Improvements in Language and Translation Modeling

Von der Fakultät für
Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
genehmigte Dissertation

vorgelegt von
Dipl.-Inform. Martin Sundermeyer
aus Dortmund

Berichter:
Prof. Dr.-Ing. Hermann Ney
Prof. Dr. Francisco Casacuberta

Tag der mündlichen Prüfung: 2. Juni 2016

Dipl.-Inform. Martin Sundermeyer
Human Language Technology and Pattern Recognition Group
RWTH Aachen University
sundermeyer@cs.rwth-aachen.de

Abstract

Virtually any modern speech recognition system relies on count-based language models. In this thesis, such models are investigated, and potential improvements are analyzed with respect to optimized discounting methods, where empirical count statistics are adjusted such that they better match unseen held-out data.

Neural networks have recently emerged as a promising alternative to count-based approaches. This work introduces the recurrent long short-term memory neural network into the field of language modeling, and provides an in-depth comparison with other neural network technologies as well as count-based approaches. Due to the high computational complexity, neural network models are combined with several speed-up techniques. In thesis, novel approximations for direct word error minimization in lattice rescoring are proposed, too.

Finally, long short-term memory neural network language models are generalized to the translation modeling problem of statistical machine translation. Two novel methods are presented, preserving word order of source and target language dependences. Both approaches allow the integration in state-of-the-art machine translation systems, complementing the improvements obtained by neural network language models.

Zusammenfassung

Nahezu alle modernen Spracherkennunssysteme bauen auf häufigkeitsbasierten Sprachmodellen auf. In dieser Dissertation werden diese Modelle untersucht und die möglichen Verbesserungen mit Hilfe von Discounting-Methoden analysiert, wobei empirische Häufigkeiten so angepasst werden, dass sie denjenigen von ungesehenen Daten besser entsprechen.

In letzter Zeit sind Neuronale Netze als vielversprechende Alternative zu häufigkeitsbasierten Sprachmodellen hinzugekommen. Diese Arbeit führt rekurrente Long-Short-Term-Memory-Neuronale-Netze in die Sprachmodellierung ein und stellt einen detaillierten Vergleich zwischen anderen Neuronale-Netze-Ansätzen sowie den häufigkeitsbasierten Sprachmodellen vor. Wegen der hohen Rechenanforderungen werden neuronale Netze mit verschiedenen Beschleunigungstechniken kombiniert. Außerdem werden in dieser Dissertation neue Approximationen zur direkten Minimierung der Wortfehlerrate auf Wortgraphen vorgestellt.

Darüber hinaus werden Long-Short-Term-Memory-Neuronale-Netze-Sprachmodelle verallgemeinert auf das Problem der Übersetzungsmodellierung aus der statistischen maschinellen Übersetzung. Zwei neue Methoden werden eingeführt, die die Reihenfolge der Wortabhängigkeiten in Quell- und Zielsprache beibehalten. Beide Ansätze lassen sich in aktuelle Übersetzungssysteme integrieren und ergänzen damit die Verbesserungen, die mit Neuronale-Netze-Sprachmodellen erzielt werden können.

Acknowledgement

At this point, I would like to express my gratitude to all the people who supported and accompanied me during the progress of this work. In particular, I would like to thank the following (inherently incomplete) list of people:

Prof. Dr.-Ing. Hermann Ney for giving me the opportunity to work on this very interesting research topic. This thesis would have not been possible without his continuous interest, advice and support.

Prof. Dr. Francisco Casacuberta for kindly agreeing to review this thesis.

Dr. rer. nat. Ralf Schlüter for many helpful discussions, as well as proofreading my research publications, sometimes on short notice.

Joern Wuebker, Kazuki Irie and Tamer Alkhouli for proofreading my thesis.

My former office mates Christian Plahl and Simon Wiesler, and later also Albert Zeyer, for the great time, during work and also beyond.

All current and former members of the speech team at the Chair of Computer Science 6: Albert Zeyer, Amr Ibrahim El-Desoky Mousa, Björn Hoffmeister, Christian Gollan, Christian Oberdörfer, Christian Plahl, David Nolden, David Rybach, Georg Heigold, Jonas Lööf, Kazuki Irie, Mahaboob Ali Basha Shaik, Markus Nußbaum-Thom, Muhammad Tahir, Pavel Golik, Stefan Hahn, Simon Wiesler and Zoltán Tüske. It was great working with you.

My colleagues from the statistical machine translation team who I also had the pleasure to work with during the course of my studies: Andreas Guta, Arnaud Dagnelies, Arne Mauser, Carmen Heger, Christoph Schmidt, Daniel Stein, David Vilar Torres, Evgeny Matusov, Gregor Leusch, Jan-Thorsten Peter, Jia Xu, Joern Wuebker, Maja Popović, Malte Nuhn, Markus Freitag, Matthias Huck, Minwei Feng, Patrick Lehnen, Saab Mansour, Saša Hasan, Stephan Peitz, Tamer Alkhouli and Yuqi Zhang.

Geoffrey Zweig and the whole team at Microsoft Research for an exciting internship and many interesting and insightful discussions.

Jean-Luc Gauvain, Lori Lamel and Ilya Oparin for the warm welcome at LIMSI CNRS during my research stay.

Finally, I would especially like to thank my brother and my parents for their support throughout my studies.

This work was partly realized as part of the Quaero programme, funded by OSEO, French State agency for innovation. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreements Nos. 287658 (EU-Bridge) and 287755 (transLectures). Experiments were partly performed with computing resources granted by JARA-HPC from RWTH Aachen University under project 'jara0085'.

Contents

Chapter 1

Introduction

Speech is one of the most natural and efficient means of human communication. The *automatic* understanding and evaluation of speech input requires the extraction of the textual information from the acoustic speech signal. This process is denoted automatic speech recognition (ASR).

The technology of ASR has numerous applications. First, it is ASR that facilitates convenient interaction with mobile devices like cell phones, watches or glasses, which often are too small for attaching fully-fledged input devices like a keyboard. On the other hand, the use of ASR technology is not limited to human computer interaction. Today, large archives of audio and video data are available. ASR technology allows the automatic transcription of these repositories. Given their textual representation, it is then possible to efficiently search and analyze the contents of the archives. And even beyond multimedia archives, there are many day-to-day scenarios where ASR technology can be helpful, e.g., when converting voicemail messages into written text and sending it directly to its recipient. Finally, ASR systems can be combined with other natural language processing related technology like machine translation. This can be used for a translation device enabling humans to communicate interactively with each other, even though they may not be able to speak a common language.

It has proven surprisingly difficult to teach an automatic system the extraction of written text from audio data by providing a set of general rules, like letter-sound analogies. By contrast, today's state-of-the-art ASR systems rely on a statistical approach. Here, a large amount of audio data including the corresponding textual transcription are provided to a system, which is intended to learn the underlying transform function itself, based on statistical models and mathematical concepts. More precisely, two main statistical models are used for speech recognition, namely the acoustic model and the language model. The former is related to estimating the probability of acoustic observations, whereas the latter operates on word information only, estimating the prior probability that a particular word sequence occurs in a given language. In such a way, a speech recognizer can resolve ambiguities of the acoustic signal. In this thesis, we will investigate how language models can be improved such that the probabilities of word sequences can be estimated more accurately, thereby considerably increasing ASR accuracy. In addition, we will transfer successful language modeling techniques to the case of statistical machine translation, leading to significant improvements in translation performance.

1.1 Statistical Speech Recognition

According to Bayes' decision rule [Bayes 63], a statistical speech recognizer aims for maximizing the posterior probability of a word sequence $w_1^N = w_1, \ldots, w_N$ given a sequence of acoustic observation vectors $x_1^T = x_1, \ldots, x_T$:

$$\{w_1^N\}_{\text{opt}} = \arg\max_{w_1^N} \{p(w_1^N | x_1^T)\} \tag{1.1}$$

$$= \arg\max_{w_1^N} \{p(x_1^T | w_1^N) \cdot p(w_1^N)\}, \tag{1.2}$$

where maximizing the posterior probability $p(w_1^N | x_1^T)$ is equivalent to maximizing the joint probability $p(w_1^N, x_1^T)$.

Eq. (1.2) shows how the joint probability $p(w_1^N, x_1^T)$ can be decomposed into the two aforementioned probabilistic models: The quantity $p(x_1^T | w_1^N)$ is referred to as the acoustic model (AM), and the quantity $p(w_1^N)$ is referred to as the language model (LM). Besides these two models, a statistical speech recognition system consists of the following four components, the interaction of which is depicted in Fig. 1.1 from [Ney 90]:

1. The *signal analysis* component (Section 1.1.1) analyzes the acoustic signal and creates feature vectors x_t for consecutive portions of the speech signal.

2. The *acoustic model* (Section 1.1.2) deals with probabilistic modeling of the acoustic observations for subword units, and from these subword units, models for entire words and word sequences are built.

3. The *language model* (Section 1.1.3) estimates the probability of arbitrary word sequences over a finite, pre-defined word vocabulary for a given language.

4. The *search* component (Section 1.1.4) tries to find the most-likely word sequence given the probabilistic models as well as the acoustic observation vectors, as extracted by the signal analysis component.

In the subsequent sections, these components will be explained in more detail.

1.1.1 Signal Analysis and Feature Extraction

The main goal of the signal analysis component of an ASR system is to extract feature vectors from the audio signal that compress the data to a minimum, containing only the information relevant to recognize the word sequence represented by the signal. Any other information like e.g. loudness, background noise, or speaker characteristics should be ideally discarded at this stage. State-of-the-art acoustic features rely on the coefficients of a fast Fourier transform (FFT) of the discretized audio signal [Rabiner & Schafer 78]. Using these coefficients, commonly either the so-called mel frequency cepstral coefficients (MFCC) [David & Mermelstein 80] or perceptual linear prediction based features (PLP) [Hermansky 90] are extracted. Both procedures incorporate knowledge how humans perceive speech. Furthermore, it is possible to enrich MFCC or PLP features by additional information that can be extracted

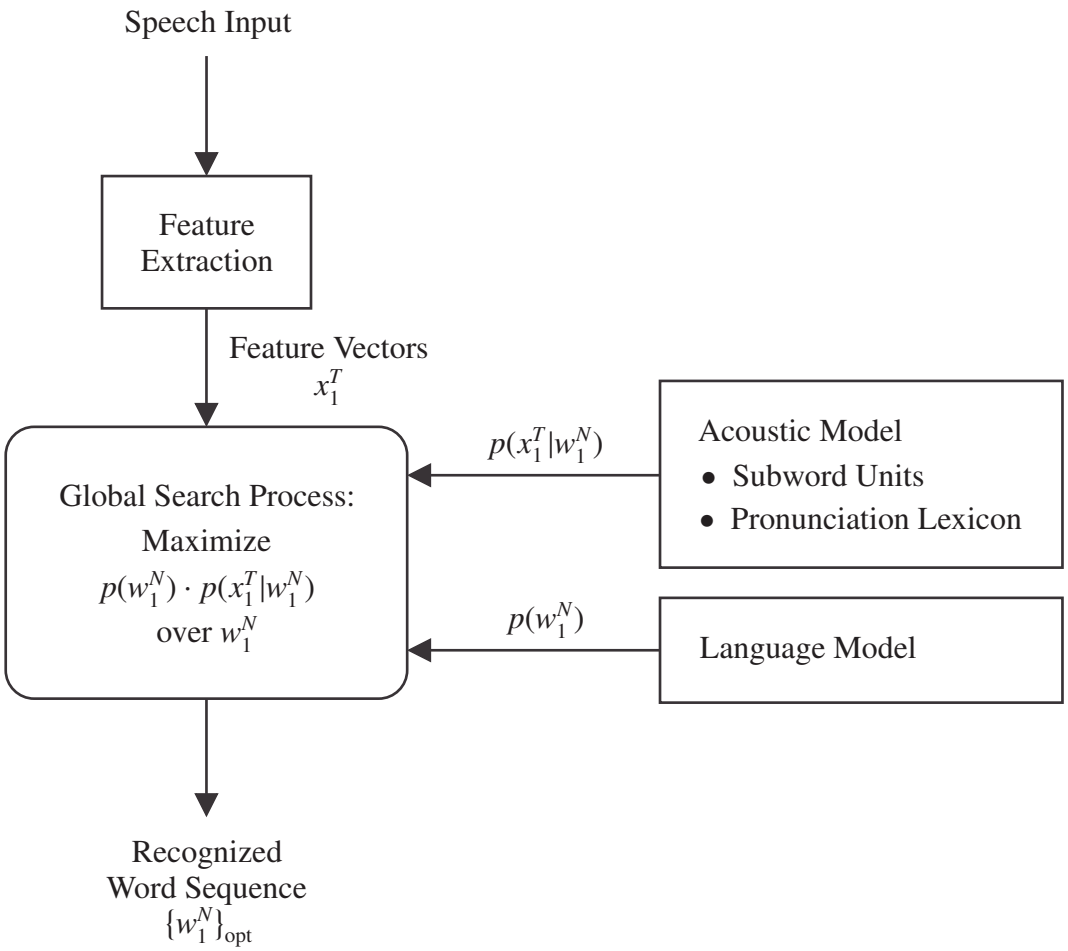

Figure 1.1 Components of a statistical speech recognition system.

e.g. from an artificial neural network modeling phone (or even sub-phone unit) posterior probabilities. This is referred to as the TANDEM approach [Hermansky & Ellis$^+$ 00].

In particular the short-term features (i.e., MFCC and PLP features) do not cover well the temporal dependencies of the acoustic signal. This can be remedied by appending first and second order derivatives to the vectors. As an alternative, a sliding window of MFCC or PLP features can be considered, concatenating consecutive vectors. As the dimensionality of the resulting feature vectors can become very high, a technique for reducing the dimensionality is needed. Usually, linear discriminative analysis (LDA) is used [Fisher 36]. Here, a linear transform is applied to the feature vector such that the variability between classes is maximized, where classes represent phonemes (or sub-phoneme units).

Recently, it was observed that when using state-of-the-art acoustic models, it is possible to train a competitive ASR system based on the raw acoustic signal only [Tüske & Golik$^+$ 14]. However, it was also found that the acoustic models needed to be considerably more complex compared to the case where a standard signal processing pipeline was applied. In addition, manually designed features still led to slightly better ASR performance. Therefore, conventional feature extraction is still an important component of today's ASR systems.

Normalizing speaker-dependent information usually strongly improves speech recognition accuracy. However, in most cases, the steps included in MFCC or PLP feature extrac-

tion are not sufficient to eliminate speaker-specific variability, and an additional processing is required. Often, vocal tract length normalization is applied to reduce the effect of gender-specific speaker characteristics [Eide & Gish 96]. Here, the speech signal is divided into segments containing only a single speaker. Even more effective is the class of speaker adaptation methods, where for a particular speaker, all the parameters of the acoustic model are adapted individually. Equivalently, for certain speaker adaptation techniques it is possible to apply a transform to the acoustic features instead [Gales 98], leaving the model parameters unchanged. An overview about speaker adaptation methods is given in [Pitz 05].

1.1.2 Acoustic Modeling

The acoustic model $p(x_1^T|w_1^N)$ estimates the probability of observing the sequence of acoustic observations x_1^T given the word sequence w_1^T. Commonly, the word sequence is split up into smaller, sub-word level units for which the probability of observing acoustic feature vectors can be estimated more easily. By concatenation of the sub-word units, an acoustic model for the complete word sequence is obtained.

In principle, it would be possible to use a modeling approach where larger units, like whole words, are used. However, in that case, the number of training observations for the individual word models would vary strongly, which would make such an approach less reliable for most data scenarios. Instead, words are broken up into their phonological units, so-called phonemes. It has been found that the pronunciation of phonemes differs depending on the previous or next phonemes. Therefore, for speech recognition often allophones are used, which are context-dependent phonemes. In most cases, a context dependence of one phoneme is sufficient, and the corresponding allophone is denoted a triphone, as it corresponds to a tuple consisting of the previous, the current and the next phoneme. It is also helpful to extend future context to the phoneme of the next word, which is referred to as across-word modeling [Hon & Lee 91, Odell & Valtchev$^+$ 94].

Strictly speaking, not triphones but generalized triphones are used in ASR: The number of triphones can be quite large, and some of them may not even occur in the training data. Therefore, triphones are clustered using a classification and regression tree (CART) based on phonetically motivated questions [Young 92].

Triphones are modeled based on Hidden Markov Models (HMMs) [Baker 75,Rabiner 89]. The main motivation of using HMMs for speech recognition is to be able to adapt to different speaking rates by aligning the acoustic feature vectors to a dynamically stretching reference model. The HMM approach relies on the notion of abstract states, which are causal for the observation of the values of a random variable, i.e., the acoustic vectors associated with a given triphone. As the states are not directly observable but only the acoustic observations that are regulated by the states, the states are called ‘hidden’. Mathematically, HMM states are introduced by extending the acoustic model in the following way

$$p(x_1^T|w_1^N) = \sum_{s_1^T} p(x_1^T, s_1^T|w_1^N), \tag{1.3}$$

where the summation is carried out over all HMM state sequences s_1^T that match the given word sequence w_1^N. According to Bayes' rule, this probability can be factorized as

$$p(x_1^T|w_1^N) = \sum_{s_1^T} \prod_{t=1}^{T} p(x_t|x_1^{t-1}, s_1^t, w_1^N) \cdot p(s_t|x_1^{t-1}, s_1^{t-1}, w_1^N). \tag{1.4}$$

Given a first-order Markov assumption [Duda & Hart$^+$ 01], Eq. (1.4) can be simplified to

$$p(x_1^T|w_1^N) = \sum_{s_1^T} \prod_{t=1}^{T} p(x_t|s_t, w_1^N) \cdot p(s_t|s_{t-1}, w_1^N). \tag{1.5}$$

In HMM terminology, the probability $p(x_t|s_t, w_1^N)$ is denoted the emission probability, and $p(s_t|s_{t-1}, w_1^N)$ is the transition probability. In other words, the assumption is that the emission probability only depends on the current state s_t, and the transition probability only depends on the previous state s_{t-1}. In Eq. (1.5), the full sum is carried out over all state sequences s_1^T. Instead, one can also approximate the sum by considering its largest summand only:

$$p(x_1^T|w_1^N) \approx \max_{s_1^T} \left\{ \prod_{t=1}^{T} p(x_t|s_t, w_1^N) \cdot p(s_t|s_{t-1}, w_1^N) \right\}. \tag{1.6}$$

The maximum approximation in Eq. (1.6) is also referred to as the Viterbi approximation. Both Eqs. (1.5) and (1.6) can be computed efficiently [Baum 72, Rabiner & Juang 86] using dynamic programming principles [Bellman 57, Viterbi 67, Ney 84].

An example HMM architecture is depicted in Fig. 1.2. Here, a triphone is modeled by three HMM states with state repetitions, which amounts to six HMM states in total, but every two states share a common emission and transition probability. This architecture was proposed in [Bakis 76]. Possible transitions are a loop transition from a state to itself, a forward transition from one state to its successor state, and a skip transition from a state to the next but one successor state. As a feature vector is commonly extracted every 10 ms time interval, the HMM architecture defines a maximum speed at which speech can be recognized. Depending on the speech data, the HMM can be adapted to account for faster speaking rates.

The transition probability from Eq. (1.5) is often modeled by a simple lookup table. The emission probability can be modeled by Gaussian mixtures

$$p(x|s, w_1^N) = \sum_{\ell=1}^{L_s} c_{s\ell} \mathcal{N}(x|\mu_{s\ell}, \Sigma, w_1^N), \tag{1.7}$$

where $\mathcal{N}(\mu, \Sigma)$ denotes a Gaussian density with mean μ and covariance matrix Σ. Throughout this thesis, we use a single globally pooled diagonal covariance matrix Σ for the Gaussian mixtures. By L_s we denote the number of mean vectors used for a given HMM state. For the mixture weights $c_{s\ell}$ we have $\sum_{\ell=1}^{L_s} c_{s\ell} = 1$. The parameters of the Gaussian mixtures are trained with the EM algorithm, maximizing the likelihood of the training data [Dempster & Laird$^+$ 77]. In a second step, this model can be refined based on a discriminative training

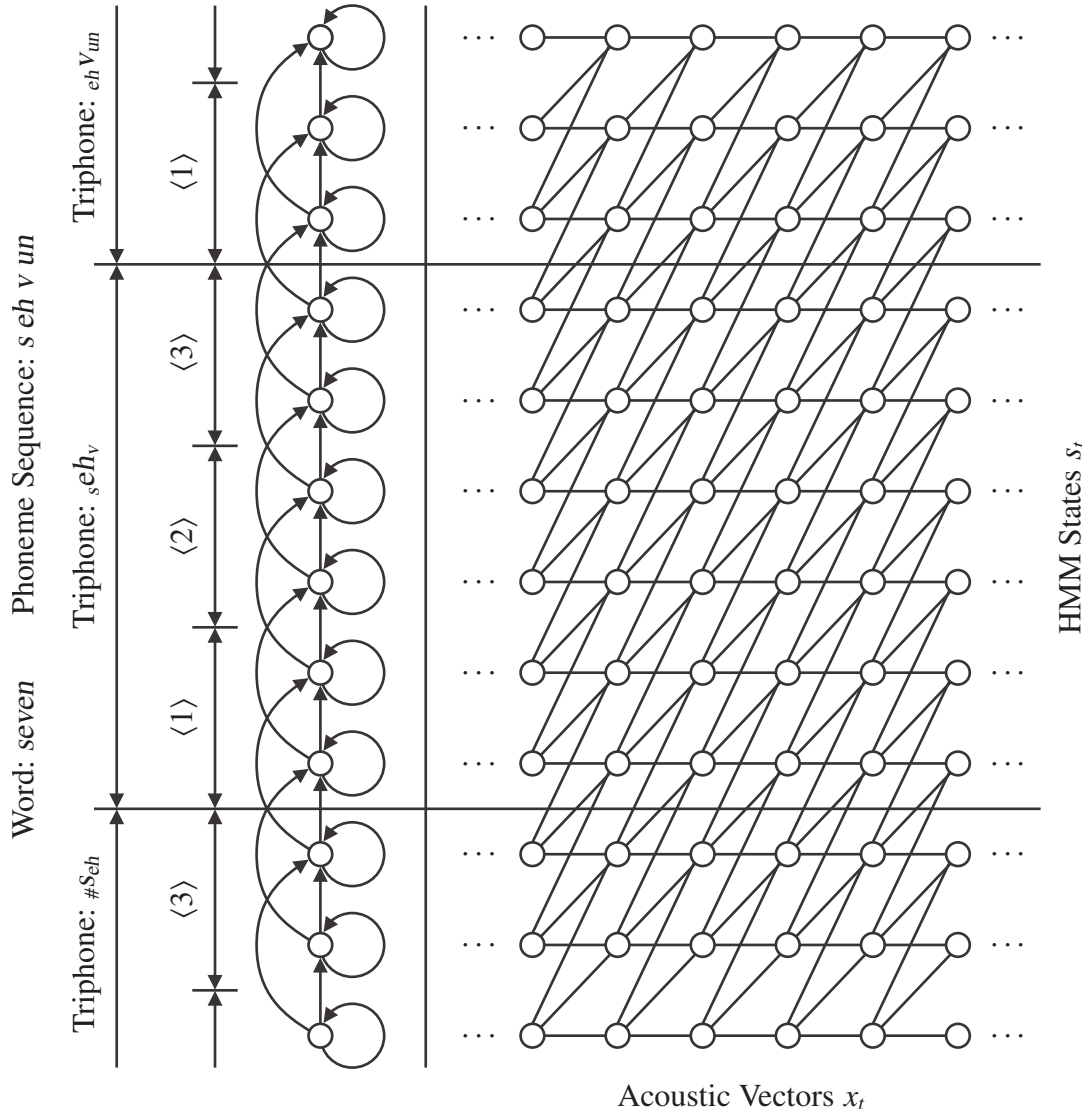

Figure 1.2 Example HMM architecture for the word 'seven'. The labels ⟨1⟩, ⟨2⟩, ⟨3⟩ denote the beginning, middle, and end part of the triphones each modeled by two tied HMM states.

criterion [Bahl & Brown$^+$ 86, Bahl & Padmanabhan$^+$ 96, Schlüter & Macherey$^+$ 01, Povey & Woodland 02].

As an alternative to Gaussian mixtures, a neural network can be used to estimate HMM state posterior probabilities [Bourlard & Morgan 93], which can then be converted to a form equivalent to the emission probability in a simple manner.

1.1.3 Language Modeling

The language model incorporates syntactic and semantic information into an ASR system by providing a prior probability for any recognition hypothesis. The dominating approach for estimating the probability $p(w_1^N)$ is to factorize it based on an m-gram approach as described

in [Bahl & Jelinek$^+$ 83]:

$$p(w_1^N) = \prod_{n=1}^{N} p(w_n|w_1^{n-1}) \tag{1.8}$$

$$= \prod_{n=1}^{N} p(w_n|w_{n-m+1}^{n-1}). \tag{1.9}$$

This means that the distribution of a word sequence is assumed to follow an $(m-1)$-th order Markov chain.

Based on this assumption, it can be shown that when estimating the m-gram probabilities from Eq. (1.9) as relative frequencies, the likelihood of the training data is maximized [Ney & Martin$^+$ 97]. However, this means that probability mass is only reserved for m-grams observed in the training data, and all other m-grams will receive zero probability mass. To be able to recognize unseen word sequences, it is necessary to attribute non-zero probability estimates to any m-gram event.

As a remedy, smoothing techniques have been developed to shift some small amount of probability mass from seen events to unseen ones [Jelinek & Mercer 80, Katz 87, Ney & Essen$^+$ 94, Kneser & Ney 95, Chen & Goodman 99]. The probability of unseen m-gram events can be estimated by generalizing the m-gram event to an $(m-1)$-gram. Two cases, namely backing-off and interpolation, are distinguished.

In case of a backing-off model, the m-gram probability is estimated according to the following general scheme:

$$p(w_n|w_{n-m+1}^{n-1}) = \begin{cases} \alpha(w_{n-m+1}^{n}) & N(w_{n-m+1}^{n}) > 0 \\ \tilde{\gamma}(w_{n-m+1}^{n-1})\beta(w_{n-m+2}^{n}) & \text{otherwise.} \end{cases} \tag{1.10}$$

If the m-gram was observed in the training data, some estimate $\alpha(w_{n-m+1}^{n})$ is used which depends on the full m-gram w_{n-m+1}^{n}. Otherwise, the left-most word of the m-gram is dropped, and a lower-order estimate $\beta(w_{n-m+2}^{n})$ is obtained, which is also multiplied by a normalization term $\tilde{\gamma}(w_{n-m+1}^{n-1})$. In the case of an interpolated model, the two branches of the backing-off case distinction are interpolated:

$$p(w_n|w_{n-m+1}^{n-1}) = \alpha(w_{n-m+1}^{n}) + \gamma(w_{n-m+1}^{n-1})\beta(w_{n-m+2}^{n}). \tag{1.11}$$

The same higher order estimate α and lower order estimate β can be used, in combination with a different normalization term γ. To obtain left-over probability mass that can be shifted between seen and unseen events, absolute discounting can be applied [Ney & Essen$^+$ 94]. In practice it is often found that interpolated models outperform its backing-off counterparts. However, it should be noted that an interpolated LM following Eq. (1.11) can always be converted to a backing-off format as in Eq. (1.10) [Chen & Goodman 99]. Such a form is more convenient to evaluate and thus used in nearly all practical applications.

1.1.4 Search

The search component of an ASR system aims at solving the optimization problem

$$\{w_1^N\}_{\text{opt}} = \arg\max_{w_1^N}\{p(w_1^N|x_1^T)\} \tag{1.12}$$

$$= \arg\max_{w_1^N}\{p(x_1^T|w_1^N)\cdot p(w_1^N)\}, \tag{1.13}$$

given the acoustic model, the language model, and the acoustic observation vectors. In the case of an HMM model, this can be re-written as

$$\{w_1^N\}_{\text{opt}} = \arg\max_{w_1^N}\left\{\left(\prod_{n=1}^{N} p(w_n|w_{n-m+1}^{n-1})\right)\cdot\left(\sum_{s_1^T}\prod_{t=1}^{T} p(x_t|s_t,w_1^N)\cdot p(s_t|s_{t-1},w_1^N)\right)\right\} \tag{1.14}$$

$$\overset{\text{Viterbi}}{=} \arg\max_{w_1^N}\left\{\left(\prod_{n=1}^{N} p(w_n|w_{n-m+1}^{n-1})\right)\cdot\left(\max_{s_1^T}\prod_{t=1}^{T} p(x_t|s_t,w_1^N)\cdot p(s_t|s_{t-1},w_1^N)\right)\right\}, \tag{1.15}$$

where in the second line the Viterbi approximation was applied. The search problem from Eq. (1.15) can be solved efficiently using dynamic programming [Bellman 57]. Many different search strategies have been applied to this particular search problem, and a good overview of the methods can be found in [Aubert 02]. There it is also concluded that, according to experimental evidence, rather different search algorithms can lead to similar performance given fixed computational resources.

To address the ASR search problem, it is possible to use a depth-first algorithm, of which the A* algorithm is an example, but also stack decoders operate according to the same principle. The idea is to follow the most promising path, expanding the search graph in a time-asynchronous manner, while maintaining an estimate of the probability of the remaining part of the word sequence [Jelinek 76, Paul 91]. In this way, it can be decided which of the hypotheses should be expanded first.

As an alternative, a breadth-first like search can be adopted [Vintsyuk 71, Baker 75, Sakoe 79, Ney 84]. Here the speech signal is evaluated in a time-synchronous way, in the sense that all competing hypotheses reflect the same portion of the acoustic signal. This facilitates efficient pruning techniques, as hypotheses can be compared more easily than in the case of a time-asynchronous, depth-first search expansion.

Pruning is especially important in the presence of large vocabularies and long-range LMs, when it is no longer possible to perform an exhaustive search where each hypothesis from the search space is examined during the optimization process. Then the goal of pruning is to obtain a reasonably good approximation of the most likely word sequence at small computational costs. A pruning technique that is widely used in the context of time-synchronous expansion is beam search. Here, hypotheses are pruned that are less likely than the current best hypothesis, multiplied by a fixed pre-defined factor [Lowerre 76, Ney & Mergel+ 87, Ortmanns & Ney 95]. Unlike for A* search, there is no guarantee that beam search will find the optimum word sequence. However, usually the number of search errors is small, so that the non-optimality of the search algorithm is not relevant in practice.

State-of-the-art search methods rely on many refinements of the basic algorithm. The pronunciations of the lexicon are usually organized as a prefix tree [Ney & Häb-Umbach+

92]. In this way, the acoustic model corresponding to prefixes that are shared by multiple words are only evaluated once. In addition, look-ahead techniques can be applied to incorporate future probability values into the pruning process as early as possible [Steinbiss & Ney$^+$ 93, Häb-Umbach & Ney 94, Odell & Valtchev$^+$ 94, Alleva & Huang$^+$ 96, Ortmanns & Ney$^+$ 96]. Research has also concentrated on fast likelihood computation, which can significantly speed-up search, as a large fraction of the overall search effort amounts to the evaluation of the acoustic model [Ramasubramansian & Paliwal 92, Fritsch 97, Bocchieri 93, Ortmanns & Ney$^+$ 97, Ortmanns 98, Kanthak & Schütz$^+$ 00].

Finally, to obtain best results, multiple decoding runs are performed over the speech signal [Evermann & Chan$^+$ 04, Sundermeyer & Nußbaum-Thom$^+$ 11]. On the one hand, a second pass allows making use of techniques like speaker adaptation, which itself requires a decoding result for computing the adaptation transform. This decoding result can be obtained in a first decoding pass where no speaker adaptation is applied. On the other hand, in an additional pass it is also possible to apply more complex models that are only applicable to a restricted search space in the form of n-best lists or word lattices. The hypotheses are reranked according to the better model, and the best hypothesis is used as the final result of the search process. Such an approach is investigated in more detail in Chapter 5.

1.1.5 Evaluation Measures

Statistical ASR systems require means to evaluate their quality. This is usually done by measuring the word error rate. In a first step, the Levenshtein distance between the hypothesis of the speech recognizer and the correct word sequence is computed [Levenshtein 66]. Afterwards, this quantity is normalized by the number of reference words.

A word error rate evaluation requires a full recognition pass on the acoustic signal, which is computationally expensive. In addition, only the combined effect of all involved statistical models can be measured. Regarding the LM, however, it is possible to evaluate the model itself, based on the so-called *perplexity*. Given a corpus of held-out data w_1^N that has not been used for training the LM, the perplexity PPL is defined as

$$\mathrm{PPL}(w_1^N) = p(w_1^N)^{-\frac{1}{N}} \tag{1.16}$$

$$= \left(\prod_{n=1}^{N} p(w_n | w_{n-m+1}^{n-1}) \right)^{-\frac{1}{N}}, \tag{1.17}$$

where Eq. (1.17) assumes an m-gram decomposition. The logarithm of the perplexity

$$\log \mathrm{PPL}(w_1^N) = -\frac{1}{N} \sum_{n=1}^{N} \log p(w_n | w_{n-m+1}^{n-1}) \tag{1.18}$$

is equivalent to the entropy of the model. Usually, the base 2 logarithm is used for computing the entropy. The perplexity can be interpreted as the number of words that the LM has to choose from on average at each word position. E.g., in the case of a uniform distribution (zerogram) the perplexity is equal to the vocabulary size.

1.2 State of the Art in Count-Based Language Modeling

Word-based LMs that rely on counts to estimate word posterior probabilities will be explored in more detail in Chapter 3, and they will serve as the baseline for all scientific work presented in subsequent chapters. Therefore, their estimation is described in more detail in this section.

1.2.1 Count-Based Language Models

Standard approaches for count-based language modeling rely for both higher order estimates α and lower order estimates β, cf. Eq. (1.10), on the raw relative frequencies [Katz 87]. This assumption can be problematic in certain cases. E.g., in many text corpora, the word *Francisco* occurs rather frequently, but it is only observed as the successor word of *San*, denoting the Californian city. By contrast, the unigram estimate $\beta(\textit{Francisco})$ will only be considered for cases where the predecessor word is not *San*. Obviously, the bigram probability $p(w, \textit{Francisco})$ is thus over-estimated for all words $w \neq \textit{San}$. Similar observations can be made for other words like *dollars*, that almost always follows numbers and country names.

This design flaw was addressed in Kneser-Ney smoothing [Kneser & Ney 95, Chen & Goodman 99]. To this end, the so-called marginal constraints were introduced, which, in the case of a unigram LM, are given by

$$p(w_n) = \sum_{w_{n-1}} p(w_n, w_{n-1}), \tag{1.19}$$

and in the case of a bigram LM can be generalized to

$$p(w_n|w_{n-1}) = \sum_{w_{n-2}} p(w_n, w_{n-2}|w_{n-1}). \tag{1.20}$$

While these conditions should be fulfilled naturally by any well-defined probability distribution, the crucial insight is that this is not the case for conventional backing-off estimates relying on relative frequencies directly. E.g., for the *San Francisco* example the summation on the right-hand side of Eq. (1.19) consists of many terms that are actually lower-order estimates for history words $w_{n-1} \neq \textit{San}$, thus over-estimating the marginal $p(\textit{Francisco})$.

In [Kneser & Ney 95] it was proposed to estimate lower-order probabilities such that the marginal constraints are fulfilled. This led to a backing-off distribution based on *modified* counts N^+, which is given below for the case of a trigram LM. For notational convenience, we let $w = w_n$, $v = w_{n-1}$, and $u = w_{n-2}$.

$$p(w|uv) = \frac{\max\{N(uv,w) - b_3, 0\}}{N(uv,\cdot)} + b_3 \frac{N_+(uv,\cdot)}{N(uv,\cdot)} p(w|uv) \tag{1.21}$$

$$p(w|v) = \frac{\max\{N^+(\cdot v,w) - b_2, 0\}}{N^+(\cdot v,\cdot)} + b_2 \frac{N_+^+(\cdot v,\cdot)}{N^+(\cdot v,\cdot)} p(w) \tag{1.22}$$

$$p(w) = \frac{\max\{N^+(\cdot,w) - b_1, 0\}}{N^+(\cdot,\cdot)} + b_1 \frac{N_+^+(\cdot,\cdot)}{N^+(\cdot,\cdot)} \frac{1}{W}. \tag{1.23}$$

The three parameters b_1, b_2, and b_3 are called discounts, and their estimation is covered in Chapter 3. Equations (1.21) to (1.23) make use of the following count definitions, all of which can be derived solely from the trigram counts $N(uv, w)$ found in the training data.

$$N(uv, \cdot) = \sum_{w} N(uv, w) \tag{1.24}$$

$$N_+(uv, \cdot) = \sum_{w:N(uv,w)>0} 1 \tag{1.25}$$

$$N^+(\cdot v, \cdot) = \sum_{w} N^+(\cdot v, w) \qquad N^+(\cdot v, w) = \sum_{u:N(uv,w)>0} 1 \tag{1.26}$$

$$N^+_+(\cdot v, \cdot) = \sum_{w:N^+(\cdot v,w)>0} 1 \tag{1.27}$$

$$N^+(\cdot, \cdot) = \sum_{w} N^+(\cdot, w) \qquad N^+(\cdot, w) = \sum_{v:N(v,w)>0} 1 \tag{1.28}$$

$$N^+_+(\cdot, \cdot) = \sum_{w:N^+(\cdot,w)>0} 1. \tag{1.29}$$

Probabilities for m-grams involving the beginning-of-sentence token are always estimated based on counts N instead of the modified counts N^+, as they cannot be reached via the backing-off branch.

In rare cases it can happen that, for a given sequence of history words uv, the above formulas lead to estimates that preserve nearly all of the probability mass for the highest order, such that the backing-off branch becomes extremely unlikely. In that cases, it can help to switch to backing-off instead of interpolation for the particular history uv only.

1.2.2 Estimation of Word Classes

Word clustering techniques are a means of grouping semantically similar words into classes, and they find application in language modeling and many related natural language processing tasks. Similar to count-based language models, the estimation of word classes relies on relative frequencies as well. The basic idea is to introduce a mapping $G : \mathcal{V} \to \mathcal{G}$ from the vocabulary $\mathcal{V}$ to the set of word classes $\mathcal{G}$, where only the number of word classes $k = |\mathcal{G}|$ is specified in advance. The counts of a word class $g \in \mathcal{G}$ can be defined analogously to the counts of words:

$$N(g) = \sum_{w:G(w)=g} N(w). \tag{1.30}$$

Word classes are commonly trained using a bigram assumption, i.e., the following modeling dependence is assumed:

$$p(w_n|w_{n-1}) = p_G(w_n|w_{n-1}) \tag{1.31}$$

$$= p(w_n|G(w_{n-1})) \cdot p(G(w_n)|G(w_{n-1})). \tag{1.32}$$

The first factor in Eq. (1.32) is usually denoted the *(class) membership probability*, and the second factor is simply a bigram LM over word classes.

Input: training corpus w_1^N, number of word classes k
Output: word class mapping G
repeat
  Create initial mapping G;
  foreach *vocabulary word* $w \in \mathcal{V}$ **do**
    foreach *word class* g **do**
      Tentatively exchange word w from class $G(w)$ to g;
      Update corresponding counts;
      Compute perplexity for tentative change;
    end
    Exchange word w from $G(w)$ to g with best perplexity;
  end
until *stopping criterion met*;
return G;

Figure 1.3 Pseudocode for the exchange algorithm.

Based on these assumptions, a maximum likelihood training criterion can be formulated

$$F(G) = -\sum_{n=1}^{N} \log\Big(p(w_n|G(w_{n-1})) \cdot p(G(w_n)|G(w_{n-1}))\Big) \tag{1.33}$$

given a training corpus w_1^N. When minimizing $F(G)$, which is equivalent to minimizing the perplexity of the training data, one can directly estimate the corresponding probabilities based on relative frequencies, i.e., no smoothing is required:

$$\arg\min_G F(G) = \arg\min_G \sum_{n=1}^{N} \log\left(\frac{N(w_n)}{N(G(w_n))} \cdot \frac{N(G(w_{n-1}), G(w_n))}{N(G(w_{n-1}))}\right) \tag{1.34}$$

$$= \arg\min_G \sum_{g_1,g_2} N(g_1,g_2) \cdot \log\left(\frac{N(g_1,g_2)}{N(g_1)N(g_2)}\right). \tag{1.35}$$

Unfortunately, all known algorithms for solving the optimization problem that is given by Eq. (1.35) involve the repeated evaluation of the training criterion $F(G)$ for a large number of word class mappings G, keeping track of the current best value of the optimization criterion. The so-called exchange algorithm, first introduced in [Kneser & Ney 91], has proven effective in finding good word classes, but other variants using bottom-up clustering exist as well [Brown & Della Pietra$^+$ 92]. The exchange algorithm is summarized in Fig. 1.3.

Regarding an initial word class mapping, a simple method is to put the most frequent words each into their own class, and reserve another class for all remaining words. For a stopping criterion, it was found empirically that running the algorithm for ten to twenty iterations almost always leads to good results, and further iterations do not add significant improvements. A more reliable stopping criterion would involve the training perplexity, stopping the training when no considerable reduction in training perplexity is obtained. In all cases, whenever two subsequent iterations do not lead to an exchange of a word from one

class to another, the algorithm can be terminated as well. The algorithm depicted in Fig. 1.3 can be accelerated considerably by a refined implementation [Martin & Ney 98]. Nevertheless, the computational effort for computing word classes can be quite high, depending on the vocabulary size, the number of word classes, and the number of running words in the training corpus.

Chapter 2

Scientific Goals

Automatic speech recognition systems are based on models that separately estimate the probability of acoustic observations and the probability of word sequences. The latter model, the LM, is not specific to speech recognition, as it does not take into account the acoustic signal. Nevertheless, the quality of the LM has a strong impact on the recognition performance of a speech recognizer.

The goal of this thesis is to investigate and improve the state of the art in language modeling. Furthermore, language modeling approaches are generalized to the case of statistical machine translation. In the following, we a give a more detailed overview about the scientific goals covered in this thesis.

Count-Based Language Modeling and Discounting Count-based LMs can be trained on huge amounts of data at small computational costs, providing probability estimates that are superior to many other modeling approaches. For count-based LMs, the Kneser-Ney method is usually considered the best smoothing algorithm, which relies on free parameters called discounts. In most scientific work, these parameters are not determined in a consistent way, therefore the true potential of these models may not fully be exploited. In this thesis we address the question which impact these parameters can have on perplexity and word error rate performance when large amounts of training data are available, and which computational costs are involved in the optimization of discount parameters. Furthermore, we investigate the effect of discounting on LM pruning.

Neural Network Language Models For language modeling, neural networks have become a powerful alternative to count-based approaches. In previous work, mainly feedforward and recurrent neural network (RNN) architectures have been investigated, where in particular recurrent networks have obtained promising results. On the other hand, it is well-known that RNNs are difficult to train, and addressing this problem might lead to improvements. One solution lies in the so-called recurrent long short-term memory (LSTM) neural network architecture that has been successfully applied in handwriting recognition, and will be investigated in this thesis. However, it should be noted that language modeling is a conceptually different problem due to the discrete nature of the textual training data. In addition, feedforward networks reportedly outperform LSTM networks on certain tasks, so that it is not clear whether LSTMs can obtain good results for language modeling, too. Fi-

nally, the LSTM paradigm also needs to be scaled to the large amounts of data commonly used for LM training.

Another scientific goal addressed in this thesis relates to the combination of count-based and neural network based LMs in the training stage. In this way, performance may be improved over the case of conventional linear interpolation of the two models, where both LMs are trained separately from each other.

Rescoring with Neural Network Language Models While neural networks show considerable improvements in language modeling, their application to speech recognition is difficult because of the high computational complexity. We address the question how long-range context neural network LMs can be applied to the rescoring of word lattices efficiently. In particular, we investigate whether a lattice-based search space can offer improved performance, or simple n-best list rescoring is sufficient.

An important aspect of this question lies in the decision rule being applied for rescoring. Previous work has only addressed the minimization of the sentence error rate, and our goal is to minimize the word error rate directly when rescoring with a neural network LM. This requires an approximate representation of the search space, including neural network LM probability estimates for any path encoded in a word lattice.

Finally, we also aim at establishing upper bounds on the potential improvements that can be obtained with a neural network LM, when no pruning errors are made and word lattice size is arbitrarily large.

Comparison of Count-Based and Neural Network Language Models Given the count-based and neural network based language modeling approaches, in this thesis we want to analyze how these methods compare in terms of perplexity, word error rate, and model size. As the training of neural network LMs depends on the use of speed-up techniques, a fair comparison of neural network LM architectures requires that all neural networks rely on the same speed-up techniques. In acoustic modeling, the concept of *deep* neural networks has shown large improvements over single-layer networks. In previous research on feedforward network LMs, this concept has not been reported to result in significant improvements. We want to verify whether long-range feedforward networks can benefit from deeper architectures, and we aim at extending the analysis to deep RNN and deep LSTM networks.

A comparative evaluation also includes providing a quantitative explanation why neural networks might improve over count-based LMs in certain scenarios. In addition, we want to identify cases where it might be more difficult to improve over the count LM baseline with a neural network. Besides, it is sometimes reported that with neural network LMs, perplexities and word error rates are no longer well-correlated, and in this thesis we intend to test this hypothesis.

Neural Network Translation Modeling Statistical machine translation systems can be improved by neural network LMs as well. However, a neural network LM can only model the probability of a translated sentence in the target language, whereas the dependence on the source language sentence is ignored. Therefore, in this thesis we want to generalize neural network LM approaches to the case of translation modeling.

In previous work, translation modeling has been successfully addressed using feedforward neural networks, but with RNNs, no notable gains have been obtained on top of neural network LMs. In previous RNN-based approaches, either word-derived vector representations were used as neural network input, or a bag-of-words model was applied. In this thesis, we want to develop novel recurrent translation models relying on word input and preserving word order at the same time. In this way, we aim at improving over a neural network LM baseline.

Chapter 3

Count-Based Language Modeling

Neural network-based approaches nowadays represent the state of the art in language modeling, and will be addressed in the following chapters. However, count-based LMs are still ahead in several regards. Besides their good performance in terms of perplexity and word error rate, they are essential for efficient decoding, and can be trained on large amounts of training data at small computational costs. Furthermore, other approaches like neural network LMs can be improved by interpolation with a count LM. Therefore, in this chapter count-based LMs will be investigated in more detail.

To obtain non-zero probability estimates, smoothing techniques are required. A variety of algorithms has been proposed, including Katz smoothing and Kneser-Ney smoothing [Katz 87, Kneser & Ney 95]. The family of Kneser-Ney smoothing variants often has been reported to give best performance [Kneser & Ney 95, Chen & Goodman 99]. Kneser-Ney smoothing relies on absolute discounting, where word posterior probabilities are estimated according to the following general form:

$$p(w_n|w_{n-m+1}^{n-1}) = \max\left\{\frac{N(w_{n-m+1}^{n}) - b}{N(w_{n-m+1}^{n-1})}, 0\right\} + \gamma(w_{n-m+1}^{n-1}) \cdot p(w_n|w_{n-m+2}^{n-1}). \tag{3.1}$$

Given an m-gram w_{n-m+1}^{n} of m consecutive words, a constant offset b is subtracted from the m-gram count $N(w_{n-m+1}^{n})$ found in the training data. This offset is denoted a *discount*. The probability $p(w_n|w_{n-m+1}^{n-1})$ is then estimated using relative frequencies, based on the discounted m-gram count, where higher-order probability estimates are interpolated with lower-order ones. The interpolation weight is denoted by γ and depends on the sequence of predecessor words w_{n-m+1}^{n-1}. The recursion in Eq. (3.1) terminates at the unigram level, interpolating unigram estimates with zerogram probabilities, cf. Section 1.2.1.

In [Kneser & Ney 95], an optimized lower-order distribution was introduced using so-called modified counts. Subsequently, [Ney & Martin$^+$ 97, Chen & Goodman 99] developed smoothing variants for multiple discounts. The general form of absolute discounting as in Eq. (3.1) was retained, but discount parameters were tied to the same value for a given m-gram count, instead of a single globally tied discount parameter.

For the discounting variants discussed in this chapter, approximation formulas have been derived to compute the values of the discount parameters. In addition, it is possible to compute the discount parameter values in an exact manner, optimizing perplexity on development data. This was first proposed in [Chen & Goodman 99], but discount parameters were only

optimized independently of each other using Powell's method [Powell 64]. At most three discount parameters per order were investigated. The experiments seemed to suggest that no improvements could be obtained by optimization when the amount of training data is large (cf. also [Goodman 01a, Siivola & Hirsimäki+ 07]). In addition, it is common practice not to optimize the discounts: Standard software implementations like [Stolcke 02, Stolcke & Zheng+ 11, Federico & Bertoldi+ 08] only offer the computation of discount parameter values based on conventional formulas. In this chapter, we address the following three questions:

1. What is the impact of using discount values which are optimized with respect to perplexity when large amounts of training data are available?
2. How can these values be computed efficiently?
3. What is the effect of optimized discounts on LM pruning?

Contrasting previous experiments, we also extend the number of discounts to much larger values. Based on [Sundermeyer & Schlüter+ 11], we run experiments on several corpora, reporting results in terms of perplexity and word error rate.

