002435     SRI:   0.008481   MIX:   0.005458     LOGP: -13254.2197
  248   NN:   0.001203     SRI:   0.000217   MIX:   0.000710     LOGP: -13257.3682
   15   NN:   0.016652     SRI:   0.023843   MIX:   0.020247     LOGP: -13259.0615
LOG PROB: -13259.061523
Words: 5643
PPL SRI: 284.372998
PPL NET: 244.805139
PPL MIX: 223.690541
OOV rate: 4.19%
```

Since the interpolation step combines two probability distributions, which both sum to 1, we can easily compute linear interpolation as $P_{INTERPOLATED} = P_{NN} * \gamma + P_{SRI} * (1- \gamma)$, $\gamma \in < 0,1 >$.

Implementation note: the only important thing when making a language model is to keep in mind that the model must not use information from the future - building a prediction model that has access to the future is of course senseless;) Also, it is good to check that the probability distribution really sums to one.

7 Experiments and results

In the first experiments with a small amount of data, I have used approximately 22 000 tokens for training, 6 600 for testing, vocabulary size was 700. Data consisted of Czech text with words divided into syllables, so that the vocabulary size was small enough to run similar tests with SNet implementation of neural networks [16] (SNet was unable to handle more than thousand words in vocabulary; the results were also slightly worse than from my implementation, mainly because weight decay and history attenuation extensions). SRI LM toolkit was used for comparison. For bigram statistics, results from SRI LM were 0.5% better in probability than from the best bigram neural network. The best single neural network (with long context) was 1% worse than the best result from SRI LM (trigram with modified Kneser-Ney discounting). Interpolation of four networks was better by 0.65% than the best result from SRI LM. After interpolation of the best results from neural networks and SRI LM, there was an improvement of 1.3% in probability over SRI LM baseline (4.5% improvement in perplexity). These results were obtained with weight decay, which has proven to be useful to improve training, at least in this case. History attenuation was also used. However, these results were quite poor and it was a question, whether the neural networks will be able to achieve better results with more data.

For the other experiments, Czech lecture transcriptions and a part of PMK+BMK were used.

	Vocabulary size	Training data size (words)	Test data size (words)	SRI PPL	NN PPL	SRI+NN PPL	**Improvement**
Czech lecture transcriptions	1 276	6 017	1 071	140.7	166.5	132.1	**6.1%**
Part of PMK+BMK corpus	5 385	31 618	6 538	183.3	190.3	168.0	**8.3%**

Table 2: Perplexity on small amount of Czech word based data

The results indicate better improvements in perplexity with more training data. It is because the neural network must see a certain word many times to place it in the right position in the multidimensional space. It was expected that with more training data, this effect may be strengthened.

The computational complexity of the used architecture is pretty high, and is linearly depended on the size of training data and vocabulary. Although the experiments were done with artificially little data (no useful language model may be computed using 31 000 words), the training time was pretty high -

for training of part of the PMK+BMK corpus it was 1.5 hour on a computer with an AMD Opteron 2,8GHz processor.

It was observed during parameter tuning that neural network with lower perplexity may not lead to bigger perplexity reduction after interpolation with statistical n-grams. The important thing for perplexity reduction is neural network's ability to discover relations in the data undiscovered by statistical n-grams. For example, bigram neural network was usually better in perplexity than 4-gram network. However, after interpolation with statistical n-grams, bigram network provided almost no new information, while 4-gram network did.

Another important thing was the use of weight decay and history attenuation - although these extensions have proven to be useful to improve the perplexity of the neural network itself, after interpolation, the behavior was again quite strange. Usually, both extensions helped to reduce the perplexity of the interpolated model by a few percent, at the cost of tuning the parameters. Because these extensions were found to hurt the training process (both speed and generalization of the model), if the parameters were not tuned, they were not used in the ongoing work. Their benefit seems to be too small.

7.1 Vocabulary size reduction

Although the training data size in the previous experiments was quite small, the training process took very long. Since the goal of this work was to train a language model based on all the spontaneous speech corpora (PMK+BMK), which consists of 1 170 000 words, it wouldn't be possible to train a neural network on this data in a reasonable time. So some improvements in implementation and architecture were made, to make the training process possible.