3.1 Estimation of Discount Parameters

The standard discount parameter formulas are derived in such a way that the leaving-one-out log-likelihood F of the training data

$$F(b) = \sum_{n=1}^{N} \log p_{\ell\text{oo}}(w_n | w_{n-m+1}^{n-1}; b) \tag{3.2}$$

is maximized. Here, $p_{\ell\text{oo}}(w_n | w_{n-m+1}^{n-1}; b)$ denotes the probability estimate of w_n given the history word sequence w_{n-m+1}^{n-1} leaving out one occurrence of the m-gram w_{n-m+1}^{n} from the training data [Ney & Martin+ 97]. In case of a single parameter b, this results in the approximate discount estimator

$$b = \frac{n_1}{n_1 + 2n_2}, \tag{3.3}$$

where n_r denotes the number of m-grams whose count (or modified count) is equal to r in the training data. When three discount parameters are used, an analogous derivation leads to the equations

$$b_1 = 1 - 2b\frac{n_2}{n_1} = b \tag{3.4}$$

$$b_2 = 2 - 3b\frac{n_3}{n_2} \tag{3.5}$$

$$b_3 = 3 - 4b\frac{n_4}{n_3}. \tag{3.6}$$

Here, the parameters b_1, b_2, and b_3 correspond to the m-gram events occurring with the counts $N(w_{n-m+1}^{n}) = 1$, $N(w_{n-m+1}^{n}) = 2$, and $N(w_{n-m+1}^{n}) \geq 3$, respectively. Using these formulas for computing the discounts may incur the following problems:

1. The above equations are (and can only be—no closed-form solutions exist) only approximations to the exact solutions.

2. Their derivations depend on backing-off smoothing. On the other hand, interpolated models as defined in Eq. (3.1) usually perform better. Nevertheless, discounts of interpolated models are computed using the backing-off discount formulas.

3. It is not clear how to set the discounts when a language model is pruned. In practice, the discounts are computed for the full model, and are not changed after pruning.

Besides, the derivation of the lower-order distribution in [Kneser & Ney 95] and [Chen & Goodman 99] relies on a single discount parameter. Using more than one parameter results in probability distributions that do not satisfy the marginal constraints [Siivola & Hirsimäki$^+$ 07]. On the other hand, increasing the number of discounts results in a more accurate modeling of the probability mass for unseen events, and in experiments, consistent improvements were obtained [Chen & Goodman 99]. For these reasons, we revisit the idea of optimizing discounts on development data. In principle, the optimization could also be carried out on the training data using leaving-one-out, but then the optimization process would be overly complex as the amount of training data usually exceeds the size of development data by several orders of magnitude [Andrés-Ferrer & Sundermeyer$^+$ 12]. Optimizing e.g. 10 discounts on the Quaero English development data, described in A.3.1, just takes about a minute on a standard CPU.

3.1.1 Interpolated Count Language Models

The interpolated Kneser-Ney smoothing method makes use of the same formula as absolute discounting for the estimation of $p(w_n|w_{n-m+1}^{n-1})$. The interpolation weight $\gamma(w_{n-m+1}^{n-1})$ is defined as

$$\gamma(w_{n-m+1}) = \frac{\sum_{k'=1}^{k} b_{k'} N_{k'}(w_{n-m+1}^{n-1})}{N(w_{n-m+1}^{n-1})}, \tag{3.7}$$

where k is the number of discounts, and

$$N_{k'}(w_{n-m+1}^{n-1}) = \begin{cases} \sum\limits_{\substack{1 \le n \le N \wedge \\ N(w_{n-m+1}^{n})=k'}} 1 & k' < k \\ \sum\limits_{\substack{1 \le n \le N \wedge \\ N(w_{n-m+1}^{n}) \ge k}} 1 & k' = k \end{cases} \tag{3.8}$$

is the number of words that have been observed k' times after the given history w_{n-m+1}^{n-1}.

The log-likelihood function F on some development data can then be defined as follows

$$F(b_1, \ldots, b_k) = \sum_{n=1}^{N} \log p(w_n|w_{n-m+1}^{n-1}; b_1, \ldots, b_k), \tag{3.9}$$

where the definition of $p(w_n|w_{n-m+1}^{n-1}; b_1, \ldots, b_k)$ is plugged in from Eq. (3.1). Unlike in Eq. (3.2), here by w_1^N we denote the word sequence representing the development data,

whereas Eq. (3.2) refers to the training data. Normally, individual discount parameters are used for each order, which results in a matrix $B \in \mathbb{R}^{m \times k}$ of discount parameters for an m-gram count LM.

We can now compute the optimum discount values by maximizing F. The corresponding optimization problem is not concave [Boyd & Vandenberghe 95, chapter 3]. To see this, we assume the case of a bigram LM with a single discount parameter, and a development data set of size $N = 1$ (the beginning-of-sentence symbol may serve as the bigram context w_0). Furthermore, we let $\tilde{F}(B) = \exp(F(B))$. Then, for $\tilde{F}$ we have

$$\tilde{F}(B) = \frac{N(w_0, w_1) - b_{21}}{N(w_0)} + \frac{N_1(w_0) b_{21}}{N(w_0)} \cdot \left(\frac{N^+(w_1) - b_{11}}{N^+(\cdot)} + \frac{N_1^+(\cdot) b_{11}}{N^+(\cdot)} \cdot \frac{1}{W} \right), \tag{3.10}$$

where W is the vocabulary size, and the superscript '+' denotes modified counts as defined in Section 1.2.1, which then are plugged into Eq. (3.8). After some calculations, for the eigenvalues $\lambda_{1,2}$ of the Hessian of $\tilde{F}(B)$ we obtain

$$\lambda_{1,2} = \pm N_1(w_0) \frac{W - N_1^+(\cdot)}{W N^+(\cdot) N(w_0)}. \tag{3.11}$$

Due to opposite signs of the eigenvalues, it follows that the Hessian matrix is not negative-semidefinite, and thus $\tilde{F}$ not concave. Finally, as $\tilde{F}$ is not concave, $\log \tilde{F} = F$ is neither. In fact, optimizing the functions $\tilde{F}$ and F is equivalent anyway, and thus showing that $\tilde{F}$ is not concave is already sufficient.

We optimize F using the improved Resilient Propagation algorithm (RPROP) including weight backtracking as described in [Igel & Hüsken 03], which was shown to give good performance when optimizing non-convex (or, in our case, non-concave) functions, and which is easy to implement. As RPROP is a first-order optimization method, we need to compute ∇F. For the discount k' of a given order m', we obtain

$$\frac{\partial F}{\partial b_{m'k'}} = \sum_{n=1}^{N} \frac{1}{p(w_n | w_{n-m+1}^{n-1})} \frac{\partial}{\partial b_{m'k'}} p(w_n | w_{n-m+1}^{n-1}), \tag{3.12}$$

and subsequently, for the highest order discounts, we have

$$\begin{aligned} \frac{\partial}{\partial b_{mk'}} p(w_n | w_{n-m+1}^{n-1}) = \frac{N_{k'}(w_{n-m+1}^{n-1})}{N(w_{m-n+1}^{n-1})} \cdot p(w_n | w_{n-m+2}^{n-1}) - \\ \begin{cases} \dfrac{1}{N(w_{n-m+1}^{n-1})}, & N(w_{n-m+1}^{n}) = k' \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3.13}$$

given that $N(w_{n-m+1}^{n-1}) > 0$. In case $N(w_{n-m+1}^{n-1}) = 0$, the derivative is zero. The derivatives with respect to lower order discounts are analogous to Eq. (3.13), where we also need to multiply by all higher order backing-off weights γ, and replace counts by modified counts.

3.1.2 Backing-Off Count Language Models

In principle, the same approach can be applied to backing-off smoothing. In this case, the probability estimate for $p(w_n|w_{n-m+1}^{n-1})$ takes on the form

$$p(w_n|w_{n-m+1}^{n-1}) = \begin{cases} \dfrac{N(w_{n-m+1}^{n}) - b}{N(w_{n-m+1}^{n-1})} & N(w_{n-m+1}^{n}) > 0 \\ \tilde{\gamma}(w_{n-m+1}^{n-1}) \cdot p(w_n|w_{n-m+2}^{n-1}) & \text{otherwise.} \end{cases} \tag{3.14}$$

The problem is that the backing-off weight

$$\tilde{\gamma}(w_{n-m+1}^{n-1}) = \frac{\gamma(w_{n-m+1}^{n-1})}{1 - \sum\limits_{w:N(w_{n-m+1}^{n-1},w)>0} p(w|w_{n-m+2}^{n-1})} \tag{3.15}$$

now also includes a normalization term which cannot be dropped in the derivative of lower-order discounts. As opposed to interpolated count LMs based on absolute discounting, the complexity of the target function F then not only depends on the size of the development data, as the summation ranges over all words from the vocabulary. For this reason, and because backing-off models usually perform worse than interpolated ones, we decided not to investigate the optimization of backing-off models in more detail.

3.2 Count Language Model Pruning

While count LMs can be trained on large corpora in a short amount of time, the resulting models can become very large in size as well. This is especially inconvenient for applications like decoding in speech recognition, where different LM probabilities are requested from the model frequently. To reduce the size of the model, in a first step a count LM can be trained on all available data, but then only a subset of the m-gram probabilities is retained in the final model. To this end, several pruning algorithms exist that can be used to remove m-grams from a count LM, and in this section, we will discuss two of the most widely used pruning techniques.

3.2.1 Count Cutoffs

A simple way of reducing the size of a count LM is to use count cutoffs: All m-grams seen less than a predefined threshold are discarded. As a result, the probability $p(w_n|w_{n-m+1}^{n-1})$ for a pruned m-gram w_{n-m+1}^{n} is modeled by the lower-order distribution. Usually, an order-specific count cutoff is used such that, in total, m parameters are needed for pruning.

Commonly, the discount parameters are estimated on the full model, and are kept constant after pruning. On the other hand, neither the formula-based estimates nor the optimized discount values are appropriate for the pruned model, because it is assumed that all m-grams are used for the LM estimation. For this reason, we include the pruning procedure in the discount optimization process. In the optimization framework, this can be easily

achieved in combination with count cutoffs by setting a discount in Eq. (3.1) to its maximum value. For example, when pruning m-grams with $N(w_{n-m+1}^{n}) = 1$, this corresponds to setting $b_1 = 1$. As a result, in interpolated Kneser-Ney smoothing only the lower-order estimate $\gamma(w_{n-m+1}^{n-1}) \cdot p(w_n|w_{n-m+2}^{n-1})$ is non-zero. All remaining discounts are optimized.

3.2.2 Stolcke Pruning

An alternative approach to LM pruning based on relative entropy was introduced in [Stolcke 98], and it is commonly referred to as entropy pruning or Stolcke pruning. In contrast to count-cutoffs, this method is self-contained, i.e., the pruning method does not rely on the count data but only on the LM itself. Let p be the probability distribution of an LM, and let p' be the distribution p where the explicit estimate $p(w_n|w_{n-m+1}^{n-1})$ has been removed. Stolcke pruning discards those m-grams for which

$$\exp(D(p\|p')) - 1 < \theta. \tag{3.16}$$

for a pre-defined threshold θ. Here, D denotes the Kullback-Leibler divergence

$$D(p\|p') = p(w_{n-m+1}^{n-1}) \sum_{w} p(w_n|w_{n-m+1}^{n-1}) \frac{\log p(w_n|w_{n-m+1}^{n-1})}{\log p'(w_n|w_{n-m+1}^{n-1})} \tag{3.17}$$

between the distributions p and p'. The left hand side of Eq. (3.16) is actually equivalent to the relative perplexity increase of the probability distribution resulting from pruning an m-gram

$$\frac{\mathrm{PPL}' - \mathrm{PPL}}{\mathrm{PPL}}, \tag{3.18}$$

where PPL and PPL$'$ are the perplexities of the probability distributions p and p', respectively (which differ from the *model* perplexities as defined in Section 1.1.5).

In [Siivola & Hirsimäki$^+$ 07] and [Chelba & Brants$^+$ 10] it was observed that this algorithm does not interact well with Kneser-Ney smoothing. In fact, Kneser-Ney smoothing introduces so-called marginal constraints, and the lower-order distributions are estimated in such a way that these constraints are fulfilled. The underlying idea is that highest-order m-gram probabilities should be estimated differently from lower order probabilities. More precisely, the estimation of a lower-order m-gram probability should take advantage of the knowledge that the higher order m-gram estimate was not seen in training. Subsequently, a lower-order Kneser-Ney estimate

$$\gamma(w_{n-m+1}^{n-1}) p(w_n|w_{n-m+2}^{n-1}) \tag{3.19}$$

rather models the probability

$$p(w_n|w_{n-m+1}^{n-1} \wedge N(w_n|w_{n-m+1}^{n-1}) = 0). \tag{3.20}$$

This is problematic because in Eq. (3.17), the joint probability $p(w_{n-m+1}^{n-1})$ is obtained by multiplying lower-order estimates from the pruned model, which, due to Eq. (3.20), results in a poor estimate of the joint probability. For this reason, in [Stolcke 10] a refinement was proposed: The quantity $p(h)$ is estimated by a Katz-smoothed LM, and we refer to the corresponding pruning method as improved Stolcke pruning.

3.3 Experimental Results

For the experimental analysis of the previously described algorithms, we concentrate on three corpora: The English Wall Street Journal (WSJ) corpus, the Quaero English and the Quaero French corpus. These are described in Appendices A.1, A.3.1, and A.3.2. By intention, in most cases we do not use linear interpolation of multiple LMs, as this is likely to mask improvements from optimizing discounts of individual LMs. Therefore, we only use a single domain for the estimation of all LMs. In the case of the WSJ data, we can make use of the standard corpus for training, and in the case of the Quaero data, we restrict to a single large training data source. The data source is chosen by relevance with respect to the development data, according to its interpolation weight in the multi-domain LM. For the Quaero English task, in this way a corpus of 508 M running words was chosen, and for Quaero French, the single-domain corpus comprised 334 M running words. Both corpora were created by collecting text data from blogs.

The implementation of the discount optimization was tested by verifying that the gradient and the difference quotient give the same result, up to small a constant value.

3.3.1 Discount Optimization

Table 3.1 summarizes perplexities for various smoothing techniques and different types of discounting. For all three corpora, the LM order was increased until no significant improvements were obtained. Thus, for the WSJ corpus a 5-gram LM was used, whereas for both the Quaero corpora a 4-gram LM was sufficient. The first half of the table shows results for standard Katz smoothing and Kneser-Ney smoothing, where backing-off and interpolated versions are distinguished. In the case of Kneser-Ney smoothing, discounts are estimated using Eq. (3.3) or, in the case of three parameters, Eqs. (3.4) to (3.6). For the second half of the Table, discount parameters optimized with respect to development perplexity are used, extending the number of parameters up to 100 per order. We also give perplexity values for the development and the test data to see whether discount optimization leads to overfitting on the development data. However, this does not seem to be the case, as the relative improvements closely match on both data sets, even when optimizing a large number of discount parameters.

In general, optimizing discounts on development data as well as using more discounts helps for all LM corpora investigated here. While the improvements in perplexity are negligible for the WSJ task, they are comparatively large for the Quaero corpora. For French, the perplexity can be decreased from 159.3 to 154.3, and for English even from 200.3 to 188.7, just by optimizing three discount parameters instead of using conventional formulas. This indicates that the approximation formulas are by no means accurate predictors of discount parameter values. For all corpora, optimized discount values significantly differ from those predicted by the formulas. For highest order discounts, on the WSJ corpus we obtain an optimized value of $b_3 = 1.89$, as opposed to $b_3 = 1.58$ for the standard discount formula. For the English Quaero data, these differences are even larger (1.37 vs. 2.60). Our optimized discounts are in accordance with previous assumptions stated in [Andrés-Ferrer & Ney 09] concerning the monotonicity constraints $k' - b_{k'} \leq k' + 1 - b_{k'+1}$, which we usually find to be fulfilled, unless the number of discounts is large. On the other hand, our values contradict

Smoothing		k	NAB		Quaero EN		Quaero FR	
			Dev	Test	Dev	Test	Dev	Test
Katz	Backing-Off+ Formulas	–	80.0	87.0	222.5	218.5	155.2	174.9
Kneser-Ney		1	77.5	84.1	221.5	218.2	153.8	172.4
	Interpolated+ Formulas	1	70.2	76.1	217.5	212.6	145.6	165.0
		3	68.7	74.4	205.2	200.3	140.8	159.3
	Interpolated+ Optimized	1	69.6	75.5	206.6	201.9	142.0	160.3
		3	68.3	74.0	195.4	190.9	137.1	154.3
		10	68.3	**73.9**	193.2	188.7	136.1	153.0
		100	**68.2**	74.0	**191.2**	**188.2**	**134.6**	**152.8**

Table 3.1 Performance of various count LM smoothing techniques on three different LM corpora, with different numbers of discount parameters, denoted by k.

the interval constraints from [Andrés-Ferrer & Ney 09], requiring $0 < b_{k'} < 1$. Also, there is no direct relation between the potential improvements by discount optimization and the amount of training data. The Quaero English corpus is the largest corpus and also exhibits the largest relative gain from better discounting.

An interesting question arising from the data presented in Table 3.1 is why improved discounting does not help for the WSJ task. While there is some noticeable gain by going from one discount parameter to three discounts, neither extending the number of discounts to larger values nor the optimization of the parameters reduces perplexity in a considerable way. The answer simply lies in the design of Modified Kneser-Ney smoothing [Chen & Goodman 99, Fig. 11], the three-discount variant of Kneser-Ney smoothing. In that work, experiments were performed on the WSJ corpus as well, comparing expected counts to actual counts. From the experimental results it was concluded that three discounts would be needed for optimum performance, and discount estimation formulas were proposed for the newly added parameters. Therefore, our results in Tab. 3.1 confirm that three formula-based discounts are sufficient for modeling the WSJ domain, but this empirical observation does not generalize well to other corpora and domains.

The results in Table 3.1 were all obtained without any count cutoffs. When tuning count cutoffs, we find that small improvements in perplexity can be obtained for formula-based discounting (cf. Tab 3.2). By contrast, in our experiments optimized discounts always lead to best performance, and then count cutoffs only result in a perplexity increase. The fact that more data—e.g., in the form of singleton events—do not pay off in perplexity shows again that standard discount formulas do not give consistent results. In the table, count cutoffs are encoded as a tuple of four values indicating the smallest count of an m-gram that is retained in the final count LM, from left to right for lower to higher LM orders. Table 3.2 also includes results in terms of word error rate. Compared to Modified Kneser-Ney smoothing, it can be seen that a larger number of optimized discounts leads to small but consistent improvements in word error rate. It is noteworthy that the computationally cheap optimization of discount parameters consistently leads to word error rate improvements, where the underlying smoothing method is the same for all models.

It should be noted that the improvements obtained by discount optimization are not due to a domain adaptation effect, as could be suspected from the results on the Quaero data,

Discounting	k	Count Cut-Offs	Perplexity		Viterbi		Consensus	
			Dev	Test	Dev	Test	Dev	Test
Formulas	3	1-1-1-1	205.2	200.3	16.3	14.2	16.0	13.8
		1-1-1-2	200.0	195.2	16.3	14.1	16.0	13.9
Optimized	10	1-1-1-1	**193.2**	**188.7**	**16.1**	**13.9**	**15.9**	**13.7**

Table 3.2 Perplexity and word error rate results for two-pass decoding with count LMs using Kneser-Ney smoothing and different discounting variants on the Quaero English task.

where data are less homogeneous than in the case of the WSJ task. In [Andrés-Ferrer & Sundermeyer+ 12], the optimization of discount parameters on the training data was investigated, and subsequently it was found that both the optimization on training and development data basically lead to the same performance.

Finally, while there is some variation in how many discount parameters turn out to be helpful for a particular corpus, this does not imply that there is a need for tuning the number of discount parameters. This is depicted in Fig. 3.1. Given an m-gram count from the training data, the Figure shows the corresponding frequency of such m-grams in the development data. In addition, it depicts the fraction of m-grams from the development data that are served by training data m-grams that have at most the given training data count. E.g., in the case of the WSJ corpus, we can see that 37,671 of the m-grams from the development data are handled by probability estimates based on singleton events in the training data. Singleton estimates make up a fraction of 17.4% of all the m-grams in the development data. It turns out that these statistics are very similar for all three corpora investigated. This indicates that when optimizing a rather small number of discount parameters like 10 or 100, most of the development data m-grams are already covered. Using more discounts is unlikely to result in perplexity improvements, as the additional discounts are ignored in the optimization criterion from Eq. (3.9). While this could be remedied by increasing the amount of development data, the general tendency that low-count discounts make up the lion's share of the development data m-grams remains the same.

3.3.2 Language Model Pruning

Figure 3.2 shows results for LM pruning on the test data when Kneser-Ney smoothing with 10 discount parameters is used. For the full model, these have been optimized on the development data. We then either use the improved Stolcke pruning method or simple count cutoffs to obtain a smaller LM. For count cutoff pruning, discount parameters are reoptimized accordingly. It can be seen that in all cases, the count cutoff performance is very close to the entropy pruning method, and often, it is even slightly better than improved Stolcke pruning, although no sophisticated criterion for choosing the m-grams to be pruned is employed. Also, it is not necessary to train a second Katz-smoothed count LM for estimating joint probabilities. In the experiments from Fig. 3.2, the count LMs were reduced to about 10% of their original size. For comparison, we also include results for the case where the reduction in size is much stronger. This is depicted in Fig. 3.3, where count LMs are pruned to up to 0.17% of the original size. Stolcke pruning was not developed for Kneser-Ney smoothing, and combining the two techniques leads to poor performance. While the perplexity increase is only small

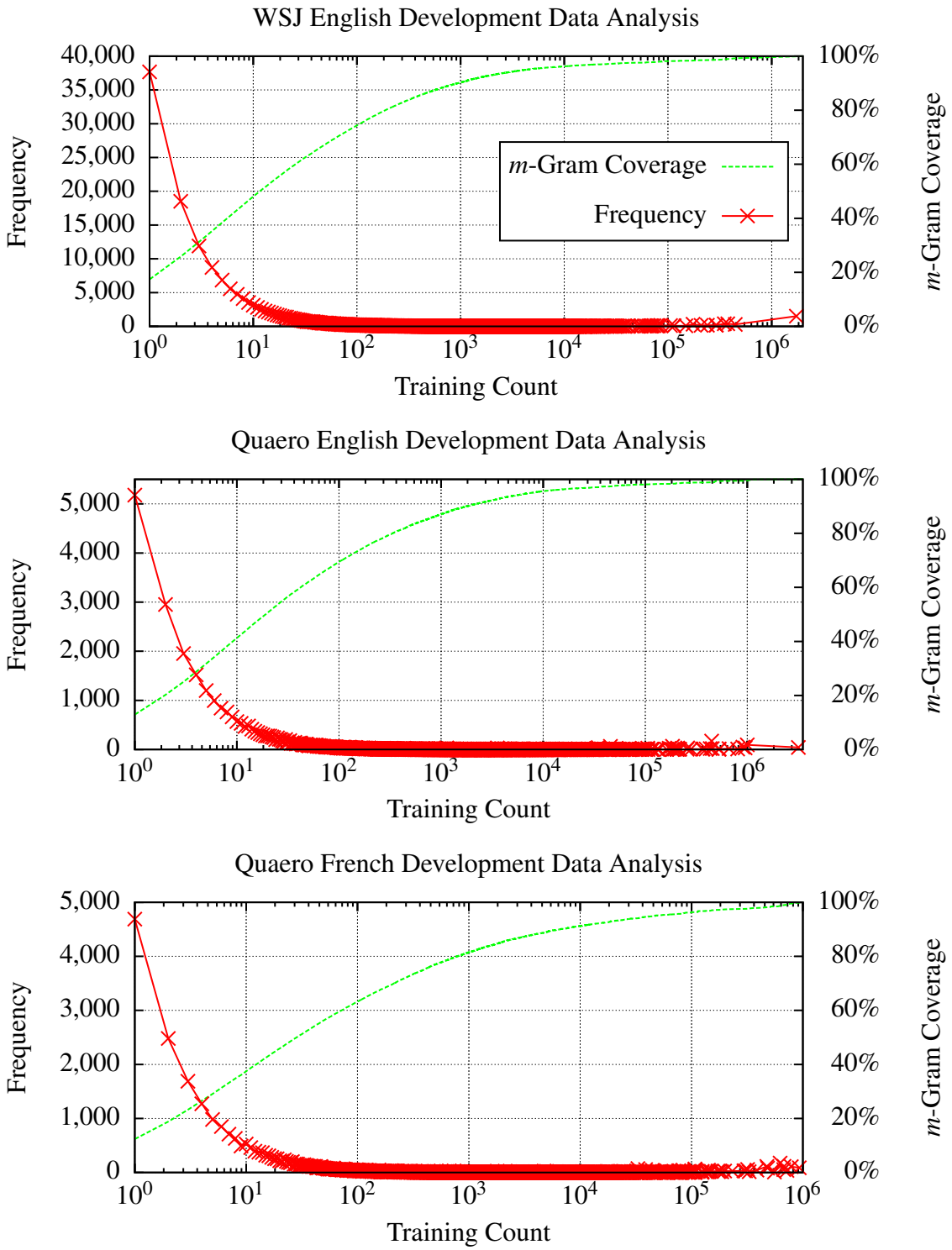

Figure 3.1 Frequencies of m-grams from the development data, sorted by their training data counts, and development data m-gram coverage up to a given training data count.

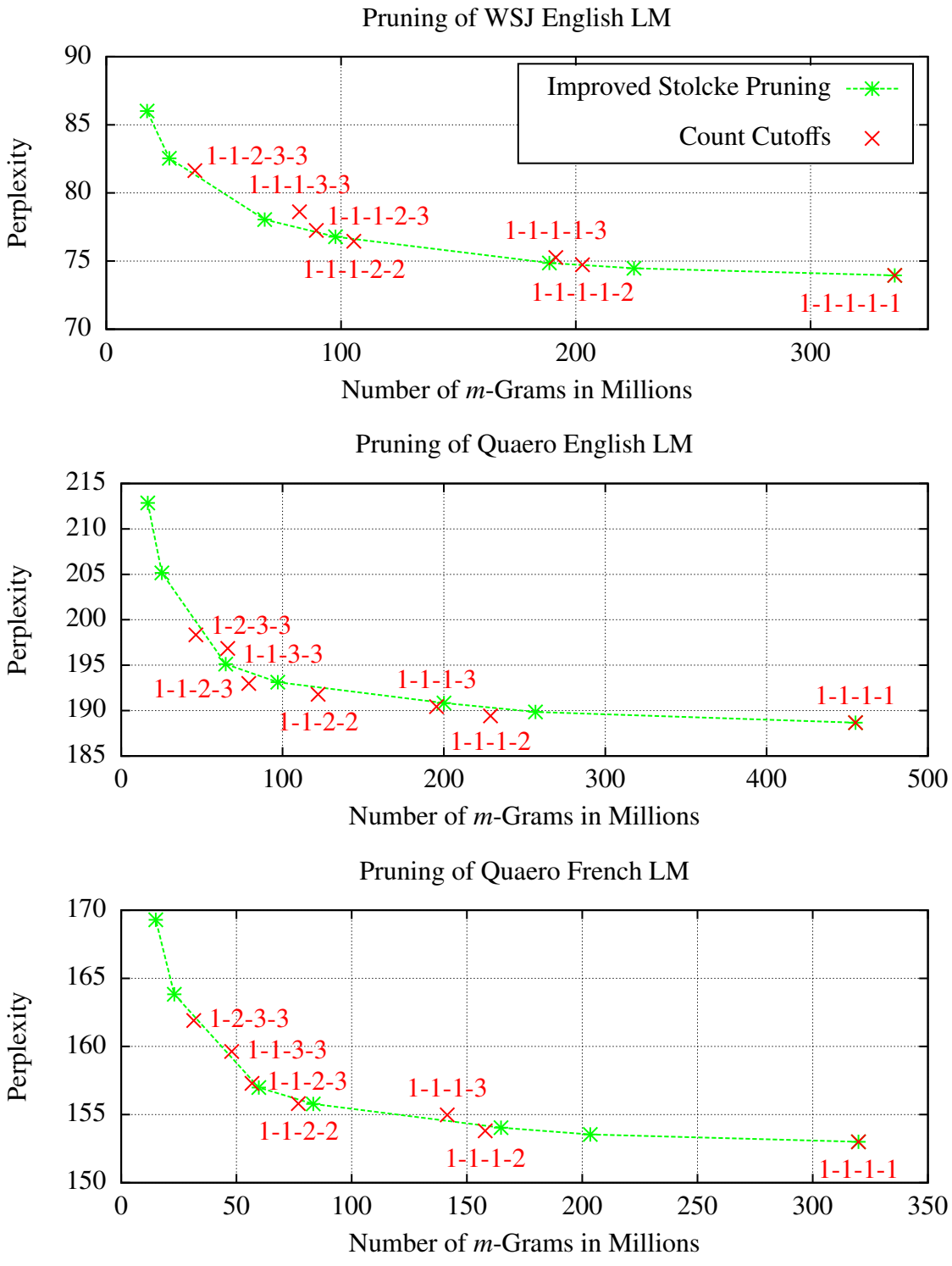

Figure 3.2 Count cutoffs vs. improved entropy pruning. Labels indicate the smallest count of an m-gram still included in the LM, from lowest to highest order (from left to right).

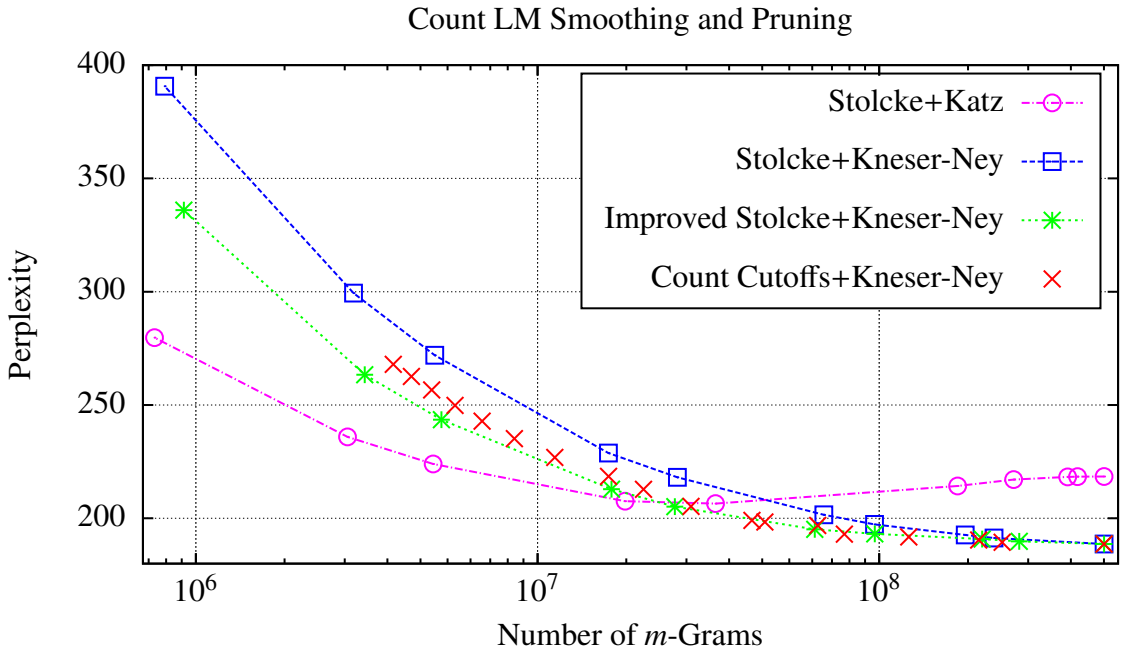

Figure 3.3 Pruning of a Katz and a Kneser-Ney-smoothed count LM with various techniques.

for a moderate pruning regime, the effect grows much larger when more m-grams are pruned. Therefore, the improved Stolcke pruning considerably outperforms standard Stolcke pruning, which indicates that the strong increase in perplexity by pruning is at least partly due to the inaccurate estimation of the joint history probabilities. The perplexity results of count cutoff pruning with discount optimization are always significantly better than the Stolcke pruning baseline, but they are slightly worse than the improved Stolcke method when it comes to large reductions in model size. Nevertheless, this shows that discount re-optimization has a significant impact on the performance of a pruned language model. Also, the count cutoff results might be improved by tuning the cutoff parameters: In the figure, cutoffs were tied to the same values for the fourgram, trigram, and bigram levels, and all unigrams were retained.

We confirm the finding that Katz smoothing in combination with Stolcke pruning performs best in case aggressive pruning is applied. However, it should be noted that the marginal constraints of Kneser-Ney smoothing are no longer fulfilled when such a model is pruned. Thus, strictly speaking, it is required to update all LM estimates for pruning, as suggested in [Kneser 96] and [Siivola & Hirsimäki$^+$ 07]. For extremal pruning scenarios, it still seams difficult to improve over Stolcke pruning of Katz-smoothed LMs, as [Siivola & Hirsimäki$^+$ 07] did not report improvements over this method even when recomputing the lower order distribution of a pruned Kneser-Ney model.

3.3.3 Language Model Interpolation

The previous experiments have shown the effectiveness of discount optimization, which serves well for count LM estimation as well as for LM pruning. On the other hand, we only analyzed the case where a single count LM is estimated. In many cases, data from

multiple domains are available. Then best performance is obtained when a separate count LM is trained on each of the data sources first, and another LM is estimated by linearly interpolating the individual LMs.

Unfortunately, it does not suffice to optimize discount parameters on each data source individually, as can be seen in Tab. 3.3. The perplexity of the resulting LM is rather increased

LM		k	Perplexity	
			Dev	Test
Kneser-Ney	Formulas	1	**132.7**	**131.2**
	Optimized	10	139.8	137.4

Table 3.3 Discounting of individual LMs and interpolation of multiple count LMs.

considerably when optimizing the discount parameters. Subsequently, even the formula-based Kneser-Ney smoothing with a single discount parameter outperforms the 10-discount optimized variant, which was one of the best configurations for single LMs in previous experiments. Therefore, for interpolated LMs, it seems necessary to investigate a revised optimization criterion which takes into account the discount parameters as well as the interpolation weights at the same time. However, as the formula-based Kneser-Ney smoothing with a single discount usually outperforms the three-discount variant when interpolating multiple LMs [Gauvain 11], it seems more difficult to obtain improvements by optimized discounting here. Consequently, in the following we will make use of optimized discounts for single-domain tasks (like Babel, cf. Appendix A.4), and in all other cases, we stick to standard interpolated Kneser-Ney smoothing with one formula-based discount parameter per order.

3.4 Summary

Optimizing discounts is a simple means of improving the performance of a Kneser-Ney-smoothed LM. We showed that the corresponding optimization problem for finding the discount values minimizing development perplexity is non-convex, and we investigated a first-order optimization approach with negligible computational costs. Even for large amounts of training data, discount optimization lead to consistent gains in perplexity as well as in word error rate. Regarding LM pruning, discount estimation in combination with count cutoffs was on par with the much more complicated improved Stolcke pruning in the case of moderate pruning. It still made up most of the difference between the original and the improved Stolcke pruning when reducing the number of m-grams to less than 1%. Also, there is virtually no need for tuning the number of discount parameters. From that point of view, we conclude it is preferable to always optimize the discounts on development data, instead of using conventional formulas.

The only exception seems to be when multiple LMs are interpolated, because then optimizing discounts of individual LMs degrades performance. An integrated optimization of discounting and interpolation would be required. However, as preliminary experiments did not suggest much potential for improvement, this approach was not investigated further.

Chapter 4

Neural Network Language Models

Count-based approaches have been very successful for language modeling. They can be trained at small computational costs, and in practice it is found difficult to obtain consistent improvements over a count LM baseline, especially when the amount of training data is large [Rosenfeld 00]. At the same time, count LMs do not exhibit a strong generalization capability, because words are only modeled as discrete events, without any notion of similarity between them. As a result, instead of attributing high probability values to semantically similar m-gram events, a count LM prefers events previously observed in the training data. Conventional solutions to this problem include the following approaches. By using word classes, semantically similar words are grouped into a single class, and a standard count LM is estimated over these classes instead of word identities [Kneser & Ney 91, Brown & Della Pietra$^+$ 92]. The resulting class-based LM can then be interpolated with a word-based LM. By allowing gaps in word histories of posterior probabilities, at least some of the conditioning word events can be retained if the full history was not seen in the training data, and a more robust probability estimation can be expected [Ney & Essen$^+$ 94, Martin & Hamacher$^+$ 99, Bilmes & Kirchhoff 03]. However, in [Goodman 01a] as well as in [Kramp 12, Botros 15] it was observed that only comparatively small improvements can be obtained when extending well-tuned standard count LMs by these techniques.

Neural networks have become increasingly popular for language modeling as they improve generalization on unseen data. First, several works had addressed language modeling-related problems using neural networks: In [Nakamura & Maruyama$^+$ 90], a neural network LM was trained on 89 word categories, reporting a considerable increase in speech recognition accuracy. In [Castaño & Vidal$^+$ 93], a recurrent neural network was used to estimate probabilities of a stochastic regular language, defined by a finite state automaton. To the best of our knowledge, the first successful application of a neural network to a real-world word-based language modeling task was demonstrated independently in [Xu & Rudnicky 00] as well as in [Bengio & Ducharme$^+$ 00] and [Bengio & Ducharme$^+$ 03].

In contrast to count-based methods, neural network LMs rely on an inherent continuous space representation of words, which is learned automatically as part of the training process. As shown in [Mikolov & Yih$^+$ 13] and [Mikolov & Sutskever$^+$ 13], the arrangement of these word vectors in a continuous space encodes their semantic relationship, which is exploited by the neural network architecture to robustly estimate probabilities of word sequences. In this chapter, we present feedforward and recurrent neural networks which have been used

for language modeling before. We outline conceptual problems of these approaches and investigate a third kind of neural network, the so-called recurrent long-short term memory neural network. We investigate speed-up techniques for neural network LMs, and we propose several refinements related to neural network training.

4.1 Feedforward Neural Networks

Neural networks were first introduced to the field of language modeling based on a feedforward architecture in [Bengio & Ducharme$^+$ 00]. Similar to count models, they rely on a Markov assumption, i.e., the probability of a word sequence w_1^N is decomposed as

$$p(w_1^N) = \prod_{n=1}^{N} p(w_n | w_{n-m+1}^{n-1}) \tag{4.1}$$

such that only the most recent $(m-1)$ preceding words are considered for predicting the current word w_n. An example of a trigram feedforward neural network LM is depicted in Fig. 4.1, which corresponds to the following set of equations:

$$y(n) = A_1 \hat{w}(n-2) \circ A_1 \hat{w}(n-1) \tag{4.2}$$
$$x_z(n) = A_2 y(n) \tag{4.3}$$
$$z(n) = \mathrm{sig}(x_z(n)) \tag{4.4}$$
$$x_p(n) = A_3 z(n) \tag{4.5}$$
$$p(w_n | w_{n-2}, w_{n-1}) = \varphi(x_p(n))\big|_{w_n} \tag{4.6}$$

Here, by A_1, A_2 and A_3 we denote the weight matrices of the neural network. We do not explicitly include a bias term in the above formulas, as this can be included in the weight matrix multiplication [Bishop 95, p. 118]. The input data $\hat{w}(n-2)$ and $\hat{w}(n-1)$ are the so-called one-hot encoded predecessor words w_{n-2} and w_{n-1}. One-hot encoding is defined as follows: For a given vocabulary $\mathcal{V}$, if $w \in \mathcal{V}$ is the j-th word of the vocabulary, we have

$$\hat{w} \in \{0,1\}^W \qquad \text{where} \qquad \hat{w}\big|_i = \begin{cases} 1 & i = j \\ 0 & \text{otherwise,} \end{cases} \tag{4.7}$$

and W denotes the vocabulary size $|\mathcal{V}|$. This encoding scheme of discrete input data for neural networks is also referred to as 1-of-W encoding [Bishop 95, p. 300]. The weight matrix A_1 is tied for all history words. The vectors $A_1 \hat{w}(n-2)$ and $A_1 \hat{w}(n-1)$ are then concatenated, indicated by the $\circ$ operator, to form the projection layer activation $y(n)$. Multiplying $y(n)$ with A_2, and applying the sigmoid activation function

$$\mathrm{sig}(x) = \frac{1}{1+\exp(-x)}, \tag{4.8}$$

which is computed element-wise for the vector $A_2 y(n)$, results in the hidden layer activation $z(n)$. By φ, we indicate the softmax activation function

$$\varphi(x) = \frac{\exp(x)}{\sum_{j=1}^{J} \exp(x_j)}, \tag{4.9}$$

which enforces normalization of the probability estimates, where J is the dimension of the vector x. Analogous to the sigmoid function sig, the softmax function φ is computed elementwise for the vector x as well.

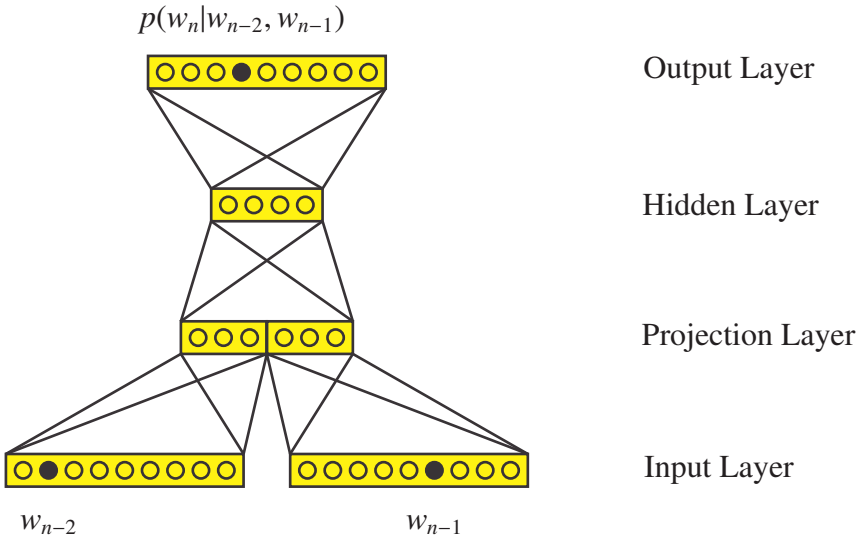

Figure 4.1 Trigram feedforward neural network architecture.

4.2 Standard Recurrent Neural Networks

Recurrent neural network LMs as introduced in [Mikolov & Karafiát+ 10] follow a very similar topology compared to feedforward neural networks. An example of a recurrent neural network (RNN) is shown in Fig. 4.2, where the RNN is defined by the following equations:

$$x_y(n) = A_1 \hat{w}(n-1) + A_2 y(n-1) \tag{4.10}$$

$$y(n) = \text{sig}(x_y(n)) \tag{4.11}$$

$$x_p(n) = A_3 y(n) \tag{4.12}$$

$$p(w_n|w_1^{n-1}) = \varphi(x_p(n))\big|_{w_n}. \tag{4.13}$$

As before, the weight matrices of the RNN are denoted by A_1, A_2, and A_3. In contrast to feedforward neural network LMs, there is a weight parameter matrix A_2 for the recurrent connections, which is multiplied by the previous hidden layer activation vector $y(n-1)$. At the very beginning of a word sequence, when $n = 1$, the vector $y(0)$ is commonly set to zero [Graves 12, p. 23] and [Sundermeyer & Schlüter+ 12], sometimes also set to one [Mikolov & Karafiát+ 10, Chen & Wang+ 14]. (In principle, it could be trained in the same way as the parameter matrices.) The RNN LM is only explicitly conditioned on the direct predecessor word, and in that way, it resembles a bigram feedforward neural network LM. On the other hand, by evaluating the RNN equations word by word for an entire sequence, the output layer result for a word w_n depends on the entire sequence of history

words w_1^{n-1}. The activations of the recurrent hidden layer then act as a memory that automatically stores previous word information, as can be seen by expanding the recursive RNN equations:

$$n = 1: \quad p(w_1) = \varphi\Big(A_2 \underbrace{\mathrm{sig}(A_1 \hat{w}(1))}_{y(1)}\Big)\Big|_{w_1} \tag{4.14}$$

$$n = 2: \quad p(w_2|w_1) = \varphi\Big(\underbrace{A_2\, \mathrm{sig}\Big(A_1 \hat{w}(2) + A_2 \underbrace{\mathrm{sig}(A_1 \hat{w}(1))}_{y(1)}\Big)}_{y(2)}\Big)\Big|_{w_2} \tag{4.15}$$

$$n = 3: \quad p(w_3|w_1^2) = \varphi\Big(\underbrace{A_2\, \mathrm{sig}\Big(A_1 \hat{w}(3) + A_2 \underbrace{\mathrm{sig}\Big(A_1 \hat{w}(2) + A_2 \underbrace{\mathrm{sig}(A_1 \hat{w}(1))}_{y(1)}\Big)}_{y(2)}\Big)}_{y(3)}\Big)\Big|_{w_3} \tag{4.16}$$

$$\vdots$$

Unlike feedforward neural networks, RNNs are thus operating on sequences instead of unrelated individual events. From a theoretic point of view, an RNN does not require approximations like the Markov assumption, and therefore the probability of a word sequence

$$p(w_1^N) = \prod_{n=1}^{N} p(w_n|w_1^{n-1}) \tag{4.17}$$

is decomposed in an exact manner. RNNs thereby represent a more accurate modeling approach than count-based LMs or feedforward neural networks.

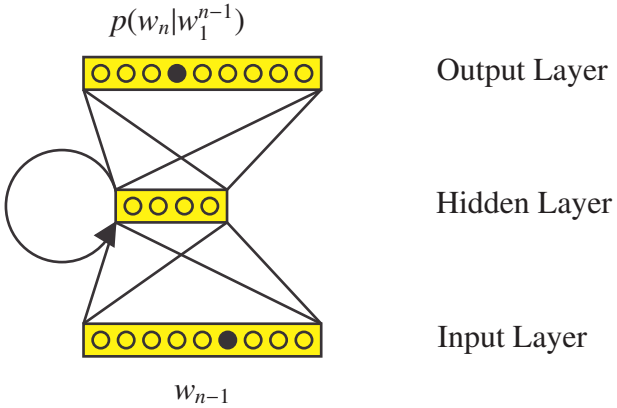

Figure 4.2 Recurrent neural network architecture.

4.3 Recurrent Long Short-Term Memory

In this section, we investigate several shortcomings of the standard RNN approach. We then outline potential solutions proposed in the neural network literature and discuss their

applicability to language modeling. Finally, we describe how one particular approach, the so-called recurrent long short-term memory neural network, can be effectively used for language modeling.

4.3.1 Vanishing and Exploding Gradient Problem

One of the main advantages of RNN LMs over feedforward neural network LMs is that no explicit dependence on a pre-defined context length has to be assumed, and, at least conceptually, it is possible to take advantage of long-range word dependences. While the RNN architecture itself facilitates the use of long-range history information, in [Bengio & Simard$^+$ 94, Hochreiter & Schmidhuber 97, Hochreiter & Bengio$^+$ 01] it was found that standard gradient-based training algorithms fall short of learning RNN weight parameters in such a way that long-range dependences can be exploited. This is due to the fact that, as dependences get longer, the gradient calculation becomes increasingly unstable. This can be seen by analyzing the error signal $\mathrm{err}(n)$ in more detail. When training an RNN, error signals are propagated back in time as described in Section 4.4.2. Given a fully connected RNN of J non-input nodes, we can compute the gradient of the error at time $(n-m)$ and node j with respect to a later time step n and a node j', for $m > 1$:

$$\frac{\partial \mathrm{err}_j(n-m)}{\partial \mathrm{err}_{j'}(n)} = \sum_{j_1=1}^{J} \cdots \sum_{j_{m-1}=1}^{J} \prod_{m'=1}^{m} \Big(\mathrm{sig}'(x_y(n-m')_{j_{m'}}) \alpha_{j_{m'-1} j_{m'}} \Big). \tag{4.18}$$

Here, we set $j_m = j$ and $j_0 = j'$. In Eq. (4.18), the summation iterates over all possible sequences of length m from RNN node j to node j'. In [Hochreiter & Schmidhuber 97, Hochreiter & Bengio$^+$ 01] the cases

$$\forall m' : \quad \left| \mathrm{sig}'(x_y(n-m')_{j_{m'}}) \alpha_{j_{m'-1} j_{m'}} \right| > 1 \tag{4.19}$$

and

$$\forall m' : \quad \left| \mathrm{sig}'(x_y(n-m')_{j_{m'}}) \alpha_{j_{m'-1} j_{m'}} \right| < 1 \tag{4.20}$$

are analyzed. It is argued that either the largest product of the summation blows up exponentially, or the smallest product decays exponentially, respectively. In the case of exploding gradients, backpropagation learning can lead to weight oscillations. In the case of vanishing gradients, no learning takes place at all. While this argument certainly does not take into account the summation over an exponentially growing number of paths, at least it can be seen that the error flow of an RNN is very sensitive to the length of the sequence m that is considered for the computation of the gradient. This phenomenon is commonly referred to as the vanishing (or exploding) gradient problem. In practice, it basically means that when training the weight parameters of an RNN with conventional backpropagation through time, only dependences of a few time steps can actually be learned—in contrast to the original motivation for using RNNs. Strictly speaking, the vanishing and exploding gradient problem is not limited to RNNs but can also occur in the case of feedforward networks. However, the depth of these networks is usually not large enough to observe this problem in practical applications.

4.3.2 Solutions

To avoid vanishing and exploding gradients, the following solutions have been addressed in the neural network language modeling literature.

1. The simplest solution to vanishing and exploding gradients probably lies in ignoring all dependences exceeding a certain context length. This can be achieved by truncating the gradient computation of the backpropagation through time algorithm after a few time steps [Williams & Zipser 95]. According to Eq. (4.18), the effect of exponential decay or growth of the error flow is then strongly limited. In addition, one can avoid exploding gradients by clipping errors, i.e., in the case that an error exceeds a predefined threshold, it is set to the threshold value [Sarle 02, Mikolov & Kombrink$^+$ 11b]. By combining both strategies for RNN training, state-of-the-art results in language modeling were obtained in [Mikolov & Karafiát$^+$ 10, Mikolov & Kombrink$^+$ 11a].

2. Based on the works [Mikolov & Karafiát$^+$ 10, Mikolov & Kombrink$^+$ 11a], it was proposed in [Mikolov & Zweig 12] to include long-range dependences in the short-term context of an RNN. To this end, a sequence of preceding words was converted into a vector representation using Latent Dirichlet Allocation [Blei & Ng$^+$ 03], which was then input to an RNN, in addition to the direct predecessor word. This resulted in improvements over a standard RNN.

3. In [Martens & Sutskever 11], an improved optimization algorithm was proposed for RNNs which makes use of higher-order information. Subsequently, this algorithm was applied to character language modeling in [Sutskever & Martens$^+$ 11]. However, despite the small character alphabet, the method required high computational efforts, which makes this approach less attractive for word-based language modeling, where much larger vocabularies are used.

4. Instead of improving the training algorithm, it is possible to modify the RNN architecture in such a way that the resulting optimization problem is more stable with respect to standard gradient-based training. This idea was proposed in [Hochreiter & Schmidhuber 97], where the recurrent long short-term memory (LSTM) neural network architecture was developed. The LSTM concept was first introduced to the field of language modeling in [Sundermeyer & Schlüter$^+$ 12], where it was shown that LSTMs can be efficiently integrated with existing neural network language modeling techniques, achieving considerable improvements over RNNs.

In an internal comparison with the authors of [Mikolov & Zweig 12] it was observed that an LSTM LM using word input alone can obtain an even slightly better perplexity than an RNN LM using word and contextual feature input. For this reason, we investigate the LSTM concept in more detail.

4.3.3 Recurrent Long Short-Term Memory Architecture

After the recurrent long short-term memory neural network was first introduced in [Hochreiter & Schmidhuber 97], the idea was subsequently extended in [Gers & Schmidhuber$^+$

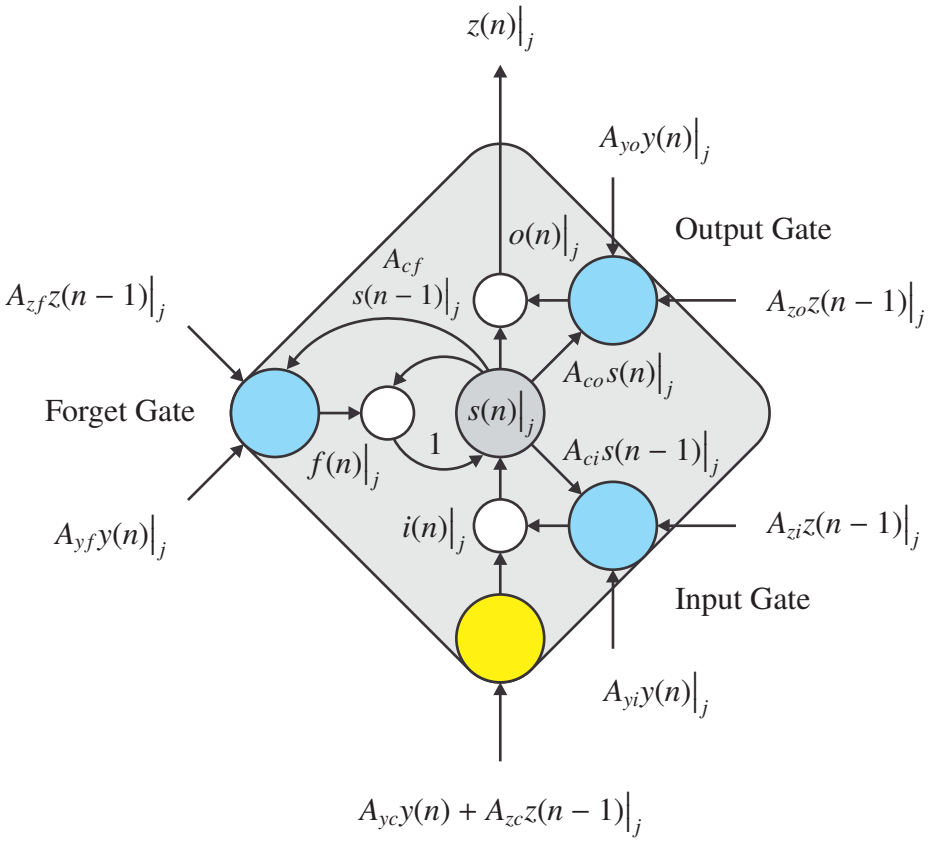

Figure 4.3 Graphical depiction of the LSTM equations for the j-th LSTM unit of a hidden layer.

00] and [Gers & Schraudolph$^+$ 02]. Today, what is commonly referred to as LSTM is the extended version including all these refinements.

In general, the performance of LSTMs strongly depends on the particular task. It was reported in [Schmidhuber 03] that LSTMs can be outperformed by simple feedforward neural networks on a time series learning problem. On the other hand, LSTMs have been very successful in the field of handwriting recognition [Graves & Liwicki$^+$ 09, Grosicki & Abed 09, Kozielski & Doetsch$^+$ 13]. For language modeling, in [Sundermeyer & Schlüter$^+$ 12] the LSTM architecture investigated is defined by the equations

$$y(n) = A_1\hat{w}(n-1) \tag{4.21}$$
$$i(n) = \mathrm{sig}(A_{yi}y(n) + A_{zi}z(n-1) + A_{ci} \odot s(n-1)) \tag{4.22}$$
$$f(n) = \mathrm{sig}(A_{yf}y(n) + A_{zf}z(n-1) + A_{cf} \odot s(n-1)) \tag{4.23}$$
$$c(n) = \tanh(A_{yc}y(n) + A_{zc}z(n-1)) \tag{4.24}$$
$$s(n) = f(n) \odot c(n-1) + i(n) \odot s(n) \tag{4.25}$$
$$o(n) = \mathrm{sig}(A_{yo}y(n) + A_{zo}z(n-1) + A_{co} \odot s(n)) \tag{4.26}$$
$$z(n) = o(n) \odot \tanh(s(n)) \tag{4.27}$$
$$p(w_n|w_1^{n-1}) = \varphi(A_2z(n))\big|_{w(n)}, \tag{4.28}$$

where A_1, A_2 as well as $A_{\{y,z,c\}\{i,f,o\}}$ and $A_{\{y,z\}c}$ are the weight matrices of the neural network. By $\odot$ we denote element-wise multiplication. The indices of the eleven LSTM weight matri-

ces in Eqs. (4.22) to (4.27), e.g., y and i in the case of A_{yi}, do not indicate a dependence on the values of a variable y or i, but are only meant to distinguish the individual matrices.

In Fig. 4.3, the equations for the j-th LSTM unit are depicted graphically. LSTMs address the vanishing and exploding gradient problem by striving for constant error flow. This is achieved by a recurrent connection with a fixed weight of one, as can be seen in Fig. 4.3. Obviously, an RNN with a recurrent connection that lacks trainable weight parameters is not suitable for real-world learning problems. Therefore, the error flow of the inner recurrency is controlled by parameterizable multiplicative units $i(n)$, $f(n)$, and $o(n)$. These are sometimes referred to as gating units, as their value lies in the $(0, 1)$ interval because of the application of the sigmoid activation function sig. An LSTM thereby exhibits two levels of recurrency, one inner level with constant error flow, and one outer level where the previous output $z(n-1)$ is used to control the gating units of time step n. Strictly speaking, since the introduction of the forget gate in [Gers & Schmidhuber$^+$ 00], the inner recurrence does not exhibit constant error flow anymore. To the best of our knowledge, there does not exist a formal proof showing that the LSTM architecture given by the above formulas is insensitive to vanishing and exploding gradients, especially when the gradient computation is not truncated. However, in experiments it is found that LSTMs indeed lead to a more robust training process compared to standard RNNs. This results from the fact that in training, the LSTM can learn gating unit parameters such that the gradient flow through the network does not explode or vanish. The overall architecture of the LSTM neural network LM is shown in Fig. 4.4. The LSTM formalism can be encapsulated as a hidden layer.

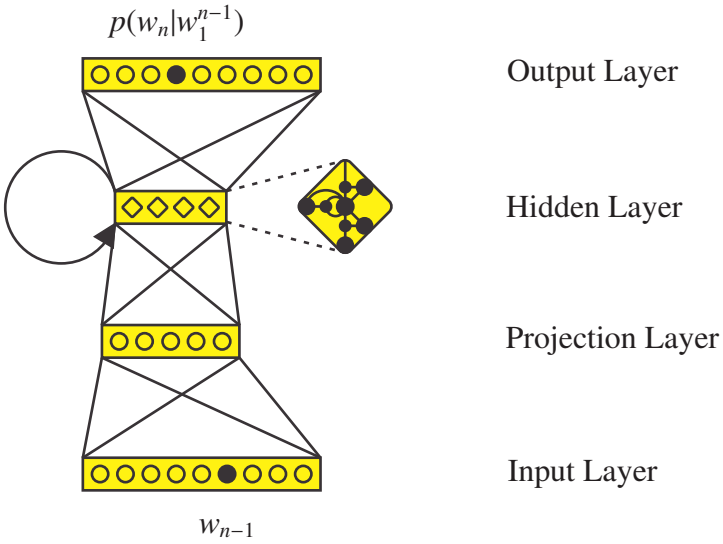

Figure 4.4 Recurrent long short-term memory neural network architecture.

4.4 Neural Network Training

When using a neural network for regression, e.g., when learning a probability distribution over classes, in most cases the so-called cross-entropy error criterion is used [Bishop 95,

p. 237]. In its most general form, this criterion assumes that the true posterior probability $p(c|x_n)$ for each class $c \in C$, given an element from the training data x_n, is known:

$$F(A) := -\sum_{n=1}^{N} \sum_{c \in C} p(c|x_n) \log p_A(c|x_n). \tag{4.29}$$

Here, $p_A(c|x_n)$ denotes the class posterior probability as defined by the neural network, given its weight parameter matrix A.

For language modeling, the true posterior probability $p(w|w_1^{n-1})$ for $w \in \mathcal{V}$ is usually unknown. It might be given by the maximum likelihood estimates on the training data. Assuming an infinite context length, we obtain

$$p(w|w_1^{n-1}) = \begin{cases} 1, & w = w_n, \\ 0, & \text{otherwise.} \end{cases} \tag{4.30}$$

Accordingly, for language modeling Eq. (4.29) can be simplified to

$$F(A) = -\sum_{n=1}^{N} \log p_A(w_n|w_1^{n-1}), \tag{4.31}$$

where $p_A(w_n|w_1^{n-1})$ is given by Eq. (4.6), (4.13) or (4.28). This formulation of the cross-entropy error is equivalent to maximum likelihood training, as well as it is equivalent to minimizing the perplexity of the training data, which means that the training criterion of count-based LMs and neural network-based LMs is essentially the same. As long as no regularization or similar techniques are used, this also implies that the optimum neural network LM weight parameters maximizing Eq. (4.31) are attained when the neural network basically has learned the relative frequencies from the training data, as in the case of unsmoothed count-based LMs [Ney & Martin$^+$ 97]. Finding the optimum weight parameters is thus undesirable. However, due to the non-convexity of the optimization problem it is unlikely that the optimum solution can be found. In addition, the optimization process is usually terminated as soon as the performance on held-out data starts degrading, long before the optimum on the training data would be reached.

4.4.1 Stochastic Gradient Descent

For the optimization problem defined in Eq. (4.31), a huge number of training algorithms exists. However, for large-scale problems like language modeling, there is virtually only a single method which is used in practice, namely the stochastic gradient descent algorithm [Bottou 12]. The algorithm is given in Fig. 4.5.

Stochastic gradient descent repeatedly computes the gradient of the training criterion with respect to the individual weight parameters and updates the weights in the direction of steepest descent. Many variations are possible, especially regarding how to decrease the learning rate over time and when to terminate the optimization process. Unless otherwise stated, we terminate the optimization if the learning rate is too small to expect noticeable changes in the held-out perplexity anymore.

Input: training data, held-out data, weight matrix A, learning rate $\eta \in \mathbb{R}^+$
Output: optimized weight matrix A
$ppl^* \leftarrow \infty$;
$A^* \leftarrow A$;
repeat
  Randomize order of training data;
  foreach *sample from training data* **do**
    Compute $\nabla F(A)$ for training sample;
    Update weight parameters: $\alpha_{ij} \leftarrow \alpha_{ij} - \eta \cdot \partial F / \partial \alpha_{ij}$;
  end
  $ppl \leftarrow$ perplexity on held-out data given A;
  if $ppl^* > ppl$ **then**
    $A^* \leftarrow A$;
    $ppl^* \leftarrow ppl$;
  else
    $\eta \leftarrow \eta/2$;
    $A \leftarrow A^*$;
  end
until *converged*;

Figure 4.5 Pseudocode for the stochastic gradient descent algorithm.

The optimization algorithm is basically the same for all neural network LM variants investigated in this chapter. The only difference lies in the methods for computing the gradient $\nabla_A F$, which is detailed in the next two sections.

4.4.2 Backpropagation

The backpropagation method [Rumelhart & Hinton$^+$ 86] relies on the repeated application of the chain rule of calculus to compute the gradient of the training criterion $F(A)$ in the case of a feedforward neural network. After computing the activation values $y(n)$, $z(n)$ and $p(w_n|w_{n-2}, w_{n-1})$ consecutively as given by Eqs. (4.2) to (4.6), the algorithm proceeds in backwards direction to obtain the corresponding derivatives $\mathrm{err}_p(n)$, $\mathrm{err}_z(n)$ and $\mathrm{err}_y(n)$, where we define

$$\mathrm{err}_p(n) = \frac{\partial F}{\partial x_p(n)} \tag{4.32}$$

$$\mathrm{err}_z(n) = \frac{\partial F}{\partial x_z(n)} \tag{4.33}$$

$$\mathrm{err}_y(n) = \frac{\partial F}{\partial y(n)}, \tag{4.34}$$

respectively. In neural network terminology, these derivatives are sometimes called errors. They can be computed as follows

$$\mathrm{err}_p(n) = p(\cdot|w_{n-2}, w_{n-1}) - \hat{w}(n) \tag{4.35}$$

$$\mathrm{err}_z(n) = (1 - z(n)) \odot z(n) \odot A_3^T \, \mathrm{err}_p(n) \tag{4.36}$$

$$\mathrm{err}_y(n) = A_2^T \, \mathrm{err}_z(n), \tag{4.37}$$

where we have used the fact that $\mathrm{sig}'(x) = (1 - \mathrm{sig}(x))\,\mathrm{sig}(x)$ holds for the derivative of the sigmoid function sig. Furthermore, T denotes matrix transposition. Given these intermediate derivatives, we can compute the gradient of the training criterion with respect to the neural network parameters

$$\nabla F(A_3; n) = \mathrm{err}_p(n) z^T(n) \tag{4.38}$$

$$\nabla F(A_2; n) = \mathrm{err}_z(n) y^T(n) \tag{4.39}$$

$$\nabla F(A_1; n) = \mathrm{err}_y(n)(\hat{w}^T(n-2) + \hat{w}^T(n-1)), \tag{4.40}$$

which is eventually used to update the weight parameters as described in Section 4.4.1. Interestingly, while the forward pass of the neural network is non-linear, the backward pass is completely linear.

It is characteristic of the operation of the backpropagation algorithm that samples from the training data are evaluated independently of each other, and likewise, Eqs. (4.35) to (4.40) compute the gradient for one observation from the training data only. This means that the sum in Eq. (4.31) degenerates into a single summand. In practice, multiple m-grams from the training data are grouped together in a so-called mini-batch. The equations of the forward and backward pass are then evaluated for each element from the mini-batch individually, the resulting gradients are accumulated and a single weight update is performed for the entire mini-batch. Mathematically this can be formulated elegantly by using matrix-matrix operations instead of matrix-vector operations, as described in [Schwenk 07] for the case of language modeling. The activation and error vectors are then replaced by activation and error matrices such that the newly added columns correspond to the activations and errors of the individual elements from the mini-batch. The above notation then changes slightly such that lower case letters (i.e., y, z, e) representing vectors are replaced by upper case letters (Y, Z, Δ, respectively) to indicate matrices.

4.4.3 Backpropagation Through Time

For training recurrent neural networks, an extension of the backpropagation algorithm is needed which is commonly referred to as backpropagation through time (BPTT). The principle was suggested in [Rumelhart & Hinton$^+$ 86] and later refined in [Werbos 88, Werbos 90, Williams & Peng 90, Williams & Zipser 95].

The underlying idea is to convert an RNN into an equivalent feedforward network. This can be achieved in the following way: Given an RNN that is evaluated on a sequence w_1^N, we can create a copy of the RNN layers $\hat{w}(n)$, $y(n)$, $z(n)$ for all time steps n, and connect the copies in such a way that the resulting network mimics the original RNN. An example is depicted in Fig. 4.6, which is the unfolded version of the RNN shown in Fig. 4.2. In principle, after unfolding the RNN, the original backpropagation algorithm as known for feedforward networks can be used for RNN training. However, unlike standard feedforward networks, the unfolded RNN gets input at each of the layers, not only the first one.

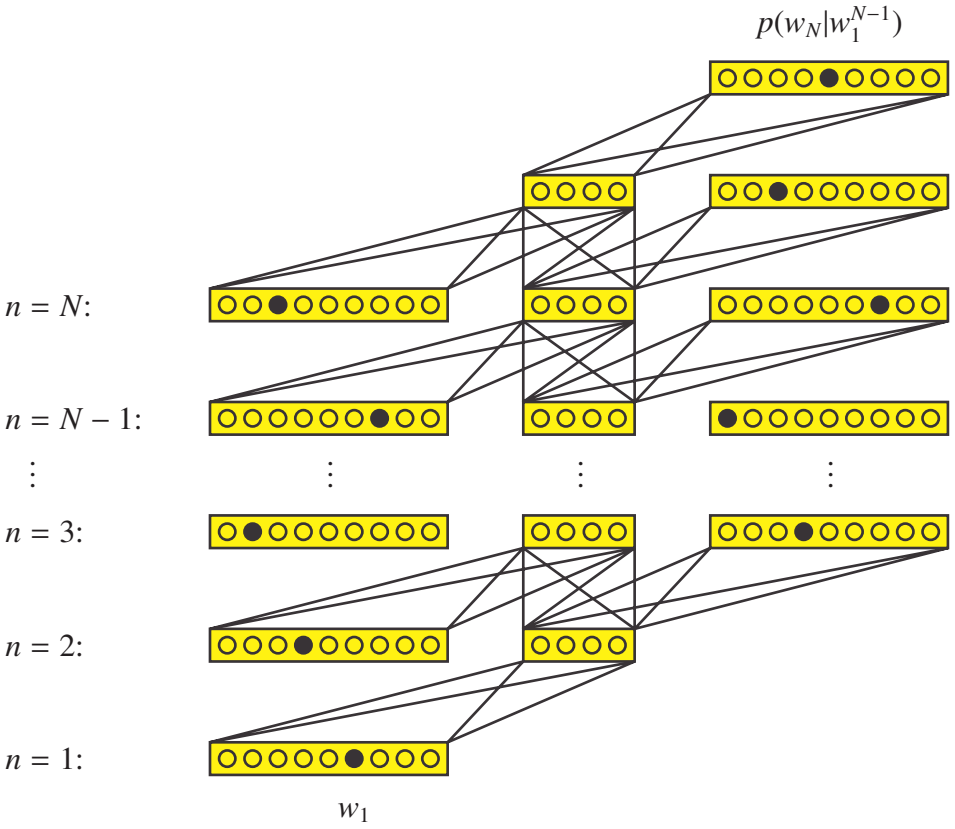

Figure 4.6 Recurrent neural network unfolded over time.

While backpropagation denotes a single algorithm for computing the gradient of feedforward networks, for RNNs many variants of the BPTT algorithm exist. These differ in several aspects. First, it is distinguished between an exact and an approximate gradient computation. This goes hand in hand with the question whether the RNN runs continuously or not. In the former case, the sequences to be processed are potentially infinite in length, in the latter the sequence length is limited a priori. For a continuously running RNN, an exact gradient virtually cannot be computed, whereas for RNNs operating on finite sequences, the computation of an exact gradient is well feasible. Second, there are variants for computing the gradient for a single element from the training data, i.e., $N = 1$ in (4.31), and as an alternative, the gradient can also be accumulated for multiple consecutive elements. This is similar to mini-batching as known from feedforward networks, however, in the RNN case, the elements have to be processed in the order given by the sequence. A more feedforward-like mini-batching for RNNs consists of evaluating multiple sequences at the same time and is described in Section 4.6.2. In this sense, mini-batches of an RNN are two-dimensional. The most important BPTT variants are listed below.

1. Real-time BPTT computes the gradient $\nabla F(A;n)$ for a single element of a sequence. Thus it is the RNN analogue of standard backpropagation as given by Eqs. (4.35) to (4.40). Its computational effort per time step is linear in the length of the sequence, as for each element n of the sequence, backpropagation has to be carried out for all the predecessor elements back to the first one.

2. Truncated BPTT approximates $\nabla F(A;n)$ by unfolding the RNN only for a fixed number of time steps, thus making the computation feasible for arbitrarily long sequences.

3. Epochwise BPTT calculates $\sum_{n=1}^{N} \nabla F(A;n)$ for the entire sequence. Only one forward and one backward pass have to be performed over the sequence, which makes this algorithm the most efficient choice for computing RNN gradients, at the limitation that it cannot be applied to problems that require continuous operation. The computational cost per time step is constant with respect to the sequence length. Epochwise BPTT also has the favorable property of computing an exact gradient, given that the internal RNN state is reset at the beginning of each sequence.

4. A hybrid BPTT variant as described in [Williams & Peng 90, Williams & Zipser 95] combines truncated and epochwise BPTT. As a result, the hybrid algorithm can be used to train continuously operating RNNs at a time complexity as low as that of epochwise BPTT, or, depending on the parameterization, even lower. The algorithm approximates $\sum_{n=n_0}^{n_1} \nabla F(A;n)$ for some $n_1 > n_0$. To speed up the computation, one can perform the forward pass of the RNN from time $n_0' < n_0$ to n_1, and the backward pass from n_1 back to n_0. In that way, the training samples $w_{n_0'}, \ldots, w_{n_0-1}$ are used only for advancing the RNN state, but they are skipped for the computation of the gradient. For continuous operation, no RNN state resets are performed between successive gradient calculations.

For efficiency reasons, only the last two variants are of interest for training RNN LMs. The advantage of epochwise BPTT lies in an exact training, but it is necessary to split the training data into sequences. Then by design it is not possible to learn context dependences spanning multiple sequences. On the contrary, the hybrid variant computes approximate gradients, but can constantly keep track of word dependences for the whole training data. The RNN LM toolkit described in [Mikolov & Kombrink$^+$ 11b] implements the hybrid algorithm, and the same algorithm was also used for training LSTM acoustic models in [Sak & Senior$^+$ 14]. The software used in [Graves & Liwicki$^+$ 09] is based on epochwise BPTT, as is the neural network language modeling toolkit presented in [Sundermeyer & Schlüter$^+$ 14].

We present the formulas for the epochwise BPTT algorithm only, the formulas for the hybrid variant are essentially the same. In the case of the RNN from Eqs. (4.10) to (4.13), the gradients with respect to the layer inputs are given by

$$\mathrm{err}_p(n) = p(\cdot|w_1^{n-1}) - \hat{w}(n) \tag{4.41}$$

$$\mathrm{err}_y(n) = (1 - y(n)) \odot y(n) \odot (A_3^T \,\mathrm{err}_p(n) + A_2^T \,\mathrm{err}_y(n+1)). \tag{4.42}$$

At the end of the sequence, when $n = N$, it is assumed that $\mathrm{err}_y(n > N) = 0$. The gradients of the weight matrices can finally be computed by

$$\nabla F(A_3) = \sum_{n=1}^{N} \mathrm{err}_p(n) y^T(n) \tag{4.43}$$

$$\nabla F(A_2) = \sum_{n=1}^{N} \mathrm{err}_y(n) y^T(n) \tag{4.44}$$

$$\nabla F(A_1) = \sum_{n=1}^{N} \mathrm{err}_y \, \hat{w}^T(n-1). \tag{4.45}$$

The corresponding derivatives for the LSTM neural network LM, as defined by Eqs. (4.21) to (4.28), are as follows

$$\mathrm{err}_p(n) = p(\cdot|w-1^{n-1}) - \hat{w}(n) \tag{4.46}$$

$$\begin{aligned}\varepsilon_c(n) = {} & A_2^T \, \mathrm{err}_p(n) + A_{zc}^T \, \mathrm{err}_z(n+1) + A_{zi}^T \, \mathrm{err}_i(n+1) + \\ & A_{zf}^T \, \mathrm{err}_f(n+1) + A_{zo}^T \, \mathrm{err}_o(n+1)\end{aligned} \tag{4.47}$$

$$\mathrm{err}_o(n) = o(n) \odot (1 - o(n)) \odot \tanh(c(n)) \odot \varepsilon_c(n) \tag{4.48}$$

$$\begin{aligned}\varepsilon_s(n) = {} & o(n) \odot \tanh(c(n)) \odot \Big(1 - \tanh(c(n))\Big) \odot \varepsilon_c(n) + \\ & f(n+1) \odot \varepsilon_s(n+1) + A_{ci} \odot \mathrm{err}_i(n+1) + \\ & A_{cf} \odot \mathrm{err}_f(n+1) + A_{co} \odot \mathrm{err}_o(n+1)\end{aligned} \tag{4.49}$$

$$\mathrm{err}_c(n) = i(n) \odot b(n) \odot (1 - b(n)) \odot \varepsilon_s(n) \tag{4.50}$$

$$\mathrm{err}_f(n) = f(n) \odot (1 - f(n)) \odot c(n-1) \odot \varepsilon_s(n) \tag{4.51}$$

$$\mathrm{err}_i(n) = i(n) \odot (1 - i(n)) \odot b(n) \odot \varepsilon_s(n) \tag{4.52}$$

$$\mathrm{err}_y(n) = A_{yi}^T \, \mathrm{err}_i(n) + A_{yf}^T \, \mathrm{err}_f(n) + A_{yc}^T \, \mathrm{err}_c(n) + A_{yo}^T \, \mathrm{err}_o(n), \tag{4.53}$$

where we have made use of the identity $\tanh'(x) = (1 - \tanh(x))\tanh(x)$ for the derivative of the tanh function. The variables ε_c and ε_s actually denote the gradient with respect to the output activations of $c(n)$ and $s(n)$, respectively:

$$\varepsilon_c(n) = \frac{\partial F}{\partial c(n)} \tag{4.54}$$

$$\varepsilon_s(n) = \frac{\partial F}{\partial s(n)}. \tag{4.55}$$

As in the standard RNN BPTT formulas, in the above formulation of the LSTM errors we have assumed epochwise BPTT as the underlying algorithm for computing the gradient. The formulas for the gradient with respect to the LSTM LM weight matrices are straightforward and completely analogous to Eqs. (4.43) to (4.45) from the standard RNN case.

4.5 Speed-Up Techniques for Neural Network Language Models

While neural networks have been found to give huge improvements over conventional count LMs, it is very costly to train them on large amounts of data. Therefore, a count LM might easily outperform a neural network when computational resources are limited, as the count LM can be trained on much more data. This motivates the use of speed-up techniques for neural networks, and in this section, we give an overview of the methods known from the literature. To accelerate neural network LM processing, it is important to reduce the computational effort incurred by the normalization at the output layer, which represents the main

bottleneck. As can be seen e.g. from Eq. (4.6), whenever a single probability estimate for a word w_n is needed, the full distribution over all vocabulary words has to be obtained. This requires the computation of the matrix-vector product $A_3 z(n)$ as well as the softmax function, where A_3 is a $(W \times J)$ matrix, the vocabulary size is denoted by W, and the hidden layer size is given by J. For most language modeling problems, the vocabulary size W exceeds the typical hidden layer size by several orders of magnitude. For this reason, the output layer normalization accounts for the vast majority of the computational effort, which applies to the forward pass as well as to the backward pass of the neural network LM. This problem has been extensively addressed in the literature, where solutions differ in whether probability normalization is enforced, or an approximate normalization is accepted.

When the normalization constraint of an LM is dropped, a simple approach is to compute only the numerator of Eq. (4.9) in an exact manner, and to come up with an approximation of the denominator. This method was investigated in [Shi & Zhang$^+$ 14a], however, only rescoring speed was improved, whereas training time remained unaffected. A similar idea was proposed independently in [Devlin & Zbib$^+$ 14] and [Shi & Zhang$^+$ 14b]: By introducing a regularization term in the training criterion, the neural network was trained in such a way that the denominator of Eq. 4.9 was approximately equal to one. Therefore, in decoding the normalization term could be dropped. Again the training time could not be reduced by this approach. A related method is the so-called noise-contrastive estimation from [Gutmann & Hyvärinen 10], which was applied to language modeling in [Mnih & Teh 12, Vaswani & Zhao$^+$ 13] and refined in [Mikolov & Sutskever$^+$ 13]. The idea is to modify the training criterion such that the normalization term can be learned as an additional parameter. In training, a sample of the vocabulary words is considered, and in decoding, the approximate normalization term is given by the learned parameter. In this way, both training and decoding time could be strongly reduced.

On the other hand, there are also speed-up techniques that still maintain normalization. To this end, one can limit the vocabulary to a small subset comprising the most frequent words [Schwenk 07]. This subset is usually called a shortlist. Obviously, its use implies that for many words from the vocabulary no neural network LM probability estimates can be obtained. Therefore it is essential to combine the neural network with a count LM covering the full vocabulary, and appropriately normalizing the two models as described in Section 4.6.4. Besides these drawbacks, shortlists are simple to implement and facilitate mini-batch parallelization and GPU-accelerated training [Schwenk & Rousseau$^+$ 12, Chen & Wang$^+$ 14]. Nevertheless, none of the techniques described above allows obtaining normalized neural network probability estimates for large vocabularies, at the same time speeding up training as well as testing. This means that analyzing perplexities of a single neural network is prohibitive with these methods. The only approach to combine all of the aforementioned properties is the word class factorization method. As it plays an essential role for this thesis, it is described in more detail in the next section.

4.5.1 Word Class Factorization

The idea of using word classes for speed-up was first proposed in [Goodman 01b] in the context of maximum entropy models. The approach readily transfers to neural networks, as a maximum entropy model can be interpreted as a single-layer neural network. Consequently,

in [Morin & Bengio 05] word classes were applied to improve the speed of neural network LMs. Here, a binary tree was constructed based on manually designed word classes, where the words were represented by paths in the tree. This lead to large speed-ups, but a significant degradation in perplexity was observed, too. In [Mnih & Hinton 08] this work was improved by a purely data-driven clustering approach as well as a refined clustering tree that allowed multiple paths for a single word. In [Le & Oparin$^+$ 11] a very similar clustering tree was investigated. Paths in the tree for a word had to be unique, but the constraint of a binary tree was relaxed to an arbitrary branching. Furthermore, the most frequent words were exempt from clustering and their probability was modeled directly. In [Mikolov & Kombrink$^+$ 11a] word classes were used for speed-up without a clustering hierarchy, which corresponds to a tree of depth one. In [Le & Oparin$^+$ 13] it was concluded that deep tree structures do not pay off in perplexity performance. In summary, the strategy from [Mikolov & Kombrink$^+$ 11a] or rather [Goodman 01b] seems to be a good compromise between model complexity and speed-up, and will be adopted in the following. Only for extremely large vocabularies, a deeper hierarchy seems necessary.

When computing a word probability based on the word class factorization technique, the probability $p(w_n|w_{n-2}, w_{n-1})$ is effectively decomposed as

$$p(w_n|w_{n-2}, w_{n-1}) = p_G(G(w_n)|w_{n-2}, w_{n-1}) \cdot p_w(w_n|w_{n-2}, w_{n-1}, G(w_n)). \tag{4.56}$$

Here, a mapping $G : \mathcal{V} \rightarrow \mathcal{G}$ is required that maps a word w_n from the vocabulary $\mathcal{V}$ to a word class $G(w_n)$ from the set of all word classes $\mathcal{G}$. For computing the probability of a given word, its class posterior probability p_G and the word posterior probability given the class p_w need to be obtained. (For single-word classes, the probability $p(w_n|w_{n-2}, w_{n-1}, G(w_n))$ is always one and can be dropped.) When no word classes are used, the computational effort for computing $p(w_n|w_{n-2}, w_{n-1})$ is proportional to the vocabulary size W. By applying Eq. (4.56), only $|\mathcal{G}|$ many classes and the words assigned to class $G(w_n)$ need to be considered. Assuming that on average we have a class size of $W/|\mathcal{G}|$ many words, this leads to a dramatic speed improvement, depending on the choice of $\mathcal{G}$. More precisely, the computational effort for the word class method is approximately proportional to

$$|\mathcal{G}| + \frac{W}{|\mathcal{G}|}, \tag{4.57}$$

and this function is minimized when $|\mathcal{G}| = \sqrt{W}$. As an example, for a vocabulary size of 10,000 words and 100 word classes, only 100 class probabilities and 100 word probabilities need to be computed. This corresponds to a speed-up factor of $10{,}000/(100 + 100) = 50$. The special cases of 1 and 10,000 word classes are equivalent to the original formulation without word classes.

Fig. 4.7 shows how the word class method can be integrated into a feedforward model. The embedding into a standard RNN or an LSTM is analogous. As opposed to the situation in Fig. 4.1, the hidden layer is connected with two output layers that are normalized separately.

Accordingly, Eq. 4.6 needs to be replaced by the two equations

$$p_w(w_n|G(w_n), w_1^{n-1}) = \varphi(A_{3,G(w_n)} y(n))\big|_{w_n} \tag{4.58}$$

$$p_G(G(w_n)|w_1^{n-1}) = \varphi(A_4 y(n))\big|_{G(w_n)}, \tag{4.59}$$

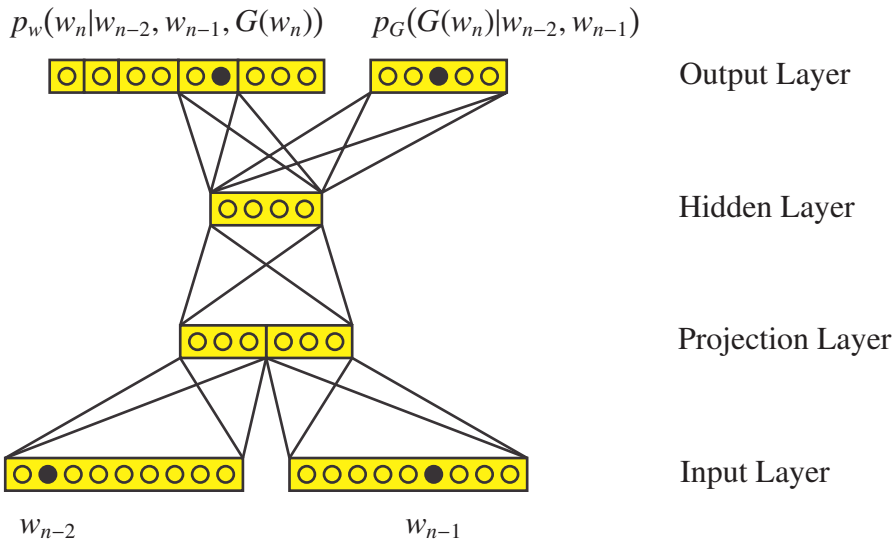

Figure 4.7 Trigram feedforward neural network architecture with word classes at the output layer.

where the matrix A_3 from Eq. (4.6) is decomposed into smaller class-specific matrices $A_{3,\cdot}$, and the weight matrix A_4 is newly introduced for learning class posterior probabilities. From Eqs. (4.58) and (4.59) it can be seen immediately that word classes incur a small overhead in model size. The speed-up comes from the fact that only a subset of the weight parameters needs to be accessed for computing a word probability.

For the training of this network, we replace Eqs. (4.35) and (4.36) by

$$\text{err}_{p_w}(n) = p_w(\cdot|w_{n-2}, w_{n-1}, G(w_n)) - \hat{w}_G(n) \tag{4.60}$$

$$\text{err}_{p_G}(n) = p_G(\cdot|w_{n-2}, w_{n-1}) - \hat{G}(n) \tag{4.61}$$

$$\text{err}_z(n) = (1 - z(n)) \odot z(n) \odot (A^T_{3,G(w_n)} \text{err}_{p_w}(n) + A^T_4 \text{err}_{p_G}(n)), \tag{4.62}$$

where we let $\hat{w}_G(n)$ be the one-hot encoding of word w_n within class $G(w_n)$, which is a vector of dimension $|\{w|w \in \mathcal{V} \wedge G(w) = G(w_n)\}|$, i.e., its dimension is the size of class $G(w_n)$. Furthermore, we let $\hat{G}(n)$ be the one-hot encoding of class $G(w_n)$. The updated formulas for the gradient with respect to the weight parameters as well as the formulas for a class-based standard RNN and LSTM then are straightforward. As can be seen from Eq. (4.60), a training signal is only specified for the words within the class $G(w_n)$. Strictly speaking, the probability $p(w|w_{n-2}, w_{n-1}, G(w_n))$ is undefined for any w with $G(w) \neq G(w_n)$, as the event is not an element of the underlying sigma-algebra of the probability distribution. Therefore, word classes equally speed-up the forward and the backward pass. Regarding the word class mapping G, several algorithms have been explored in the literature:

1. In [Mikolov & Kombrink$^+$ 11a] it was proposed to use word classes induced by frequency binning. As a result, all the classes are approximately equal in size, but usually there is no semantic relationship between the words of a given class, except that they occur with similar frequency.

2. [Zweig & Makarychev 13] and [Shi & Zhang$^+$ 13] investigated perplexity-based word classes as discussed in Section 1.2.2 for neural network LM speed-up, and it was found

that they lead to significantly better perplexities than word classes computed by frequency binning.

3. In [Zweig & Makarychev 13], a regularization term for perplexity-based word classes was suggested, reducing the computational effort of the neural network further. To this end, word and class frequencies in the training data were taken into account in the word clustering process.

4. The word classes from [Le & Oparin$^+$ 11] are based on a clustering of word vectors that were obtained by training a shortlist neural network LM in the first place.

Algorithm (4) incurs multiple trainings of a neural network LM. Internal comparisons did not indicate an improvement over perplexity-based word classes, whereas the computational cost of the neural network approach (4) is considerably higher. Among the first three variants, algorithm (2) performs best in terms of perplexity, making this variant the default choice for the experiments presented in this thesis.

It should be noted that there is an inherent mismatch between the training criterion of perplexity-based word classing algorithms and the word class decomposition applied by the neural network LM, as given by Eq. (4.56). In fact, the decomposition cannot be used for the training of word classes, because it represents an exact mathematical identity. Therefore Eq. (4.56) is independent of the word class mapping G, which is undesirable for a word class training criterion.

In summary, word classes are an effective means for reducing the computational costs of neural network LMs. However, gains can only be expected when the number of words considered for a given LM context is small, e.g. when the neural network LM is applied to reranking n-best lists. If the full probability distribution needs to be obtained instead, as is the case e.g. in speech decoding, word classes will slow down the computation.

4.5.2 Maximum Entropy Features

Following the discussion in Section 4.5, instead of reducing the output layer size, it would be likewise effective to reduce the size of the hidden layer for improving the speed of neural network LMs. On the other hand, a smaller hidden layer usually degrades perplexity performance. The reduction in model size can be circumvented by combining the neural network LM with a different type of model, the complexity of which does not depend on the hidden layer.

This idea was first realized in [Bengio & Ducharme$^+$ 03], where direct connections were integrated into the neural network architecture. Given the feedforward model from Eqs. (4.2) to (4.6), an additional weight matrix A'_3 was introduced such that Eq. (4.6) becomes

$$p(w_n|w_{n-2}, w_{n-1}) = \varphi(A_3 z(n) + A'_3 y(n))\big|_{w_n}, \tag{4.63}$$

thereby connecting the projection and the output layer directly and thus bypassing the hidden layer. However, no significant improvements were obtained with this architecture. This approach can be refined by modifying Eq. (4.63) to

$$p(w_n|w_{n-m+1}^{n-1}) = \varphi\left(A_3 z(n) + \sum_{m'=1}^{m-1} A_{4,m'} f(w_{n-m'}^{n-1})\right)\Bigg|_{w_n}, \tag{4.64}$$

where $f(w_{n-m'}^{n-1})$ denotes the one-hot encoding of an m'-gram $w_{n-m'}^{n-1}$, and by $A_{4,\cdot}$ we refer to additional weight matrices. Here, the input and output layer are directly connected, instead of the projection layer and output layer. Obviously, when using direct connections for higher context sizes m, the number of parameters for $A_{4,\cdot}$ becomes too large to store in memory. As a remedy, in [Mikolov & Deoras$^+$ 11] it was suggested that a hash table is used for maintaining the weight matrices $A_{4,\cdot}$, which can be accessed by computing a hash value for the context words $w_{n-m'}^{n-1}$. In that work, the model was successfully introduced within an RNN framework. The hidden layer of the neural network could be roughly halved, while the speech recognition performance was still slightly better than that of an RNN without direct connections. The hash table for the additional weight parameters adds a huge overhead in memory size, but the computational overhead for direct connections is negligible. A detail of the training seems important: While the RNN hidden layer was trained with the hybrid BPTT variant, the direct connection weights were trained in an online fashion, i.e., with a mini-batch size of one.

As the direct connections are mathematically equivalent to a maximum entropy model with binary features $f(w_{n-m'}^{n-1})$, the direct connections as defined by Eq. (4.64) are usually referred to as maximum entropy features for neural network LMs, and we will analyze them in Section 4.7.2

4.6 Neural Network Language Modeling Extensions

Neural networks open a wide range of opportunities for incorporating task-specific knowledge at various levels. In this section, we describe several refinements of neural networks which are specific to language modeling.

4.6.1 Input Data Standardization

It is well-known that backpropagation training of neural networks can benefit from various degrees of normalization. E.g., in [Bishop 95, p. 298] as well as in [Sarle 99, Sarle 02] it was recommended to standardize the input data of a neural network, regardless of whether the data are continuous or categorical. In [Le Cun & Kanter$^+$ 91] the centering of input and also hidden units was investigated, and in [Schraudolph 12] even centering the error signals was analyzed. Standardization of input features for neural network language modeling was first investigated in [Sundermeyer & Schlüter$^+$ 12].

Given a word sequence w_1^N, by centering the input data we refer to the transform

$$\hat{w}_i(n) := \hat{w}_i(n) - \langle \hat{w}_i \rangle, \tag{4.65}$$

and by standardizing the input data we denote the transform

$$\hat{w}_i(n) := \frac{\hat{w}_i(n) - \langle \hat{w}_i \rangle}{\sigma_i} \tag{4.66}$$

of the i-th dimension of the one-hot encoding representation $\hat{w}(n)$ of the word w_n. The variables $\langle \hat{w}_i \rangle$ and σ_i indicate the mean and the standard deviation of dimension i in the

training data w_1^N, as defined by

$$\langle \hat{w}_i \rangle = \frac{1}{N} \sum_{n=1}^{N} \hat{w}_i(n) \tag{4.67}$$

$$\sigma_i^2 = \frac{1}{N-1} \sum_{n=1}^{N} \big(\hat{w}_i(n) - \langle \hat{w}_i \rangle\big)^2. \tag{4.68}$$

Fortunately, it is not necessary to accumulate the above statistics explicitly, which can be seen as follows.

$$\langle \hat{w}_i \rangle = \frac{1}{N} \sum_{n=1}^{N} \delta_{w_n i} \tag{4.69}$$

$$= \frac{1}{N} \cdot N_i \tag{4.70}$$

$$= p_i. \tag{4.71}$$

By N_i, we denote the count of the i-th word in the training data, and p_i is the corresponding relative frequency. Furthermore, we let

$$\delta_{wi} = \begin{cases} 1 & \text{word } w \text{ is the } i\text{-th element of the vocabulary} \\ 0 & \text{otherwise} \end{cases} \tag{4.72}$$

be the Kronecker delta. Overall, this shows that the mean value of the i-th dimension of the input data is simply the probability of occurrence of the i-th element of the vocabulary in the training data.

For the entries σ_i on the main diagonal of the covariance matrix of the training data, we find

$$\sigma_i^2 = \frac{1}{N-1} \sum_{n=1}^{N} \left(\hat{w}_i^2(n) - 2\hat{w}_i(n)\langle \hat{w}_i \rangle + \langle w_i \rangle^2 \right) \tag{4.73}$$

$$= \frac{1}{N-1} \left(N_i - 2p_i N_i + N p_i^2 \right) \tag{4.74}$$

$$= \frac{1}{N-1} \left(N_i(1-2p_i) + N p_i^2 \right) \tag{4.75}$$

$$= \frac{N_i(1-2p_i) + N p_i^2}{N-1} \tag{4.76}$$

$$\overset{N\to\infty}{=} p_i(1-2p_i) + p_i^2 \tag{4.77}$$

$$= p_i - p_i^2. \tag{4.78}$$

In summary, the transform of the input data given by Eq. (4.66) can be obtained in an efficient manner.

In textbooks like [Bishop 95, p. 299] it is also advised to eliminate dependences between different input dimensions. One-hot encoded input data obviously are inherently correlated, e.g., we have

$$\hat{w}_j(n) = 1 - \sum_{i \neq j} \hat{w}_i(n) \tag{4.79}$$

for any dimension j of the vector $\hat{w}(n)$. However, to transform the data such that these correlations are eliminated it is necessary to compute the eigenvalues of the covariance matrix of the training data, which seems unfeasible for reasonably large vocabulary sizes.

4.6.2 Sequence Definitions for Recurrent Neural Networks

The standardization of input features aims at improving the training such that better neural network weight parameters can be found, and the optimization process converges faster. Another way to speed up the training of neural network LMs lies in evaluating multiple words at the same time. As described in Sections 4.4.2 and 4.4.3, this can be done by processing multiple training events in parallel and accumulating the corresponding gradients. In the case of feedforward networks, there is no interdependence between the training of several m-grams, which makes mini-batching simple and efficient. The situation for an RNN is depicted in Fig. 4.8, where a mini-batch of size k is used, and the index j simultaneously iterates over the word positions in all of the k sentences from the training data. By w_{ij} we

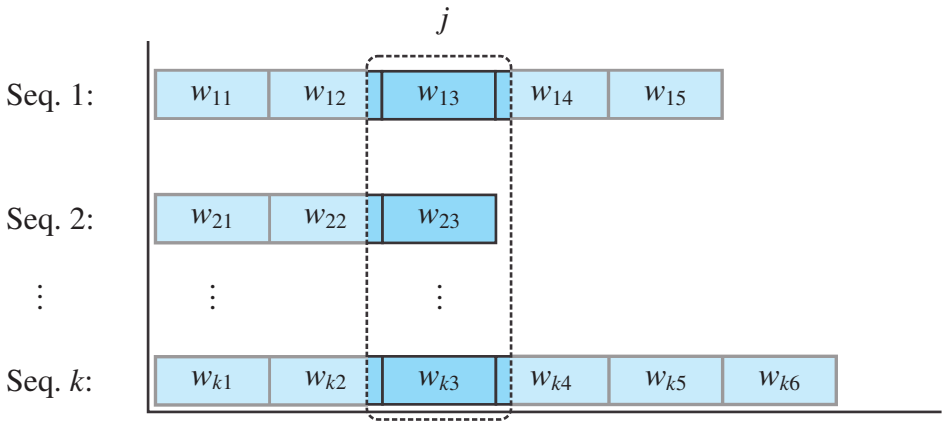

Figure 4.8 Example of the evaluation of the j-th word position, given a mini-batch of size k. Here, the sequences considered by the RNN are defined by the sentences, as found verbatim in the training data.

denote the j-th word of the i-the sentence in the mini-batch. The main problem here lies in the varying length of the individual sentences. Mathematically, this can be remedied e.g. by sorting the sentences in the mini-batch by their length in descending order, and reducing the dimensions of the activation and error matrices by one as soon as a sentence from the mini-batch ends. However, this means that with increasing j mini-batching becomes more and more inefficient, because less sentences are evaluated in parallel. For this reason, it is necessary to balance the lengths of the sequences within a mini-batch, as suggested in [Chen & Wang+ 14] for the hybrid BPTT variant, and independently implemented for the toolkit described in [Sundermeyer & Schlüter+ 14], which uses epochwise BPTT.

The first approach is to concatenate multiple sentences up to a maximum number of words for the resulting sequences, which is depicted in Fig. 4.9. In this way, sequence lengths are much more similar than the raw sentence lengths, and as a result, BPTT can be performed more efficiently. This strategy implies that the beginning of a sequence also coincides with the beginning of a sentence. (The only exception would be when a natural sentence exceeds the pre-defined maximum sequence length, which can happen e.g. in the

case of conversational speech transcriptions. Then it is necessary to break up a sequence in the middle of a sentence.) This behavior may be beneficial for epochwise BPTT, where RNN state resets occur at the beginning of a sequence. In that way, the RNN may learn that from the initial RNN state, the initial words from a sentence need to be predicted. Furthermore, this way of defining sequences facilitates the learning of across-sentence dependences with the epochwise BPTT algorithm.

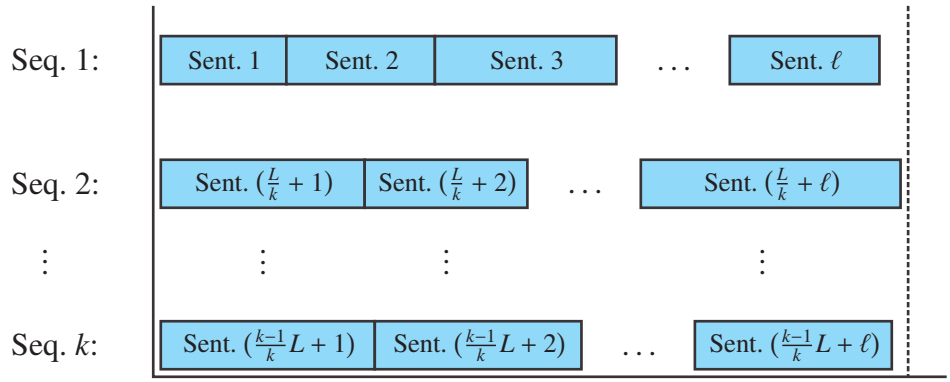

Figure 4.9 Example of the concatenation approach for forming RNN sequences from multiple consecutive sentences. For notational convenience, we assume that each of the k sequences from the mini-batch is composed of exactly ℓ sentences, and that the training data consist of L sentences.