It is clear that the overall performance depends mainly on two things: amount of training data and the vocabulary size. From these, it is possible to optimize the latter one. As was already mentioned, the neural network is able to increase the amount of learned data by assigning words to a right position in multidimensional space, so that words with similar use in an utterance lie near each other. However, it was also mentioned that with small amount of training data, the network is unable to find the right position, because there is simply not enough occurrences of a certain word to compute reliable statistics. We may exploit this fact to increase overall performance without hurting the training process.

There are two ways, both very similar, how to reduce vocabulary size by throwing out the least used words. We may assign all the rarely used words to one token (for example, <rare>), and compute statistics for these words by assuming uniform distribution. For example, if the predicting model assigns probability 0.5 to the <rare> token, with 100 rare words assigned to the <rare> token, the probability estimation for each rare word would be 0.005. Since the vocabulary consists mostly of rare words, we may easily reduce its size to a half or a third by throwing all the words, which occurred less then two or three times in the training data. This is quite the same solution as was used in [1].

The other way, used by [7], is to use so-called shortlists. The principle is quite the same: neural network is used to predict only *s* most frequent words. All words from the vocabulary are still considered as input of the neural network.

$$\hat{P}(w_t|h_t) = \begin{cases} \hat{P}_N(w_t|h_t)P_S(h_t) & \text{if } w_t \in \text{shortlist} \\ \hat{P}_B(w_t|h_t) & \text{else} \end{cases} \tag{6.3}$$

$$P_S(h_t) = \sum_{w \in shortlist(h_t)} \hat{P}_B(w|h_t) \tag{6.4}$$

where P_N denotes probabilities in the shortlist computed by the neural network and P_B probabilities obtained from standard 4-gram back-off LM. This means that the neural network redistributes the probability mass of words in the shortlist. However, this approach needs to evaluate probabilities of all the words in the shortlist using standard back-off model.

In my implementation, I have used the first solution, merging all the rare words into one token. However, it's hard to predict behavior of such a solution; defining threshold of the least frequent words that will be present in the vocabulary does not tell us anything about the reduction of the vocabulary size. So the more natural solution seems to be using the shortlists; however, this approach as implemented by [7] would need better cooperation with back-off LM, since it is needed to compute probability mass assigned by the back-off model to all the words in the shortlist. So the used simple solution is to define the shortlist size and discard all the words past this limit in a vocabulary sorted by frequency of words. Thus, neural network learns only to predict words that are in the shortlist. Final probability for a word *w* in a given context *h* is then computed as

$$P(w \mid h) = P_N(w \mid h) * \gamma + P_B(w \mid h) * (1 - \gamma)$$

(6.3)

The reason why [7] are using a more complicated solution is because they reduce the vocabulary size to 2 000 words, while in the experiments presented here, the shortlist is typically 15 000 - 20 000 words.

However, neural network trained this way will output zero probability estimation for words that are not present in the short list. Although we are primarily interested in perplexity of an interpolated model (so the zero probability from neural network is not a problem), it may be useful to report results obtained using only the neural network. For this case, it is possible to use previous solution (merging words into one <rare> token), or simply redistributing part of the probability over the rare words uniformly.

Surprisingly, results obtained with well reduced vocabulary are sometimes even better than those computed with full vocabulary. The reason is probably significant reduction of parameters of the neural networks. Discarding all the words occurring less then three times in the training data is probably the most effective solution. Although it is possible to reduce the vocabulary further, the results are going worse with bigger reduction.

7.2 Final training and evaluation using perplexity

With the implementation of shortlists, it was possible to train a neural network based language model using all the data from *Pražský mluvený korpus* and *Brněnský mluvený korpus* (PMK+BMK). This corpus consists of 1 170 000 words. For evaluation purposes, it was divided into three parts - 1 155 000 words for training data, 5 500 words for validation data and 9 500 words for testing. The baseline was a 4-gram back-off language model using modified Kneser-Ney smoothing, learned by SRI LM toolkit.

Full vocabulary computed on the training data contained 68 500 words, from which only 20 300 were used more than two times. Starting value for learning rate was set to 0.1 in both networks. Experiments were made to see how much important is this value - how it affects the number of training epochs and overall perplexity results.