Furthermore, we can also ignore the segmentation of the training words into sentences completely, and wrap sentences at arbitrary positions after a fixed number of words. This is depicted in Fig. 4.10. For this type of processing, BPTT is most efficient, but sequences do not necessarily start at the beginning of a sentence.

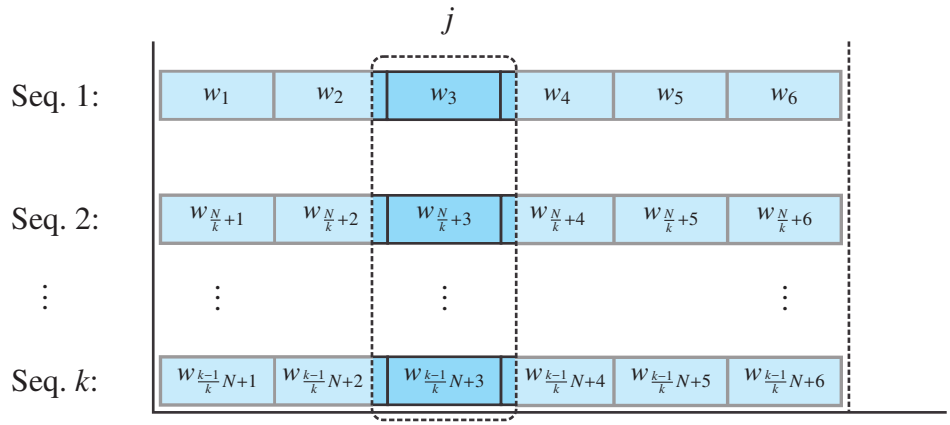

Figure 4.10 Example of the strategy where a sequence is built from a fixed number of consecutive words from the training data.

4.6.3 Effect of a Projection Layer

Another aspect of neural network LMs is the size of the resulting model, which is directly tied to the number of nodes in a layer as well as the total number of layers. As can be seen from

Figs. 4.1, 4.2 and 4.4, the neural network architectures discussed in the previous sections differ in this regard: While the standard RNN only consists of a single recurrent hidden layer, the feedforward network as well as the LSTM architecture make use of an additional hidden layer directly following the input layer. This layer is often denoted the projection layer, as it can be interpreted to project discrete word information into a continuous space. In principle, any of the three neural network architectures can be combined with a projection layer, or the layer can be left out. However, in previous experiments [Sundermeyer & Schlüter$^+$ 12, Freiberg 13, Sundermeyer & Schlüter$^+$ 14] the relevance of such a layer was investigated. In all cases it was observed that the additional layer does not have a crucial effect on perplexity. Especially when no activation function is used for the projection layer, this result is not surprising: By adjusting the weights of the following hidden layer, the projection layer can be omitted, while the resulting network is still equivalent to the network including the layer.

Nevertheless, it makes sense to incorporate a projection layer, at least for feedforward and LSTM neural network LMs: Even when speed-up techniques as those described in Section 4.5 are used, the main computational effort still accounts for the weight multiplication between the last hidden layer and the output layer. For a feedforward network, the hidden layer size grows linearly with increasing m-gram order, and as a certain dimension for a continuous space word representation is required to obtain good performance, the projection layer can be quite large for higher order LMs. Therefore, it is preferable not to connect the projection layer directly to the output layer, but to have an intermediate hidden layer that compresses the weight matrix dimensions. This argument does not apply to recurrent neural networks in general, because they only model a bigram dependence explicitly.

For LSTMs in particular, the effect of a projection layer is as follows. Let J be the size of an LSTM layer, and let I and K be the size of the previous and the next layer, respectively. From Eqs. (4.22) to (4.26) it can be seen immediately that there are $4IJ$ many weights connecting the LSTM layer with its predecessor layer, and JK many weights connecting the LSTM layer with its successor layer. When omitting the projection layer, we have $I = W$ for a vocabulary size of W. When adding a projection layer with $I \ll W$, we can reduce the number of neural network parameters at the input virtually to one fourth, without a degradation in performance. Therefore for most experiments in this work, feedforward and LSTM neural network LMs are combined with a projection layer.

4.6.4 Word Probability Normalization

In many practical applications, a count LM is trained on huge text corpora, where the LM vocabulary W is derived from these training data. When a neural network LM is used, it is often found that it is not possible to come up with estimates for all of the words from the original vocabulary: If a shortlist is used, by design no neural network LM probabilities are available for words which are not part of the shortlist. If word classes are used, the full vocabulary can be covered. However, in most cases it is computationally too expensive to train the neural network on all of the data that were used to train the count LM. As a result, a subset of the count LM data has to be used, which does not necessarily contain all of the words in the count LM vocabulary. In both cases there is a mismatch between the vocabularies of the count LM and the neural network LM, which makes perplexities incomparable and requires appropriate normalization.

In [Emami & Mangu 07] two normalization variants were considered for extending the neural network LM probabilities $p_{\text{NN}}(w_n|w_{n-m+1}^{n-1})$ to the full vocabulary. In the first variant, a zero probability is assumed for all words not in the vocabulary $\mathcal{V}_{\text{NN}}$ of the neural network:

$$p(w_n|w_{n-m+1}^{n-1}) = \begin{cases} p_{\text{NN}}(w_n|w_{n-m+1}^{n-1}) & w_n \in \mathcal{V}_{\text{NN}} \\ 0 & \text{otherwise.} \end{cases} \tag{4.80}$$

Unless the neural network LM is interpolated with the count LM, the above approach is not suitable for computing perplexities. As an alternative, the estimates $p_{\text{count}}(w_n|w_{n-m+1}^{n-1})$ of the count LM can be used directly for words not in the neural network vocabulary:

$$p(w_n|w_{n-m+1}^{n-1}) = \begin{cases} p_{\text{NN}}(w_n|w_{n-m+1}^{n-1}) \cdot \gamma(w_{n-m+1}^{n-1}) & w_n \in \mathcal{V}_{\text{NN}} \\ p_{\text{count}}(w_n|w_{n-m+1}^{n-1}) & \text{otherwise,} \end{cases} \tag{4.81}$$

where we let

$$\gamma(w_{n-m+1}^{n-1}) = \sum_{w \in \mathcal{V}_{\text{NN}}} p_{\text{count}}(w|w_{n-m+1}^{n-1}). \tag{4.82}$$

In this way, non-zero probabilities are obtained for all vocabulary words, but it is not possible to compute perplexities using the neural network LM alone. In addition, the computation of the normalization term γ is very costly. In terms of speech recognition performance, no difference between Eqs. (4.80) and (4.81) was observed.

In [Park & Liu$^+$ 10] a modified neural network architecture was proposed that included a so-called out-of-shortlist node at the input and the output layer. This simply means that the training data are preprocessed such that all out-of-vocabulary (OOV) words with respect to the neural network vocabulary are replaced by a new token $\langle\text{unk}\rangle$, which accumulates the probability mass of OOV words. In that case, it was suggested extending neural network LM probabilities to the full vocabulary by

$$p(w_n|w_{n-m+1}^{n-1}) = \begin{cases} p_{\text{NN}}(w_n|w_{n-m+1}^{n-1}) & w_n \in \mathcal{V}_{\text{NN}} \\ \tilde{p}_{\text{count}}(w_n|w_{n-m+1}^{n-1}) \cdot p_{\text{NN}}(\langle\text{unk}\rangle|w_{n-m+1}^{n-1}) & \text{otherwise,} \end{cases} \tag{4.83}$$

with the definition

$$\tilde{p}_{\text{count}}(w_n|w_{n-m+1}^{n-1}) = \frac{p_{\text{count}}(w_n|w_{n-m+1}^{n-1})}{\sum_{w \notin \mathcal{V}_{\text{NN}}} p_{\text{count}}(w|w_{n-m+1}^{n-1})}. \tag{4.84}$$

As before, the computational effort for this kind of normalization is high, and no perplexity evaluation of the stand-alone neural network LM is possible.

A much simpler variant was suggested in [Sundermeyer & Oparin$^+$ 13], extending neural network LM probabilities according to

$$p(w_n|w_{n-m+1}^{n-1}) = \begin{cases} p_{\text{NN}}(w_n|w_{n-m+1}^{n-1}) & w_n \in \mathcal{V}_{\text{NN}} \\ \frac{1}{|\mathcal{V}_{\text{count}} \setminus \mathcal{V}_{\text{NN}}|+1} \cdot p_{\text{NN}}(\langle\text{unk}\rangle|w_{n-m+1}^{n-1}) & \text{otherwise.} \end{cases} \tag{4.85}$$

In this way, effectively a zerogram distribution is assumed over words that do not appear in the neural network LM vocabulary. Unlike the other normalization schemes, Eq. (4.85) does not incur any computational overhead, and at the same time, probabilities do not depend on an additional count LM. Therefore, in this thesis we stick to Eq. (4.85) for perplexity normalization.

4.6.5 Integrated Training

In most cases, when a neural network LM is used, it is interpolated with a count LM, i.e., the resulting probability is computed according to the formula

$$p(w_n|w_1^{n-1}) = (1-\lambda)p_{\text{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_{\text{NN}}(w_n|w_1^{n-1}) \tag{4.86}$$

for an interpolation weight $\lambda \in [0,1]$, an m-gram count LM p_{count} and e.g. an RNN LM $p_{\text{NN}}(w_n|w_1^{n-1})$. Usually, significant improvements are obtained by interpolation, as the two LMs can be considered complementary, and often the count LM is also trained on more data than the neural network.

On the other hand, in Eq. (4.86) there are three training processes involved, where the count LM, the neural network, and the interpolation parameter λ are optimized separately. From that point of view it seems interesting to analyze a modified neural network training criterion, where the fact that the neural network is interpolated with a count LM at a later time is already reflected. To this end, we can modify the training criterion $F(A)$ with respect to the neural network parameters A from Eq. (4.31) in the following way

$$F(A,\lambda) = -\sum_{n=1}^{N} \log((1-\lambda)p_{\text{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_{\text{NN}}(w_n|w_1^{n-1};A)). \tag{4.87}$$

We refer to this approach as the integrated training of a count LM and a neural network LM. In contrast to similar methods like e.g. maximum entropy features as described in Section 4.5.2, the integrated training does not require adapting the neural network equations for the forward pass. It should be noted that Eq. (4.87) implies that we no longer fulfill the natural pairing property of a training criterion and its corresponding activation function [Bishop 95, p. 232], which slightly complicates the resulting mathematics.

In the following, we give the derivation of the error terms that need to be computed for training the model according to the new criterion. Here, we assume that both the count LM and the neural network LM rely on the same vocabulary, and that we have $N = 1$ in Eq. (4.87). We obtain

$$\frac{\partial F}{\partial p_{\text{NN}}(w|w_1^{n-1})} = -\frac{\lambda\delta_{ww_n}}{(1-\lambda)p_{\text{count}}(w|w_{n-m+1}^{n-1}) + \lambda p_{\text{NN}}(w|w_1^{n-1})} \tag{4.88}$$

for the derivative of the training criterion with respect to the word probability at the output layer, and for the derivative with respect to the input activation at the output layer, we have

$$\begin{aligned}
\text{err}_{pw}(n) &= \sum_{w'\in\mathcal{V}} \frac{\partial F}{\partial p_{\text{NN}}(w'|w_1^{n-1})} \cdot \frac{\partial p_{\text{NN}}(w'|w_1^{n-1})}{\partial x_{pw}} && (4.89)\\
&= \sum_{w'\in\mathcal{V}} \frac{\lambda p_{\text{NN}}(w|w_1^{n-1}) p_{\text{NN}}(w'|w_1^{n-1})\delta_{w'w_n}}{(1-\lambda)p_{\text{count}}(w'|w_{n-m+1}^{n-1}) + \lambda p_{\text{NN}}(w'|w_1^{n-1})} - \\
&\qquad \sum_{w'\in\mathcal{V}} \frac{\lambda p_{\text{NN}}(w|w_1^{n-1})\delta_{w'w_n}\delta_{ww'}}{(1-\lambda)p_{\text{count}}(w'|w_{n-m+1}^{n-1}) + \lambda p_{\text{NN}}(w'|w_1^{n-1})} && (4.90)\\
&= \frac{\lambda p_{\text{NN}}(w|w_1^{n-1}) p_{\text{NN}}(w_n|w_1^{n-1}) - \lambda p_{\text{NN}}(w_n|w_1^{n-1})\delta_{ww_n}}{(1-\lambda)p_{\text{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_{\text{NN}}(w_n|w_1^{n-1})}, && (4.91)
\end{aligned}$$

where x_{pw} is the component of the vector x_p corresponding to word w, and by $\delta_{ww'}$ we denote the Kronecker delta. In Eq. (4.90) we have made use of the identity

$$\frac{\partial p_{\mathrm{NN}}(w'|w_1^{n-1})}{\partial x_{pw}} = p_{\mathrm{NN}}(w|w_1^{n-1})\delta_{ww'} - p_{\mathrm{NN}}(w|w_1^{n-1})p_{\mathrm{NN}}(w'|w_1^{n-1}) \tag{4.92}$$

that holds for the softmax function φ, as used in Eq. (4.28). By setting $\lambda = 1$ in Eq. (4.91), we obtain $\mathrm{err}_{pw}(n) = p_{\mathrm{NN}}(w|w_1^{n-1}) - \delta_{ww_n}$, which is identical to the result from Eq. (4.35).

We can refine the above derivation by combining it with the word classing speed-up technique from Section 4.5.1 that is necessary for training the integrated model on large vocabularies. This leads to the following derivative with respect to the word posterior probability

$$\begin{aligned}&\frac{\partial F}{\partial p_w(w'|w_1^{n-1}, G(w'))} \\ &\quad = -\frac{\lambda p_G(G(w')|w_1^{n-1})\delta_{w'w_n}}{(1-\lambda)p_{\mathrm{count}}(w'|w_{n-m+1}^{n-1}) + \lambda p_G(G(w')|w_1^{n-1})p_w(w'|w_1^{n-1}, G(w'))},\end{aligned} \tag{4.93}$$

and with respect to the class posterior probability, we have

$$\begin{aligned}&\frac{\partial F}{\partial p_G(g'|w_1^{n-1})} \\ &\quad = -\frac{\lambda p_w(w_n|w_1^{n-1}, g')\delta_{g'G(w_n)}}{(1-\lambda)p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1})p_w(w_n|w_1^{n-1}, G(w_n))}.\end{aligned} \tag{4.94}$$

In a second step, we can compute the error terms in the following way. For the errors of the word output layer, we obtain

$$\mathrm{err}_{p_w w'}(n) = \sum_{w'' \in G(w_n)} \frac{\partial F}{\partial p_w(w''|w_1^{n-1}, G(w_n))} \frac{\partial p_w(w''|w_1^{n-1}, G(w_n))}{\partial x_{p_w w'}} \tag{4.95}$$

$$\begin{aligned}&= \sum_{w'' \in G(w_n)} \frac{\lambda p_G(G(w'')|w_1^{n-1})\delta_{w''w_n} p_w(w'|w_1^{n-1}, G(w_n))p_w(w''|w_1^{n-1}, G(w''))}{(1-\lambda)p_{\mathrm{count}}(w''|w_{n-m+1}^{n-1}) + \lambda p_G(G(w'')|w_1^{n-1})p_w(w''|w_1^{n-1}, G(w''))} - \\ &\quad \sum_{w'' \in G(w_n)} \frac{\lambda p_G(G(w'')|w_1^{n-1})\delta_{w''w_n} p_w(w'|w_1^{n-1}, G(w'))\delta_{w'w''}}{(1-\lambda)p_{\mathrm{count}}(w''|w_{n-m+1}^{n-1}) + \lambda p_G(G(w'')|w_1^{n-1})p_w(w''|w_1^{n-1}, G(w''))}\end{aligned} \tag{4.96}$$

$$\begin{aligned}&= \frac{\lambda p_G(G(w_n)|w_1^{n-1})p_w(w'|w_1^{n-1}, G(w_n))p_w(w_n|w_1^{n-1}, G(w_n))}{(1-\lambda)p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1})p_w(w_n|w_1^{n-1}, G(w_n))} - \\ &\quad \frac{\lambda p_G(G(w_n)|w_1^{n-1})p_w(w'|w_1^{n-1}, G(w'))\delta_{w'w_n}}{(1-\lambda)p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1})p_w(w_n|w_1^{n-1}, G(w_n))}\end{aligned} \tag{4.97}$$

$$\begin{aligned}&= \lambda p_G(G(w_n)|w_1^{n-1}) \cdot \\ &\quad \frac{p_w(w'|w_1^{n-1}, G(w_n))p_w(w_n|w_1^{n-1}, G(w_n)) - p_w(w'|w_1^{n-1}, G(w_n))\delta_{w'w_n}}{(1-\lambda)p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1})p_w(w_n|w_1^{n-1}, G(w_n))},\end{aligned} \tag{4.98}$$

where the error is only defined for words w' that occur in the class $G(w_n)$ under consideration at time step n. Similarly, the error of the class output layer is given by

$$\begin{aligned}
\mathrm{err}_{p_G g'}(n) &= \sum_{g'' \in \mathcal{G}} \frac{\partial F}{\partial p_G(g''|w_1^{n-1})} \frac{\partial p_G(g''|w_1^{n-1})}{\partial x_{p_G g'}} && (4.99)\\
&= \sum_{g'' \in \mathcal{G}} \frac{\lambda p_w(w_n|w_1^{n-1}, g'') \delta_{g'' G(w_n)} p_G(g'|w_1^{n-1}) p_G(g''|w_1^{n-1})}{(1-\lambda) p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1}) p_w(w_n|w_1^{n-1}, G(w_n))} - \\
&\quad \sum_{g'' \in \mathcal{G}} \frac{\lambda p_w(w_n|w_1^{n-1}, g'') \delta_{g'' G(w_n)} p_G(g'|w_1^{n-1}) \delta_{g' g''}}{(1-\lambda) p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1}) p_w(w_n|w_1^{n-1}, G(w_n))} && (4.100)\\
&= \frac{\lambda p_w(w_n|w_1^{n-1}, G(w_n)) p_G(g'|w_1^{n-1}) p_G(G(w_n)|w_1^{n-1})}{(1-\lambda) p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1}) p_w(w_n|w_1^{n-1}, G(w_n))} - \\
&\quad \frac{\lambda p_w(w_n|w_1^{n-1}, G(w_n)) p_G(g'|w_1^{n-1}) \delta_{g' G(w_n)}}{(1-\lambda) p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1}) p_w(w_n|w_1^{n-1}, G(w_n))} && (4.101)\\
&= \lambda p_w(w_n|w_1^{n-1}, G(w_n)) \cdot \\
&\quad \frac{p_G(g'|w_1^{n-1}) p_G(G(w_n)|w_1^{n-1}) - p_G(g'|w_1^{n-1}) \delta_{g' G(w_n)}}{(1-\lambda) p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_G(G(w_n)|w_1^{n-1}) p_w(w_n|w_1^{n-1}, G(w_n))}. && (4.102)
\end{aligned}$$

Again, we can verify that by setting $\lambda = 1$ we obtain the result from Eq. (4.60) and Eq. (4.61).

One important aspect of the integrated training approach lies in the choice of the training data. It does not seem promising to train the count LM and the neural network LM on the same data. The reason is that the count LM would contribute probabilities that are close to the relative frequencies, which to some extent can be considered optimal estimates for the training data [Ney & Martin$^+$ 97]. Furthermore, the training criterion given in Eq. (4.87) is minimized when the interpolated probabilities $(1-\lambda) p_{\mathrm{count}}(w_n|w_{n-m+1}^{n-1}) + \lambda p_{\mathrm{NN}}(w_n|w_1^{n-1})$ as a whole are equal to the relative frequencies. Then it follows immediately that minimizing Eq. (4.87) with respect to $p_{\mathrm{NN}}(w_n|w_1^{n-1})$ results in relative frequencies for the neural network LM probabilities as well. However, this is identical to the optimum solution of the independent training of the neural network LM according to Eq. (4.31). In other words, the training criteria Eqs. (4.31) and (4.87) are equivalent unless the count LM estimates significantly differ from the relative frequencies. This is achieved by training the count LM on data that are disjoint with respect to the neural network LM training data. In this way, the neural network may learn to better complement the count LM.

4.7 Experimental Results

The techniques discussed in this chapter are evaluated experimentally on the Treebank corpus. A description of the corpus can be found in Section A.2.

4.7.1 Neural Network Architectures

Table 4.1 lists the perplexities that can be obtained with different types of language models. The count LM makes use of modified Kneser-Ney smoothing [Kneser & Ney 95, Chen & Goodman 99] without count cutoffs, and the order is a 5-gram. The feedforward neural network LM is a 6-gram. In the case of both the count LM and the feedforward neural network LM no improvements could be obtained by increasing the order further. All parameters of the different neural networks were tuned to obtain the best possible perplexity performance. We can see that any neural network variant significantly outperforms the count LM. The

LM	Dev	Test
Count-Based	148.0	141.2
Feedforward	132.6	127.6
RNN	128.8	122.5
LSTM	**113.9**	**108.0**

Table 4.1 Stand-alone perplexities on the Treebank corpus for different language models. The feedforward result is taken from [Freiberg 13].

difference between feedforward and recurrent neural network LMs is rather small, whereas the overall best result on the test data is obtained by a large margin by the LSTM. Since first results were published in [Sundermeyer & Schlüter+ 12], the LSTM performance could be improved further by additional parameter tuning. A more detailed comparison of neural network LM variants, including speech recognition results, is presented in Chapter 6.

4.7.2 Speed-Up Techniques

In the following, we investigate the impact of maximum entropy features as well as word classes in more detail.

Maximum Entropy Features

Concerning the standard RNN, we can add in maximum entropy features, denoted by RNN-ME, as described in Section 4.5.2. The corresponding result is given in Table 4.1. Here, a hash table storing 500 million parameters was used in combination with 5-gram features, i.e., the last four history words were considered to address the weight parameters of the direct connections. In a preliminary experiment, we did not obtain improvements by increasing the order of the features further. By using maximum entropy features, a performance very similar to that of the LSTM can be achieved. From that point of view, the RNNME model might not only be considered a speed-up technique as in [Mikolov & Deoras+ 11] but even an improvement of the RNN model. However, it should be noted that the maximum entropy extension requires a considerable amount of additional parameters. Usually, it is found that best performance is achieved when the hash table is considerably larger than the RNN model itself. As a result, it is currently not possible to train neural network LMs with maximum entropy features on large data sets using GPUs due to their limited memory size. Furthermore, the maximum entropy extension basically can be interpreted as a maximum entropy

LM that is combined with the underlying RNN. It is well-known that pure word-based maximum entropy LMs are very similar to count LMs in terms of perplexity performance [Chen & Rosenfeld 00]. Therefore, it can be expected that the integration of a maximum entropy model in an RNN LM, as represented by the RNNME model, obtains a perplexity which is similar to that of an RNN interpolated with a count LM. This is verified in Table 4.2. It is

LM	NN Weight	Dev	Test
RNN	1	128.8	122.5
Count-Based+RNN	0.64	109.4	105.0
RNNME	1	112.3	109.2
Count-Based+RNNME	0.82	108.9	105.9

Table 4.2 Neural network (NN) interpolation weights and perplexity results including interpolation with the count LM.

observed that the relative improvement by interpolation with the count LM is much smaller for the RNNME model than for the RNN. Similarly, the LM interpolation weight of the RNNME is much higher than that of the RNN, indicating that the count LM contains little information that is not already learned by the RNNME.

In addition, the RNN without maximum entropy features generalizes better on the test data than the RNNME model after interpolation with the count LM. A similar observation was made in [Mikolov & Deoras$^+$ 11] on a larger data set, where an RNNME only achieved minor speech recognition improvements compared to an RNN when both models were interpolated with a count LM. Nevertheless, the RNNME could be trained using a hidden layer size which was halved in comparison to the RNN, thereby accelerating the training and the evaluation of the model. Compared to word classes, the potential speed improvements from maximum entropy features seem much smaller, and due to the large memory requirement, in the following we concentrate on the basic neural network variants without the maximum entropy extension. Besides, the inclusion of maximum entropy features is independent of neural network variants, and any of the three architectures could be enriched in this way.

Word Class Factorization

As an alternative speed-up technique, we analyze the word class factorization of the output layer. For all the experiments, we consider an LSTM neural network with a hidden layer size of 200 each for the projection layer and the hidden layer. In principle, an arbitrary word class mapping G can be used, as long as a unique word class is assigned to every word from the vocabulary. For the experimental analysis, we investigate word classes based on frequency binning and classes that were obtained using a perplexity-based training criterion. For the latter, we distinguish between a bigram and a trigram decomposition of the likelihood of the training data. Table 4.3 summarizes the results that can be obtained with these word classes. Unlike in the case of the results in [Sundermeyer & Schlüter$^+$ 12], here we make use of a projection layer to reduce the overall number of parameters.

We observe that the particular choice of word classes has a significant impact on performance. In [Zweig & Makarychev 13, Shi & Zhang$^+$ 13] it was found that bigram perplexity-based word classes considerably improve over frequency-based word classes when combined

with a standard RNN including maximum entropy features. From the results in Table 4.3 we find that this result readily transfers to the case of LSTM neural network LMs. E.g., we can decrease the perplexity on the test data from 127.1 to 114.9 only by replacing frequency-based classes with bigram classes at the output layer. Furthermore, we extend previously reported results by increasing the order of the word class training criterion to a trigram dependence. Unfortunately this does not lead to an additional decrease in perplexity. Only when the number of word classes is very small, trigram word classes offer an advantage over bigram classes. As noted in Section 4.5.1, the class factorization does not introduce any modeling assumptions, and therefore any choice of word classes might be equally well-suited for neural network LM training, which contradicts experimental evidence. Our explanation of this result is as follows. Word classes lead to a separation of the training data into disjoint sets, such that the weight parameters $A_{3,g}$ for a class g in Eq. (4.58) are only trained on the set of training events associated with class g. By contrast, the weight parameters A_4 in Eq. (4.59) are trained on all of the training events. From that point of view, we can expect to obtain a reliable estimate of the class posterior probability $p_G(G(w_n)|w_1^{n-1})$, but a less reliable estimate of the word posterior probability $p_w(w_n|w_1^{n-1}, G(w_n))$. In case the word classes have been derived by frequency binning, there is no semantic relationship between the words within a class, and the class posterior probability $p_G(G(w_n)|w_1^{n-1})$ alone must be a poor estimate of the probability $p(w_n|w_1^{n-1})$, but the word posterior probability estimate $p_w(w_n|w_1^{n-1}, G(w_n))$ cannot compensate for this effect as it is trained on too few data. In case perplexity-based word classes are used, words within a class are semantically similar. Subsequently, the word posterior probability $p_G(G(w_n)|w_1^{n-1})$ alone is already a good estimate of $p(w_n|w_1^{n-1})$, thereby reducing the error that might by introduced by the factor $p_w(w_n|w_1^{n-1}, G(w_n))$ due to insufficient training data.

In principle, for any of the three word classing criteria, using very few or very many classes should lead to good neural network LM perplexities, as these scenarios are close to the case where no word classes are used at all. For frequency classes, this observation is well-supported by the experimental results presented in Fig. 4.11 (a), which graphically depicts perplexity performance and relates it to the speed-up per epoch over the case without word classes. When perplexity-based bigram word classes are used, the resulting neural network

Classes	Frequency		Bigram		Trigram	
	Dev	Test	Dev	Test	Dev	Test
50	135.4	129.6	126.8	120.3	127.4	120.3
100	135.0	130.5	127.0	119.8	**124.4**	**117.9**
200	134.3	128.4	123.1	117.5	132.7	125.4
400	136.0	128.9	125.5	120.1	139.5	132.9
600	134.7	129.5	**121.7**	**114.9**	136.9	129.9
800	131.7	127.1	123.2	117.3	136.5	130.0
1,000	**131.2**	**124.8**	121.7	116.9	135.5	129.4

Table 4.3 Perplexities on the Treebank corpus of an LSTM LM using word classes at the output layer. Here, bigram and trigram word classes were obtained minimizing the criterion from Eq. (1.33) and its generalization to trigrams, respectively. The resulting classes were then plugged into Eq. (4.56) for neural network training.

LM perplexity on the test data is more stable with respect to the number of word classes, as can be seen in Fig. 4.11. By contrast, for perplexity-based trigram word classes, the optimum perplexity is attained for a small number of word classes. In word class training, overfitting is prevented by limiting the number of classes or the number of context classes taken into account by the training criterion. The longer the context and the more word classes are used, the more the class model resembles an un-smoothed word-based model, which is known to generalize poorly on unseen data. For trigram word classes, this effect can only be remedied by limiting the number of word classes. Accordingly, in Fig. 4.11 (c), the best perplexity is obtained at about 100 classes only. A similar behavior was also observed in the context of count-based LMs in [Botros 15].

Fig. 4.11 also shows the speed-up obtained by using word classes, where the runtime results are based on the `rwthlm` LSTM implementation from [Sundermeyer & Schlüter+ 14]. Here, the training of a single epoch using word classes at the output layer is compared with using the full vocabulary without word classes. Even though the vocabulary size is only 10,000 for the Treebank corpus, the training time per epoch can be reduced by more than a factor of ten. As described in Section 4.5.1, the speed-up should be largest for $\sqrt{10{,}000} = 100$ classes. This holds rather well in practice, where the maximum speed-up is achieved at about 150 classes, for all word classing criteria. The offset is simply due to the fact that for the theoretic result it was assumed that all word classes contain the same number of words, which is not the case for any of the word classing techniques. Therefore it is noteworthy that the observed speed-up by word classes is very consistent among different criteria. In Fig. 4.12 the size of the individual word classes is shown for the three criteria.

According to Fig. 4.12 (a), class sizes differ greatly for word classes derived by frequency binning. While a significant amount of classes only comprises a single word, beyond single-word classes we see an exponential increase in the class size. This is directly related to Zipf's law, which basically states that for the relative frequency p_i of the i-th word from the vocabulary, we have

$$p_i \propto \frac{1}{i}. \tag{4.103}$$

Frequency binning fills a class with words sorted by relative frequency in descending order until the average relative class frequency $1/|\mathcal{G}|$ is reached, where $\mathcal{G}$ denotes the set of word classes. Therefore, we have

$$\frac{1}{|\mathcal{G}|} \propto \sum_{i=i_0}^{i_1} \frac{1}{i} \approx \log \frac{i_1}{i_0} \tag{4.104}$$

for a class comprising the word indices $i_0, \ldots, i_1$, where we have used the relation $\sum_{i=1}^{k} \frac{1}{i} \approx \log k + \text{const}(k)$ for the k-th harmonic number, which shows the exponential growth in class size for frequency binning.

The class sizes for bigram and trigram word classes are depicted in Fig. 4.12 (b). Compared to frequency binning, the class sizes are much more balanced, and there is virtually no difference between the bigram and trigram criterion.

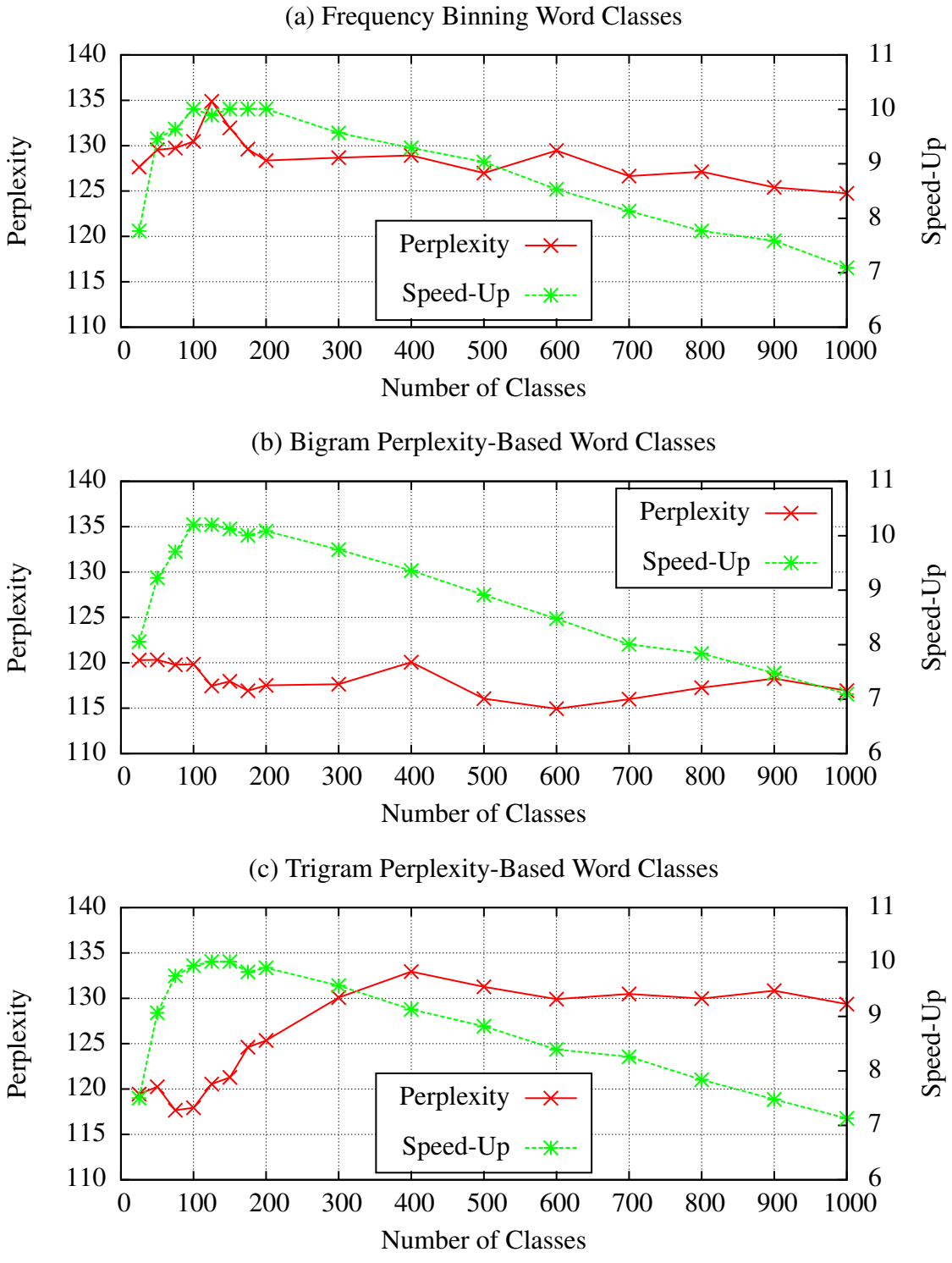

Figure 4.11 Perplexity performance of an LSTM using word classes and speed-up over the baseline method without word classes on the Treebank test corpus.

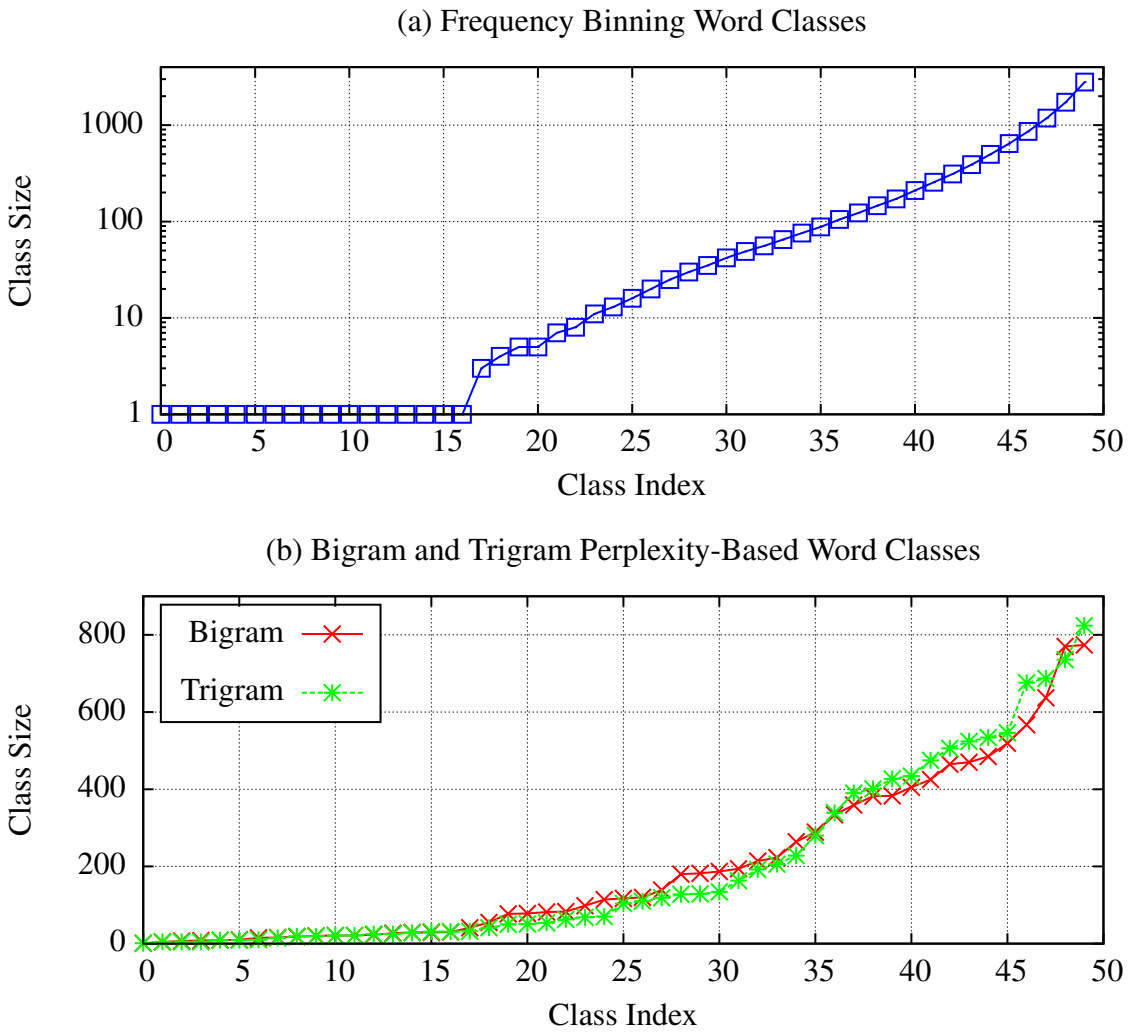

Figure 4.12 Distribution of class sizes for different word class training criteria. For all criteria, 50 word classes were trained on the Treebank corpus. In the upper figure, a logarithmic scale for the vertical axis is used.

4.7.3 Sequence Definitions

In Section 4.6.2 three different types of sequences have been proposed that can be used in combination with epochwise BPTT. We evaluate the impact of sequence types on perplexity. Furthermore, we can also vary the length of the resulting sequences, which is analyzed as well. In all the experiments, we use the same sequence definitions for the training data as well as the test data.

Sequence Lengths

In Fig. 4.13 the perplexity in terms of the sequence length is depicted on the Treebank development and test data. All experiments are performed using an LSTM with a projection layer and hidden layer each of size 200. Conceptually, it can be expected that short sequences are not optimal, as the LSTM cannot learn longer contexts, and the LSTM history is reset

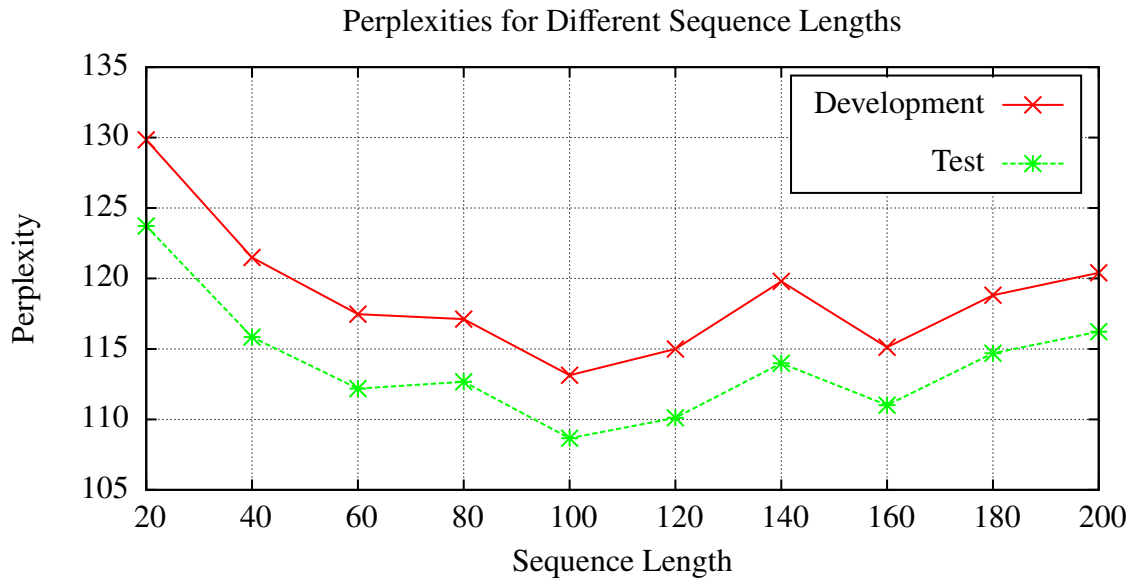

Figure 4.13 Perplexity results in terms of the sequence length on the Treebank corpus. Sequences were composed of a fixed number of consecutive words.

at the beginning of each sequence. On the other hand, when sequences get too long, weight updates are less frequent and therefore convergence of stochastic gradient descent training is worse, so that both effects need to be balanced. From Fig. 4.13 it can be seen that the optimum perplexity is obtained at 108.7 on the test data, for a sequence length of 100 words. Another aspect relates to the preprocessing of the data. In the experiments, we make use of the preprocessing proposed in [Mikolov & Karafiát$^+$ 10], where the order of the sentences was shuffled. From that point of view, it does not seem necessary to train the LSTM using overly long sequences, because two subsequent sentences may not be related.

Sequence Definitions

The effect of the different sequence types is shown in Table 4.4 for the Treebank data. By verbatim, we denote a processing where the sentences are directly used as sequences for epochwise BPTT. By the concatenated sequence type, we refer to the concatenation of multiple sentences up to a maximum length of 100 words, and for the fixed sequence type, sequences are always exactly 100 words long. The effect of different sequence types is small,

Sequence Type	Dev	Test
Verbatim	116.5	111.3
Concatenated	115.4	109.9
Fixed	**113.9**	**108.0**

Table 4.4 Perplexity results on the Treebank corpus for different sequence definitions.

where best performance is obtained for a fixed sequence length. Nevertheless it seems to be beneficial to concatenate multiple sentences instead of only processing sentences inde-

pendently. Due to this experiment, all previous results were based using the fixed sequence type.

For comparison, here we also extend our results to the French Quaero corpus as described in Section A.3.2, where the natural order of the sentences is preserved. The neural network LM is composed of a projection layer and an LSTM layer, where both layers comprise 500 units. Due to the large hidden layer size, we restrict the amount of training data to 27 M running words per epoch, corresponding to 30 M running words when counting sentence boundary tokens. The perplexities are given in Table 4.5. Again, we do not find a significant impact on perplexity performance by sequence definitions. It does not seem to be crucial whether a sequence starts at the beginning of a sentence or not. As perplexities

Sequence Type	Dev	Test
Concatenated	**92.3**	**104.1**
Fixed	93.4	104.8

Table 4.5 Perplexity results on the Quaero French corpus for different sequence definitions.

are slightly lower for the concatenated sequence type, we stick to this variant for all experiments on the Quaero data. A verbatim sequence definition was not investigated, because the sentence length varies greatly within the Quaero corpus. As the corpus is much larger than the Treebank corpus, for efficiency reasons multiple sequences were processed in parallel, which is only beneficial when sequences are similar in length.

4.7.4 Projection Layer

Most neural network LM architectures make use of a projection layer. Its effect on perplexity performance is summarized in Table 4.6, where different kinds of projection layers are combined with an LSTM hidden layer of size 200. The projection layers are always of the same size as the LSTM layer. Three different cases are distinguished: Either the input data

Projection Layer	Dev	Test
None	113.4	109.7
Linear	**113.0**	**109.5**
Sigmoid	125.9	121.1

Table 4.6 Perplexities on the Treebank corpus for different combinations of projection layers with an LSTM hidden layer. Results are reported based on a modified version of the RNNLIB toolkit.

are directly fed into the LSTM layer, or the input data are projected onto a continuous space using a projection layer first, where in turn we distinguish an identity activation function or a sigmoid activation function. In the case of an identity function, the mapping from words to the continuous space is completely linear.

Based on these results, we conclude that for LSTM networks, a linear projection layer does not have a significant impact on perplexity, and from a conceptual point of view it might also be left out. A sigmoidal projection layer even seems to make the learning problem

more difficult, and to the best of our knowledge, its use has never been reported in the language modeling literature. (According to [Bengio & Ducharme$^+$ 03], a non-linear activation function would not add useful information, which we can confirm here on an experimental level.) However, as noted in Sections 4.6.3 and 6.1.5, a projection layer can greatly reduce the number of parameters of an LSTM neural network LM. For this reason, the `rwthlm` toolkit [Sundermeyer & Schlüter$^+$ 14] does not support LSTM networks without a projection layer, and subsequently, the above results were obtained using a modified version of the RNNLIB software.

For RNN LMs, a projection layer is not needed to limit the model size. The effect on perplexity on the Treebank corpus is shown in Table 4.7. Again, it can be seen that a projection layer only gives very small improvements on the test data. Therefore, in the following we will not use a projection layer for standard RNN architectures.

Projection Layer	Dev	Test
None	**128.8**	122.5
Linear	129.4	**122.4**

Table 4.7 Different projection layer configurations for a standard RNN. The result for the RNN without a projection layer was obtained with the `rnnlm` toolkit from [Mikolov & Kombrink$^+$ 11b].

Unlike for recurrent neural network variants, for feedforward networks, the projection layer fulfills a dual purpose. On the one hand, it maps discrete word information to a continuous space, on the other hand, it introduces a tying of weight parameters, as for any history word, the same weight matrix is used. This tying is used in virtually any work on feedforward networks, e.g. in [Bengio & Ducharme$^+$ 00, Bengio & Ducharme$^+$ 03, Schwenk 07, Le & Oparin$^+$ 11, Vaswani & Zhao$^+$ 13, Devlin & Zbib$^+$ 14]. We can compare this tied projection matrix to the case where an individual matrix is used for each history word. Regarding Eq. (4.2), this means that not the projection vectors are concatenated, but the matrix A_1 is multiplied with a vector consisting of the concatenated one-hot encoded representations of the history words. The resulting numbers are given in Table 4.8. We find that we obtain nearly identical perplexities, regardless of whether the projection layer weights are tied across history word positions or not. In summary, a tied projection layer is clearly favorable for feedforward networks, and will be used for all experiments in this thesis.

Projection Layer	Dev	Test
Tied	132.6	127.6
Untied	**132.6**	**127.3**

Table 4.8 Perplexities on the Treebank corpus for a tied and untied projection layer of a feedforward neural network LM. The numbers are from [Freiberg 13].

4.7.5 Input Data Standardization

The standardization of input features has become a common preprocessing step in acoustic modeling, where neural networks are trained on real-valued input features, cf. Section 1.1.1.

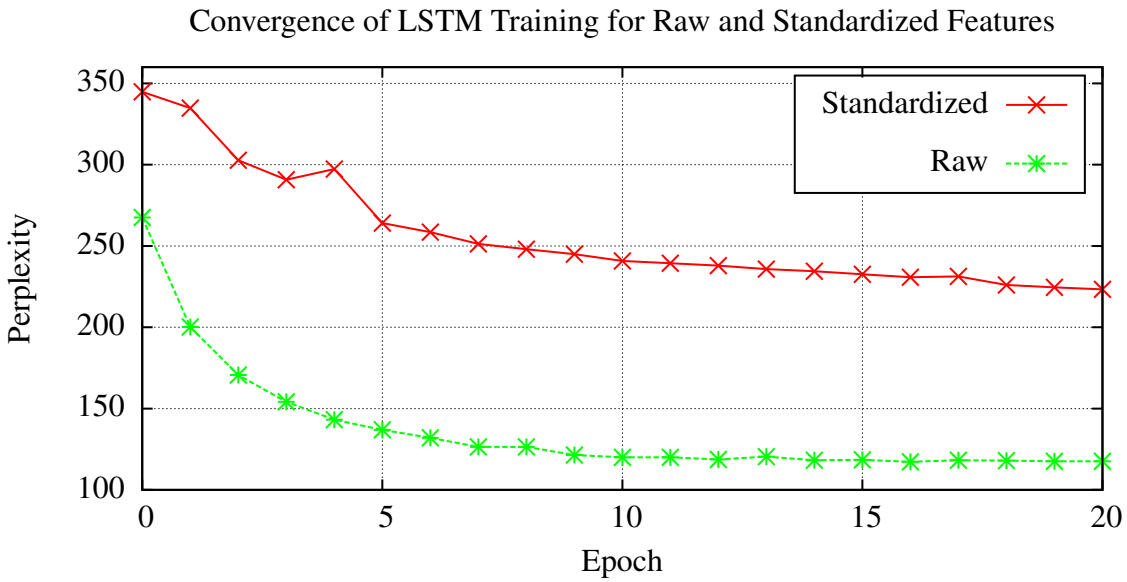

Figure 4.14 Convergence speed on the Treebank development corpus for raw one-hot encoded input data and standardized input features.

For the discrete word input features used in language modeling, we obtained the following convergence behavior, depicted in Fig. 4.14. For the experiment, an LSTM with 200 hidden nodes, including a projection layer, was used. We stick to the exact formula (4.76) instead of the asymptotic limit given in Eq. (4.78).

It can be observed that by standardizing the one-hot encoded input data, the convergence speed of the neural network LM was actually slowed down. However, even without any standardization, the components of a one-hot encoded feature vector already lie in the $[0, 1]$ interval. Furthermore, standardization transforms the sharp peaks of one-hot encoding to a smeared word representation, where all vector components are non-zero. Thus, it might be harder to learn the correspondence between input words and the output word to be predicted.

In addition, one-hot encoding is also favorable in terms of processing speed, as the sparsity of the input data can be exploited for fast calculation of the projection layer activations. Because no non-linearity is used, the matrix-vector multiplication from Eq. (4.2) simplifies to a table lookup of the weight parameters corresponding to the given input word [Le & Allauzen$^+$ 10]. By contrast, the complexity of the projection layer calculation in the case of standardized input features would be the same as that of the output layer.

4.7.6 Integrated Training

In analogy to the integration of a maximum entropy model into the neural network architecture, in Section 4.6.5 the integration of a count LM into the neural network training criterion was proposed.

For the experimental analysis, we do not stick to the Treebank corpus due to the need for a cross-validation-like training. Instead, we make use of the Wall Street Journal corpus described in Section A.1, which was already investigated for the experiments presented in

Section 3.3.1. The count LM is a Kneser-Ney smoothed 5-gram LM which was trained on the standard training set of the corpus. Concerning the neural network, an LSTM was trained with a linear projection layer and a hidden layer each comprising 300 hidden nodes. The LSTM training data consisted of 13.9 M running words which were disjoint with respect to the count LM training data of 227.3 M running words. For speed-up, the 20 K vocabulary was partitioned into 50 word classes using frequency binning. These models and in particular the linear interpolation of the two LMs form the baseline for the integrated training approach.

In a next step, we trained an LSTM with the exact same configuration according to the criterion given in Eq. (4.87). One question arising is how to determine the interpolation weight λ of the integrated training. Conceptually, it is possible to consider the interpolation weight in the same way as the remaining neural network parameters, and optimize λ using SGD as well. However, this might be problematic, because λ values can be expected to fluctuate over the course of the training process. At the beginning of training, the neural network weights are randomly initialized, resulting in poor neural network probability estimates. The likelihood of the training data is therefore maximized by decreasing λ. As training converges, the λ value will start to increase again. Also, as can be seen from Eqs. (4.94), (4.98) and (4.102), the error terms of the neural network parameters are scaled by the interpolation weight λ, i.e., the learning speed is sensitive to the current value of λ, and it is not clear whether typical learning rate schedules as the one described in Fig. 4.5 work well for this scenario. For simplicity, we performed a grid search over a range of λ values. By running multiple LSTM trainings, where λ was kept fixed, the interpolation weight was optimized.

The resulting perplexities on the development data are shown in Fig. 4.15. It can be seen that the integrated approach results in significantly lower perplexities in relation to the stand-alone LSTM as well as the count LM. We can compare the λ optimization curve of the integrated training with the one obtained by standard linear interpolation. We observe that both curves exhibit a similar behavior, where the optimum is attained at an interpolation weight of $\lambda = 0.4$ for the integrated approach, and a weight of $\lambda = 0.5$ in the case of the independently trained LMs. Perplexity results are summarized in Table 4.9, which also includes test set performance.

LM	Dev	Test
Count-Based	70.2	76.1
LSTM	77.1	83.1
Linear Interpolation	**53.3**	**57.8**
Integrated Training	55.0	59.7

Table 4.9 Perplexities of the count LM, the LSTM LM, their linear interpolation in the integrated training approach. All models are trained independently of each other.

On the development data, we find that the integrated training approach does not improve performance over the baseline method, even though the results are close, where the perplexities are 53.3 and 55.0 for the linear interpolation and the integrated training, respectively. The same tendency can also be observed on the test set. The situation on the test data indicates that there is no advantage of optimizing the interpolation parameter directly on the development data (conventional interpolation) over an implicit optimization via early stopping (integrated training).

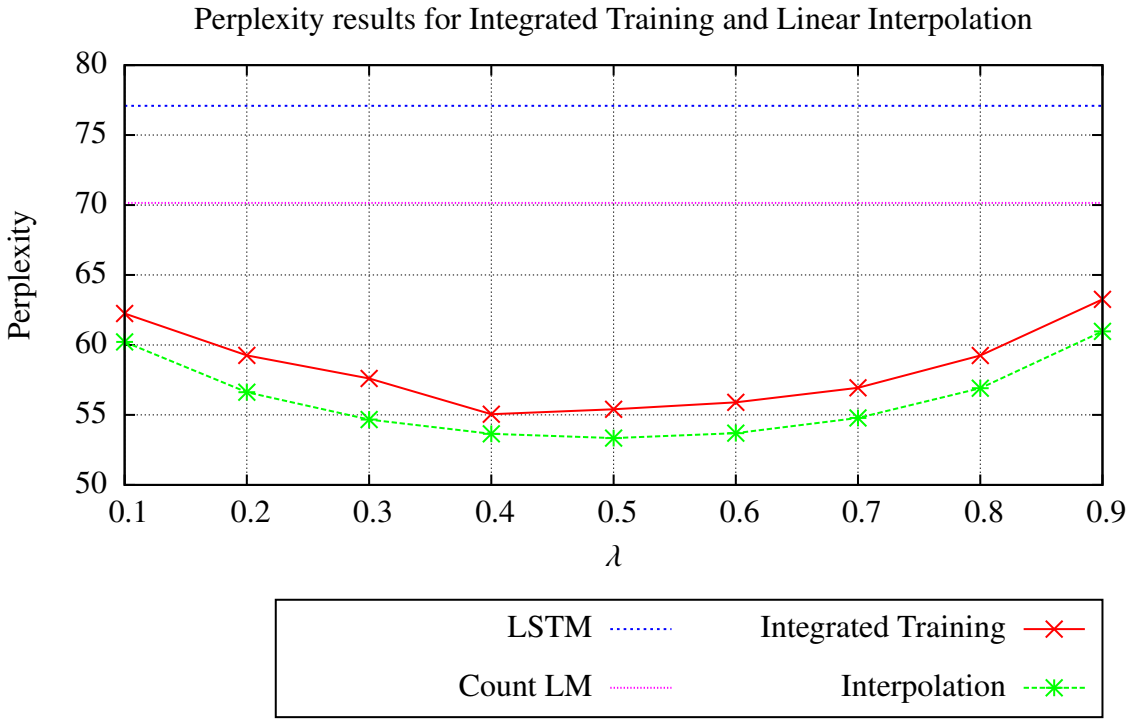

Figure 4.15 Perplexity results on the WSJ 20 K development data for linear interpolation of independently trained models, compared to the integrated training.

The correctness of the experimental result was verified by comparing the performance of the models on the training data. Here, the integrated model obtained a perplexity of 41.7 for the optimized interpolation weight of 0.4, whereas the perplexity of the conventional interpolation was only 43.2 in that case. It can thus be concluded that integrated training only improves the training criterion, but the improvements do not generalize to unseen data.

In summary, it seems surprisingly difficult to improve over the linear interpolation of separately trained LMs by integrating multiple models into a single framework. Beyond the integrated training approach, a similar result was already found in Section 4.7.2 in the case of a maximum entropy model that was combined with an RNN.

4.8 Summary

In this chapter, we gave an overview of existing feedforward and recurrent neural network LM architectures and presented algorithms for learning the neural network weight parameters. We outlined drawbacks of conventional neural network architectures. Subsequently, we introduced the LSTM concept to the field of language modeling, which addresses the commonly observed training instabilities of standard RNN approaches. The training method that we employ for LSTM LMs differs from widely used RNN LM training techniques in that we do not make use of an approximate gradient computation. Instead, we obtain an exact gradient. Our experimental results indicate that the LSTM method outperforms other neural network LMs by a considerable margin. This is investigated in more detail in Chapter 6.

These favorable results indicate that there is no strict need for approximations in the gradient computation to obtain state-of-the-art performance.

We discussed two methods how the training of neural network LMs can be accelerated. Regarding the first, maximum entropy features can be integrated into an RNN, but according to experimental evidence, they might anticipate improvements that could also be obtained by the interpolation with a count LM. Furthermore, we investigated a word class factorization of the output layer. We found that this method is well suitable for combination with the LSTM concept. Experiments showed that even on a corpus with a small-sized vocabulary, a speed-up of one order of magnitude could be obtained, at a small degradation in perplexity.

Furthermore, we proposed several ways how to build sequences for LSTM training from the training sentences and evaluated the techniques in experiments. In addition, we showed how neural network probabilities can be accurately normalized for perplexity evaluation on large vocabularies without additional costs.

Finally, we investigated several ways how to improve neural network LMs, including different types of projection layers, the standardization of word input features, and a modified training criterion for the integration of a count LM into a neural network LM. With any of the three approaches, it turned out to be difficult to improve perplexity performance of an LSTM LM baseline further when the LSTM was interpolated with a count LM. However, a projection layer was found to be very useful, as it reduced the number of parameters of feedforward and LSTM networks, without having a significant effect on perplexities.

Chapter 5

Rescoring with Neural Network Language Models

In the previous chapter, we have discussed neural network techniques for computing probabilities of arbitrary word sequences. However, the application of neural network LMs to practical problems is not straightforward. In the following, we will concentrate on the use of neural network LMs for speech recognition, but the methods presented can easily be transferred to other applications like e.g. machine translation or handwriting recognition.

The main problem lies in the long-range context of neural network LMs, which complicates the search problem encountered in speech recognition. While standard count LM probability estimates usually do not depend on more than four predecessor words, feedforward networks can easily take into consideration twice as many predecessor words, and for recurrent networks, in theory the context length is unbounded. In addition, a conventional m-gram count LM often does not even make use of the $(m-1)$ most recent history words, but backs off to lower orders, so that ultimately only a bigram or even a unigram dependence may actually be needed in decoding. These properties of count LMs are exploited by the so-called recombination technique in standard speech recognition decoders, cf. [Ney & Ortmanns 00].

The scientific problem addressed in this chapter amounts to how long-range neural network LMs can be used for search in the absence of backing-off hierarchies that facilitate recombination. This leads to the question how many of the predecessor words are actually exploited by recurrent neural network variants. Obviously, there must be a limitation on the maximum context dependence of an RNN or LSTM, as such a model cannot store arbitrarily long history information in its hidden layer activations.

5.1 Decision Rules

The problem of search is related to the decision rule that is used for decoding. In fact, it is always Bayes' decision rule that search is based on, but there are differences in the way it is applied to speech recognition. As noted in Section 1.1, most commonly the word

sequence $\hat{w}_1^{\hat{N}}$ is chosen according to the rule

$$\hat{w}_1^{\hat{N}} = \arg\max_{w_1^N} \left\{ p(w_1^N) p(x_1^T | w_1^N) \right\} \tag{5.1}$$

for a given acoustic observation x_1^T. This is equivalent to the decision rule

$$\hat{w}_1^{\hat{N}} = \arg\min_{w_1^N} \left\{ \sum_{\tilde{w}_1^{\tilde{N}}} p(\tilde{w}_1^{\tilde{N}} | x_1^T) \cdot \left(1 - \delta_{\tilde{w}_1^{\tilde{N}} w_1^N}\right) \right\}, \tag{5.2}$$

where δ denotes the Kronecker delta. As can be seen from Eq. (5.2), this decision rule minimizes the sentence error rate in speech recognition. The decision rule is associated with the Viterbi method, where the term Viterbi also implies that according to Eq. (5.1) the joint probability $p(w_1^N, x_1^T)$ is not maximized for the most likely word sequence but only for the most likely path through the search graph.

In speech recognition usually the word error rate is used to evaluate the performance of a recognizer. Therefore, it seems more promising to apply the following decision rule instead, that minimizes the word error rate

$$\hat{w}_1^{\hat{N}} = \arg\min_{w_1^N} \left\{ \sum_{\tilde{w}_1^{\tilde{N}}} p(\tilde{w}_1^{\tilde{N}} | x_1^T) \cdot L(\tilde{w}_1^{\tilde{N}}, w_1^N) \right\}, \tag{5.3}$$

where L denotes the Levenshtein distance between the word sequences $\tilde{w}_1^{\tilde{N}}$ and w_1^N.

It seems infeasible to evaluate this decision rule during speech decoding due to the non-local loss function. Consequently, decision rule 5.3 was first introduced in [Stolcke & Konig$^+$ 97] by an evaluation on n-best lists only. As the search space given by n-best lists is very limited, the idea was transferred to word lattices in [Mangu & Brill$^+$ 99, Mangu & Brill$^+$ 00], which greatly improved the performance over the n-best list approach. The main idea is to introduce a multiple word alignment which simplifies the evaluation of the loss function such that it can be evaluated locally on the word level. The multiple word alignment is referred to as a *confusion network*, and the confusion network-based algorithm for approximately minimizing the word error rate was named the *consensus method*.

5.2 Application of Neural Network Language Models to Automatic Speech Recognition

The two decision rules described in the previous section can be applied to different search space representations in speech recognition. More precisely, we can distinguish whether an n-best list, a word lattice, or the full search space encountered during speech decoding forms the basis of the summation carried out in Eqs. (5.2) and (5.3).

The simplest approach for turning the perplexity improvements of neural network LMs into word error rate reductions is by rescoring n-best lists. The neural network LM probability is computed for each entry of the list, and the best-scoring hypothesis is selected as the final result [Mikolov & Karafiát$^+$ 10]. In [Kombrink & Mikolov$^+$ 11] and [Si & Zhang$^+$

13], an improved n-best list rescoring was investigated, reducing the computational effort by caching and compressing the lists into a prefix tree. Both aforementioned decision rules can be applied when rescoring n-best lists, and the neural network LM can also be used without approximations, regardless of the context size of the neural network. However, the small search space size does not show the full potential of neural network language modeling techniques, as observed in [Sundermeyer & Oparin$^+$ 13].

A better representation of the search space is obtained by rescoring lattices instead of n-best lists. Lattices are usually created with a count LM using a context size of at most four words. If a feedforward neural network LM is used, it is possible to simply replace the count LM estimates with those of the neural network model, and to use standard rescoring algorithms directly on the lattice [Schwenk 07]. The replacement step may require an expansion of the lattice [Weng & Stolcke$^+$ 98]. E.g., if the order of the count LM is m, and the neural network LM is of order $(m+1)$, a word arc in the lattice may have two different predecessor paths of length $(m+1)$ that match only in the most recent m word positions. As a result, assigning a probability to the word arc according to the neural network LM would be ambiguous, and the two paths have to be kept separate for rescoring. In practice, lattice expansion can dramatically increase the size of a lattice, and it becomes too expensive if the difference in LM order is large. In the case of an RNN, the expanded lattice degenerates into a prefix tree. For these reasons, with higher order neural network LMs an approximate rescoring is necessary for lattices. When the Viterbi method is used, only the single best path, and, to some extent, its competing hypotheses need to be evaluated with the neural network LM. In the case of the consensus method, neural network LM probability estimates are needed for all hypotheses encoded in the lattice, because the algorithm requires the computation of word posterior probabilities, which is commonly performed on a word lattice.

For neural network LMs, lattice rescoring was first addressed in [Deoras & Mikolov$^+$ 11], where a hill climbing algorithm was presented. In [Auli & Galley$^+$ 13], push-forward rescoring for RNN LMs was introduced in a machine translation setting. Both algorithms are based on the Viterbi method. They do not facilitate the rescoring of all hypotheses in a word lattice with an RNN LM, therefore it is prohibitive to minimize the word error rate directly based on the consensus method. This was subsequently addressed in [Liu & Wang$^+$ 14] and [Sundermeyer & Tüske$^+$ 14]. While [Liu & Wang$^+$ 14] created a rescored lattice directly based on an algorithm from [Liu & Gales$^+$ 13] without consideration of acoustic model scores, in [Sundermeyer & Tüske$^+$ 14] the rescored lattice was obtained as a by-product of a push-forward processing, taking advantage of all available knowledge sources.

Recently, research also focussed on using RNN LMs directly in decoding. Only the Viterbi method was investigated. In [Huang & Zweig$^+$ 14], a fine-grained cache architecture was presented to reduce the computational overhead incurred by the RNN, but compared to push-forward rescoring, a degradation in word error rate was observed. In [Hori & Kubo$^+$ 14], an RNN LM was integrated into a weighted finite state transducer-based decoder, which helped reducing the latency of the speech recognition process, but only led to very small gains in word error rate over lattice rescoring. In addition, no higher order expansion of the RNN was considered, and pruning was kept fixed at tight values for lattice rescoring. In summary, there is little evidence that it is necessary to integrate an RNN LM into decoding to obtain optimum word error rate results. However, as noted in [Sundermeyer & Oparin$^+$ 13] and [Sundermeyer & Tüske$^+$ 14], it is important to obtain a rescored lattice incorporating

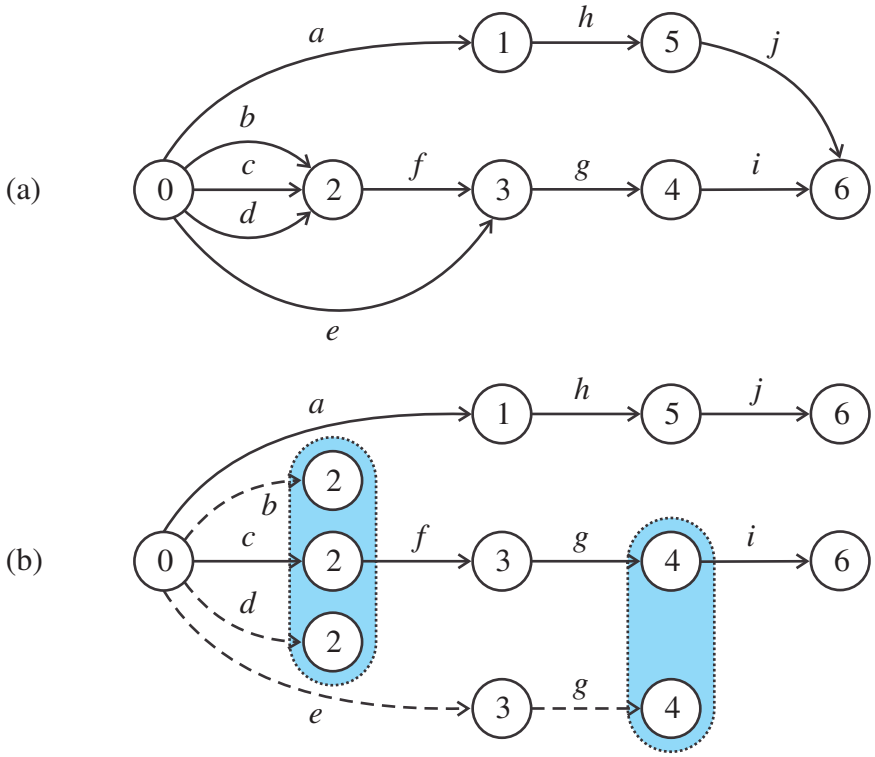

Figure 5.1 (a) Example word lattice and (b) a corresponding traceback tree. Dashed arcs correspond to paths that will be pruned, and background blue indicates recombined paths.

neural network LM probabilities to facilitate the consensus method for improved speech recognition performance.

5.3 Push-Forward Rescoring

As the methods presented in this chapter are based on push-forward rescoring from [Auli & Galley$^+$ 13], we introduce this algorithm in more detail. Given a lattice-based search space, the algorithm extracts the single best hypothesis according to the neural network. To this end, it partially expands the word lattice into a prefix tree.

A lattice can be represented as an undirected graph consisting of nodes and arcs. An example is depicted in Fig. 5.1 (a), where the vocabulary consists of the words *a* to *j*. Each arc is labeled with a word recognized in decoding, and each node is associated with a corresponding word end time. For a node, a list (implemented as a heap data structure) of hypotheses is maintained, where a hypothesis stores the RNN hidden layer that was obtained by evaluating the RNN on the word labels observed on the individual path from the start node of the lattice up to the current node. The nodes are visited in topological order, starting from the initial node of the lattice. At a given node, for each hypothesis and each outgoing arc, the RNN state from the hypothesis is expanded by the word label of the arc, and the resulting RNN state is added to the list of hypotheses at the successor node. The pseudocode of the algorithm is given in Fig. 5.2.

```
Input: nodes V, arcs E, initial node s, final node f, traceback T, pruning threshold k
Output: best scoring hypothesis h and traceback T
V' ← SortTopologically(V);
foreach v ∈ V' do
    H(v) ← MaxHeap();
end
H(s).Push(0);
foreach v ∈ V' do
    foreach (v, v') ∈ E do
        foreach h ∈ H(v) do
            h' ← EvaluateRNN(h, Word(v, v'));
            SetPredecessor(h, h', T);
            H(v').Push(h');
            while |H(v')| > k do
                H(v').Pop();
            end
        end
    end
end
while |H(f)| > 1 do
    H(f).Pop();
end
return (H(f).Top(), T);
```

Figure 5.2 Pseudocode for push-forward rescoring of word lattices with an RNN LM, adapted from [Auli & Galley+ 13].

To reduce the computational effort, the number of hypotheses per node is limited to a maximum value k, retaining only the best scoring hypotheses, which we refer to as cardinality pruning. In principle, it suffices to only use this pruning technique, but then it can be observed that often the hypotheses not being pruned are essentially the same. To increase the diversity among hypotheses after pruning, it is possible to retain only the single best hypothesis for a given m-gram [Boulanger-Lewandowski & Bengio+ 13], which we denote by recombination pruning. This means that recombination is enforced, even though from a conceptual point of view an RNN LM does not allow recombination—unless there are word sequences, leading to the current lattice node, that are identical.

5.3.1 Extensions

Push-forward rescoring was introduced in the context of a machine translation task. In speech recognition, word end time information is associated with the lattice nodes and can be used for more efficient pruning. As noted in [Sundermeyer & Tüske+ 14], to this end the lattice nodes can be sorted by time in ascending order. Traversing the lattice in a time-synchronous fashion has the advantage of allowing to compare different hypotheses that correspond to the same portion of the acoustic signal. This facilitates efficient beam pruning

on a lattice node level like in speech decoding [Ney & Ortmanns 00]. The most likely hypothesis for a word end time is obtained, and all hypotheses having a probability lower than the best one, multiplied by an empirically determined factor, are pruned.

In addition, other than in decoding, it can be exploited that the complete acoustic and LM probabilities from the count LM are already present on the lattice arcs. Therefore, it is possible to incorporate the probabilities of future word arcs at the current word position, either by taking into account the sum over all future word arcs, or by considering the single best path only. This is important for minimizing the computationally expensive evaluation of the neural network LM.

Another issue arising in rescoring relates to the interdependence of consecutive word sequences. If the speech data is separated into multiple utterances, it is most convenient to rescore each utterance independently of the others, which also simplifies parallelized rescoring. However, conceptually it would be necessary to consider any hypothesis for the previous utterance as a candidate RNN state initialization for rescoring the current utterance. As a compromise, in this work the rescoring is performed either independently, or the single best previous hypothesis is used as initialization for the rescoring of the current utterance.

5.4 Approximations for Lattice Rescoring

Push-forward rescoring focusses on the Viterbi method for minimizing the sentence error. On the other hand, many speech applications require a set of multiple hypotheses, e.g. the consensus method for minimizing word error [Mangu & Brill$^+$ 99], the computation of word confidences which relates to obtaining word posterior probabilities [Wessel & Schlüter$^+$ 01], or keyword search [Fiscus & Ajot$^+$ 07, Mangu & Soltau$^+$ 13]. Thus, in this section two extensions of the push-forward algorithm are described that not only extract the single best hypothesis from a lattice with respect to a neural network LM, but also create a new lattice including neural network LM probability estimates for all paths from the original lattice. To this end, it is helpful to make use of a so-called traceback data structure [Ney & Ortmanns 00]. The traceback is a prefix tree that stores all the paths that were considered by push-forward rescoring, as summarized in Fig. 5.2. The tree structure makes it possible to extend a partial path by another word arc from the lattice in constant time, regardless of the length of the partial path. In Fig. 5.1 (b), an example traceback is shown after a hypothetical rescoring run on the lattice from Fig. 5.1 (a), where we assume that the paths *b–f–g–i* and *d–f–g–i* have been pruned at node 2, and the path *e–g–i* has been pruned at node 4. As a result, the example traceback tree contains three paths not ending at the final node 6, which is indicated by dashed arrows. The two extensions of push-forward rescoring described in the following make use of this data structure.

5.4.1 Replacement Approximation

We first restrict to the case where the new lattice has the same topology as the original one, i.e., only the LM probabilities are replaced. We can sort the paths from the traceback tree by the word end time of the last node and the probability of the path in descending order. Then, for each path from the traceback tree, the word labels are followed in reverse order, and the

LM probabilities associated with the traceback arcs are written on the corresponding arcs of the word lattice. In case a lattice arc is visited a second time, the LM probability of the arc does not get updated again. In this way, an arc gets assigned the word probability it obtained on the overall best path in the traceback tree. In addition, if the lattice encodes an n-best list, this approximate lattice rescoring strategy leads to an exact rescoring. The pseudocode for this algorithm is given in Fig. 5.3.

Input: nodes V, arcs E, initial node s, final node f, traceback T, pruning threshold k
Output: Updated LM probabilities for the arcs E of the input lattice
`RescorePushForward`(V, E, s, f, T, k);
foreach $e \in E$ **do**
  $p_{\text{LM}}(e) \leftarrow -\infty$;
end
$visited \leftarrow \emptyset$;
$T' \leftarrow$ `SortTopologicallyAndByProbability`(T);
foreach $t' \in T'$ **do**
  $t \leftarrow t'$;
  while $visited \neq T' \wedge t \notin visited$ **do**
    $e \leftarrow$ `LatticeArc`(t);
    if $p_{\text{LM}}(e) < 0$ **then**
      $p_{\text{LM}}(e) \leftarrow p_{\text{LM}}(t)$;
    end
    $visited \leftarrow visited \cup \{t\}$;
    $t \leftarrow$ `Predecessor`(t);
  end
end

Figure 5.3 Pseudocode for rescoring with the replacement approximation. Only the probabilities on the lattice arcs E are replaced.

5.4.2 Traceback Approximation

As an alternative, we can directly make use of the traceback tree and convert it into a lattice, incorporating the neural network LM probabilities. Obviously, this will lead to an increased size of the rescored lattice because of the on-the-fly expansion. In a first step, we create a new final node, and connect all nodes from the traceback tree, that correspond to the final node of the original lattice, to the new final node via ε transitions.

Paths from the traceback tree that do not end at the final node have been pruned during push-forward rescoring. A lattice may not contain pruned paths, thus pruned paths need to be extended to paths ending at the final node, in order to include them in the new lattice. We recombine a path that has been pruned at a particular lattice node with another path that survived pruning at this node. In the example of Fig. 5.1 (b), there are two paths that have been expanded up to lattice node 4: The pruned path e–g, and the path c–f–g that has survived pruning at node 4. Recombining the paths means that we merge the two traceback

nodes corresponding to lattice node 4 in the new lattice. As a result, we extend the pruned path e–g to a complete path e–g–i that ends at the final lattice node 6.

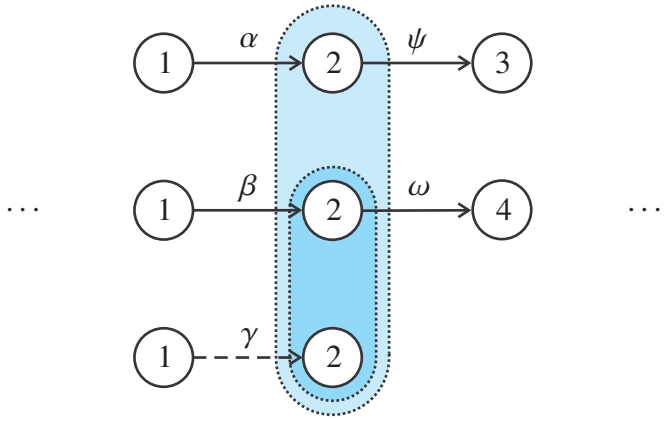

Figure 5.4 Example for the traceback approximation, when multiple paths for recombination are available.

In the general case, there may be multiple surviving paths that can be used for recombination with a pruned path. For recombination we choose the partial path whose probability up to the current lattice node is closest to the probability of the path that has been pruned. Furthermore, we ensure that the probability of the path that is used for recombination is also higher than that of the pruned path. Because of recombination pruning, a surviving path is not necessarily better than a pruned one: Recombination pruning will keep only the best path ending in a given m-gram, and if paths ending in two different m-grams w_{n-m+1}^{n} and $\tilde{w}_{n-m+1}^{n}$ survive beam pruning, then the second best path ending in context w_{n-m+1}^{n} may be better than the best path ending in $\tilde{w}_{n-m+1}^{n}$. We ensure that the path for recombination is better than the pruned one to enforce that the best path in the rescored lattice is the same as the best path obtained by push-forward rescoring. In other words, we want to avoid the creation of a better path. An example is depicted in Fig. 5.4, showing a part of a traceback tree. Numbers indicate nodes from the original lattice, and Greek letters denote probabilities of the corresponding partial paths. We assume that the paths α–ψ and β–ω survive pruning, but γ is pruned. Therefore we need to continue the γ-path by either the ψ-path or the ω-path. We assume $\beta < \gamma < \alpha$ as well as $\beta\omega < \alpha\psi$ and $\alpha\psi < \gamma\omega$, implying $\psi < \omega$. The best path found by push-forward rescoring is $\alpha\psi$, therefore we may not recombine γ with β, as this would create the new path $\gamma\omega$ that is more likely than the previously best path.

The original lattice contains the same number of paths as the rescored lattice, but it can be larger than the original one. The increase in lattice size can be dynamically controlled by adjusting the pruning parameters of push-forward rescoring. The pseudocode of the algorithm is summarized in Fig. 5.5.

5.4.3 Rescoring Setups

From the two decision rules investigated in speech recognition and the different rescoring variants described in the previous sections, the following five rescoring setups can be derived.

1. Push-forward rescoring with the Viterbi method. The algorithm from Fig. 5.2 is applied to directly find the most likely word sequence.

```
Input: nodes V, arcs E, initial node s, final node f, traceback T, pruning threshold k
Output: Rescored and expanded word lattice
RescorePushForward(V, E, s, f, T, k);
// Convert traceback to lattice format.
pruned ← PrunedTracebackNodes(T, f);
final ← FinalTracebackNodes(T, f);
f' ← AddNewFinalLatticeNode();
WriteNodes(T, f', pruned);
WriteArcs(T, pruned);
// Recombine pruned paths with non-pruned paths.
foreach v ∈ V do
    C(v) ← []; // candidate nodes for recombination
end
foreach t ∈ T do
    if t ∉ pruned then
        C(LatticeNode(t)).Append(t);
    end
end
foreach v ∈ V do
    SortByProbability(C(v));
end
foreach t ∈ pruned do
    v ← LatticeNode(t);
    foreach t' ∈ C(v) do
        if p(t') ≥ p(t) then
            break;
        end
    end
    WriteArc(t, t');
end
// Add epsilon arcs to new final node.
foreach t ∈ final do
    WriteEpsilonArc(t, f');
end
```

Figure 5.5 Pseudocode for lattice rescoring with the traceback approximation. An expanded lattice different from the input lattice (V, E) is created, incorporating neural network LM probabilities.

2. Replacement approximation with the Viterbi method. A rescored word lattice is obtained with the replacement approximation from Fig. 5.3. Afterwards, the best path according to the Viterbi method is obtained from the lattice.

3. Replacement approximation with the consensus method. The rescored word lattice is obtained in the same way as (2), but the consensus method for minimizing the word error is applied to the lattice.

4. Traceback approximation with the Viterbi method. The traceback approximation is used to create a rescored and expanded word lattice. The best word sequence is obtained according to the minimum sentence error decision rule with the Viterbi method.

5. Traceback approximation with the consensus method. A rescored word lattice is obtained as in (4), but the consensus method is used for obtaining the best word sequence.

For the experimental analysis, we will investigate all the variants and show relations between the rescoring results obtained by these methods.

Furthermore, there is the need for appropriately scaling LM probabilities. The input lattices for rescoring are created with a count LM only, therefore, it seems reasonable to apply a different LM scale for rescoring, where a significantly improved LM is used. For the variants (2) to (5), we can even make use of a third LM scale when applying the Viterbi method or the consensus method to the rescored lattice.

5.5 Experimental Results

We make use of the Quaero French system described in Appendix A.3.2. For evaluating the algorithms described in Section 5.3 and 5.4, we trained an LSTM neural network LM with a projection layer and a hidden layer each of size 300 on the French 100 M word training corpus. For speed-up, 1,000 perplexity-based word classes were used to factorize the output layer. The perplexities of this LM are shown in Tab. 5.1. For comparison, we include perplexity values of a Kneser-Ney-smoothed 4-gram count LM, once trained on the same data as the LSTM and once trained on the full data set. From the results it can be observed that the count model is considerably improved by adding the additional data, reducing the perplexity by 16.9% relative on the development data and by 15.6% relative on the test data, i.e., the additional training data are relevant for the domain of the test data. Even though the LSTM is trained on the smaller data set, it still obtains consistently better perplexities than the count

LM	Running Words	Perplexity	
		Dev	Test
Count LM	100 M	123.9	144.6
	1.6 B	102.9	122.0
LSTM	100 M	98.6	114.9
+ Count LM	1.6 B	**79.9**	**94.4**

Table 5.1 Perplexity results on the Quaero French development and test data.

LM trained on the full data set. The lexicon of the speech recognizer contains 200 K words, but not all of them are covered by the reduced training data set of 100 M running words. For this reason, we include a special out-of-vocabulary (OOV) token in the LSTM LM vocabulary, and we map all words from the recognizer lexicon not observed in the training data to this token. By penalizing the OOV token as described in Section 4.6.4, we enforce that probability distributions are normalized, and perplexities can be compared among different kinds of LMs. By interpolating the large count LM and the LSTM, we obtain further improvements, resulting in a perplexity of 79.9 on the development data, and 94.4 on the test data. This interpolated model will be used throughout the French rescoring experiments.

5.5.1 Recombination Order

We investigate the optimum order that is used for recombination pruning in combination with push-forward rescoring, where the beam pruning parameter is kept fixed at a large value. The corresponding results can be found in Fig. 5.6. It can be seen that the negative logarithmic probability of the best lattice rescoring hypothesis continuously decreases when increasing the recombination order, and the curve saturates at an order of about 9, which corresponds to a 10-gram recombination. Basically, the same tendency is reflected in the word error rate curve. For achieving the best performance, a 10-gram recombination is necessary, which is used in all subsequent experiments.

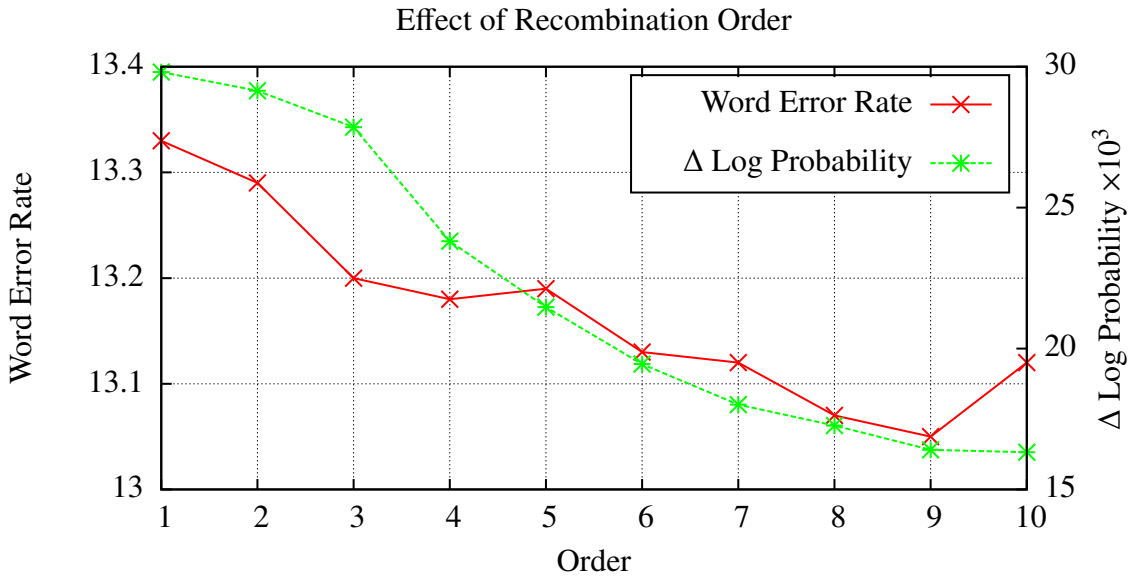

Figure 5.6 Word error rates for different recombination orders, and corresponding negative log probability of the best path.

The experiment can be interpreted in such way that the LSTM effectively makes use of a 10-gram context. On the other hand, it cannot be said whether the LSTM has learned context dependences beyond a 10-gram that do not pay off in noticeable improvements in probability or word error rate. Conversely, the experimental result might also show that the LSTM could not learn dependences of more than 10 consecutive words. Finally, while there certainly exist

grammatical phenomena spanning more than 10 consecutive words in general, it is not clear to which extent these are present in the training data, i.e., they might not have been learned because they were not relevant for minimizing perplexity.

5.5.2 Pruning Techniques

It seems important to experimentally analyze the refined pruning techniques discussed in Section 5.3.1 in combination with push-forward rescoring. From standard speech decoding it is well-known that the particular choice of pruning techniques has a strong impact on the resulting word error rate performance as well as the decoding speed [Ney & Ortmanns 00]. From a scientific point of view, it is necessary to find out whether advanced pruning techniques allow obtaining larger word error rate improvements, which would be prohibitive to obtain with basic techniques like cardinality pruning alone.

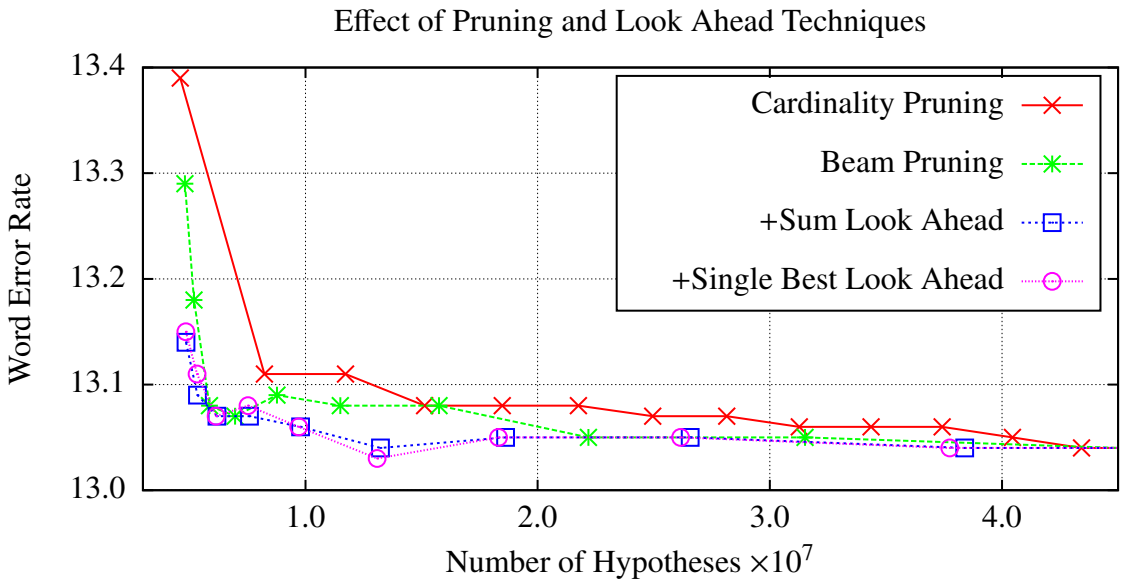

Figure 5.7 Tradeoff between the number of hypotheses evaluated during search and the resulting word error rate, for various pruning and look ahead techniques.

Fig. 5.7 depicts the effect of probability-based pruning techniques. With all of the pruning methods that are investigated, the same best word error rate can be achieved. The main difference between the techniques lies in the size of the search space that needs to be considered to obtain a certain word error rate result. We can gradually decrease the search space size by using beam pruning instead of cardinality pruning, and further reductions are obtained by combining beam pruning with look ahead. (Look ahead does not have any effect on cardinality pruning, because it computes the best successor for a given lattice node, and cardinality pruning only considers a single lattice node at a time. By contrast, look ahead can only pay off when hypotheses assigned to different lattice nodes are compared during the pruning process.) There is virtually no difference whether the sum over all future paths is computed, or whether only the single best path is taken into account for the look ahead. The search space size is directly proportional to the search effort, as the effort is dominated by

the evaluation of the neural network LM equations. In summary, it is important to make use of advanced pruning techniques either if the search effort should be reduced to a minimum, or if the best possible word error rate needs to be obtained. In the former case, we can reduce the word error rate from 13.4% to 13.1%. In the latter case, we can reduce the search space size from 43 million to 13 million hypotheses. Then the real time factor for push-forward rescoring is 0.5 on a single core of an Intel X5675 CPU with double floating point precision.

5.5.3 Decision Rules and Rescoring Algorithms

In Table 5.2, word error rate results are given on the French development and test data for different search space sizes using push-forward rescoring. Here, the baseline refers to two pass speaker-adapted decoding with the large count LM, which gives a WER of 16.4% on the test data. The baseline lattice has a density of 166, where density is measured in lattice arcs per reference token. From the lattice, 100-best, 1,000-best or even 10,000-best lists can be extracted. We actually extract more than 100, 1,000 or 10,000 hypotheses to ensure that hypotheses are unique on the word sequence level, excluding acoustic model tokens. The precise number of hypotheses is determined by binary search. The interpolation of the count LM and the LSTM LM is then used to rescore the n-best lists without approximations, leading to a reduction in WER of 1.6% absolute on the 100-best lists up to 1.7% absolute on the 10,000-best lists. We can also apply push-forward rescoring to word lattices instead of n-best lists, obtaining a word error rate of 14.6%. This 0.1% absolute gain is a very small improvement over the result on 10,000-best lists, so the huge increase in search space size does not pay off in word error rate when sentence error is minimized.

	Lattice Density		Viterbi	
	Dev	Test	Dev	Test
Baseline	124	166	14.4	16.4
100-best	104	106	13.4	14.8
1,000-best	1,337	1,365	13.2	14.7
10,000-best	16,490	17,122	13.1	14.7
Lattice	124	166	13.1	14.6
+Dependent			**12.8**	**14.4**
Expanded Lattice	392	523	13.0	14.6
+Dependent			**12.8**	**14.4**

Table 5.2 Word error rate results and lattice densities based on push-forward rescoring.

By using the consensus method and optimizing for minimum word error, the count LM baseline result is improved from 16.4% to 15.9% absolute, as shown in Table 5.3. We can then apply the replacement approximation for neural network LM rescoring, where LM probabilities are replaced on the arcs of the n-best lists or the lattices. Viterbi results on n-best lists are the same as obtained with push-forward rescoring. With the consensus method only a small improvement of 0.2% absolute on the rescored n-best lists is observed in relation to the Viterbi method—unless the n-best lists are extremely large. This is due to the fact that n-best lists encode a relatively small number of hypotheses, and many of the hypotheses even

	Lattice Density		Viterbi		Consensus	
	Dev	Test	Dev	Test	Dev	Test
Baseline	124	166	14.4	16.4	14.1	15.9
100-best	104	106	13.4	14.8	13.4	14.7
1,000-best	1,337	1,365	13.2	**14.7**	13.0	14.5
10,000-best	16,490	17,122	13.1	**14.7**	12.9	14.4
Lattice	124	166	13.2	14.8	12.9	14.4
+Dependent			13.0	14.8	12.7	14.4
Expanded Lattice	392	523	13.2	14.8	12.8	14.4
+Dependent			**12.9**	**14.7**	**12.6**	**14.3**

Table 5.3 Word error rate results and lattice densities based on rescoring with the replacement approximation.

differ in a few word positions only. In comparison, a word lattice from our setup can easily contain 10^{30} hypotheses, while being considerably smaller than e.g. a 1,000-best list. From Tab. 5.3 it can be seen that, on average, the lattices from the French setup are smaller by more than a factor of 8 compared to 1,000-best lists. When using the replacement approximation for lattice rescoring, the density of the rescored lattices remains 166, as for the baseline. The consensus method leads to a reduced word error rate of 14.4%. However, it can be observed that the Viterbi result of 14.8% on the rescored lattices is slightly worse than the 14.7% that we obtained by push-forward rescoring of lattices or 10,000-best lists. This indicates that here the constraint of keeping the size of the original lattice negatively affects performance.

As an alternative, we investigate the traceback approximation, where the LM contexts are dynamically expanded as needed. The results are given in Table 5.4. We leave out results for n-best lists as they are identical to those in Table 5.3 for the replacement approximation. By construction, the word error rate obtained by the Viterbi method on the rescored lattices

	Lattice Density		Viterbi		Consensus	
	Dev	Test	Dev	Test	Dev	Test
Baseline	124	166	14.4	16.4	14.1	15.9
Lattice	592	828	13.1	14.6	12.6	14.2
+Dependent			**12.8**	**14.4**	**12.5**	14.2
Expanded Lattice	880	1,209	13.0	14.5	12.6	14.2
+Dependent			**12.8**	**14.4**	**12.5**	**14.1**

Table 5.4 Word error rate results and lattice densities based on rescoring with the traceback approximation.

and the push-forward rescoring result are exactly the same on the development and on the test data, which indicates that the traceback lattices are well-defined. Furthermore, with the consensus method, the word error rate can be be reduced further to 12.6% on the development data and to 14.2% on the test data, a notable gain of 0.5% and 0.4% absolute, respectively. At the same time, 1,000-best lists are still larger by more than a factor of two compared to the rescored lattices, and even with 10,000-best lists the consensus word error rate is worse compared to traceback lattices.

In all previously described rescoring experiments the utterances were considered independently of each other. Instead, we can initialize the LSTM state for the current utterance with the best LSTM state from the end of the previous utterance, as described in Section 5.3.1. For all five rescoring strategies, in Tables 5.2, 5.3 and 5.4 we include word error rate results for the case of such a dependent rescoring. This implies that rescoring needs to be performed sequentially, only the audio recordings can be processed in parallel. We mainly see improvements by a dependent rescoring for the Viterbi method, but when the consensus method is applied, these improvements mostly vanish, therefore the increased latency resulting from sequential processing seems hardly justified.

The baseline lattices generated by the RWTH speech decoder [Rybach & Bisani$^+$ 11] are not guaranteed to be expanded to a unique context of any order. Therefore, a unigram expansion needs to be assumed. We also analyzed the effect of a static expansion of the baseline lattices to a unique 4-gram context. As shown in Tables 5.2, 5.3 and 5.4, this only gives tiny improvements over the unexpanded case, or even no change in word error rate at all. This behavior can be expected, because due to beam pruning, it does not make a difference whether the original lattice is expanded or not. An exception is that at each lattice node it is enforced that at least one hypothesis survives pruning, which accounts for the tiny differences observed by expansion. We notice that for the independent rescoring of the expanded lattices there actually is a difference in the word error rate between push-forward rescoring and the Viterbi method applied to the traceback lattices (14.6% vs. 14.5%). This is only due to an additional LM scale for the traceback approximation, which was described in Section 5.4.3. When using the same scales, both error rates are exactly the same.

In principle, as indicated in Figs. 5.6 and 5.7, most of the gain in word error rate can be obtained with small recombination orders and tight pruning parameters. However, as observed in Tables. 5.3 and 5.4, it seems necessary to take these settings to the limit and apply the consensus method in combination with the traceback approximation to improve over simple n-best list rescoring. Nevertheless, it should be noted that 10,000-best lists are very large and lead to an enormous computational effort for rescoring. Unlike the lattice-based algorithms presented in this section, 10,000-best lists are hardly relevant for practical applications.

We also analyze the improvements of lattice rescoring over n-best list rescoring in terms of statistical significance. To this end, we make use of the techniques introduced in [Bisani & Ney 04], which rely on bootstrapping statistics and which can be applied at the word error rate level, as opposed to conventional approaches that only rely on sentence error rate. In particular, we investigate bootstrapping of word error rate *differences*, which turned out to provide tighter and more consistent confidence intervals than the analysis based on absolute word error rates.

For the analysis, we focus on the Consensus method and compare independent rescoring of 10,000-best lists with independent lattice rescoring using the traceback approximation (lines 4 and 2 of Tables 5.3 and 5.4, respectively). The latter rescoring technique is also used for the comparison in Chapter 6. Table 5.5 shows the number of words, utterances and speakers for the development and test data. (For the significance analysis, we rely on the manual speaker annotation, whereas for recognition, the speakers were inferred using an automatic procedure.) Furthermore, the absolute difference in word error rate ΔWER between the two rescoring approaches is given. The corresponding confidence intervals based on bootstrap

	Dev	Test
Words	37,786	44,545
Utterances	1,383	1,725
Speakers	106	76
ΔWER	−0.25	−0.21

Table 5.5 Number of words, utterances and speakers for the development and test data, as well as difference in WER between 10,000-best list and lattice rescoring.

estimates are summarized in Tab. 5.6, where we have set the bootstrap sample size to 10,000. We present results for bootstrapping from the distribution of speakers, which is the preferred method. For comparison, we also include numbers for the case of bootstrapping from the distribution of utterances. We find that the bootstrap estimate $\Delta\text{WER}_{\text{boot}}$ of the word error rate difference is equal to the true word error rate difference in all four cases. With a confidence of 90%, the word error rate difference lies in the $[\Delta\text{WER}_{\text{boot}}^{-0.05}, \Delta\text{WER}_{\text{boot}}^{+0.05}]$ interval. As the distribution of word error rate differences is approximately Gaussian for large bootstrap sample sizes, the confidence interval is also given by $\Delta\text{WER}_{\text{boot}} \pm 1.64\text{se}_{\text{boot}}$. E.g., for the speaker-wise bootstrap, with 90% confidence the lattice-based rescoring will result in a WER improvement between −0.34% and −0.09%. It should be noted that this interval does not foresee any degradation when applying lattice-based rescoring compared to 10,000-best list rescoring. Consequently, the probability of improvement poi_{boot} of the lattice-based approach is 99.9%. The probability is very high, which is due to the fact that two identical systems are compared, that only differ in the size of the rescoring search space. For most of the utterances and speakers, applying the same neural network LM to the larger lattice-based search space will result in a reduction in word error rate over the *n*-best list. By contrast, if two different acoustic models are used like in [Bisani & Ney 04], word error rate differences are less systematic and therefore the probability of improvement is smaller.