Learning rates		Training epochs		Perplexity of interpolated models
0.3	0.2	11	10	222.6
0.1	0.1	12	11	222.3
0.06	0.03	14	12	223.9

Table 3: Final perplexity (after interpolation with back-off LM) and required training epochs for bigram and n-gram neural network with different starting learning rates (with shortlist 15 000 words).

It seems that the starting learning rate value is not much important and a well chosen value may only improve the number of required training epochs.

Shortlist	Validation	Test	Lecture
10 000	248.7	273.8	534.9
15 000	248.5	276.5	541.6
20 000	247.1	276.2	540.9
25 000	246.2	272.5	533.6

Table 4: Perplexities of bigram neural network (after interpolation with KN 4-gram) with different size of the shortlist on various data.

As it can be seen, determining the optimal shortlist length is not an easy task. The reason of this random performance lies in the neural network itself - the starting weights of the network are chosen randomly, so a network with the same parameters and architecture trained two times may provide significantly different results. The only way how to provide reliable results would be to train a network many times and selecting the one giving the best results, otherwise any tuning of parameters must be based more on experience than on some particular results.

	Validation data	Test data	Lecture transcription
KN 4-gram	284.37	299.96	613.39
Neural network	242.77	274.70	533.17
Interpolation	221.34	250.30	486.68
Improvement	**22.2%**	**16.6%**	**20.7%**

Table 5: Perplexity of language models trained on PMK+BMK corpus, rare words are merged into one token