		Dev	Test
Utterance-Wise Bootstrap	$\Delta\text{WER}_{\text{boot}}$	−0.25	−0.21
	$1.64\ \text{se}_{\text{boot}}$	0.15	0.14
	$\Delta\text{WER}_{\text{boot}}^{-0.05}$	−0.40	−0.35
	$\Delta\text{WER}_{\text{boot}}^{+0.05}$	−0.10	−0.06
	poi_{boot}	99.6	99.1
Speaker-Wise Bootstrap	$\Delta\text{WER}_{\text{boot}}$	−0.25	−0.21
	$1.64\ \text{se}_{\text{boot}}$	0.16	0.12
	$\Delta\text{WER}_{\text{boot}}^{-0.05}$	−0.41	−0.34
	$\Delta\text{WER}_{\text{boot}}^{+0.05}$	−0.08	−0.09
	poi_{boot}	99.1	99.9

Table 5.6 Utterance-wise and speaker-wise confidence intervals based on bootstrap estimates of WER differences between 10,000-best list and lattice rescoring.

5.5.4 Keyword Search

We also investigate the potential improvements in keyword search by lattice rescoring with LSTM LMs. It is state-of-the-art to use a bigram count LM for keyword search [Knill & Gales$^+$ 13, Tüske & Nolden$^+$ 14]. Best results are obtained for extremely large lattices, therefore we only investigate the replacement approximation here.

For the experiments, a system for Babel Assamese keyword search trained on the full language pack according to the 'BaseLR' conditions is used. The system is described in Appendix A.4. The lattices created by the decoder contain 7,900 arcs per second. A count LM, four bigram feedforward neural network LMs and four LSTM LMs were trained. The corresponding perplexities of the models and their linear interpolation is summarized in Table 5.7.

LM	Perplexity
Count LM	244.0
4 × Feedforward LM	226.1
4 × LSTM LM	185.0
Count LM+4 × Feedforward LM	219.0
Count LM+4 × LSTM LM	**181.6**

Table 5.7 Perplexities of the count LM and various neural network LMs on the Babel Assamese development data.

The performance of keyword search is evaluated using the maximum term-weighted value (MTWV) measure from [Fiscus & Ajot$^+$ 07, Mangu & Soltau$^+$ 13]. Higher scores indicate better performance. The results can be found in Table 5.8. For the baseline system, we obtain an MTWV score of 0.4936. By interpolating the count LM with the four bigram feedforward neural network LMs, we improve the MTWV score to 0.5001. If we replace the feedforward network by an LSTM instead, we can finally increase this value to 0.5060, improving the baseline score by 2.5% relative.

LM	MTWV
Count LM	0.4936
+4 × Bigram Feedforward LM	0.5001
+4 × LSTM LM	**0.5060**

Table 5.8 Keyword search results for Babel Assamese.

5.5.5 Cheating Experiment

Whenever an experiment involves a search procedure that relies on pruning, the question arises how much performance is sacrificed to improper pruning decisions, and how good the models would actually perform when search errors could be eliminated. In this section, we investigate the so-called cheating experiment to address this issue.

When computing the word error rate, the correct sentence is already known, and its acoustic model probability and its LM probability can be obtained. In the case of a modeling error,

the recognized word sequence obtains a higher probability than the correct word sequence. In the case of a pruning error, the recognized word sequence has a probability less than the one of the correct word sequence. Conversely, if the probability of the recognition output is higher than that of the correct word sequence, it is clear that the error is due to inaccurate probabilistic models. If the probability of the recognition output is lower than that of the correct word sequence, no such argument can be made.

In terms of the rescoring experiments carried out in this work, it is either an insufficient lattice size or incorrectly set pruning parameters in rescoring that account for search errors. By increasing the pruning thresholds of the initial lattice generation or the search performed during rescoring, ultimately any possible word sequence—and in particular the correct one—is hypothesized. Instead of handling extremely large word lattices, we can virtually add the correct sentence to a given lattice in a cheating experiment. In this way it is guaranteed that the correct sentence can be hypothesized in rescoring. If the word error rate improves considerably by the cheating experiment, this may indicate that the lattice size or the search parameters are insufficient.

The word error rate results for the cheating experiment are given in Table 5.9, where a manual segmentation was used. Overall we observe that the cheating experiment does not

	Viterbi		Consensus	
	Dev	Test	Dev	Test
Count LM	14.2	15.8	13.7	15.5
+Cheat	13.7	15.4	13.3	15.2
Count LM+LSTM LM	12.5	14.2	12.3	13.9
+Cheat	12.1	13.7	12.0	13.7

Table 5.9 Word error rate results for the cheating experiment.

lead to large improvements, in particular when the consensus method is used to minimize word error rate directly, which is the preferred decision rule throughout the experiments in the thesis. Most importantly, the gap in word error rate is even slightly smaller for the neural network rescorings than for the count LM baselines. For count LMs, the effect of pruning parameters is well understood. The fact that the cheating gap for neural network rescoring is smaller than for count LMs shows that the neural network rescoring is not more error-prone than count LM decoding, and we conclude that pruning parameters are well chosen.

The underlying idea of the cheating experiments relies on the assumption that each word from the correct word sequence can also be recognized, and thus it must be contained in the LM vocabulary. In practice there are OOV words, for which no acoustic or LM probability can be obtained. For the experiments reported in Table 5.9, we thus handle OOV words in the following way. First, we create n-best lists comprising recognition hypotheses for the utterances containing OOV words. Then we compute a Levenshtein alignment between the correct word sequence and each of the recognition hypotheses. We choose the hypothesis with minimum word error where the OOV word from the reference is edited by a substitution operation in the Levenshtein distance. The OOV word in the correct word sequence is then replaced by the corresponding word according to the alignment. In this way, we obtain an OOV-free reference transcription of the speech data. However, for computing the Levenshtein alignment, a proper preprocessing of the correct word sequence and the recognized

word sequence is necessary. Otherwise, systematic differences in orthography like "we're" vs. "we are" can lead to strongly inaccurate results. Due to the preprocessing, the words from the Levenshtein alignment differ from those found in the LM vocabulary, and thus cannot always be distinguished from OOV words. Also, there is no guarantee that each OOV word is edited by a substitution operation in the Levenshtein alignment. These special cases make it difficult to obtain a reference transcription without OOV words.

The word error rates from the cheating experiment may be considered a lower bound on the word error rate that can be achieved given an acoustic model and a language model. Regarding the Viterbi method, there are always hypotheses that have been pruned and that may result in a higher probability than the correct word sequence when they are included in the search space. Regarding the consensus method, the cheating experiment may introduce a bias towards the correct word sequence when computing word posterior probabilities. Nevertheless, the experiment gives an indication whether pruning parameters need to be adapted, and it demonstrates the minimum word error rate that could be obtained when search errors are avoided.

5.6 Summary

The search problem associated with neural network LMs has received lots of attention in the literature. We gave on overview of existing techniques, describing the push-forward method, that is suitable for the rescoring of word lattices with RNN LMs and LSTM LMs. In speech recognition, two decision rules are commonly used, where either the sentence error or the word error is minimized. Previous methods only facilitated the minimization of the sentence error, and this work was extended such that the minimization of the word error was possible as well.

In experiments it was found that the word lattices created by this method were well-defined in the sense that they led to the optimum Viterbi word error rate, while they also allowed considerable improvements based on the consensus method. In comparative experiments it was observed that n-best lists were significantly larger than the proposed lattice-based approach. At the same time, n-best lists led to higher word error rates. These findings contrast other research on lattice rescoring, where no improvements over n-best lists could be obtained, regardless of the size of the rescored word lattices.

Furthermore, pruning techniques from speech decoding were transferred to lattice rescoring with neural networks, and their impact on word error rate as well as rescoring speed was analyzed. For recurrent neural network LMs, the concept of recombination can be considered a pruning technique, and recombination pruning provided experimental insights into the maximum context length that is exploited by an LSTM LM. The pruning techniques in turn were applied in a cheating experiment, where the impact of search errors was analyzed. The corresponding results indicated that only small gains could be achieved by increasing pruning thresholds, and the neural network LM search space showed slightly less room for improvement than the count LM search space.

The methods in this chapter concentrated on lattice rescoring for large-scale speech recognition. So far few work has integrated a neural network LM into decoding, only small-sized tasks were investigated, and no considerable improvements compared to lattice rescoring

were obtained. Besides the fact that the application of neural network LMs to decoding for large-scale setups is computationally demanding, neither small-scale experiments nor the cheating experiment indicate a need for decoding integration.

Chapter 6

Comparison of Count-Based and Neural Network Language Models

Research in neural network language modeling has mainly concentrated on three different architectures, namely feedforward, recurrent, and recurrent LSTM neural networks, which have been described previously.

These methods have in common that they rely on word input only, and the same kind of speed-up techniques like word classes can be used in all cases. Thus, the architectures can essentially model the same kind of probability events on large-scale tasks. On the other hand, the architectures differ in many aspects like the hidden layer configuration, the number of parameters for a given hidden layer size, or the training algorithm. From a conceptual point of view, a feedforward model seems preferable, as it is simpler than recurrent variants, and it relies on a fixed context of predecessor words, which can be helpful in applications like decoding and rescoring. Therefore, in this chapter we address the question whether the increased complexity of recurrent models pays off in performance.

To this end, we compare the models on a large-scale speech recognition task, where we aim for the best possible word error rate that can be obtained with each architecture, given a fixed amount of training data. Furthermore, we relate the neural network results to conventional count LMs, and present an analysis why neural network LMs perform better than count LMs. Finally, we also identify weaknesses of neural network LMs, where they fall short of showing significant improvements over a count LM baseline.

The comparison of different neural network architectures has received some attention in the literature. In [Mikolov & Kombrink$^+$ 11a] and [Arısoy & Sainath$^+$ 12], a feedforward LM and an RNN LM were compared in terms of perplexity on one million and on 24 million running words of Wall Street Journal data, respectively. In both cases, the perplexity of the RNN was consistently lower by more than 10% relative. However, on the larger corpus, perplexities were not directly comparable due to different vocabulary sizes.

A more detailed study based on perplexities was presented in [Oparin & Sundermeyer$^+$ 12], where it was investigated for what type of events a count LM or a neural network LM performs best. This analysis was extended to word error rate results for the first time in [Sundermeyer & Oparin$^+$ 13], where feedforward LMs and recurrent LSTM LMs were trained on 27 million running words of French broadcast conversational data. The LSTM obtained a perplexity which was lower by more than 15% relative compared to the feedforward model, and

these improvements also carried over to word error rate reductions. Only in [Le & Allauzen$^+$ 12a] it was observed that an RNN LM was outperformed by a feedforward architecture. A feedforward 10-gram obtained a perplexity which was lower by 5% relative on a large-scale English to French translation task, and the final performance in terms of BLEU was identical for both networks. For efficiency reasons, the RNN was not trained with backpropagation through time. A recurrent-like m-gram model was used in training, instead of a standard RNN LM.

To the best of our knowledge, only [Arısoy & Sainath$^+$ 12] investigated the potential gains by deep network architectures for language modeling in speech recognition. Virtually no improvements in perplexity or word error rate were observed when a feedforward neural network LM was interpolated with a count LM that was trained on the same amount of data. Very recently, in [Pascanu & Gulcehre$^+$ 14] the construction of deep RNNs was analyzed. However, even with a deep RNN, the perplexities obtained on one million running words of Wall Street Journal Data were only on par with the perplexity of a single-layer LSTM [Sundermeyer & Schlüter$^+$ 14]. No speech recognition results were provided.

The comparison presented here considerably extends previous analyses from [Arısoy & Sainath$^+$ 12] and [Sundermeyer & Oparin$^+$ 13]. We make use of state-of-the-art word classes and take into account most recent developments in lattice rescoring from [Sundermeyer & Tüske$^+$ 14]. In addition, we revisit the idea of deep neural networks for language modeling, providing new insights in relation to [Arısoy & Sainath$^+$ 12]. The results have also been published in [Sundermeyer & Ney$^+$ 15].

6.1 Experimental Results

The experiments were performed on the Quaero English task. The corresponding system is described in Appendix A.3.1. We made use of a 4-gram count LM. Due to long training times, unless stated otherwise, neural network LMs were trained on 50 M running words, and the count LM was trained on all available data. Neural networks were trained until convergence, regardless of the number of epochs. For all neural network variants, we tied the projection layer and hidden layer sizes to the same value, except for the projection layer of the feedforward neural network, which was kept constant in all experiments. Furthermore, for all variants the output layer was factorized using 1,000 word classes. Word classes were obtained with the exchange algorithm from [Kneser & Ney 91] based on a perplexity criterion with bigram dependences.

For the feedforward network, the dimension of the word features was set to 300, which we found to give best performance in preliminary experiments. The feedforward neural network LM was a 10-gram, thus, in total, the projection layer comprised 2,700 units. Training was performed with the `rwthlm`[1] toolkit [Sundermeyer & Schlüter$^+$ 14].

The RNN LM was trained with the `rnnlm` software from [Mikolov & Kombrink$^+$ 11b], which has been used by many researchers in the language modeling community and has become a widely accepted standard. In addition, its training algorithm is well-suited to reduce the problem of vanishing and exploding gradients encountered with RNNs. We made use of a modified version of the toolkit to allow perplexity-based word classes at the output layer.

[1] http://www-i6.informatik.rwth-aachen.de/web/Software/rwthlm.php

This has a considerable impact on perplexity performance, which was verified experimentally in Section 4.7.2. Following recommendations given in [Mikolov & Kombrink+ 11b], we truncated BPTT after six time steps by setting '`-bptt`' to 6, and we trained the RNN in block mode with '`-bptt-block`' equal to 10. The '`-min-improvement`' parameter was set to 1.

Regarding the LSTM LM, the settings that were found to work best in Section 4.7 were chosen for this experimental comparison. To speed-up LSTM training, mini-batches composed of four sequences were processed in parallel. In a preparatory experiment, this level of parallelization was not found to affect performance.

6.1.1 Perplexity Results

We first investigate the performance of count-based LMs in relation to feedforward, recurrent, and recurrent LSTM neural network LMs. In Fig. 6.1, the different types of LMs are compared in terms of perplexity on the English development data. For the sake of this comparison, we do not consider the full 150 K recognition vocabulary, but only the subset of words that occur in the 50 M word training data set, which amounts to 128 K words. Here, all LMs including the count LM were trained on the 50 M word data set. For this comparison, individual count LMs were trained on each data source, and the resulting models were then linearly interpolated such that the develompent perplexity is minimized.

We find that increasing the hidden layer size of the feedforward neural network considerably reduces the perplexity from 163.1 to 136.5. Furthermore, we can make the feedforward neural network deeper by adding another hidden layer. In comparison with a single-layer feedforward network, the perplexity is always improved by the second hidden layer. However, the perplexity reductions achieved are not very large. When switching to a recurrent neural network, at first the perplexities for the feedforward model are lower. E.g., for a hidden layer size of 100 nodes, the perplexity of the feedforward neural network LM is 163.1, whereas the RNN LM perplexity is still at 169.1. On the other hand, as soon as we increase the hidden layer size, we observe further gains over the feedforward architecture. Eventually, the RNN with a single hidden layer of size 600 obtains a perplexity of 125.2, whereas the perplexity of a feedforward network with even two hidden layers of size 600 is 130.9. By replacing the RNN with an LSTM, we can still obtain perplexities that are much lower. The perplexity of the LSTM with one hidden layer of size 600 is 107.8, which corresponds to a relative improvement of 13.9% over the RNN. By adding a second LSTM layer, we find that perplexities still drop further to a value of 100.5. We did not investigate two-layer RNNs as `rnnlm` does not support RNN architectures with multiple hidden layers. In principle, such networks could be trained with `rwthlm`. However, this toolkit implements epochwise BPTT, which computes gradients over full sequences and thus requires neural network architectures that are robust with respect to the vanishing and exploding gradient problem, like LSTM networks. An overview of the stand-alone perplexities on the development data as well as on the test data is also given in Table 6.1.

While feedforward neural networks are far from obtaining the best perplexity in this experiment, their implementation is simple and they might be trainable in shorter time than more complex variants like e.g. LSTM networks. From that point of view, the disadvantage of worse performance might in practice be remedied by just training them on more data—

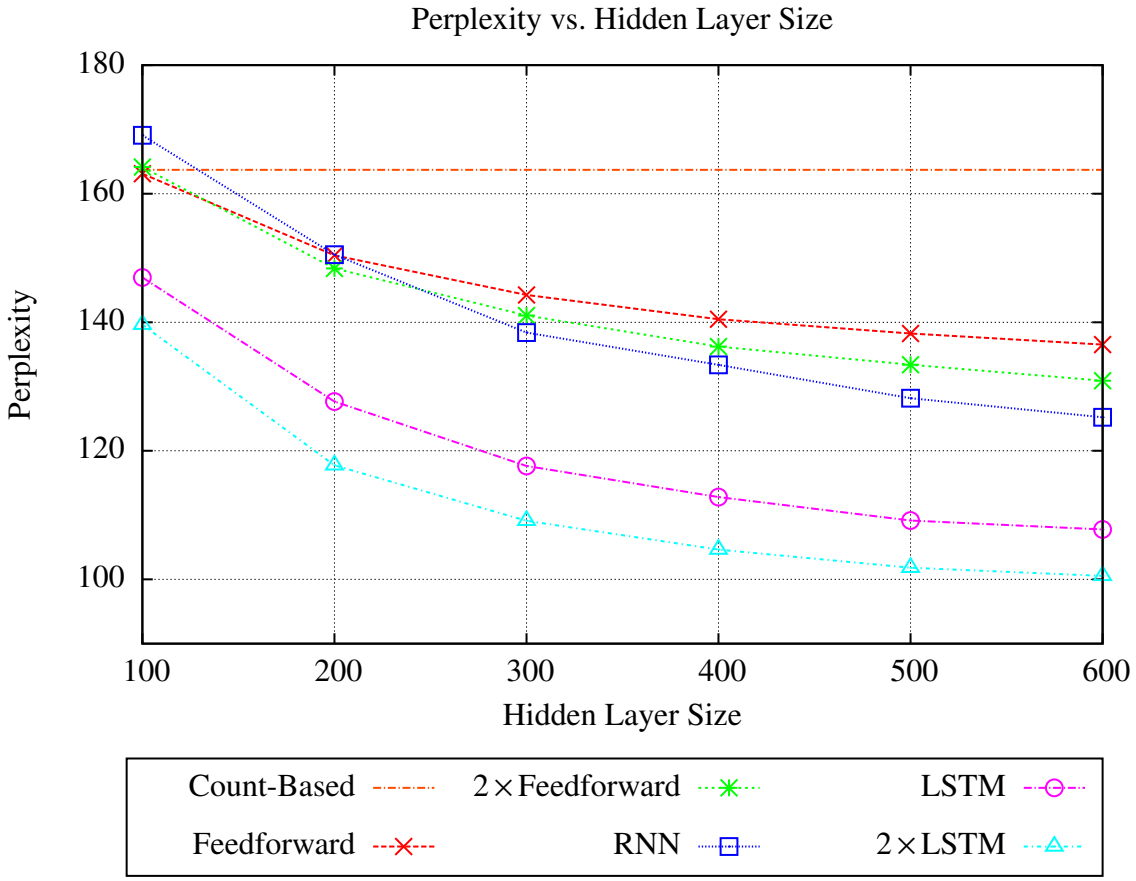

Figure 6.1 Stand-alone perplexity on the English development data for different hidden layer configurations and neural network architectures. All models are estimated on the same amount of data. The vocabulary is restricted to words observed in the training data.

under the assumption that more training data are available. Therefore, as an outlook, we also trained a feedforward neural network LM on 500 M running words with 2 hidden layers comprising 800 hidden units each. This was one of the experiments with the longest runtime carried out in this thesis. In spite of the increase in training data and the larger hidden layer configuration, the perplexity of the feedforward model was 109.8 on the development data. According to Fig. 6.1, this is still significantly higher than the best LSTM perplexity of 100.5 that was obtained on a tenth of the training data only.

6.1.2 Convergence Behavior

In Fig. 6.2 the convergence behavior of the neural network architectures is depicted. The curve is smoothed such that for a given epoch, the best development perplexity observed until this epoch is shown. The LSTM converges fastest. After 15 epochs, no considerable improvements are obtained anymore, whereas both the feedforward neural network and the RNN LM benefit from a continued training. In the case of the feedforward neural network LM and the LSTM LM, the learning rate is only halved when the development perplexity increases, as described in Fig. 4.5 in Section 4.4.1. For the RNN, the learning rate is halved at

Hidden Layer	Feedforward		LSTM		RNN	
	Dev	Test	Dev	Test	Dev	Test
100	163.1	162.2	147.0	145.6	169.1	166.5
200	150.4	148.2	127.7	126.0	150.5	151.4
300	144.2	142.9	117.6	116.7	138.4	137.9
400	140.5	141.6	112.8	112.4	133.4	133.9
500	138.2	138.0	109.2	109.1	128.2	128.4
600	136.5	135.6	107.8	107.2	125.2	126.4
2×100	164.1	163.5	139.6	139.3		
2×200	148.4	145.9	117.7	117.0		
2×300	141.1	143.3	109.1	109.6		
2×400	136.2	134.6	104.6	105.8		
2×500	133.4	133.2	101.8	102.4		
2×600	130.9	129.8	**100.5**	**101.0**		

Table 6.1 Perplexities of neural network LMs on the English development and test data, without interpolation. The count LM obtains perplexities of 163.7 and 161.4 on the development and test data, respectively.

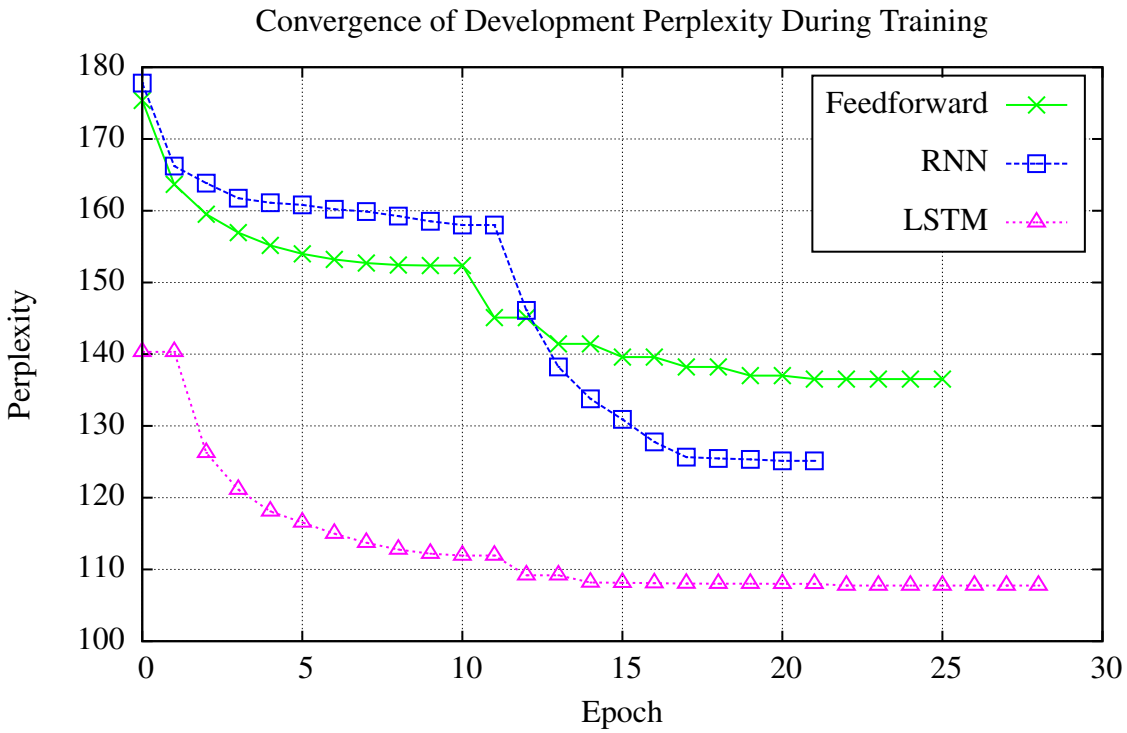

Figure 6.2 Evolution of the development perplexity during training. All networks were trained with a single hidden layer of size 600.

each epoch as soon as the development perplexity has increased for the first time, according to the learning rate schedule implemented in the `rnnlm` toolkit. This explains why the RNN LM improves quickly after the eleventh epoch.

LM	With Shuffling		Without Shuffling	
	Dev	Test	Dev	Test
Feedforward	152.3	153.9	136.5	135.6
LSTM	114.6	114.2	**107.8**	**107.2**

Table 6.2 Effect of shuffling on final perplexity. Neural network LMs make use of a single hidden layer comprising 600 units.

In total, a large number of words is processed during training that ranges from 1.0 B to 1.4 B running words. Regarding the large feedforward model, even 23 B running words were seen in training. Sometimes, these numbers are referred to as the number of words that have been used for training, or even larger values, when sampling techniques are used [Le & Allauzen$^+$ 12a, Schwenk & Rousseau$^+$ 12]. We use considerably more data in our experiments. However, only the number of words per epoch is counted and reported as the amount of training data in this thesis, regardless of how many times they have been encountered in training.

6.1.3 Shuffling

In our experiments, we also confirm the finding from [Mikolov & Deoras$^+$ 11] that during neural network training, it is important to process in-domain data towards the end of an epoch. In Table 6.2 perplexities are given for the case where the data were processed in this order during training, and where the order is not maintained due to shuffling. For both the feedforward neural network and the LSTM, data are shuffled on the sequence level. We do not include results for standard RNNs. As these are trained with the hybrid BPTT variant, it does not make sense to shuffle the data, because the sentences in the training corpus fall in a natural order, and the RNN state is never reset in training. We find that due to the non-homogeneity of the data, final perplexities can be considerably improved when not shuffling the data. As a result, any of the neural network architectures significantly outperforms a Kneser-Ney smoothed count LM. This observation contradicts experience from acoustic modeling, where it is often found that shuffling the training data is an important step to obtain the best possible performance.

6.1.4 Order-Wise Perplexities

We can analyze the perplexity results in more detail by computing order-wise perplexities as introduced in [Oparin & Sundermeyer$^+$ 12]. The perplexity for the m'-th order $\mathrm{PPL}_{m'}$ is defined as

$$\mathrm{PPL}_{m'} = \exp\Big(-\frac{1}{|\mathcal{I}_{m'}|}\sum_{n\in\mathcal{I}_{m'}}\log p(w_n|w_1^{n-1})\Big), \tag{6.1}$$

where we let

$$\mathcal{I}_{m'} = \{n|N(w_{n-m'+1}^n) > 0 \wedge N(w_{n-m'}^n) = 0\} \tag{6.2}$$

for training data counts $N(\cdot)$ and a test data word sequence w_1^N. Thus, for the standard perplexity as defined in Section 1.1.5, we have

$$\text{PPL} = \exp\left(\sum_{m'=1}^{m} \frac{|\mathcal{I}_{m'}|}{N} \log \text{PPL}_{m'}\right), \tag{6.3}$$

where m denotes the order of the count LM. This means that the words from the test data are partitioned according to the order of the m-gram hit from the count LM, and perplexities are computed on the corresponding subsets of m-gram events.

We transfer the partitioning of word positions on the test data from the count LM to the neural network variants (even though these may not rely on any kind of order information). In Tab. 6.3 these results are summarized. It can be seen that the count LM obtains very low

Order	Number of Events	Perplexity			
		Count-Based	Feedforward	RNN	LSTM
4	10 K	16.2	21.9	20.7	17.0
3	13 K	88.0	78.5	73.5	63.5
2	11 K	609.3	382.8	349.2	302.3
1	3 K	49,234.2	18,929.6	17,980.5	14,405.1
Total	36 K	161.4	135.7	126.5	107.2

Table 6.3 Order-wise perplexities on the test data for the count-based LM and three neural network architectures with a single hidden layer of size 600.

Order	Number of Events	Perplexity	
		Feedforward	LSTM
4	10 K	20.3	15.8
3	13 K	75.6	60.6
2	11 K	373.8	288.9
1	3 K	17,999.8	12,333.4
Total	36 K	129.8	101.0

Table 6.4 Order-wise perplexities for neural network LMs with two hidden layers of size 600 each on the test data.

perplexities in case of a 4-gram hit, but for lower orders, these perplexities increase strongly. The same holds true for all neural network LMs as well. On the other hand, we observe that neural networks obtain better perplexities than the count LM for all orders except the highest. Only an LSTM with two hidden layers of 600 units slightly outperforms the count LM on the highest order, from 16.2 to 15.8, as can be seen in Table 6.4, which gives perplexities for deep neural network LMs. This is due to the fact that for the highest order, the count LM probability estimates are very close to the relative frequencies. For lower orders, the count LM estimate does not take into account one or more of the history words, and the ability of a neural network to exploit all of the history word information results in better perplexities. For example, when a 4-gram count LM backs off to a unigram, the three most recent history

words are ignored for probability estimation. By contrast, a feedforward 10-gram LM can still map all of the nine history words to a continuous space and take them into consideration for estimating probabilities, regardless of whether the 10-gram was observed in training or not.

6.1.5 Number of Parameters

At this point, the number of neural network parameters is of importance, which differs for the individual neural network architectures. Let W be the number of vocabulary words, S be the shortlist size, i.e., the number of single-word classes, P be the projection layer size, H the number of word classes, m the order of the feedforward neural network LM, and k the hidden layer size. Then the number of a single-layer feedforward neural network LM $N_{\text{Feedforward}}$ is given by

$$\begin{aligned}
N_{\text{Feedforward}}(W,S,P,H,m,k) = {} & WP+ && \text{input layer to projection layer} \\
& Pmk+ && \text{projection layer to hidden layer} \\
& (W-S)k+ && \text{hidden layer to word output layer} \\
& Hk+ && \text{hidden layer to class output layer} \\
& P+ && \text{projection layer bias} \\
& k+ && \text{hidden layer bias} \\
& (W-S)+ && \text{word output layer bias} \\
& H+ && \text{class output layer bias} && (6.4) \\
= {} & WP + (Pm + W - S + H + 1)k + W - S + H. && && (6.5)
\end{aligned}$$

For an RNN LM, we obtain the formula

$$\begin{aligned}
N_{\text{RNN}}(W,H,k) = {} & Wk+ && \text{input layer to hidden layer} \\
& k^2+ && \text{hidden layer to hidden layer} \\
& Wk+ && \text{hidden layer to word output layer} \\
& Hk && \text{hidden layer to class output layer} && (6.6) \\
= {} & k^2 + (2W + H)k. && && (6.7)
\end{aligned}$$

This formula is slightly simpler because the `rnnlm` toolkit also includes weight parameters for the word posterior probability of single-word classes, and it does not provide bias terms. However, according to our experiments, this does neither affect the total number of parameters nor the perplexity performance in a significant way. Finally, for the LSTM, the number of parameters can be computed according to

$$\begin{aligned}
N_{\text{LSTM}}(W,S,H,k) = {} & Wk+ && \text{input layer to projection layer} \\
& 4k^2+ && \text{projection layer to LSTM layer}
\end{aligned}$$

$$
\begin{aligned}
&4k^2+ && \text{LSTM layer to gating units} \\
&3k+ && \text{LSTM peepholes} \\
&(W-S)k+ && \text{LSTM layer to word output layer} \\
&Hk+ && \text{LSTM layer to class output layer} \\
&k+ && \text{projection layer bias} \\
&4k+ && \text{LSTM layer bias} \\
&(W-S)+ && \text{word output layer bias} \\
&H+ && \text{class output layer bias} \qquad (6.8) \\
&= 8k^2 + (2W - S + H + 8 + W)k + W - S + H. && \qquad (6.9)
\end{aligned}
$$

The formulas for two-layer neural network LMs are completely analogous. The relation between the number of parameters and the hidden layer size is depicted in Fig. 6.3. Conceptually, the number of parameters grows quadratically with the hidden layer size, for all types of neural networks, with the exception of single-layer feedforward networks, where the dependence is only linear. However, we see that the growth is quasi-linear for the range of hidden layer sizes investigated. The reason is that the number of parameters is strongly influenced by the number of vocabulary words. In the limit, for recurrent neural network variants we roughly have $2kW$ parameters, and for feedforward variants there are about kW parameters, due to the tying of the projection layer parameters over all the history words. In particular, LSTM networks use more parameters for the hidden connections than standard RNNs, but this is masked by the input and output dimensions of the network.

On the other hand, only a small fraction of the parameters at the input and output layer are actually taken into consideration when processing a single word due to word classing. E.g., in the training corpus, the number of words per class on average is 140, where we take into account the class frequency in the data, so the effort for computing the class posterior probability is much higher than that for the word posterior probability of the output layer. Therefore, the bottom of Fig. 6.3 also shows the number of parameters that are accessed on average when computing word probabilities with any of the neural network architectures. The resulting numbers give an indication about the processing speed of an optimized implementation of the neural networks. However, there is no direct dependence between the number of accessed parameters and the corresponding performance in terms of perplexity.

6.1.6 Word Error Rate Results

In Tab. 6.5, results are given on the test data, in terms of perplexity and word error rate (WER). For comparison, we also include character error rates (CER). In all cases, we interpolate a neural network LM with the large count LM trained on 3.1 B words. Here, we did not investigate the performance of neural networks without interpolating the count LM: In rescoring, the search space is initially generated with the count LM, therefore it is hard to clearly separate the influence of the count model and the neural network on the word error rate level. We rescore lattices with the traceback approximation and apply the consensus method. Utterances are considered independently of each other, with the exception of standard RNNs: For the RNN results, we train the networks with `rnnlm` and then convert them to `rwthlm` format. In this way, we can evaluate all neural networks using the same rescoring

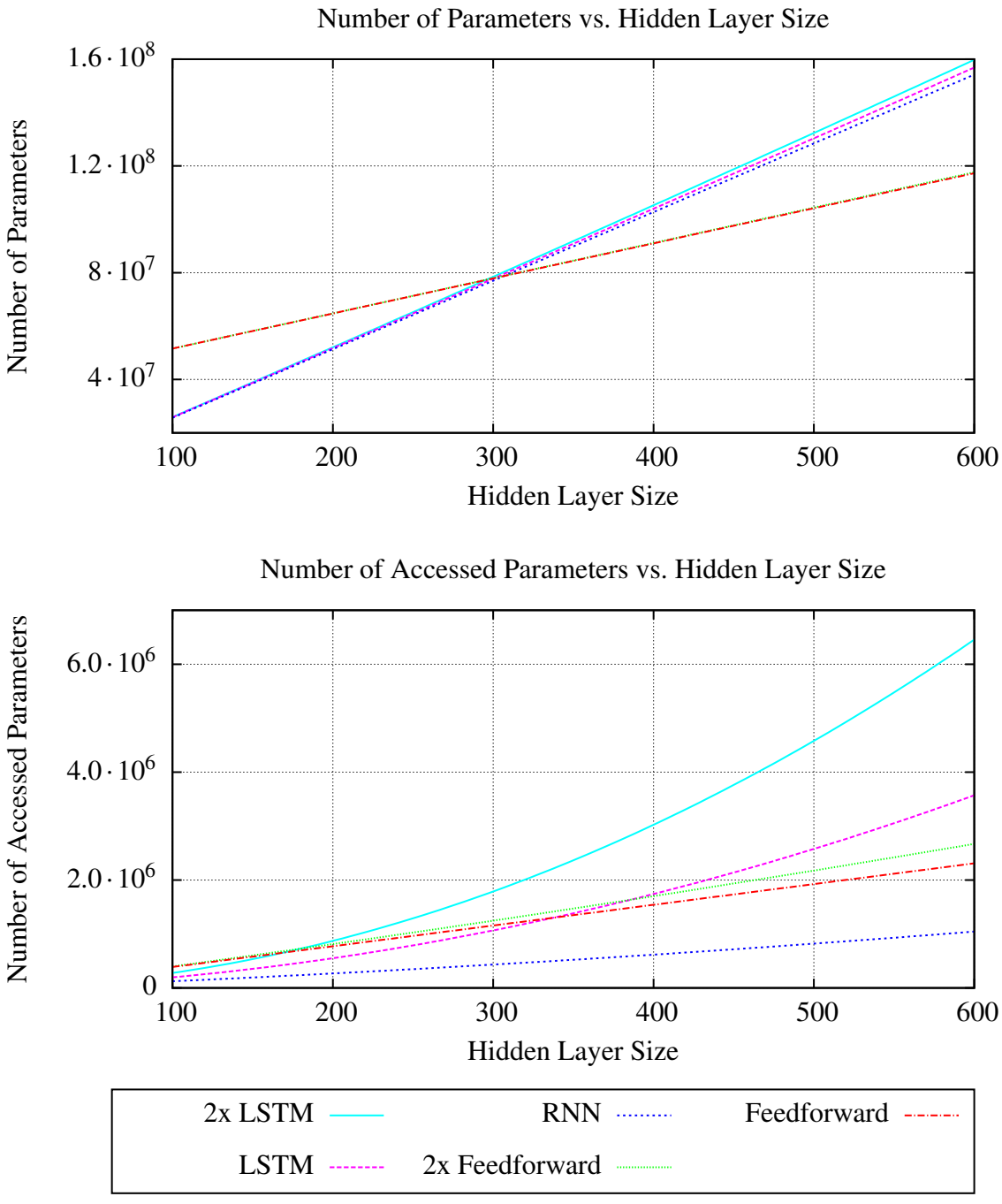

Figure 6.3 Comparison of the total number of parameters as well as the average number of accessed parameters for different neural network configurations. A corresponding 4-gram count LM comprises 63 M n-grams, without count-cutoffs.

LM	Hidden Layers	Perplexity	CER	WER
Count-Based	–	131.2	7.6	12.4
Count-Based+ Feedforward	100	121.1	7.5	11.8
	200	116.6	7.3	11.6
	300	114.7	7.3	11.5
	400	114.1	7.2	11.5
	500	113.4	7.2	11.5
	600	112.5	7.2	11.5
	2×100	121.2	7.5	11.9
	2×200	115.7	7.3	11.5
	2×300	115.4	7.2	11.5
	2×400	112.2	7.1	11.3
	2×500	111.0	7.1	11.3
	2×600	110.2	7.2	11.3
Count-Based+ RNN	100	121.0	7.5	11.8
	200	117.6	7.3	11.7
	300	112.6	7.3	11.4
	400	111.5	7.2	11.3
	500	108.9	7.1	11.2
	600	108.1	7.0	11.1
Count-Based+ LSTM	100	115.3	7.3	11.7
	200	106.8	7.1	11.2
	300	102.4	6.9	11.0
	400	99.9	6.9	10.9
	500	97.9	6.9	10.9
	600	96.7	6.8	10.8
	2×100	111.0	7.2	11.4
	2×200	101.6	7.0	11.0
	2×300	97.5	6.8	10.8
	2×400	95.2	6.9	10.8
	2×500	93.1	**6.6**	10.5
	2×600	**92.0**	6.7	**10.4**

Table 6.5 Performance of different types of language models on the English test data. Neural network LMs are always interpolated with the large count LM.

technique. However, as `rnnlm` training is dependent, rescoring with standard RNNs has to be performed in a dependent way, too. Overall, we see that any neural network LM gives substantial improvements in perplexity and word error rate. Feedforward models fall behind the performance of recurrent variants. Furthermore, adding another hidden layer clearly reduces the word error rate. As single-layer networks may still improve by increasing its hidden layer further, it is not obvious that a deeper model is required to obtain optimum performance. Nevertheless, for the ranges of hidden layer sizes investigated here, we can conclude that deep architectures are beneficial. It is especially remarkable that LSTM networks benefit the most from an additional layer in our experiments, because a single-layer LSTM network can al-

ready be considered deep, as it corresponds to a deep feedforward network when unfolding it over time for training.

We find that absolute word error rate reductions are nearly doubled with a deep LSTM compared to a deep feedforward neural network LM. The feedforward model improves the word error rate from 12.4% to 11.3%, whereas the LSTM obtains a word error rate of 10.4%. In addition, it appears there is no more room for improvement by increasing the hidden layer in the case of the feedforward network, because no effect on word error rate is observed when increasing the hidden layer size from 400 to 600.

The results for the RNN LM might be optimistic, as it is the only LM that was trained and evaluated in a fully dependent manner, whereas e.g. the LSTM hidden state was reset about every 100 words in training and in perplexity evaluation. However, the default of the `rnnlm` toolkit is a dependent training, and we did not want to artificially limit the RNN performance by training the sentences independently. Due to the different training algorithms, a perfectly consistent evaluation seems difficult. Therefore, in Table 6.6 the complete picture for all combinations of training and testing modes is given, which covers perplexity evaluation as well as rescoring. We leave out the case of an independent training and a dependent testing

Training	Testing	Perplexity	CER	WER
independent	independent	109.6	7.1	11.2
dependent	independent	111.6	7.3	11.4
dependent	dependent	**108.1**	**7.0**	**11.1**

Table 6.6 Perplexities, character and word error rate for the RNN LM with a hidden layer size of 600 units for different training and rescoring configurations.

phase, as then the RNN would learn that a sentence boundary token never occurs at the beginning of a sentence and assigns a zero probability to this event. However, in the case of a dependent testing scheme, exactly this event would be encountered. In the end, the result given in Table 6.5 is the best from Table 6.6.

6.1.7 Correlation Between Word Error Rate and Perplexity

In Fig. 6.4, the word error rate on the test data is depicted as a function of the perplexity. Both axes are scaled logarithmically as suggested in [Klakow & Peters 02]. By fitting a polynomial to the data, we find that the experimentally observed data points can be represented quite accurately by the function

$$\mathrm{WER}(\mathrm{PPL}) = 1.42 \cdot \mathrm{PPL}^{0.44}, \tag{6.10}$$

where PPL denotes perplexity. At the bottom of Fig. 6.4, the experimental analysis is extended to very high perplexities and word error rates, which were obtained by bigram and unigram count LMs. We find that even for such a wide range of perplexities and word error rates, the regression curve is only slightly affected, where we obtain a factor of 1.87, and an exponent of 0.39. We thereby confirm the strong correlation between the two quantities. In particular, we observe that this functional dependence between perplexity and word error rate seems to hold independently of the type of LM that is used, and while [Klakow & Peters 02] analyzed count-based LM variants only, the findings can also be transferred to the case of

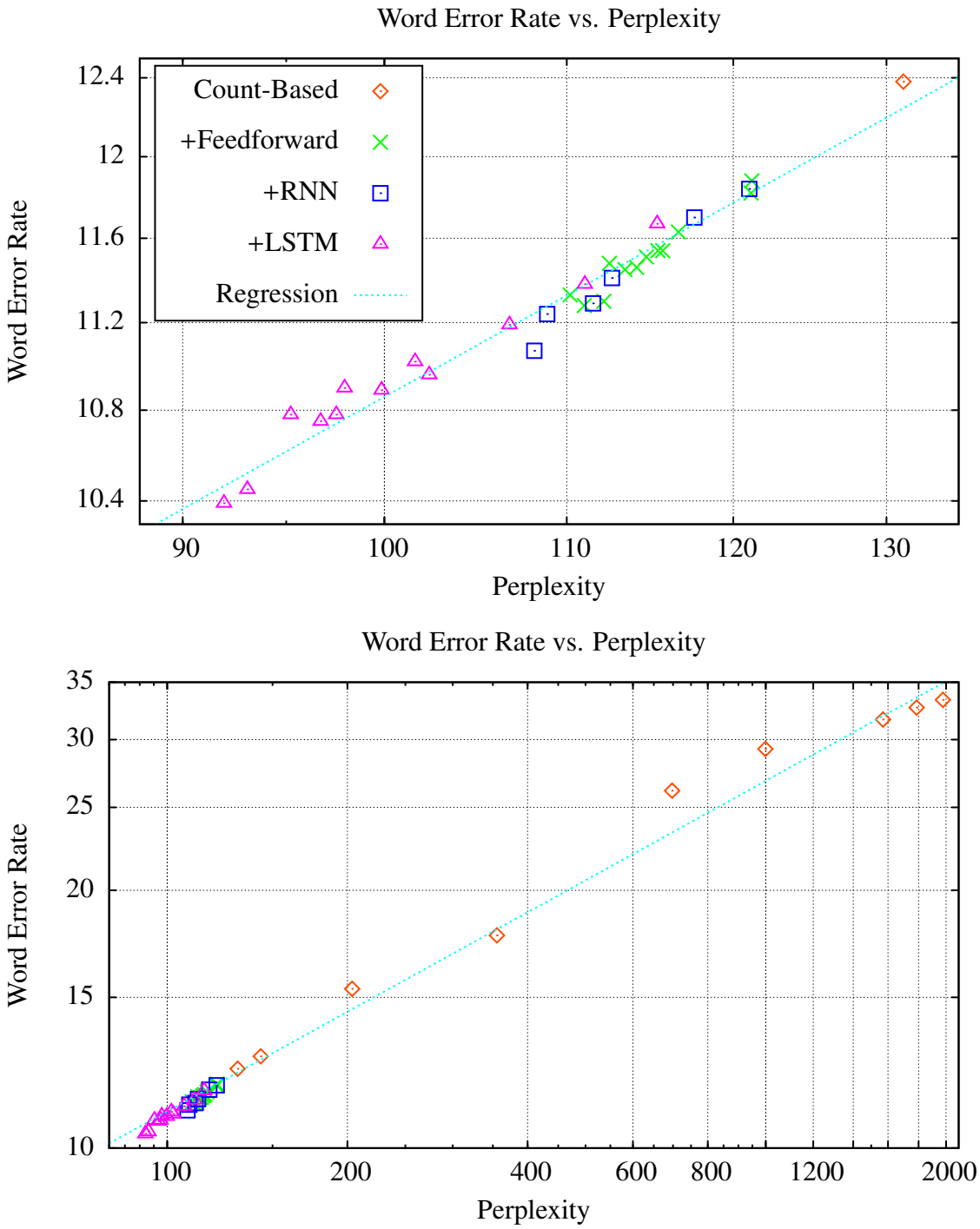

Figure 6.4 Experimental analysis of the relationship between perplexity and word error rate on the English test data. Additional count LM results are taken from [Irie 15].

neural network LMs. In language modeling, the correlation between perplexity and word error rate can thus be considered an advantage in comparison to acoustic modeling, where it is more difficult to find a relation between the word error rate and conventional cross validation criteria, like the frame error rate.

6.1.8 Cheating Experiment

In analogy to the experiment performed in Section 5.5.5, we can analyze the potential of further word error rate improvements on the English speech recognition task as well. A manual segmentation is used. The approach for creating a reference transcription without unknown words is exactly the same as in Section 5.5.5. The results are summarized in Table 6.7.

	Viterbi		Consensus	
	Dev	Test	Dev	Test
Count LM	14.1	12.0	13.9	11.8
+Cheat	13.6	11.5	13.5	11.5
Count LM+LSTM LM	12.1	10.1	12.0	9.8
+Cheat	11.7	9.7	11.6	9.6

Table 6.7 Word error rate results for the cheating experiment.

Basically, we can draw the same conclusions as in the case of the cheating experiment for the French speech recognition task. The lower bounds on the best possible word error rate are rather close to the word error rates that are actually obtained, in particular for the consensus method. Again, the gap in word error rate is slightly smaller for the neural network LM rescoring than for the count LM decoding. Therefore, we conclude that for the English setup, neural network LM rescoring does not introduce an additional source of error to the search problem, and pruning parameters are chosen accurately.

6.1.9 Character-Based Language Models

In the previous experiments, any neural network LM obtained considerable improvements in perplexity over a count LM baseline. However, it seems that this is not the case in all training data scenarios. A LM may also be estimated on characters instead of words. Such a character-based LM is e.g. helpful in recognizing out-of-vocabulary terms in handwriting recognition [Kozielski & Rybach$^+$ 13], as all characters are usually known in advance, in contrast to the full words.

Fig. 6.5 depicts the perplexity performance of a feedforward network, a recurrent LSTM and a count LM for various orders, trained on 50 M characters from the Quaero English corpus. The vocabulary consists of 142 characters, including a special whitespace character indicating word boundaries. The neural networks are both configured as those that obtained best performance in the word-based experiments, i.e., they comprise a projection layer and two hidden layers with 600 nodes, respectively. The feedforward neural network LM makes use of a history of nine preceding words, each of which is mapped to a 300-dimensional

feature vector. The count LM makes use of three discount parameters per order that were optimized with respect to perplexity [Sundermeyer & Schlüter+ 11]. Discount optimization is necessary as no character was observed only a single time in the training data, cf. Eq. (3.4).

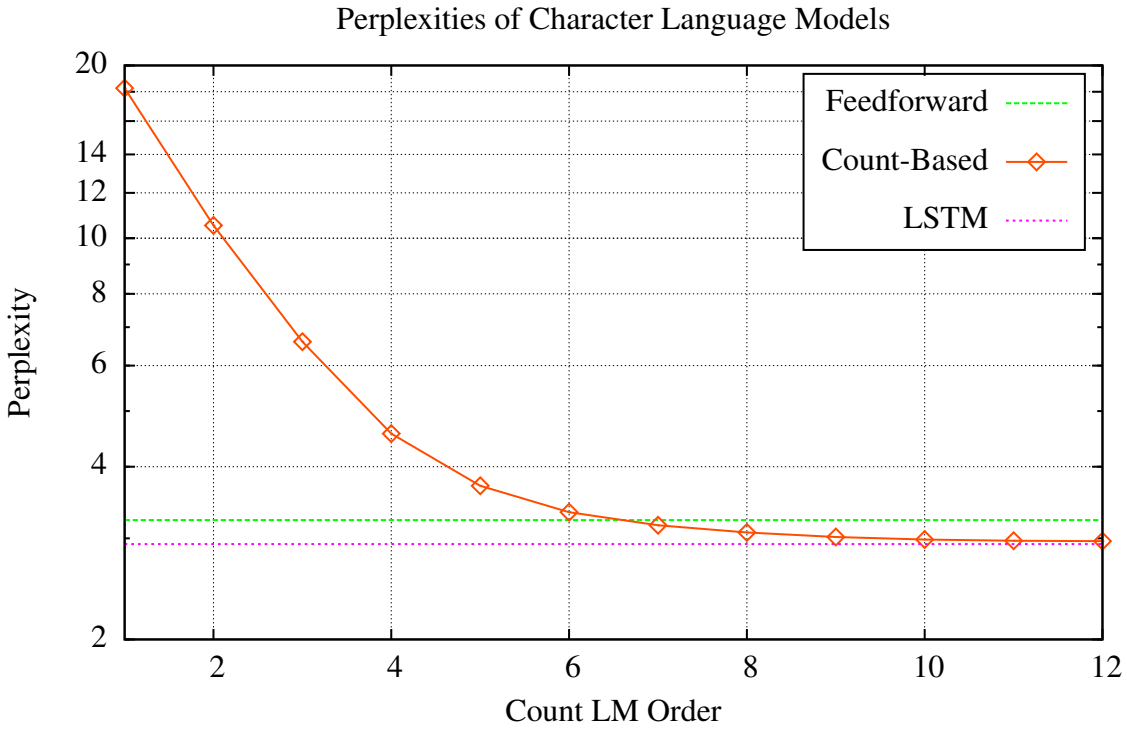

Figure 6.5 Perplexities of character LMs on the English test data.

It can be observed that even an LSTM model only gives a tiny gain in perplexity over a well tuned count LM baseline, from 2.97 to 2.93. The 10-gram feedforward neural network LM roughly corresponds to a 7-gram count LM, and falls behind the perplexity of higher-order count LMs as well as the LSTM. In preliminary experiments it was even observed that when less training data are available, all of the neural network LMs obtain perplexities significantly higher than a count LM. It seems that the smoothing of neural network LMs mainly benefits from large vocabulary sizes. When vocabularies are small, word clustering in a high dimensional continuous space is less effective. For character-based LMs, long context dependences are more important, and these can be accurately modeled by a count LM in a table lookup like fashion.

6.2 Summary

We presented an in-depth comparison of count-based and neural network based language modeling techniques for a large-scale English speech recognition setup. It was found that any neural network LM performed better than a well-tuned count LM baseline. On the other hand, the architecture of a neural network had a strong impact on the resulting performance, where RNNs showed improvements over feedforward networks, and LSTMs in turn outperformed

standard RNNs. Furthermore, we observed that deep neural networks were beneficial for language modeling. Any neural network architecture could benefit from stacking multiple hidden layers, where largest improvements were obtained for LSTMs.

The improvements in perplexity also led to considerable gains in speech recognition accuracy, reducing the word error rate of a state-of-the-art baseline system from 12.4% to 10.4% absolute. In general, perplexities and word error rates were well correlated, where the word error rate was roughly proportional to the square root of the perplexity.

Our experimental analysis also revealed that when the vocabulary size is small and long-range dependences play a role—as found with character-based LMs—neural network LMs seem to lose their advantage compared to count-based LMs, and both variants are basically on par in terms of perplexity.

Finally, we note that our experimental results indicate that, unlike standard RNNs, LSTM networks can be trained by computing exact gradients, backpropagating error information over long sequences without any kind of truncation or clipping of gradient values. This finding from Chapter 4 readily extends to large vocabularies and text corpora as used for the comparative experiments.

Chapter 7

Neural Network Translation Modeling

When translating a sequence of words from one language into another, the LM helps identifying sentences that are likely to occur in the target language. At this point, a statistical machine translation system can be easily improved by taking advantage of neural network LMs as described in the previous chapters. However, this means that the target sentence is considered completely independently of the source sentence. The LM will only evaluate whether a candidate translation hypothesis *a priori* occurs frequently in the target language, but the probabilities do not reflect whether the hypothesis might also be an accurate translation of the given source sentence.

At the same time, neural networks offer a flexible framework allowing arbitrary feature input. The basic idea of this chapter therefore is to extend the approaches presented for language modeling to the case of translation modeling, taking into account the dependence of the target side on its corresponding source side. In Chapter 6, it was found that recurrent and in particular recurrent LSTM neural networks perform considerably better than feedforward variants. Therefore it seems particularly promising to investigate recurrent LSTM architectures for the problem of translation modeling. The main challenge arising from recurrent neural network-based translation modeling lies in the difference in length between the source and the target sentence. In language modeling, the input word sequence as well as the sequence of teacher signals at the output layer are in fact the same word sequence, shifted by a constant offset of one word. In translation modeling, there is no direct analogy between input data and teacher signal. Thus, this chapter proposes novel approaches how to deal with the alignment problem as met in recurrent neural network translation modeling.

To the best of our knowledge, the earliest attempts to apply neural networks in machine translation were presented in [Castaño & Casacuberta$^+$ 97b, Castaño & Casacuberta 97a, Castaño & Casacuberta 99], where neural networks were used for example-based machine translation. More recently, in [Schwenk 12] a feedforward neural network translation model was proposed. The model was limited to phrases of a fixed maximum length, and all target words were predicted at once. In [Le & Allauzen$^+$ 12b] translation models were presented based on a word-factored and a tuple-factored m-gram approach, where best results were obtained using the word factorization. No maximum phrase length was assumed. Regarding large vocabulary machine translation, in [Schwenk 12, Le & Allauzen$^+$ 12b] neural networks were used for rescoring, whereas [Vaswani & Zhao$^+$ 13] and [Devlin & Zbib$^+$ 14] also applied feedforward neural networks in decoding. To avoid the costly output normalization

step, [Vaswani & Zhao+ 13] used noise contrastive estimation, and [Devlin & Zbib+ 14] augmented the training objective function to produce approximately normalized probabilities directly (cf. Section 4.5). The former work only made use of a neural network LM, whereas the latter investigated translation and joint models similar to those from [Le & Allauzen+ 12b], and pre-computed the first hidden layer beforehand, resulting in large speed-ups. Major improvements were reported over strong baselines. Regarding recurrent neural network architectures, RNNs were used with full source sentence representations in [Kalchbrenner & Blunsom 13]. These continuous representations were obtained by applying a sequence of convolutions, and the result was fed into the hidden layer of an RNN LM. However, rescoring results did not indicate improvements over the state of the art. Similarly, [Auli & Galley+ 13] included source sentence representations built either by using latent semantic analysis or by concatenating word embeddings. This approach produced no notable gain over systems already including an RNN LM. In [Hu & Auli+ 14] RNNs were used with minimum translation units, which are phrase pairs undergoing certain constraints. At the input layer, each of the source and target phrases were modeled as a bag of words, while the output phrase was predicted word-by-word assuming conditional independence. While previous work on RNN translation modeling always relied on some kind of preprocessing of the source sentence to address the alignment problem, in [Sundermeyer & Alkhouli+ 14] for the first time an RNN was investigated that made use of source word identities directly as neural network input. Word-based LSTM translation models were proposed as well as phrase-based models, which represent the generalization of word-based models to arbitrary alignments between the source and the target sentence. Experiments showed consistent improvements in translation quality over well-tuned baselines including an LSTM LM. This approach will be described more detail in this chapter. Very recently, in [Sutskever & Vinyals+ 14] an alignment-free LSTM translation approach was suggested, resulting in considerable improvements over a baseline system. However, it was also found in an evaluation that better performance could be obtained without the proposed methods.

In the following, word- and phrase-based translation models are introduced. Let f_1^J be a source sentence, and let e_1^I be its translation into a target language. Unlike in the case of language modeling, we are not interested in modeling the joint probability $p(e_1^I)$ but the posterior probability $p(e_1^I|f_1^J)$ instead. Similar to the rescoring techniques described in Chapter 5, we can then generate a set of candidate hypotheses that might be appropriate translations of the source sentence f_1^J, and rescore the hypotheses using the posterior probabilities $p(e_1^I|f_1^J)$ as obtained from the neural network translation model. Based on the updated translation scores, which include the negative logarithmic translation model probability, the final result is obtained.

7.1 Word-Based Translation Models

For the definition of the word-based translation model, in the first place we assume that we have $I = J$, i.e., source and target sentence are equal in length. This leads to the assumption of a one-to-one correspondence between the i-th word from the source side and the i-th word from the target side. We present the mathematical details of the model, and in a second step, we show how a one-to-one correspondence between the source sentence and the target

sentence can be ensured. Given a source sequence f_1^I and a target sequence e_1^I, we define the posterior translation probability as follows:

$$p(e_1^I|f_1^I) = \prod_{i=1}^{I} p(e_i|e_1^{i-1}, f_1^I) \tag{7.1}$$

$$= \prod_{i=1}^{I} p(e_i|e_1^{i-1}, f_1^{i+d}) \tag{7.2}$$

$$= \prod_{i=1}^{I} p(e_i|f_1^{i+d}). \tag{7.3}$$

We denote formulation (7.1) as the bidirectional joint model. The terminology is adapted from [Devlin & Zbib$^+$ 14], where it was introduced for the case of feedforward neural networks. The term bidirectional refers to the fact that at word position i, the posterior probability is conditioned on the source sentence in backwards direction f_1^i as well as in forwards direction f_i^I, resulting in a dependence on the full source sentence f_1^I. This model can be simplified by several independence assumptions. First, by dropping the dependence on future source information, a so-called unidirectional joint model (JM) is obtained, as given in Eq. (7.2). Here, by $d \in \mathbb{N}_0$ we denote a delay parameter, which is always set to zero unless stated otherwise. The idea of the delay parameter is to allow for dependences on future source side words in a simple manner. However, the parameter requires additional tuning on held-out data, and while the inclusion of future context may be helpful, it also has the downside of introducing a time lag between the i-th target word and its associated i-th source word, which eventually makes learning of dependences more difficult [Graves 12, p. 25]. Finally, assuming conditional independence of the previous target sequence in Eq. (7.3) results in a unidirectional translation model (TM). Analogously, we can define a bidirectional translation model by keeping the dependence on the full source sentence f_1^I, but ignoring the previous target sequence e_1^{i-1}:

$$p(e_1^I|f_1^I) = \prod_{i=1}^{I} p(e_i|f_1^I). \tag{7.4}$$

Table 7.1 summarizes the four word-based translation model variants. The neural network architecture corresponding to the unidirectional word-based model is illustrated in Fig. 7.1, where me make use of a word class mapping G as described in Section 4.5.1. The equations of the LSTM unidirectional translation model are analogous to the case of an LSTM LM, as defined by Eqs. (4.21) to (4.28), except that at the input side, we input a source side word, and at the output side, we present a target side word, i.e., the input and output vocabularies differ. In the case of a unidirectional joint model, we replace Eq. (4.21) by

$$y(i) = A_0 \hat{f}(i+d) + A_1 \hat{e}(i-1) \tag{7.5}$$

such that an additional weight matrix A_0 is introduced for mapping a one-hot encoded source word $\hat{f}(i)$ into continuous space, and the resulting vector $A_0\hat{f}(i)$ is added to the projection layer activation $y(i)$. In the next section, bidirectional RNNs are described, which are the basis of the bidirectional translation model variants as defined by Eqs. (7.1) and (7.4).

	Translation Model	Joint Model
Unidirectional	$\prod_{i=1}^{I} p(e_i \mid f_1^i)$	$\prod_{i=1}^{I} p(e_i \mid e_1^{i-1}, f_1^i)$
Bidirectional	$\prod_{i=1}^{I} p(e_i \mid f_1^I)$	$\prod_{i=1}^{I} p(e_i \mid e_1^{i-1}, f_1^I)$

Table 7.1 Word-based variants for LSTM translation modeling, inducing different factorizations of the probability $p(e_1^I|f_1^J)$.

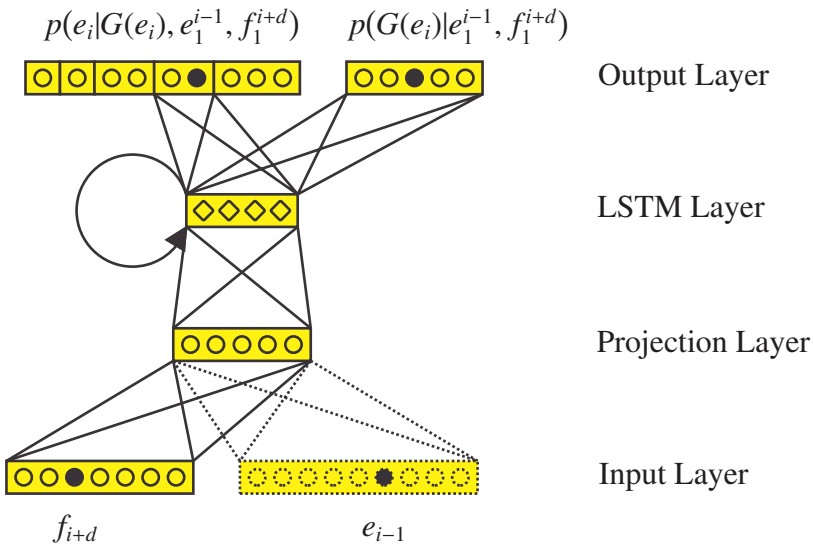

Figure 7.1 Architecture of a recurrent unidirectional translation model. By including the dotted parts, a *joint* model is obtained.

7.1.1 Bidirectional Recurrent Neural Networks

Bidirectional recurrent neural networks were first proposed in [Schuster & Paliwal 97] and applied to speech recognition. They have since been used for different tasks like parsing [Henderson 04], handwriting recognition [Graves & Liwicki$^+$ 09] and spoken language understanding [Mesnil & He$^+$ 13]. While the unidirectional translation models include an unbounded source sentence history, they are still limited in the number of future source words that are included. Bidirectional models provide a flexible means to also include an unbounded future context, which, unlike the delayed unidirectional model, requires no tuning of the delay parameter d. A bidirectional RNN joint translation model is defined by the following equations.

$$x^+(i) = A_0 \hat{f}(i) + A_1 \hat{e}(i-1) \tag{7.6}$$

$$x^-(i) = A_0 \hat{f}(i) \tag{7.7}$$

$$y^+(i) = \mathrm{sig}(A_2 x^+(i) + A_3 y^+(i-1)) \tag{7.8}$$

$$y^-(i) = \mathrm{sig}(A_4 x^-(i) + A_5 y^-(i+1)) \tag{7.9}$$

$$z(i) = \mathrm{sig}(A_6 y^+(i) + A_7 y^-(i) + A_8 z(i-1)) \tag{7.10}$$

$$p_e(e_i|G(e_i), e_1^{i-1}, f_1^I) = \varphi(A_{9,G(e_i)} z(i))\Big|_{e_i} \tag{7.11}$$

$$p_G(G(e_i)|e_1^{i-1}, f_1^I) = \varphi(A_{10} z(i))\Big|_{G(e_i)} \tag{7.12}$$

The extension to an LSTM hidden layer instead of the standard RNN is straightforward. The main difference of the bidirectional RNN to a unidirectional RNN lies in the duplication of the recurrent hidden layer $y(i)$, which now consists of a hidden layer $y^+(i)$, where information is processed in forwards direction, and another hidden layer $y^-(i)$, operating in backwards direction. As can be seen from the dependences in the recurrent equations (7.8) and (7.9), bidirectional neural networks require a modified evaluation order in training as well as in testing. While unidirectional recurrent neural network variants can proceed from left to right over the source sequence, the RNN layer $y^+(i)$ requires the evaluation of the complete source sentence from left to right, and the layer $y^-(i)$ requires a processing from right to left. Only then it is possible to compute the activations of the hidden layer $z(i)$, which merges the independent forward and backward processing branches. Jointly, the two branches include the full source sentence f_1^I. The gradient with respect to the input activations of the layers is given by the equations

$$\mathrm{err}_{p_e}(i) = p_e(\cdot|e_1^{i-1}, G(e_i), f_1^I) - \hat{e}_G(i) \tag{7.13}$$

$$\mathrm{err}_{p_G}(i) = p_G(\cdot|e_1^{i-1}, f_1^I) - \hat{G}(i) \tag{7.14}$$

$$\mathrm{err}_z(i) = (1 - z(i)) \odot z(i) \odot (A_{9,G(e_i)}^T \mathrm{err}_{p_e}(i) + A_{10}^T \mathrm{err}_{p_G}(i) + A_8^T \mathrm{err}_z(i+1)) \tag{7.15}$$

$$\mathrm{err}_{y^-}(i) = (1 - y^-(i)) \odot y^-(i) \odot (A_7^T \mathrm{err}_z(i) + A_5^T \mathrm{err}_{y^-}(i-1)) \tag{7.16}$$

$$\mathrm{err}_{y^+}(i) = (1 - y^+(i)) \odot y^+(i) \odot (A_6^T \mathrm{err}_z(i) + A_3^T \mathrm{err}_{y^+}(i+1)), \tag{7.17}$$

where we stick to the notational conventions introduced in Section 4.5.1. Fig. 7.2 illustrates the bidirectional architecture. The figure as well as Eqs. (7.6) to (7.12) show the deep variant of the bidirectional model, where the forward and backward layers converge into another recurrent hidden layer. A shallow variant can be obtained if the parallel layers converge into the output layer directly. In the implementation, the forward and backward layers converged into an intermediate identity layer, and the aggregate was weighted and fed into the next layer.

7.1.2 Resolving Alignment Ambiguities

Word-based recurrent models are only defined for one-to-one-aligned source-target sentence pairs. In this work, we always evaluate the model in the order of the target sentence. However, we experiment with several different ways to resolve ambiguities due to unaligned or multiply aligned words. To that end, we introduce two additional tokens, $\epsilon_{aligned}$ and $\epsilon_{unaligned}$. Unaligned words are either removed or aligned to an extra $\epsilon_{unaligned}$ token on the opposite side. If an $\epsilon_{unaligned}$ is introduced on the target side, its position is determined by the aligned source word that is closest to the unaligned source word in question, preferring left to right. To resolve one-to-many alignments, we use an IBM-1 translation table to decide for one of the alignment connections to be kept. The remaining words are also either deleted or aligned to additionally introduced $\epsilon_{aligned}$ tokens on the opposite side. Fig. 7.3 shows an

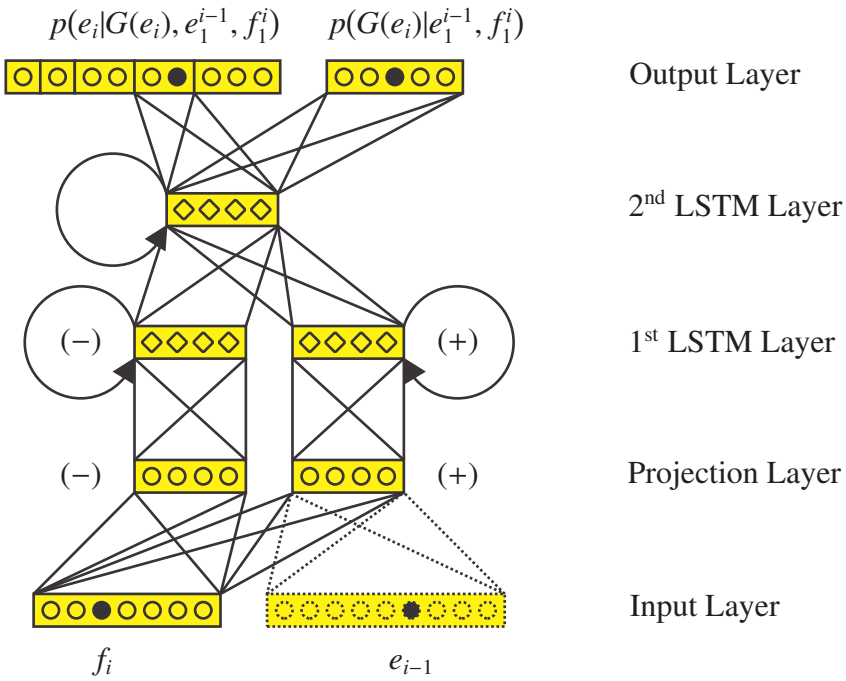

Figure 7.2 Architecture of a recurrent bidirectional translation model. By (+) and (−), we indicate a processing in forward and backward time direction, respectively. The inclusion of the dashed parts leads to a bidirectional *joint* translation model. A tied source word projection matrix is used for the forward and backward branches.

example sentence from the IWSLT data, where all ϵ tokens are introduced. In this way, a one-to-one alignment is obtained, and by enriching the source and target vocabulary by the additional ϵ tokens, the pre-processed source and target sentences are used as the basis of the four word-based translation model variants.

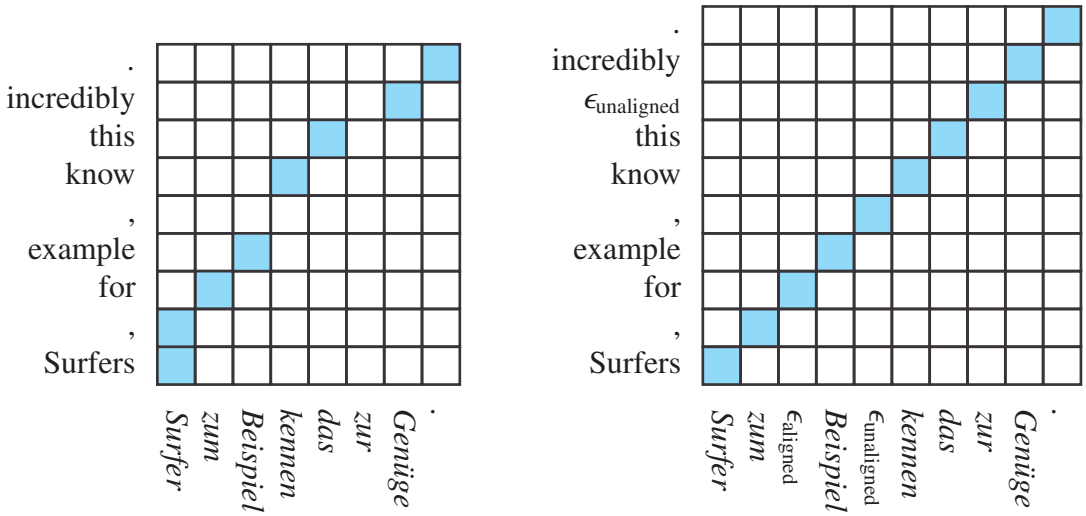

Figure 7.3 Example sentence from the German to English IWSLT data. The one-to-one alignment is created by introducing $\epsilon_{aligned}$ and $\epsilon_{unaligned}$ tokens.

7.2 Phrase-Based Translation Models

One of the conceptual disadvantages of word-based modeling as introduced in the previous section is that there is a mismatch between training and testing conditions: During neural network training, the vocabulary has to be extended by additional ϵ tokens, and a one-to-one alignment is used which does not reflect the situation in decoding. In phrase-based machine translation, more complex alignments in terms of multiple words on both the source and the target side are used, that allow the decoder to make use of richer short-distance dependences and that are crucial for the performance of the resulting system.

From this perspective, it seems interesting to harmonize the alignments used in neural network training as well as in machine translation decoding. However, it is difficult to use the phrases themselves as the vocabulary of the LSTM. Usually, the huge number of potential phrases in comparison to the relatively small amount of training data makes the learning of continuous phrase representations difficult due to data sparsity. This was confirmed by results presented in [Le & Allauzen$^+$ 12b], which showed that a word-factored translation model outperformed a phrase-factored version. Therefore, we continue relying on source and target word vocabularies for building phrase representations. However, a direct correspondence between a source and a target word is no longer assumed, which was enforced in word-based models.

Fig. 7.4 shows an example phrase alignment $\mathcal{A}$, where a sequence of source words $\tilde{f}_k$ is directly mapped to a sequence of target words $\tilde{e}_k$ for $1 \leq k \leq 6$. In the general case, we denote the number of phrase pairs in an alignment by K. We then decompose the target sentence posterior probability in the following way:

$$p(e_1^I, \mathcal{A}|f_1^J) = \prod_{k=1}^{K} p(\tilde{e}_k, \mathcal{A}|\tilde{e}_1^{k-1}, \tilde{f}_1^K) \tag{7.18}$$

$$= \prod_{k=1}^{K} p(\tilde{e}_k, \mathcal{A}|\tilde{e}_1^{k-1}, \tilde{f}_1^k), \tag{7.19}$$

where the phrase-based joint model in Eq. 7.18 corresponds to a bidirectional LSTM, and Eq. 7.19 only requires a unidirectional LSTM. In principle, the word-based translation model given by Eqs. (7.1) to (7.3) equally depends on an alignment $\mathcal{A}$, however, it does not occur in the formulas as the source sentence and the target sentence have already been aligned in a preprocessing step. In practice, the dependence on the alignment is not considered due to the maximum approximation in decoding. By leaving out the conditioning on the target side, we obtain a phrase-based bidirectional translation model, or a phrase-based translation model. Analogously to Table 7.1, the phrase-based translation model variants are summarized in Table 7.2.

As there is no one-to-one correspondence between the words within a phrase, the basic idea of the phrase-based approach is to let the neural network learn the dependences itself, and to present the full source side of the phrase to the network before letting it predict target side words. Then the probability for the target side of a phrase can be computed, in the case of Eq. 7.19, by

$$p(\tilde{e}_k, \mathcal{A}|\tilde{e}_1^{k-1}, \tilde{f}_1^k) = \prod_{\ell=1}^{|\tilde{e}_k|} p((\tilde{e}_k)_\ell, \mathcal{A}|(\tilde{e}_k)_1^{\ell-1}, \tilde{e}_1^{k-1}, \tilde{f}_1^k), \tag{7.20}$$

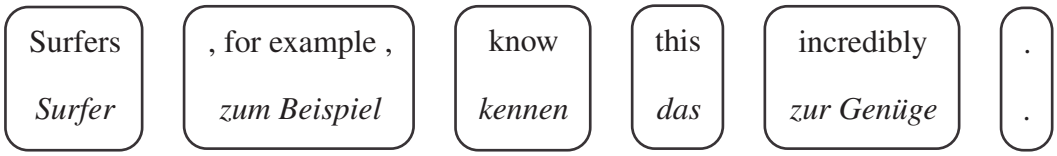

Figure 7.4 Example phrase alignment for a sentence from the IWSLT training data. Target words are printed in normal face, while source words are printed in italic face.

where $(\tilde{e}_k)_\ell$ denotes the ℓ-th word of the k-th aligned target phrase. The decomposition for the remaining three phrase-based translation modeling variants are analogous to Eq. 7.20. We feed the source side of a phrase into the neural network one word at a time. Only when the presentation of the source side is finished we start estimating probabilities for the target side. Therefore, we do not let the neural network learn a target distribution until the very last source word was processed. In this way, the conventional RNN training scheme, where an input sample is directly followed by its corresponding teacher signal, is broken up. Similarly, the presentation of the source side of the next phrase only starts after the prediction of the current target side is completed.

To this end, a no-operation token is introduced, denoted by ε, which is not part of the vocabulary (which means it cannot be input to or predicted by the neural network). When the ε token occurs as input, it indicates that no input needs to be processed by the neural network. When the ε token occurs as a teacher signal for the LSTM, the output layer distribution is ignored, and it does not even have to be computed. In both cases, all the other layers are still processed during forward and backward passes such that the LSTM state can be advanced even without additional input or output. For a source phrase $\tilde{f}_k$, we include $(|\tilde{e}_k| - 1)$ many ε symbols at the end of the phrase. Conversely, for a target phrase $\tilde{e}_k$, we include $(|\tilde{f}_k| - 1)$ many ε symbols at the beginning of the phrase.

Fig. 7.5 depicts the evaluation of the phrase-based joint model at the second phrase-pair from the example alignment shown in Fig. 7.4. The training of the German phrase "*zum Beispiel*" and its English translation "*, for example ,*" is shown. At the input layer, the source words are fed in one word at a time, while ε tokens are presented at the target side input layer and the output layer (with the exception of the very first time step, where we still have the last target word from the previous phrase as input instead of ε). With the last word of the source phrase "*Beispiel*" being presented to the network, the full source phrase is stored in the hidden layer, and the neural network is then trained to predict the target phrase words at

	Translation Model	Joint Model
Unidirectional	$\prod_{k=1}^{K} p(\tilde{e}_k, \mathcal{A} \mid \tilde{f}_1^k)$	$\prod_{k=1}^{K} p(\tilde{e}_k, \mathcal{A} \mid \tilde{e}_1^{k-1}, \tilde{f}_1^k)$
Bidirectional	$\prod_{k=1}^{K} p(\tilde{e}_k, \mathcal{A} \mid \tilde{f}_1^K)$	$\prod_{k=1}^{K} p(\tilde{e}_k, \mathcal{A} \mid \tilde{e}_1^{k-1}, \tilde{f}_1^K)$

Table 7.2 Phrase-based variants for LSTM translation modeling, resulting in different factorizations of the probability $p(e_1^I, \mathcal{A} | f_1^J)$.

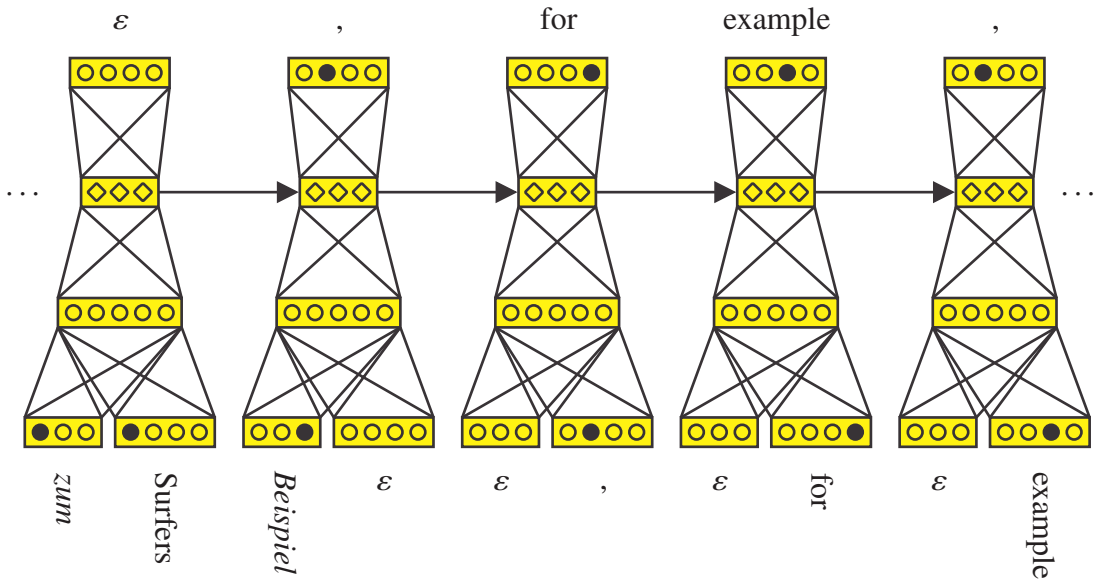

Figure 7.5 An LSTM phrase-based joint translation model, unfolded over time.

the output layer. Subsequently, the source input is ε, and the target input is the most recent target side history word.

To obtain a phrase-aligned training sequence for the phrase-based translation models, a forced alignment of the training data can be computed, applying the leaving-one-out principle as described in [Wuebker & Mauser+ 10].

7.3 Experimental Results

All translation experiments were performed with the Jane toolkit [Vilar & Huck+ 10, Wuebker & Huck+ 12]. The largest part of the experiments was carried out on the IWSLT 2013 German to English shared translation task. The corresponding system is described in Appendix A.5. To confirm our results, additional experiments were run on the Arabic to English and Chinese to English tasks of the DARPA BOLT project. Details on the BOLT systems can be found in Appendix A.6.1 and A.6.2. In all cases, neural network models were added on top of the best RWTH evaluation systems by rescoring 1,000-best lists, as word lattices are not supported by the Jane toolkit. All results are measured in case-insensitive BLEU [Papineni & Roukos+ 02] and TER [Snover & Dorr+ 06] on a single reference.

7.3.1 Phrase-Based Translation Models

The results for the phrase-based translation modeling approach are summarized in Table 7.3. Here, we also compare against a feedforward joint model as described by [Devlin & Zbib+ 14], with a source window of 11 words and a target history of three words. Instead of part-of-speech tags, 1,000 word classes were obtained according to a bigram perplexity-based training criterion. A shortlist of 16 K words was used. The remaining words were only distinguished on the class level. All neural networks were trained on the TED portion of

		Dev		Test	
		BLEU	TER	BLEU	TER
Baseline		33.5	45.8	30.9	48.4
+Feedforward	JM	34.4	44.9	31.9	47.6
+Unidirectional LSTM	TM	34.3	44.9	**32.1**	47.5
	JM	34.3	45.0	32.0	47.5
	JM (10-best)	34.4	44.8	32.0	47.3
+Bidirectional LSTM	TM	34.3	45.2	32.0	47.5
	TM (deep)	34.5	**44.6**	32.0	**47.1**
	JM	34.3	45.0	31.7	47.5
	JM (deep)	**34.8**	44.9	31.9	47.5

Table 7.3 Results for the IWSLT 2013 German to English task with different phrase-based LSTM models.

the data, comprising 138 K segments. At this point, an LSTM LM is not yet included in the baseline, which will be reported in Section 7.3.3.

It can be seen that all neural network variants give consistent gains in BLEU as well as in TER, where six out of seven LSTM phrase-based translation modeling variants outperform the feedforward approach from [Devlin & Zbib$^+$ 14] on the test data. A phrase-based model can also be trained on multiple variants for the phrase alignment, to improve generalization towards unseen phrase alignments. For the experiments, 10-best alignments were analyzed in comparison to the single best alignment, which resulted in a small improvement of 0.2 TER on both the development and the test data in the case of the joint model. In general, when adding another hidden layer, small improvements in translation quality can be observed. However, we did not obtain consistent gains by using bidirectional models. To some extent, future information is already considered in unidirectional phrase-based models by feeding the complete source side before predicting the target side, which explains why bidirectional models were not found to be effective.

7.3.2 Word-Based Translation Models

In this section, we experimentally analyze word-based translation models. First, we investigate details of the one-to-one alignment that impact performance of the neural network approaches, and determine optimum settings for subsequent experiments. We then present the results of the previously described word-based variants, and we relate the performance of bidirectional LSTMs to time-delay LSTM networks.

One-to-One Alignment Variants

In a preliminary experiment, different setups were evaluated with a unidirectional word-based LSTM translation model. Given the two ϵ tokens $\epsilon_{\text{aligned}}$ and $\epsilon_{\text{unaligned}}$ that can be introduced on the source as well as on the target side to enforce a one-to-one correspondence, the following alignment variants were distinguished:

1. one-to-one alignments without any ϵ token,

2. without $\epsilon_{\text{unaligned}}$,

3. with both ϵ tokens, but ϵ inserted only on the source side,

4. with both ϵ tokens, but ϵ inserted only on the target side, and

5. with both ϵ tokens on both sides.

Going without a particular ϵ token implies that the word that would have been aligned to that ϵ token is deleted from the sentence to ensure a one-to-one correspondence. The results can be found in Tab. 7.4. We use the setup with all ϵ tokens in the following experiments, which showed the best BLEU performance.

	Dev		Test	
	BLEU	TER	BLEU	TER
Baseline	33.5	45.8	30.9	48.4
+No ϵ token	34.2	45.3	31.8	47.7
+No $\epsilon_{\text{unaligned}}$	34.4	44.8	31.7	47.4
+Source side only	34.5	44.6	31.9	**47.0**
+Target side only	34.5	45.0	31.9	47.5
+All ϵ tokens	**34.6**	**44.5**	**32.0**	47.1

Table 7.4 Comparison of including different sets of ϵ tokens into the one-to-one alignment on the IWSLT 2013 German to English task using the unidirectional word-based LSTM translation model.

Word-Based Translation Modeling

Our results on the IWSLT German to English task are summarized in Table 7.5. The delay parameter d from Eqs. (7.2) and (7.3) is set to zero. We observe that for all LSTM translation model variants, we achieve substantial improvements over the baseline on the test data, ranging from 0.9 BLEU up to 1.6 BLEU. These results are also consistent with the improvements in terms of TER, where we achieve reductions from 0.8 TER up to 1.8 TER. While the unidirectional word-based variants are on par with phrase-based variants, the word-based approach is considerably improved by including future source side information through bidirectional LSTMs, outperforming all other neural network approaches.

Interestingly, for the word-based LSTM models, on the test data it can be seen that TMs perform better than JMs, even though TMs do not take advantage of target side history words. However, exploiting this extra information does not always need to result in a better model, as the target side words are only derived from the given source side, which is available to both TMs and JMs. By adding another LSTM layer that combines forward and backward time directions, we obtain our overall best model.

Time-Delay Translation Models

In Fig. 7.6 we compare the word-based bidirectional TM with a unidirectional TM that uses different time delays $d = 0, \ldots, 4$. For a delay $d = 2$, the same performance is obtained as with

		Dev		Test	
		BLEU	TER	BLEU	TER
Baseline		33.5	45.8	30.9	48.4
+Feedforward	JM	34.4	44.9	31.9	47.6
+Unidirectional LSTM	TM	34.6	44.5	32.0	47.1
	JM	34.7	44.7	31.8	47.4
+Bidirectional LSTM	TM	34.7	44.9	32.3	47.0
	TM (deep)	34.8	44.3	**32.5**	46.7
	JM	34.7	44.5	32.1	47.0
	JM (deep)	**34.9**	**44.1**	32.2	**46.6**

Table 7.5 Results for the IWSLT 2013 German to English task with different word-based LSTM models.

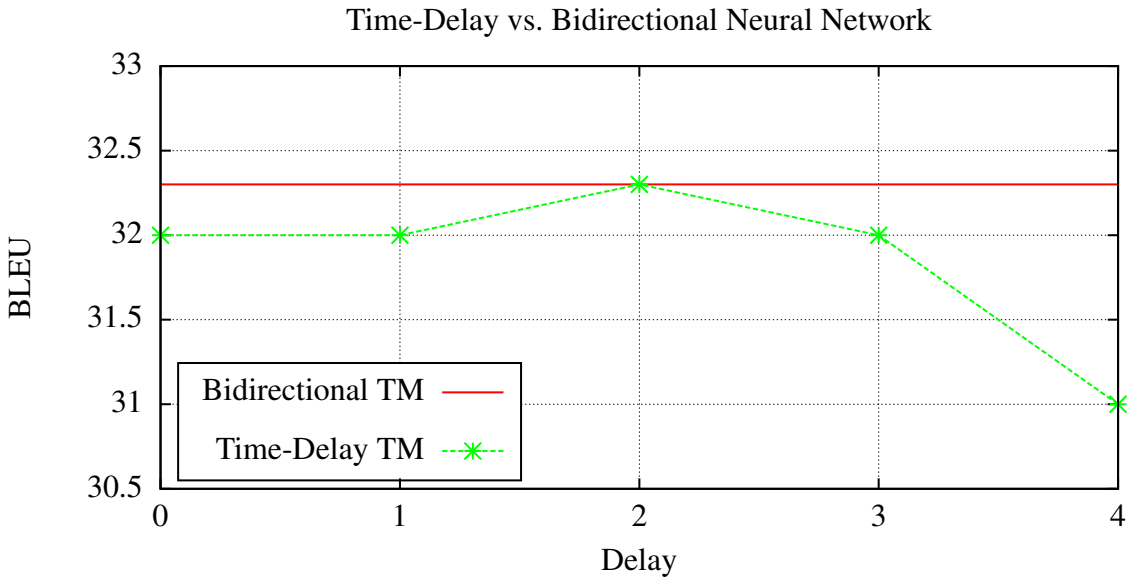

Figure 7.6 BLEU scores on the IWSLT test data for the bidirectional TM in comparison to unidirectional time-delay TMs, for different delay values.

the bidirectional model. This shows that a bidirectional model is not required for optimum performance, but when using a time-delay instead, this comes at the price of manually tuning the delay parameter.

7.3.3 Model Combination

Table 7.6 shows different model combination results for the IWSLT task, where an LSTM LM is included in the baseline. Adding a deep word-based bidirectional JM or TM to the recurrent language model improves the baseline by 1.1 BLEU or 1.2 BLEU, respectively. A phrase-based model substantially improves over the LSTM LM baseline, but performs not as good as its word-based counterparts. By adding four different translation models, including models in reverse word order and reverse translation direction, we are able to

improve translation quality even further. However, especially on the test data, the gains from model combination saturate quickly.

	Dev		Test	
	BLEU	TER	BLEU	TER
Baseline+LSTM LM	34.3	44.8	31.5	47.8
+Word-Based Bidirectional TM (deep)	34.9	**43.7**	**32.7**	**46.1**
+Word-Based Bidirectional JM (deep)	**35.0**	44.4	32.6	46.5
+Phrase-Based Bidirectional JM (deep)	34.8	44.6	32.3	47.2
+4 LSTM models	**35.2**	**43.4**	**32.7**	**46.0**

Table 7.6 Results for the IWSLT 2013 German to English task with different LSTM models. All results include an LSTM LM.

Apart from the IWSLT track, we also analyze the performance of our translation models on the BOLT Chinese to English and Arabic to English translation tasks. Due to the large amount of training data, we concentrate on models giving best performance in the IWSLT experiments. The results can be found in Tab. 7.7 and 7.8. In both cases, we see consistent improvements over the LSTM LM baseline, improving the Arabic to English system by 0.6 BLEU and 0.5 TER on the first test set. This can be compared to the rescoring results for the same task reported by [Devlin & Zbib$^+$ 14], where they achieved 0.3 BLEU, despite the fact that they used multiple references for scoring, whereas in our experiments we rely on a single reference only. The models are also able to improve the Chinese to English system by 0.5 BLEU and 0.5 TER on the second test set.

	Test 1		Test 2	
	BLEU	TER	BLEU	TER
Baseline	25.2	57.4	26.8	57.3
+Bidirectional TM (deep)	25.6	**56.6**	26.8	56.7
+Bidirectional JM (deep)	25.9	56.9	27.4	56.7
+LSTM LM	25.6	57.1	27.5	56.7
+Bidirectional TM (deep)	25.9	56.7	27.3	56.8
+Bidirectional JM (deep)	**26.2**	**56.6**	**27.9**	**56.5**

Table 7.7 Results for the BOLT Arabic to English task with different word-based LSTM models.

7.3.4 Analysis

To investigate whether bidirectional models benefit from future source information, we compare the single-best output of a system reranked with a unidirectional model to the output reranked with a bidirectional model. We choose the models to be translation models in both cases, as they predict target words independent of previous predictions, given the source information, cf. Eqs. (7.3) and (7.4). This makes it easier to detect the effect of including future source information or the lack thereof. The examples are taken from the IWSLT task, where we include the one-to-one source information, reordered according to the target side.

	Test 1		Test 2	
	BLEU	TER	BLEU	TER
Baseline	18.3	63.6	16.7	63.0
+Bidirectional TM (deep)	18.7	63.3	17.1	62.6
+Bidirectional JM (deep)	18.5	**63.1**	17.2	**62.3**
+LSTM LM	18.8	63.3	17.2	62.8
+Bidirectional TM (deep)	**18.9**	**63.1**	**17.7**	**62.3**
+Bidirectional JM (deep)	18.8	63.3	17.5	62.5

Table 7.8 Results for the BOLT Chinese to English task with different word-based LSTM models.

Reference:
Source: nicht **so wie** ich
Target: not **like** me

Hypothesis 1:
1-to-1 Source: **so** ich ϵ nicht **wie**
1-to-1 Target: **so** I do n't **like**

Hypothesis 2:
1-to-1 Source: nicht **so wie** ich
1-to-1 Target: not $\boldsymbol{\epsilon}$ **like** me

In this example, the German phrase "*so wie*" translates to "*like*" in English. The bidirectional model prefers hypothesis 2, making use of the future word "*wie*" when translating the German word "*so*" to ϵ, because it has future insight that this move will pay off later when translating the rest of the sentence. This information is not available to the unidirectional model, which prefers hypothesis 1 instead.

Reference:
Source: das taten wir dann auch und verschafften uns so **eine Zeit lang** einen Wettbewerbs Vorteil .
Target: and we actually did that and it gave us a competitive advantage **for a while** .

Hypothesis 1:
1-to-1 Source: das ϵ ϵ ϵ wir dann auch taten und verschafften uns so **eine Zeit lang** einen Wettbewerbs Vorteil .
1-to-1 Target: that 's just what we ϵ ϵ did and gave us ϵ **a time ,** a competitive advantage .

Hypothesis 2:
1-to-1 Source: das ϵ ϵ ϵ wir dann auch taten und verschafften uns so einen Wettbewerbs Vorteil $\boldsymbol{\epsilon}$ **eine Zeit lang** .
1-to-1 Target: that 's just what we ϵ ϵ did and gave us ϵ a competitive advantage **for a** $\boldsymbol{\epsilon}$ **while** .

Here, the German phrase "*eine Zeit lang*" translates to "*for a while*" in English. Bidirectional scoring favors hypothesis 2, while unidirectional scoring favors hypothesis 1. It seems that

the unidirectional model translates "*Zeit*" to "*time*" as the object of the verb "*give*" in hypothesis 1, being blind to the remaining part "*lang*" of the phrase which changes the meaning. The bidirectional model, to its advantage, has the full source information, allowing it to make the correct prediction.

7.4 Summary

Based on experience in the field of language modeling, it was motivated why LSTMs might be particularly suitable for modeling translation probabilities. A word-based and a phrase-based LSTM translation approach were presented in this chapter. The former is simple and performs well in practice, while the latter represents the generalization to arbitrary alignments and is more consistent with the phrase-based paradigm. Both approaches inherently evade data sparsity problems as they work on words in their lowest levels of processing. Our experiments show the models are able to achieve notable improvements over baselines containing a state-of-the-art LSTM LM. To the best of our knowledge, no previous work on recurrent translation modeling used word identities as neural network input directly, and no previous work reported improvements over a neural network LM when a recurrent neural network translation model was used.

For each of the two approaches presented, four different factorizations of the target sentence posterior probability were considered. In particular, two bidirectional variants were described, that allow modeling past and future dependences of any length. Bidirectional models were also compared with a simpler approach, where a fixed time delay of the source sentence was introduced. While the bidirectional and the time-delay LSTM network achieved the same BLEU performance, the latter required a manual tuning of the delay parameter, which is not necessary for the bidirectional method. Besides its good performance in practice, the bidirectional architecture is of theoretical interest as it allows the exact modeling of posterior probabilities.

Chapter 8

Scientific Contributions

The goal of this thesis was to analyze the state of the art in language and translation modeling, and to improve existing models by refined approaches. In more detail, the following scientific contributions have been made.

Discounting for Count-Based Language Models The optimization of discount parameters, as used in count-based LMs that rely on absolute discounting, was investigated. It was proven that the resulting optimization problem is not concave, and thus it can only be expected that local optima are found during the optimization process.

A simple optimization framework for discount parameters was introduced, where the perplexity on held-out data is minimized. In this way, discount parameters can be optimized at negligible computational costs. In particular, the optimization is independent of the size of the training data. Furthermore, the effect of optimized discount parameters was analyzed with respect to LM pruning. In all cases, small but consistent gains were observed when optimized discounts were used instead of conventional approximation formulas. Previous work indicated that no improvements can be obtained when the amount of training data is large, and a small number of discount parameters is sufficient. Both assumptions did not hold in our experiments.

Neural Network Language Models A comprehensive overview of neural network LM variants was presented. A conceptual drawback of state-of-the-art recurrent neural network LMs was outlined. As a solution, it was proposed for the first time to use recurrent LSTM neural networks for language modeling, and considerable improvements over any other neural network LM were observed. To reduce the number of parameters, it was suggested that LSTMs are combined with a linear projection layer. Furthermore it was shown that training times of LSTM neural network LMs can be reduced significantly by applying speed-up techniques like word classes.

Several aspects of training were investigated in detail, in particular the integrated training of neural network LMs and count-based LMs. While the performance on the training data could be improved, these gains did not generalize to unseen test data.

Rescoring with Neural Network Language Models In this thesis, the rescoring of neural network LMs was analyzed. Two competing concepts were investigated: The simple rerank-

ing of *n*-best lists and the rescoring of word lattices. In this way, the open question was addressed whether the increased search space of word lattices can provide a benefit for neural network LM rescoring. In a first step, the push-forward rescoring algorithm was adapted to LSTM neural network LMs, extending it by concepts known from first-pass decoding. Second, approximation strategies were developed that allow the creation of well-defined lattices incorporating neural network LM probabilities for all paths from the original first-pass lattice. In this way it was possible to minimize the word error rate in rescoring, whereas most other work only considers the minimization of the sentence error rate. Considerable improvements over *n*-best lists were obtained, while in other work a degradation in word error rate was reported when rescoring lattices instead of *n*-best lists. Finally, we performed a cheating experiment, providing an upper bound on the word error rate improvements that can be obtained with neural network LMs.

Comparison of Count-Based and Neural Network-Based Language Models This thesis introduced the LSTM concept to language modeling, and it also addressed the question how neural network LM architectures compare with each other and with count-based LMs.

In an experimental study it was found that LSTM networks outperform any other language modeling technique. Large improvements on top of a state-of-the-art speech recognition system could be obtained, even though the baseline count LM was trained on all available data. In addition, experiments showed for the first time consistent improvements with deep neural networks, in particular for recurrent LSTM architectures. For neural network training, more than one billion words were processed, and in a comparative experiment, for the training of a feedforward network even twenty billion words were used. In addition, state-of-the-art word classes, that were inferred based on a bigram training criterion, were used for the comparison to obtain the best possible results.