Table 5 shows final results on validation and test data sets. These results were obtained with these parameters: all words occurring less then three times were merged to <rare> token (vocabulary reduction to 20 300 words), starting learning rates for both networks were 0.1, sizes of hidden layers were 30 neurons for the bigram network and 50 for the n-gram network. Length of the context for n-gram network was chosen to be 5 words. Perplexity on lecture transcription is also reported, since the language model is aimed to model this type of data. Training time was 46 hours on a computer with an AMD Opteron 2,8GHz processor.

The final results are comparable with [1], who has achieved a 20.1% perplexity reduction on validation data set and 21.5% reduction on test data on a Brown corpus (English text, 800 000 words for training, 200 000 validation and 181 000 testing, rare words occurring less then 3 times merged into one token, reducing vocabulary size to 16 400 words).

	Validation data	Test data	Lecture transcription
KN 4-gram	284.4	299.9	613.4
Neural network	235.8	290.9	549.2
Interpolation	217.6	256.8	497.6
Improvement	**23.5%**	**14.4%**	**18.9%**

Table 6: Perplexity of language models trained on PMK+BMK corpus, simple shortlist implementation with 20 000 of the most frequent words.

As it can be seen, although the implementation with shortlists performs better on the validation data than implementation with merging words into one token, it is worse on the test data.

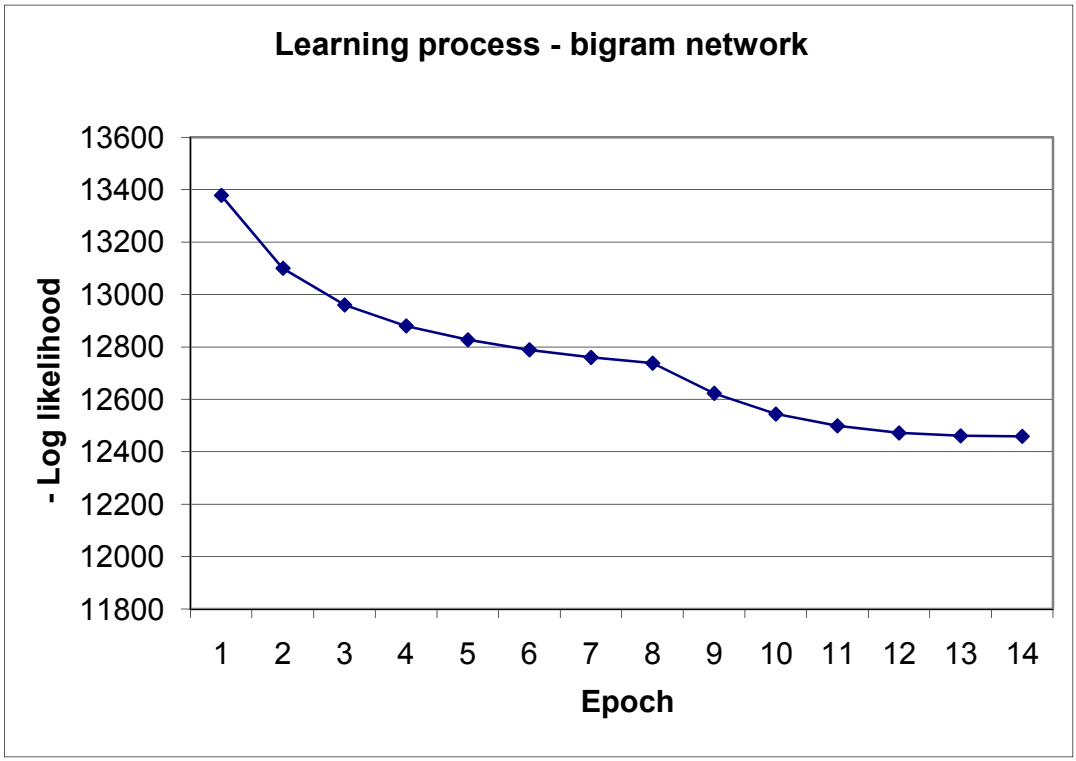


Figure 5: Log likelihood of validation data while learning bigram neural network language model. It can be easily seen that the training algorithm started to divide learning rate after epoch 8.

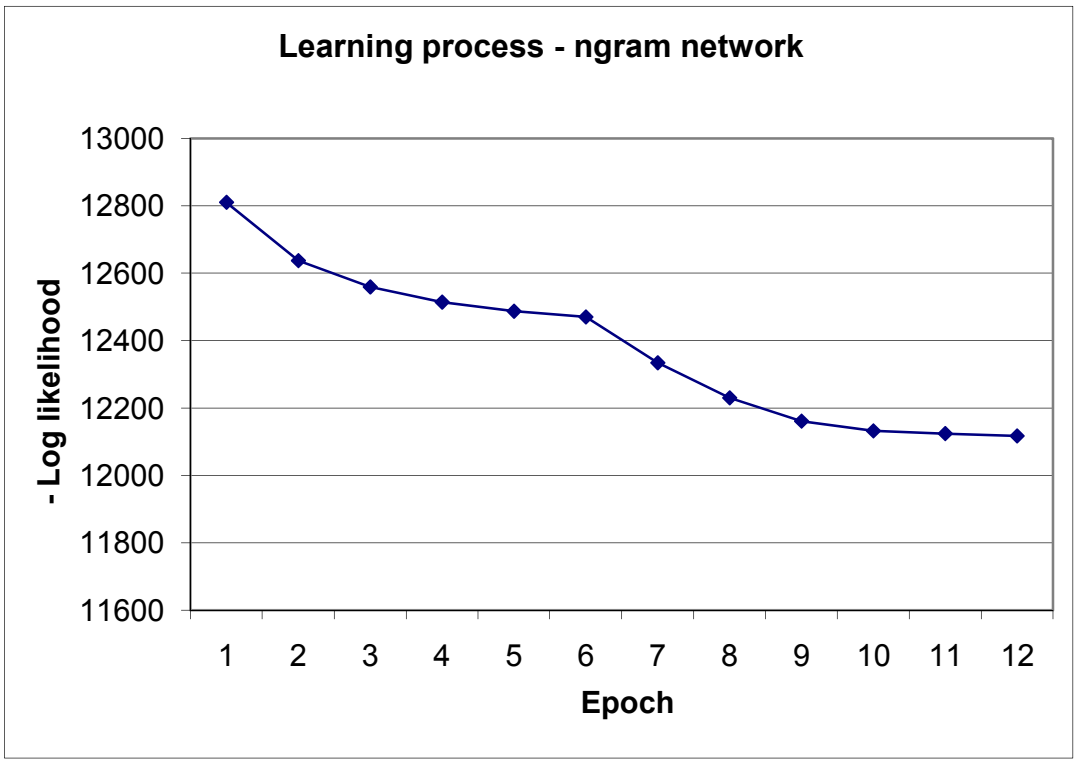


Figure 6: Learning process of the n-gram neural network.

7.3 Lattice rescoring

Since this whole work is aimed on improving speech recognition of spontaneous Czech speech, experiments with lattices were also performed. Lattice is an oriented graph containing possible hypothesis for some utterance, for example:

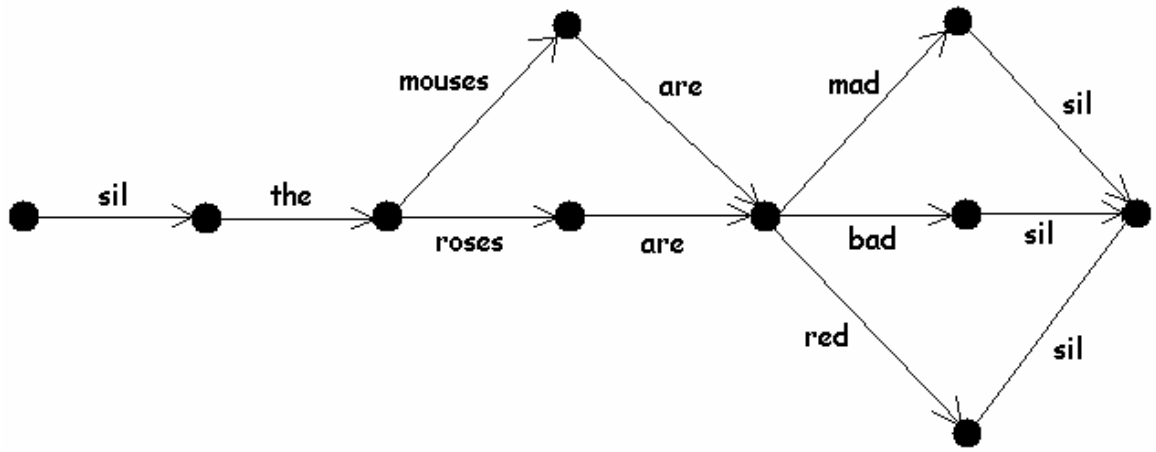


Figure 7: Sample lattice (the language and acoustic score values are not included)

It is the ultimate goal of language modeling to assign the highest probability to the hypothesis that is the most meaningful in a given context. It is obvious that in the sample lattice, the path "the roses are red" should have higher probability than "the mouses are mad".

Lattices are produced by some decoding system from the acoustic data using a simple back-off language model. They contain acoustic and language score for each edge. The acoustic score represents, how much likely a certain word was really said according to the acoustic models. The language score represents, how much likely the speaker intended to say it.

The first experiments with lattice rescoring were done with a data from Speecon and TEMIC, which consists of 1 695 Czech non-spontaneous sentences. The original language score in those lattices was computed using trigram back-off language model trained on gigabytes of textual data. On the other hand, neural network LM was trained only on PMK+BMK corpus. This resulted in high OOV rate in lattices for the NN LM - 15% of words were not ever seen by the neural network LMs, and another 8% were words that occurred in the training data less than three times. So, neural network language score was taken as an additional information, instead of replacing the old score.

First step to rescore lattices is to generate N-best lists from them. N-best list is a set of N best hypotheses in a lattice with their acoustic and language scores. Neural network LM was used to compute probability for each of this hypothesis; the OOV words were treated as rare words, thus they were given a small probability to avoid zero probability problem. Since N-best lists tend to contain redundant data, it was very useful to implement a cache based on a hash function, since during evaluation, neural network does not change and for the same history, the output will be always the same. Cache hit was about 92.44% (mainly because N-best list length aligning; the true cache hit can be around 70%).

The final score for each hypothesis was then computed as

$$LM\_SCORE = \gamma * NNLM + (1-\gamma) * LM$$
$$SCORE = AC\_SCORE + LM\_SCALE * LM\_SCORE + WORD\_COUNT * WI\_PENALTY$$

where NNLM denotes the probability of hypothesis computed by the neural network language model, LM denotes old language score, γ is an interpolation coefficient, LM_SCALE is a language score scaling factor, WORD_COUNT is the number of words in the hypothesis and WI_PENALTY is a word insertion penalty score.

The parameters that significantly affect overall accuracy are LM_SCALE and WI_PENALTY. The first one is used to strengthen the influence of the language model score - since there is more variability in the acoustic data, they have much lesser likelihood and would otherwise dominate in the overall score. Word insertion penalty is then used to balance between word insertions and deletions mistakes.

For experimental purposes, evaluation was done on all the available data. First, LM_SCALE and WI_PENALTY were tuned to obtain the best results with N-best lists (N=300), achieving baseline accuracy 81.65%. After incorporating score obtained from neural network and tuning the parameters (LM_SCALE, WI_PENALTY and γ), the accuracy went up to 82.01%. However, such an improvement is very modest, since it was obtained only on validation data. Oracle accuracy (best possible path through lattice with the highest accuracy, computed using the correct transcription) of lattices was 95.5%, so there was a plenty of space to improve.

The reason why neural networks didn't help much is probably because the network was trained on spontaneous speech transcriptions, while the Speecon + TEMIC data contained mostly artificial

sentences (no speaker mistakes etc). Also, lattices contained mostly words unknown to the neural network.

Another experiment, much closer to the PMK+BMK corpora, was a lecture data recognition - containing spontaneous speech with technical terms. A lecture consisted of 873 sentences, with original accuracy 54.24%. Since the experiments were made again on all the available data, first thing to do was to determine the best possible results with tuning of the LM_SCALE and WI_PENALTY parameters.

	Accuracy (%)
Original lattices	54.24
Tuned parameters	**56.82**
NN LM	55.80
old LM + PMK + BMK	56.86
old LM + NN LM	**57.99**

Table 7: Accuracy on lecture data

The experiments were made on N-best lists with N=100. After parameter tuning, obtained accuracy went up to 56.82%. This value was considered as a baseline. After substituting old language model score with the one obtained with neural network, accuracy went down to 55.80% - however, the original language model was trained on a huge data so it was expected that neural network alone will perform much worse (on the other hand, just by using 100-best list is a way of using the old language model). Another experiment was to determine the effect of PMK+BMK data - after computing a language model based on this corpora and an interpolation with old language model score, accuracy went up to 56.86%, which is almost no improvement over the baseline. Finally, after interpolating score from the neural network and the old language model, accuracy went up to 57.99%. The rescoring process took approximately only 0.5 hour, mainly because the use of cache.

8 Conclusion and future work

The neural networks working in continuous space are able to significantly reduce perplexity of state of the art 4-gram back-off language model by exploiting similarity of certain words. Their use in any natural language processing task should be beneficial, be it machine translation, optical character recognition or speech processing.

The main outcome of this work is an implementation of neural network based language model. The perplexity reduction over 20% is itself a very good result in comparison with other approaches for advanced language modeling (factored language models, class based models), at the cost of computational complexity needed for training the model. However, rescoring of N-best lists can be quite fast, working in much less than 1xRT, so the use in a practical system is possible.

The neural network based language model should provide the best improvements if it would be trained on huge data - this is however impossible with the current implementation. There are ways how to speed up the whole implementation many times (70x or more using hierarchical neural network) - this should be investigated in the future work.

There are also other possible uses of this work - since the neural network has some sense of nearness between words, it can be used for example to generate additional training data for the standard back-off language model. Examples of near words can be found in Appendix A.

One possible advantage of the neural network LM over standard back-off LM is the memory requirements - as can be seen in (6.2) and (6.4), memory requirements do not increase with the amount of training data, only with the size of vocabulary. So the resulting language model may be only a few megabytes in size, even if trained on huge data.

Although the final perplexity reduction seems to be quite big, we are still very far from the optimal solution, artificial intelligence. But it can be said that the neural network based language model is more intelligent than the simple back-off LM. To improve things further, we may choose two paths. The first and easy one is to take some other successful language modeling techniques like cache language models (or better trigger LMs and topic LMs) and factored language models and build a complex model. This approach has one main advantage: it will surely work. The disadvantage is that it will not lead to AI, since it is an ad hoc solution.

The second way would be to propose completely new technique for language modeling. Main weaknesses of current language modeling lie in its simplicity, taking word units as the ultimate and often the only source of information. To identify sub-word information, new techniques, like factored language models, must be employed. On the other hand, to capture information over long contexts of words, techniques like cache models, trigger models or topic language models were developed. All these techniques aim in fact the same problem.

The other main weakness, inability to follow more information sources in one time, is partially solved by class based LMs and NN LMs. However, these techniques are crude and there is a plenty of space for improvements.

I believe that there exists a general solution able to capture information from a natural signal. However, finding this solution is a pure research with uncertain results, which may not be directly applicable in practical systems.

Bibliography

[1] Yoshua Bengio, Réjean Ducharme and Pascal Vincent. 2003. A neural probabilistic language model. Journal of Machine Learning Research, 3(2):1137–1155.

[2] Joshua Goodman, Eugene Charniak. 1999. The State of The Art in Language Modeling.

[3] Ircing Pavel. Large Vocabulary Continuous Speech Recognition of Highly Inflectional language (Czech). 2003. Ph.D thesis, University of West Bohemia in Pilsen Department of Cybernetics

[4] Petr Podveský. Speech Recognition of Czech Using Finite-State Machines. Ph.D thesis, Ústav formální a aplikované lingvistiky, MFF UK

[5] Holger Schwenk and Jean-Luc Gauvain. Training Neural Network Language Models On Very Large Corpora. Published in Joint Conference HLT/EMNLP, pages 201–208, oct 2005

[6] Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. International Conference on Speech and Language Processing, 2002, pp. II: 901–904.

[7] Holger Schwenk and Jean-Luc Gauvain. 2005. Building Continuous Space Language Models for Transcribing European Languages.

[8] Jeff Hawkins and Sandra Blakeslee. 2004. On Intelligence. Times Books, Henry Holt and Company, New York, NY 10011. In Press.

[9] Mathew V. Mahoney. 2000. Fast Text Compression with Neural Networks.

[10] Joshua Goodman. 2006. Language Models for Handwriting.

[11] Shannon, C.E. 1948. A Mathematical Theory of Communication. Bell System Technical Journal, 27, pp. 379–423 & 623–656.

[12] C. S. Wallace and D. L. Dowe. 1999. Minimum message length and Kolmogorov complexity. Computer Journal (special issue on Kolmogorov complexity), 42(4):270--283.

[13] Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In ICSLP, pages II: 901–904.

[14] J. A. Bilmes and K. Kirchhoff. 2003. Factored language models and generalized parallel backoff. In Proceedings of HLT/NACCL, pp. 4--6.

[15] Andrei Alexandrescu, Katrin Kirchhoff. 2006. Factored Neural Language Models.

[16] S. Kontár. 2006. Parallel training of neural networks for speech recognition. In Proc. 12th International Conference on Soft Computing MENDEL'06.

Appendix A

Example of the nearest words in multidimensional space learned automatically by the bigram neural network:

The nearest words to 'STO' and their distance in the continuous space:
0.000000 STO
0.485603 PADESÁT
0.634387 SET
0.715822 TISÍCE
0.548773 ŠEDESÁT
0.331836 STA
0.740655 PATNÁCT
0.644968 DEVADESÁT
0.265452 STĚ
0.568353 DALŠÍCH
0.744552 SEDUMDESÁT
0.722997 PRVNÍCH
0.733424 NULA
0.463603 VOSUMDESÁT
0.735826 SEDUMNÁCT
0.697874 ŠTYŘICET
0.785682 OSUMNÁCT

The nearest words to 'ONI':
0.166965 VONI
0.642922 DĚCKA
0.738023 UČITELÉ

The nearest words to 'PRÁCE':
0.338162 PRÁCI
0.444156 ZAMĚSTNÁNÍ
0.453964 DĚTÍ

The nearest words to 'NIMA':
0.254859 NÍ

0.408227 NĚ
0.366906 NICH
0.277304 NĚJ
0.447615 NIM
0.344229 JU
0.314955 VÁS
0.383597 NĚHO
0.428735 NĚM
0.480196 KLIDNĚ
0.264886 NÍM
0.468224 NÁHODOU
0.325690 NI
0.461070 DÁVNO
0.304047 TEBOU
0.464210 NĚMU

The nearest words to 'MLADÝ':
1.382202 NOVÝ
1.348571 RÁDI
1.292689 STARŠÍ
0.833468 STARÝ
1.217794 OSTATNÍ
0.977726 CIZÍ
1.372153 TYTO
0.744946 SLUŠNÝ
1.352882 PŘÍJEMNÝ
1.372255 PLNÝ
1.169210 HODNÝ
1.000128 MLADÉ