Regardless of the underlying language modeling technique, perplexities and word error rates were strongly correlated in the experiments carried out in this work. Furthermore, in this thesis it was also observed that LSTM neural network LMs can be trained using exact gradients, without techniques like gradient clipping or truncation, which are applied in most other works.

Implementation of Neural Network Language Models The experiments carried out in this thesis require an efficient implementation of the feedforward, recurrent and recurrent LSTM neural network LM architectures as defined in Sections 4.1, 4.2, and 4.3.3. In addition, an implementation of the stochastic gradient descent training algorithm is needed that is based on the backpropagation and backpropagation through time variants for computing the gradient with respect to the aforementioned neural network models.

To the best of our knowledge, no publicly available software supports the training of LSTM neural network LMs in particular, or neural network LMs in general using perplexity-based word classes. In addition, no implementation of lattice rescoring algorithms is publicly available beyond the simple replacement of probability values on lattice arcs, which requires a lattice expansion and is therefore infeasible in case the neural network LM order is large.

For this reason, a novel toolkit was developed to carry out state-of-the-art experiments on considerable amounts of data as presented in this thesis. The toolkit contains multiple tests to

verify the correctness of the gradient implementation, and it is available for download under an open source license [Sundermeyer & Schlüter$^+$ 14].

LSTM Neural Networks for Translation Modeling LSTM neural networks have been shown to give strong improvements for language modeling in this thesis. The generalization of LSTM LMs to the problem of translation modeling for statistical machine translation was subsequently investigated, too. Two novel translation model variants were introduced, both of which are based on sequential word input, and rely on either word or phrase alignments. With both approaches, it was possible to maintain word order of source and target language dependences. These models represent the first successful attempt to obtain considerable improvements with a recurrent translation model architecture on top of a baseline system already including a neural network LM.

The impact of source and target history dependences was analyzed. Unlike for language modeling, for translation modeling the explicit use of future dependences can be helpful, and in this thesis a time delay LSTM as well as a bidirectional LSTM neural network were introduced, which led to additional gains in translation quality.

Chapter 9

Outlook

Neural networks represent a major breakthrough in speech recognition and related natural language processing technologies. Regarding the research problems covered in this thesis, many questions still remain unanswered, and these problems will need to be addressed in future research. The following areas appear to be of particular interest.

Improved Training Algorithms Large improvements have been obtained with deep recurrent LSTM neural network models. It would be interesting to see the improvements that are possible when much more data could be used for training the LSTM, and the size of individual layers as well as the number of stacked layers could be increased until no further improvements are obtained. Such an experiment would be computationally very expensive.

While neural networks can be trained on huge amounts of data by using massively parallelized systems, see e.g. [Dean & Corrado$^+$ 12], it is not clear to which extent these algorithms are applicable to language modeling or LSTMs, and whether the approximations involved would affect performance. In the end, a better optimization algorithm might need to be found for learning complex neural network language and translation models on large amounts of data at acceptable computational costs.

Features for Language and Translation Modeling All models investigated in this thesis relied on mere words as training data. Such a modeling approach seems favorable due to its simplicity, allowing the integration into standard speech recognition and machine translation systems.

On the other hand, neural networks represent a framework that allows the easy incorporation of other discrete or continuous-valued features that may lead to additional improvements. Future research might investigate the potential of features that are either derived from word identities or represent meta information that can be helpful for improving performance.

Improved Neural Network Modeling The research carried out in this thesis indicates that the performance of a neural network language or translation model strongly depends on the network architecture that is used. According to comparative experiments, LSTMs seem to give best results. However, it is likely that the LSTM paradigm can be improved, and future research will certainly concentrate on finding neural networks that better exploit the available data and result in lower perplexities and higher BLEU scores.

Decoding for Machine Translation According to experimental evidence, it seems unlikely that using neural network LMs directly for first-pass speech decoding would result in significant improvements compared to lattice rescoring. However, the decoding problem of machine translation is considerably more complex, and more promising results have been reported when using feedforward neural network language and translation models directly in decoding. Future work may consider using recurrent neural network variants for machine translation decoding directly.

Appendix A

Corpora and Systems

In this chapter, the training corpora are summarized that were used for the experimental evaluations. In addition, automatic speech recognition as well as statistical machine translation systems are described that form the baseline for improved language and translation models developed in this thesis.

A.1 Wall Street Journal

The Wall Street Journal-based Continuous Speech Recognition Language Model Corpus (WSJ) has been widely used for language modeling experiments. Table A.1 summarizes

Corpus		Running Words	Vocabulary
Train	Full	227.3 M	20 K
	Reduced	13.3 M	
Dev		207.8 K	
Test		207.2 K	

Table A.1 Language modeling data from the Wall Street Journal corpus.

the number of running words for the individual data sets. To the best of our knowledge, besides the data devoted to LM training, the WSJ corpus only contains a so-called 'set-aside' data set. From this corpus, we extracted another reduced training set, a development and a test set. The reduced training set was used for the approach presented in Section 4.6.5, where two disjoint training corpora are required.

A.2 Treebank

The Treebank-3 corpus is a subsection of the WSJ corpus, comprising roughly one million running words. This corpus has been used by man language modeling researchers, especially the preprocessed version as used in [Mikolov & Karafiát$^+$ 10], which is publicly available for download and thus allows a direct comparison to the results that can be obtained with various language modeling techniques. Further details are summarized in Table A.2.

Corpus	Running Words	Vocabulary
Train	930 K	
Dev	74 K	10 K
Test	82 K	

Table A.2 LM training data of the Treebank corpus.

A.3 Quaero

Within the project *Quaero*[1], state-of-the-art speech recognition systems for several European languages were developed, two of which are considered for the experiments in this thesis. Both systems obtained the best word error rate results in the final 2013 evaluation of the project.

A.3.1 English

For acoustic modeling, a Gaussian mixture model was trained on manually transcribed broadcast news and broadcast conversations, comprising 250 hours of speech in total. The system included multilingual bottleneck multi-layer perceptron (MLP) features as proposed in [Tüske & Schlüter$^+$ 13], following the tandem approach from [Hermansky & Ellis$^+$ 00]. The MLP features were trained on a superset of 840 hours of English, French, German, and Polish data. More details on the acoustic setup of the system can be found in [Sundermeyer & Nußbaum-Thom$^+$ 11]. Table A.3 summarizes the corpora that were used for training the LMs of the English system. For all the available training data Kneser-Ney-smoothed count

Corpus		Running Words	Vocabulary
Train	Full	3.1 B	150 K
	Reduced	50 M	
Dev		39 K	
Test		35 K	

Table A.3 LM training corpora for Quaero English.

LMs [Kneser & Ney 95] were estimated on each of the individual text corpora. Then these LMs were linearly interpolated, optimizing the interpolation weights such that the perplexity on the development data was minimized. Finally, the corpora having a non-negligible interpolation weight were selected for the full training data set, comprising 3.1 B running words. On these data, the final Kneser-Ney model was estimated that was used for first-pass decoding. As it is too time-consuming to train a neural network LM on the full data set, a subset of 50 M running words was selected. The data set was chosen from the full data set by including the in-domain data first, and then adding more data from the second most relevant data source. To sort the data sources by relevance, the interpolation weights of the large Kneser-Ney model were considered, where the weights were normalized with respect to the number of running words per corpus. Compared to this simple approach we did not

[1] http://www.quaero.org

obtain perplexity improvements by more sophisticated data selection strategies like [Moore & Lewis 10].

A.3.2 French

The Quaero French speech recognition system closely resembles the architectural details of the English system. The Gaussian mixture model was trained on 350 hours of broadcast news and broadcast conversations. Multilingual MLP features were extracted for the French data based on the same MLP model as used for the English system.

The French LM data are shown in Table A.4. The French data set for neural network

Corpus		Running Words	Vocabulary
Train	Full	1.6 B	200 K
	Reduced	100 M	
Dev		35 K	
Test		41 K	

Table A.4 LM training data for Quaero French.

LM training comprises 100 M running words that were selected based on the same selection techniques as described in the previous section. The neural network LM experiments on the French data were run with a smaller hidden layer size compared to the English experimental settings, thereby allowing a larger amount of training data using a fixed quota of computational resources.

A.4 Babel Assamese

For the IARPA project *Babel*, an Assamese speech recognition system was developed that was also applied to keyword search. The full language pack setup was used according to the 'BaseLR' conditions. The acoustic training data comprised 120 hours of manually transcribed conversational telephone speech, where the audio data were separated by speaker. However, after segmentation and silence normalization only 50 hours remained. Similar to the Quaero systems, mono-lingual hierarchical bottleneck MLP features were extracted from the training data and included in the feature set of a Gaussian mixture based acoustic model. More details about the Assamese system can be found in [Tüske & Schlüter$^+$ 13, Tüske & Nolden$^+$ 14].

The LM training data are given in Table A.5. Due to the small amount of available data, for Kneser-Ney smoothed count LMs as well as neural network LMs, the full amount of training data was used. Within the project, no test data are distributed among project partners.

A.5 IWSLT English to German

The data of the 2013 evaluation campaign of the International Workshop on Spoken Language Translation (IWSLT) were used for the English to German translation system. The

Corpus	Running Words	Vocabulary
Train	405.9 K	22 K
Dev	66.5 K	

Table A.5 Babel Assamese LM training corpora.

state-of-the-art baseline is a phrase-based statistical machine translation system [Koehn & Och+ 03] tuned with MERT [Och 03]. The system was trained on all available bilingual data, comprising 4.3 M sentence pairs in total. It contains a hierarchical reordering model [Galley & Manning 08] and a 7-gram count LM based on word classes [Wuebker & Peitz+ 13]. A 4-gram count LM with modified Kneser-Ney smoothing was used. For count LM training, in addition to the target side of the bilingual data, parts of the Shuffled News corpus and the LDC English Gigaword corpus were selected using a cross-entropy difference criterion [Moore & Lewis 10]. This resulted in a total of 1.7 B running words for count LM training.

Corpus		Sentences	Source Vocabulary	Target Vocabulary
Train	Full	4.3 M	41 K	30 K
	Reduced	137.9 K		
Dev		0.9 K		
Test		1.6 K		

Table A.6 Bilingual training data for the English to German IWSLT 2013 system.

An overview of the bilingual training data is given in Table A.6. All neural network LMs and neural network translation models were trained on the in-domain TED portion of the data, denoted the reduced training data set in the table.

A.6 BOLT

Within the DARPA-funded project *BOLT*, several machine translation systems were developed. The most competitive Arabic to English and the Chinese to English translation systems developed by RWTH serve as baseline systems for neural network language and translation modeling experiments in this thesis.

A.6.1 Arabic to English

The Arabic to English system is a standard phrase-based decoder trained on 6.6 M sentences using 17 dense features. The statistics of the bilingual training data are summarized in Table A.7. The two test sets were chosen from the 'discussion forum' domain. The reduced training corpus is used for training the neural network LMs and the neural network translation models.

Corpus		Sentences	Source Vocabulary	Target Vocabulary
Train	Full	6.6 M	139 K	88 K
	Reduced	921.7 K		
Test 1		1.5 K		
Test 2		1.1 K		

Table A.7 Bilingual training data for the BOLT Arabic to English system.

A.6.2 Chinese to English

For the Chinese to English translation task, a hierarchical phrase-based system was trained on 3.7 M sentences with 22 dense features, including an advanced orientation model [Huck & Wuebker+ 13]. The available bilingual training corpora are described in Table A.8. For neural network training, a subset of 370 K sentences was selected, corresponding to 9 M running words.

Corpus		Sentences	Source Vocabulary	Target Vocabulary
Train	Full	3.7 M	55 K	76 K
	Reduced	369.2 K		
Test 1		1.8 K		
Test 2		1.1 K		

Table A.8 Bilingual training data for the BOLT Chinese to English system.

Appendix B

Publications

According to § 5.3 of the doctoral guidelines of RWTH Aachen University, Department of Computer Science, October 29th 2014, a list of referenced publications of the author of this thesis is given, describing his contributions.

- [Sundermeyer & Ney+ 15]: Martin Sundermeyer implemented the software, performed the experiments, and wrote the paper.
- [Sundermeyer & Alkhouli+ 14]: Martin Sundermeyer suggested using bidirectional LSTM networks for translation modeling. He also suggested using ε tokens for a novel LSTM training paradigm. He trained LSTM LMs for the IWSLT and BOLT baseline systems for all language pairs and provided JARA computing resources for the training of LSTM language and translation models. He contributed the LSTM LM code base for the translation model implementations. He implemented the phrase-based translation models, performed all phrase-based experiments including MERT optimization and scoring, and wrote major parts of the paper.
- [Sundermeyer & Tüske+ 14] Martin Sundermeyer suggested approximation techniques for lattice rescoring with neural network LMs, implemented the software, performed the experiments, and wrote the paper.
- [Sundermeyer & Schlüter+ 14] Martin Sundermeyer implemented the software, performed the experiments, and wrote the paper.
- [Sundermeyer & Oparin+ 13] Martin Sundermeyer implemented the LSTM software and supervised the implementation of the feedforward neural network by Ben Freiberg. Martin Sundermeyer performed all LSTM experiments as well as all system combination experiments, and wrote the paper.
- [Sundermeyer & Schlüter+ 12] Martin Sundermeyer suggested using LSTM neural networks for language modeling, implemented the software, performed the experiments, and wrote the paper.
- [Oparin & Sundermeyer+ 12] Martin Sundermeyer performed first experiments on order-wise perplexities.

- [Andrés-Ferrer & Sundermeyer$^+$ 12] Martin Sundermeyer contributed source code for discount optimization on held-out data and ran rescoring experiments for all word error rate experiments.
- [Sundermeyer & Schlüter$^+$ 11] Martin Sundermeyer proved the non-concavity of the optimization problem, implemented the software, performed the experiments, and wrote the paper.
- [Sundermeyer & Nußbaum-Thom$^+$ 11] Martin Sundermeyer developed the Quaero French speech recognition system and wrote the paper.

List of Figures

List of Tables

Bibliography

[Alleva & Huang+ 96] P. Alleva, X.D. Huang, M.Y. Hwang: Improvements on the pronunciation prefix tree search organization. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 1, pp. 133–136, Atlanta, GA, USA, May 1996.

[Andrés-Ferrer & Ney 09] J. Andrés-Ferrer, H. Ney: Extensions of absolute discounting (Kneser-Ney method). In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 4729–4732, Taipei, Taiwan, April 2009.

[Andrés-Ferrer & Sundermeyer+ 12] J. Andrés-Ferrer, M. Sundermeyer, H. Ney: Conditional leaving-one-out and cross-validation for discount estimation in Kneser-Ney-like extensions. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 5013–5016, Kyoto, Japan, March 2012.

[Arısoy & Sainath+ 12] E. Arısoy, T.N. Sainath, B. Kingsbury, B. Ramabhadran: Deep neural network language models. In *NAACL-HLT Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pp. 20–28, Montréal, QC, Canada, June 2012.

[Aubert 02] X.L. Aubert: An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech and Language*, Vol. 16, No. 1, pp. 89–114, Jan. 2002.

[Auli & Galley+ 13] M. Auli, M. Galley, C. Quirk, G. Zweig: Joint language and translation modeling with recurrent neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1044–1054, Seattle, WA, USA, Oct. 2013.

[Bahl & Brown+ 86] L.R. Bahl, P.F. Brown, P.V. de Souza, R.L. Mercer: Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 49–52, Tokyo, Japan, May 1986.

[Bahl & Jelinek+ 83] L.R. Bahl, F. Jelinek, R.L. Mercer: A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 5, pp. 179–190, March 1983.

[Bahl & Padmanabhan+ 96] L.R. Bahl, M. Padmanabhan, D. Nahamoo, P.S. Gopalakrishnan: Discriminative training of Gaussian mixture models for large vocabulary speech

recognition systems. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 613–616, Atlanta, GA, USA, May 1996.

[Baker 75] J.K. Baker: Stochastic modeling for automatic speech understanding. In D.R. Reddy, editor, *Speech Recognition*, pp. 512–542. Academic Press, New York, NY, USA, 1975.

[Bakis 76] R. Bakis: Continuous speech recognition via centisecond acoustic states. In *ASA Meeting*, Washington, DC, USA, April 1976.

[Baum 72] L.E. Baum: An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In O. Shisha, editor, *Inequalities*, Vol. 3, pp. 1–8. Academic Press, New York, NY, USA, 1972.

[Bayes 63] T. Bayes: An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions*, Vol. 53, pp. 370–418, Dec. 1763.

[Bellman 57] R.E. Bellman: Dynamic programming. Princeton University Press, 1957.

[Bengio & Ducharme$^+$ 00] Y. Bengio, R. Ducharme, P. Vincent: A neural probabilistic language model. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 933–938, Denver, CO, USA, Nov. 2000.

[Bengio & Ducharme$^+$ 03] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin: A neural probabilistic language model. *Journal of Machine Learning Research*, Vol. 3, pp. 1137–1155, Feb. 2003.

[Bengio & Simard$^+$ 94] Y. Bengio, P. Simard, P. Frasconi: Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 157–166, March 1994.

[Bilmes & Kirchhoff 03] J.A. Bilmes, K. Kirchhoff: Factored language models and generalized parallel backoff. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pp. 4–6, Edmonton, AB, Canada, May 2003.

[Bisani & Ney 04] M. Bisani, H. Ney: Bootstrap estimates for confidence intervals in ASR performance evaluation. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 409–412, Montréal, QC, Canada, May 2004.

[Bishop 95] C.M. Bishop: *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK, 1995.

[Blei & Ng$^+$ 03] D.M. Blei, A.Y. Ng, M.I. Jordan: Latent dirichlet allocation. *Journal of Machine Learning Research*, Vol. 3, pp. 993–1022, Jan. 2003.

[Bocchieri 93] E. Bocchieri: Vector quantization for the efficient computation of continuous density likelihoods. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 692–695, Minneapolis, MN, April 1993.

[Botros 15] R. Botros: Efficient training of word classes and application to language modeling. Master's thesis, RWTH Aachen University, Department of Computer Science, April 2015.

[Bottou 12] L. Bottou: Stochastic gradient descent tricks. In G. Montavon, G.B. Orr, K. Müller, editors, *Neural Networks: Tricks of the Trade*, pp. 421–436. Springer, Berlin and Heidelberg, Germany, 2012.

[Boulanger-Lewandowski & Bengio+ 13] N. Boulanger-Lewandowski, Y. Bengio, P. Vincent: High-dimensional sequence transduction. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 3178–3182, Vancouver, BC, Canada, May 2013.

[Bourlard & Morgan 93] H.A. Bourlard, N. Morgan: *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publihsers, Norwell, MA, USA, 1993.

[Boyd & Vandenberghe 95] S. Boyd, L. Vandenberghe: *Convex Optimization*. Cambridge University Press, New York, NY, USA, 1995.

[Brown & Della Pietra+ 92] P.F. Brown, V.J. Della Pietra, P.V. deSouza, R.L. Mercer: Class-based n-gram models of natural language. *Computational Linguistics*, Vol. 18, No. 4, pp. 467–479, March 1992.

[Castaño & Casacuberta 97a] M. Castaño, F. Casacuberta: A connectionist approach to machine translation. In *European Conference on Speech Communication and Technology (Eurospeech)*, pp. 91–94, Rhodes, Greece, Sept. 1997.

[Castaño & Casacuberta+ 97b] M. Castaño, F. Casacuberta, E. Vidal: Machine translation using neural networks and finite-state models. In *International Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, pp. 160–167, Santa Fe, NM, USA, July 1997.

[Castaño & Casacuberta 99] M. Castaño, F. Casacuberta: Text-to-text machine translation using the RECONTRA connectionist model. In *International Work-Conference on Artificial and Natural Neural Networks (IWANN)*, pp. 683–692, Alicante, Spain, May 1999.

[Castaño & Vidal+ 93] M.A. Castaño, E. Vidal, F. Casacuberta: Inference of stochastic regular languages through simple recurrent networks. In *IEE Colloquium on Grammatical Inference: Theory, Applications and Alternatives*, pp. 16/1–16/6, Colchester, UK, April 1993.

[Chelba & Brants+ 10] C. Chelba, T. Brants, W. Neveitt, P. Xu: Study on interaction between entropy pruning and Kneser-Ney smoothing. In *Interspeech*, pp. 2422–2425, Makuhari, Chiba, Japan, Sept. 2010.

[Chen & Goodman 99] S.F. Chen, J. Goodman: An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, Vol. 13, No. 4, pp. 359–393, Oct. 1999.

[Chen & Rosenfeld 00] S.F. Chen, R. Rosenfeld: A survey of smoothing techniques for ME models. *IEEE Transactions on Speech and Audio Processing*, Vol. 8, No. 1, pp. 37–50, Jan. 2000.

[Chen & Wang+ 14] X. Chen, Y. Wang, X. Liu, M.J.F. Gales, P.C. Woodland: Efficient GPU-based training of recurrent neural network language models using spliced sentence bunch. In *Interspeech*, pp. 641–645, Singapore, Sept. 2014.

[David & Mermelstein 80] S. David, P. Mermelstein: Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 87, No. 4, pp. 1738–1752, Aug. 1980.

[Dean & Corrado+ 12] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A.Y. Ng: Large scale distributed deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1223–1231, Lake Tahoe, NV, USA, Dec. 2012.

[Dempster & Laird+ 77] A. Dempster, N. Laird, D. Rubin: Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, Vol. 39, pp. 1 – 38, 1977.

[Deoras & Mikolov+ 11] A. Deoras, T. Mikolov, K. Church: A fast re-scoring strategy to capture long-distance dependencies. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1116–1127, Edinburgh, Scotland, UK, July 2011.

[Devlin & Zbib+ 14] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, J. Makhoul: Fast and robust neural network joint models for statistical machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1370–1380, Baltimore, MD, USA, June 2014.

[Duda & Hart+ 01] R.O. Duda, P.E. Hart, D.G. Stork: *Pattern Classification*. John Wiley & Sons, New York, NY, USA, 2001.

[Eide & Gish 96] E. Eide, H. Gish: A parametric approach to vocal tract length normalization. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 346–349, Atlanta, GA, USA, May 1996.

[Emami & Mangu 07] A. Emami, L. Mangu: Empirical study of neural network language models for arabic speech recognition. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 147–152, Kyoto, Japan, Dec. 2007.

[Evermann & Chan+ 04] G. Evermann, H.Y. Chan, M.J.F. Gales, T. Hain, X. Liu, L. Wang, D. Mrva, P.C. Woodland: Development of the 2003 CU-HTK conversational telephone speech transcription system. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 249–252, Montréal, QC, Canada, May 2004.

[Federico & Bertoldi+ 08] M. Federico, N. Bertoldi, M. Cettolo: IRSTLM: An open source toolkit for handling large scale language models. In *Interspeech*, pp. 1618–1621, Brisbane, QLD, Australia, Sept. 2008.

[Fiscus & Ajot+ 07] J.G. Fiscus, J. Ajot, J.S. Garofolo: Results of the 2006 spoken term detection evaluation. In *Special Interest Group on Information Retrieval Workshop*, pp. 51–57, Amsterdam, Netherlands, July 2007.

[Fisher 36] R.A. Fisher: The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, Vol. 7, pp. 179–188, 1936.

[Freiberg 13] B.J. Freiberg: Comparison of feedforward and recurrent neural network language models. Master's thesis, RWTH Aachen University, Department of Computer Science, June 2013.

[Fritsch 97] J. Fritsch: ACID/HNN: A framework for hierarchical connectionist acoustic modeling. In S. Furui, B.H. Juang, W. Chou, editors, *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 164–171, Santa Barbara, CA, USA, Dec. 1997.

[Gales 98] M.J.F. Gales: Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech and Language*, Vol. 12, pp. 75–98, 1998.

[Galley & Manning 08] M. Galley, C.D. Manning: A simple and effective hierarchical phrase reordering model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 848–856, Waikiki, Honolulu, Hawaii, USA, Oct. 2008.

[Gauvain 11] J.L. Gauvain: Personal communication, 2011.

[Gers & Schmidhuber+ 00] F.A. Gers, J. Schmidhuber, F. Cummins: Learning to forget: Continual prediction with LSTM. *Neural Computation*, Vol. 12, No. 10, pp. 2451–2471, Oct. 2000.

[Gers & Schraudolph+ 02] F.A. Gers, N.N. Schraudolph, J. Schmidhuber: Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, Vol. 3, pp. 115–143, Aug. 2002.

[Goodman 01a] J. Goodman: A bit of progress in language modeling. *Computer Speech and Language*, Vol. 15, No. 4, pp. 403–434, Oct. 2001.

[Goodman 01b] J. Goodman: Classes for fast maximum entropy training. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 561–564, Salt Lake City, UT, USA, May 2001.

[Graves 12] A. Graves: *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, Berlin and Heidelberg, Germany, 2012.

[Graves & Liwicki+ 09] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, J. Schmidhuber: A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 31, No. 5, pp. 855–868, May 2009.

[Grosicki & Abed 09] E. Grosicki, H.E. Abed: ICDAR 2009 handwriting recognition competition. In *International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1398–1402, Barcelona, Spain, July 2009.

[Gutmann & Hyvärinen 10] M. Gutmann, A. Hyvärinen: Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pp. 33–40, Sardinia, Italy, May 2010.

[Häb-Umbach & Ney 94] R. Häb-Umbach, H. Ney: Improvements in beam search for 10000-word continuous-speech recognition. *IEEE Transactions on Speech and Audio Processing*, Vol. 2, No. 2, pp. 353–356, April 1994.

[Henderson 04] J. Henderson: Discriminative training of a neural network statistical parser. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1–8, Barcelona, Spain, July 2004.

[Hermansky 90] H. Hermansky: Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, Vol. 87, No. 4, pp. 1738–1752, June 1990.

[Hermansky & Ellis+ 00] H. Hermansky, D. Ellis, S. Sharma: Tandem connectionist feature extraction for conventional HMM systems. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 1635–1638, Istanbul, Turkey, June 2000.

[Hochreiter & Schmidhuber 97] S. Hochreiter, J. Schmidhuber: Long short-term memory. *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, Nov. 1997.

[Hochreiter & Bengio+ 01] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In J.F. Kolen, S.C. Kremer, editors, *A Field Guide to Dynamical Recurrent Neural Networks*, pp. 237–244. IEEE Press, New York, NY, USA, 2001.

[Hon & Lee 91] H.W. Hon, K.F. Lee: Recent progress in robust vocabulary-independent speech recognition. In *DARPA Speech and Natural Language Processing Workshop*, pp. 258–263, Pacific Grove, Feb. 1991.

[Hori & Kubo+ 14] T. Hori, Y. Kubo, A. Nakamura: Real-time one-pass decoding with recurrent neural network language model for speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 6414–6418, Florence, Italy, May 2014.

[Hu & Auli+ 14] Y. Hu, M. Auli, Q. Gao, J. Gao: Minimum translation modeling with recurrent neural networks. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 20–29, Gothenburg, Sweden, April 2014.

[Huang & Zweig+ 14] Z. Huang, G. Zweig, B. Dumoulin: Cache based recurrent neural network language model inference for first pass speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 6404–6408, Florence, Italy, May 2014.

[Huck & Wuebker+ 13] M. Huck, J. Wuebker, F. Rietig, H. Ney: A phrase orientation model for hierarchical machine translation. In *ACL Workshop on Statistical Machine Translation (WMT)*, pp. 452–463, Sofia, Bulgaria, Aug. 2013.

[Igel & Hüsken 03] C. Igel, M. Hüsken: Empirical evaluation of the improved Rprop learning algorithms. *Neurocomputing*, Vol. 50, No. 1, pp. 105–123, Jan. 2003.

[Irie 15] K. Irie: Unpublished results, Chair of Computer Science 6, RWTH Aachen University, April 2015.

[Jelinek 76] F. Jelinek: Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, Vol. 64, No. 10, pp. 532–556, April 1976.

[Jelinek & Mercer 80] F. Jelinek, R.L. Mercer: Interpolated estimation of Markov source parameters from sparse data. In E.S. Gelsema, L.N. Kanal, editors, *Pattern Recognition in Practice*, pp. 381–397. North-Holland Publ. Company, Amsterdam, Netherlands, 1980.

[Kalchbrenner & Blunsom 13] N. Kalchbrenner, P. Blunsom: Recurrent continuous translation models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1700–1709, Seattle, WA, USA, October 2013.

[Kanthak & Schütz+ 00] S. Kanthak, K. Schütz, H. Ney: Using SIMD instructions for fast likelihood calculation in LVCSR. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 1531–1534, Istanbul, Turkey, June 2000.

[Katz 87] S.M. Katz: Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 35, No. 3, pp. 400–401, March 1987.

[Klakow & Peters 02] D. Klakow, J. Peters: Testing the correlation of word error rate and perplexity. *Speech Communication*, Vol. 38, No. 1, pp. 19–28, May 2002.

[Kneser 96] R. Kneser: Statistical language modeling using a variable context length. In *International Conference on Spoken Language Processing (ICSLP)*, pp. 494–497, Philadelphia, PA, USA, Oct. 1996.

[Kneser & Ney 91] R. Kneser, H. Ney: Forming word classes by statistical clustering for statistical language modelling. In *Proceedings of the International Conference on Quantitative Linguistics (QUALICO)*, pp. 221–226, Trier, Germany, Sept. 1991.

[Kneser & Ney 95] R. Kneser, H. Ney: Improved backing-off for m-gram language modelling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 181–184, Detroit, MI, USA, May 1995.

[Knill & Gales+ 13] K.M. Knill, M.J.F. Gales, S.P. Rath, P.C. Woodland, C. Zhang, S.X. Zhang: Investigation of multilingual deep neural networks for spoken term detection. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 138–143, Olomouc, Czech Republic, Dec. 2013.

[Koehn & Och+ 03] P. Koehn, F.J. Och, D. Marcu: Statistical phrase-based translation. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pp. 127–133, Edmonton, AB, Canada, May 2003.

[Kombrink & Mikolov+ 11] S. Kombrink, T. Mikolov, M. Karafiát, L. Burget: Recurrent neural network based language modeling in meeting recognition. In *Interspeech*, pp. 2877–2880, Florence, Italy, Aug. 2011.

[Kozielski & Doetsch+ 13] M. Kozielski, P. Doetsch, H. Ney: Improvements in RWTH's system for off-line handwriting recognition. In *International Conference on Document Analysis and Recognition (ICDAR)*, pp. 935–939, Washington, DC, USA, Aug. 2013.

[Kozielski & Rybach+ 13] M. Kozielski, D. Rybach, S. Hahn, R. Schlüter, H. Ney: Open vocabulary handwriting recognition using combined word-level and character-level language models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 8257 – 8261, Vancouver, BC, Canada, May 2013.

[Kramp 12] M. Kramp: Investigations on statistical language modeling. Master's thesis, RWTH Aachen University, Department of Computer Science, Aug. 2012.

[Le & Allauzen+ 10] H.S. Le, A. Allauzen, G. Wisniewski, F. cois Yvon: Training continuous space language models: some practical issues. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 778–788, MIT, MA, USA, Oct. 2010.

[Le & Allauzen+ 12a] H.S. Le, A. Allauzen, F. Yvon: Measuring the influence of long range dependencies with neural network language models. In *NAACL-HLT Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pp. 1–10, Montréal, QC, Canada, June 2012.

[Le & Allauzen+ 12b] H.S. Le, A. Allauzen, F. Yvon: Continuous space translation models with neural networks. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 39–48, Montréal, QC, Canada, June 2012.

[Le & Oparin+ 11] H.S. Le, I. Oparin, A. Allauzen, J.L. Gauvain, F. Yvon: Structured output layer neural network language model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 5524–5527, Prague, Czech Republic, May 2011.

[Le & Oparin+ 13] H.S. Le, I. Oparin, A. Allauzen, J.L. Gauvain, F. Yvon: Structured output layer neural network language models for speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, Vol. 21, No. 1, pp. 197–206, Jan. 2013.

[Le Cun & Kanter+ 91] Y. Le Cun, I. Kanter, S.A. Solla: Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, Vol. 66, No. 18, pp. 2396–2399, May 1991.

[Levenshtein 66] V.I. Levenshtein: Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics - Doklay*, Vol. 10, No. 10, pp. 707–710, 1966.

[Liu & Gales+ 13] X. Liu, M.J.F. Gales, P.C. Woodland: Use of contexts in language model interpolation and adaptation. *Computer Speech and Language*, Vol. 27, No. 1, pp. 301–321, Jan. 2013.

[Liu & Wang+ 14] X. Liu, Y. Wang, X. Chen, M.J.F. Gales, P.C. Woodland: Efficient lattice rescoring using recurrent neural network language models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 4941–4945, Florence, Italy, May 2014.

[Lowerre 76] B. Lowerre: *A Comparative Performance Analysis of Speech Understanding Systems*. Ph.D. thesis, Carnegie Mellon University, Department of Computer Science, Pittsburgh, PA, 1976.

[Mangu & Brill+ 99] L. Mangu, E. Brill, A. Stolcke: Finding consensus among words: Lattice-based word error minimization. In *European Conference on Speech Communication and Technology (Eurospeech)*, pp. 495–498, Budapest, Hungary, Sept. 1999.

[Mangu & Brill+ 00] L. Mangu, E. Brill, A. Stolcke: Finding consensus in speech recognition: word error minimization and other applications of confusion networks. *Computer Speech and Language*, Vol. 14, No. 4, pp. 373–400, Oct. 2000.

[Mangu & Soltau+ 13] L. Mangu, H. Soltau, K. Hong-Kwang, B. Kingsbury, G. Saon: Exploiting diversity for spoken term detection. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 8282–8286, Vancouver, BC, Canada, May 2013.

[Martens & Sutskever 11] J. Martens, I. Sutskever: Learning recurrent neural networks with Hessian-free optimization. In *International Conference on Machine Learning (ICML)*, pp. 1033–1040, Bellevue, Washington, USA, June 2011.

[Martin & Hamacher+ 99] S. Martin, C. Hamacher, J. Liermann, F. Wessel, H. Ney: Assessment of smoothing methods and complex stochastic language. In *European Conference on Speech Communication and Technology (Eurospeech)*, pp. 1939–1942, Budapest, Hungary, Sept. 1999.

[Martin & Ney 98] S. Martin, H. Ney: Algorithms for bigram and trigram word clustering. *Speech Communication*, Vol. 24, No. 1, pp. 19–37, April 1998.

[Mesnil & He+ 13] G. Mesnil, X. He, L. Deng, Y. Bengio: Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech*, pp. 3771–3775, Lyon, France, Aug. 2013.

[Mikolov & Deoras+ 11] T. Mikolov, A. Deoras, D. Povey, L. Burget, J. Černocký: Strategies for training large scale neural network language models. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 196–201, Hawaii, USA, Dec. 2011.

[Mikolov & Karafiát+ 10] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, S. Khudanpur: Recurrent neural network based language model. In *Interspeech*, pp. 1045–1048, Makuhari, Chiba, Japan, Sept. 2010.

[Mikolov & Kombrink+ 11a] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, S. Khudanpur: Extensions of recurrent neural network language model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 5528–5531, Prague, Czech Republic, May 2011.

[Mikolov & Kombrink+ 11b] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, J. Černocký: RNNLM - recurrent neural network language modeling toolkit. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Dec. 2011. Show and Tell session, Toolkit.

[Mikolov & Sutskever+ 13] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean: Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3111–3119, Lake Tahoe, NV, USA, Dec. 2013.

[Mikolov & Yih+ 13] T. Mikolov, W. Yih, G. Zweig: Linguistic regularities in continuous space word representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pp. 746–751, Atlanta, GA, USA, June 2013.

[Mikolov & Zweig 12] T. Mikolov, G. Zweig: Context dependent recurrent neural network language model. In *IEEE Spoken Language Technology Workshop (SLT)*, pp. 234–239, Miami, FL, USA, Dec. 2012.

[Mnih & Hinton 08] A. Mnih, G. Hinton: A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1081–1088, Vancouver, BC, Canada, Dec. 2008.

[Mnih & Teh 12] A. Mnih, Y. Teh: A fast and simple algorithm for training neural probabilistic language models. In *International Conference on Machine Learning (ICML)*, pp. 1751–1758, Edinburgh, Scotland, June 2012.

[Moore & Lewis 10] R. Moore, W. Lewis: Intelligent selection of language model training data. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 220–224, Uppsala, Sweden, July 2010.

[Morin & Bengio 05] F. Morin, Y. Bengio: Hierarchical probabilistic neural network language model. In *International Workshop on Artificial Intelligence and Statistics*, pp. 1–7, Barbados, Jan. 2005.

[Nakamura & Maruyama+ 90] M. Nakamura, K. Maruyama, T. Kawabata, K. Shikano: Neural network approach to word category prediction for English texts. In *International Conference on Computational Linguistics (COLING)*, pp. 213–218, Helsinki, Finland, Aug. 1990.

[Ney 84] H. Ney: The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Transactions on Speech and Audio Processing*, Vol. 32, No. 2, pp. 263–271, April 1984.

[Ney 90] H. Ney: Acoustic modeling of phoneme units for continuous speech recognition. In *European Signal Processing Conference (EUSIPCO)*, pp. 65–72, Barcelona, Spain, Sept. 1990.

[Ney & Essen+ 94] H. Ney, U. Essen, R. Kneser: On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, Vol. 8, No. 1, pp. 1–38, Jan. 1994.

[Ney & Häb-Umbach+ 92] H. Ney, R. Häb-Umbach, B.H. Tran, M. Oerder: Improvements in beam search for 10000-word continuous speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 1, pp. 9–12, San Francisco, CA, USA, March 1992.

[Ney & Martin+ 97] H. Ney, S. Martin, F. Wessel: Statistical language modeling using leaving-one-out. In S. Young, G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech Processing*, pp. 174–207. Kluwer Academic Publishers, Dordrecht, Netherlands, 1997.

[Ney & Mergel+ 87] H. Ney, D. Mergel, A. Noll, A. Paeseler: A data-driven organization of the dynamic programming beam search for continuous speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 833–836, Dallas, TX, USA, April 1987.

[Ney & Ortmanns 00] H. Ney, S. Ortmanns: Progress in dynamic programming search for LVCSR. *Proceedings of the IEEE*, Vol. 88, No. 8, pp. 1224–1240, Aug. 2000.

[Och 03] F.J. Och: Minimum error rate training in statistical machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 160–167, Sapporo, Japan, July 2003.

[Odell & Valtchev+ 94] J.J. Odell, V. Valtchev, P.C. Woodland, S.J. Young: A one-pass decoder design for large vocabulary recognition. In *ARPA Spoken Language Technology Workshop*, pp. 405–410, Plainsboro, NJ, March 1994.

[Oparin & Sundermeyer+ 12] I. Oparin, M. Sundermeyer, H. Ney, J.L. Gauvain: Performance analysis of neural networks in combination with n-gram language models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 5005–5008, Kyoto, Japan, March 2012.

[Ortmanns 98] S. Ortmanns: *Effiziente Suchverfahren zur Erkennung kontinuierlich gesprochener Sprache*. Ph.D. thesis, RWTH Aachen, Department of Computer Science, Aachen, Germany, Nov. 1998.

[Ortmanns & Ney 95] S. Ortmanns, H. Ney: An experimental study of the search space for 20000-word speech recognition. In *European Conference on Speech Communication and Technology (Eurospeech)*, Vol. 2, pp. 901–904, Madrid, Spain, Sept. 1995.

[Ortmanns & Ney$^+$ 96] S. Ortmanns, H. Ney, A. Eiden: Language-model look-ahead for large vocabulary speech recognition. In *International Conference on Spoken Language Processing (ICSLP)*, Vol. 4, pp. 2095–2098, Philadelphia, PA, Oct. 1996.

[Ortmanns & Ney$^+$ 97] S. Ortmanns, H. Ney, T. Firzlaff: Fast likelihood computation methods for continuous mixture densities in large vocabulary speech recognition. In *European Conference on Speech Communication and Technology (Eurospeech)*, Vol. 1, pp. 139–142, Rhodes, Greece, Sept. 1997.

[Papineni & Roukos$^+$ 02] K. Papineni, S. Roukos, T. Ward, W.J. Zhu: BLEU: a method for automatic evaluation of machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 311–318, Philadelphia, PA, USA, July 2002.

[Park & Liu$^+$ 10] J. Park, X. Liu, M.J.F. Gales, P.C. Woodland: Improved neural network based language modelling and adaptation. In *Interspeech*, pp. 1041–1044, Makuhari, Chiba, Japan, Sept. 2010.

[Pascanu & Gulcehre$^+$ 14] R. Pascanu, C. Gulcehre, K. Cho, Y. Bengio: How to construct deep recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, pp. 1–13, Banff, AB, Canada, April 2014.

[Paul 91] D.B. Paul: Algorithms for an optimal A^* search and linearizing the search in the stack decoder. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 1, pp. 693–696, Toronto, ON, Canada, May 1991.

[Pitz 05] M. Pitz: *Investigations on Linear Transformations for Speaker Adaptation and Normalization*. Ph.D. thesis, RWTH Aachen University, Department of Computer Science, Aachen, Germany, March 2005.

[Povey & Woodland 02] D. Povey, P.C. Woodland: Minimum phone error and I-smoothing for improved discriminative training. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 1, pp. 105–108, Orlando, FL, May 2002.

[Powell 64] M.J.D. Powell: An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, Vol. 7, No. 2, pp. 155–162, 1964.

[Rabiner 89] L.R. Rabiner: A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, Vol. 77, No. 2, pp. 257–286, Feb. 1989.

[Rabiner & Juang 86] L. Rabiner, B.H. Juang: An introduction to hidden Markov models. *IEEE ASSP Magazine*, Vol. 3, No. 1, pp. 4–16, 1986.

[Rabiner & Schafer 78] L.R. Rabiner, R.W. Schafer: *Digital Processing of Speech Signals*. Prentice Hall, Upper Saddle River, NJ, USA, 1978.

[Ramasubramansian & Paliwal 92] V. Ramasubramansian, K.K. Paliwal: Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *IEEE Transactions on Speech and Audio Processing*, Vol. 40, No. 3, pp. 518–528, March 1992.

[Rosenfeld 00] R. Rosenfeld: Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, Vol. 88, No. 8, pp. 1270–1278, Aug. 2000.

[Rumelhart & Hinton$^+$ 86] D.E. Rumelhart, G.E. Hinton, R.J. Williams: Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, The PDP Research Group, editors, *Parallel Distributed Processing*, Vol. 1, pp. 318–362. The MIT Press, Cambridge, MA, USA, London, UK, 1986.

[Rybach & Bisani$^+$ 11] D. Rybach, M. Bisani, P. Dreuw, C. Gollan, S. Hahn, G. Heigold, B. Hoffmeister, S. Kanthak, P. Lehnen, J. Lööf, D. Nolden, M. Pitz, M. Sundermeyer, Z. Tüske, S. Wiesler, A. Zolnay, R. Schlüter, H. Ney: RASR – the RWTH Aachen University open source speech recognition toolkit. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Dec. 2011. Show and Tell session, Toolkit.

[Sak & Senior$^+$ 14] H. Sak, A. Senior, F. Beaufays: Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Interspeech*, pp. 338–342, Singapore, 9 2014.

[Sakoe 79] H. Sakoe: Two-level DP-matching - a dynamic programming-based pattern matching algorithm for connected word recognition. *IEEE Transactions on Speech and Audio Processing*, Vol. 27, pp. 588–595, Dec. 1979.

[Sarle 99] W.S. Sarle: Ill-conditioning in neural networks. `ftp://ftp.sas.com/pub/neural/illcond/illcond.html`, Sept. 1999.

[Sarle 02] W.S. Sarle: Neural network frequently asked questions, part 2 of 7: Learning, periodic posting to the usenet newsgroup comp.ai.neural-nets. `ftp://ftp.sas.com/pub/neural/FAQ.html`, May 2002.

[Schlüter & Macherey$^+$ 01] R. Schlüter, W. Macherey, B. Müller, H. Ney: Comparison of discriminative training criteria and optimization methods for speech recognition. *Speech Communication*, Vol. 34, pp. 287–310, 2001.

[Schmidhuber 03] J. Schmidhuber: Long short-term memory: Tutorial on LSTM recurrent networks. `http://people.idsia.ch/~juergen/lstm/sld001.htm`, Jan. 2003.

[Schraudolph 12] N.N. Schraudolph: Centering neural network gradient factors. In G. Montavon, G.B. Orr, K. Müller, editors, *Neural Networks: Tricks of the Trade*, pp. 205–223. Springer, Berlin and Heidelberg, Germany, 2012.

[Schuster & Paliwal 97] M. Schuster, K.K. Paliwal: Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, Vol. 45, No. 11, pp. 2673–2681, Nov. 1997.

[Schwenk 07] H. Schwenk: Continuous space language models. *Computer Speech and Language*, Vol. 21, No. 3, pp. 492–518, July 2007.

[Schwenk 12] H. Schwenk: Continuous space translation models for phrase-based statistical machine translation. In *International Conference on Computational Linguistics (COLING)*, pp. 1071–1080, Mumbai, India, Dec. 2012.

[Schwenk & Rousseau+ 12] H. Schwenk, A. Rousseau, M. Attik: Large, pruned or continuous space language models on a GPU for statistical machine translation. In *NAACL-HLT Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pp. 11–19, Montréal, QC, Canada, June 2012.

[Shi & Zhang+ 13] Y. Shi, W.Q. Zhang, J. Liu, M.T. Johnson: RNN language model with word clustering and class-based output layer. EURASIP Journal on Audio, Speech, and Music Processing, July 2013.

[Shi & Zhang+ 14a] Y. Shi, W.Q. Zhang, M. Cai, J. Liu: Empirically combining unnormalized NNLM and back-off n-gram for fast n-best rescoring in speech recognition. EURASIP Journal on Audio, Speech, and Music Processing, April 2014.

[Shi & Zhang+ 14b] Y. Shi, W.Q. Zhang, M. Cai, J. Liu: Variance regularization of RNNLM for speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 4893–4897, Florence, Italy, May 2014.

[Si & Zhang+ 13] Y. Si, Q. Zhang, T. Li, J. Pan, Y. Yan: Prefix tree based n-best list rescoring for recurrent neural network language model used in speech recognition system. In *Interspeech*, pp. 3419–3423, Lyon, France, Aug. 2013.

[Siivola & Hirsimäki+ 07] V. Siivola, T. Hirsimäki, S. Virpioja: On growing and pruning Kneser-Ney smoothed n-gram models. *IEEE Transactions on Audio, Speech, and Language Processing*, Vol. 15, No. 5, pp. 1617–1624, July 2007.

[Snover & Dorr+ 06] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, J. Makhoul: A study of translation edit rate with targeted human annotation. In *Conference of the Association for Machine Translation in the Americas (AMTA)*, pp. 223–231, Cambridge, MA, USA, Aug. 2006.

[Steinbiss & Ney+ 93] V. Steinbiss, H. Ney, R. Häb-Umbach, B. Tran, U. Essen, R. Kneser, M. Oerder, H. Meier, X. Aubert, C. Dugast, D. Geller: The Philips research system for large-vocabulary continuous-speech recognition. In *European Conference on Speech Communication and Technology (Eurospeech)*, pp. 2125–2128, Berlin, Germany, Sept. 1993.

[Stolcke 98] A. Stolcke: Entropy-based pruning of backoff language models. In *DARPA Broadcast News Transcription and Understanding Workshop*, pp. 1–5, Lansdowne, VA, USA, Feb. 1998.

[Stolcke 02] A. Stolcke: SRILM – an extensible language modeling toolkit. In *International Conference on Spoken Language Processing (ICSLP)*, pp. 901–904, Denver, CO, USA, Sept. 2002.

[Stolcke 10] A. Stolcke: Srilm user mailing list. `http://www.speech.sri.com/pipermail/srilm-user/2010q3.txt`, Sept. 2010.

[Stolcke & Konig+ 97] A. Stolcke, Y. Konig, M. Weintraub: Explicit word error minimization in n-best list rescoring. In *European Conference on Speech Communication and Technology (Eurospeech)*, pp. 163–166, Rhodes, Greece, Sept. 1997.

[Stolcke & Zheng+ 11] A. Stolcke, J. Zheng, W. Wang, V. Abrash: SRILM at sixteen: Update and outlook. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Dec. 2011. Show and Tell session, Toolkit.

[Sundermeyer & Alkhouli+ 14] M. Sundermeyer, T. Alkhouli, J. Wuebker, H. Ney: Translation modeling with bidirectional recurrent neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14–25, Doha, Qatar, Oct. 2014.

[Sundermeyer & Ney+ 15] M. Sundermeyer, H. Ney, R. Schlüter: From feedforward to recurrent LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Vol. 23, No. 3, pp. 13–25, March 2015.

[Sundermeyer & Nußbaum-Thom+ 11] M. Sundermeyer, M. Nußbaum-Thom, S. Wiesler, C. Plahl, A.E.D. Mousa, S. Hahn, D. Nolden, R. Schlüter, H. Ney: The RWTH 2010 Quaero ASR evaluation system for English, French, and German. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 2212–2215, Prague, Czech Republic, May 2011.

[Sundermeyer & Oparin+ 13] M. Sundermeyer, I. Oparin, J.L. Gauvain, B. Freiberg, R. Schluter, H. Ney: Comparison of feedforward and recurrent neural network language models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 8430–8434, Vancouver, BC, Canada, May 2013.

[Sundermeyer & Schlüter+ 11] M. Sundermeyer, R. Schlüter, H. Ney: On the estimation of discount parameters for language model smoothing. In *Interspeech*, pp. 1433–1436, Florence, Italy, Aug. 2011.

[Sundermeyer & Schlüter+ 12] M. Sundermeyer, R. Schlüter, H. Ney: LSTM neural networks for language modeling. In *Interspeech*, pp. 194–197, Portland, OR, USA, Sept. 2012.

[Sundermeyer & Schlüter+ 14] M. Sundermeyer, R. Schlüter, H. Ney: rwthlm – the RWTH Aachen University neural network language modeling toolkit. In *Interspeech*, pp. 2093–2097, Singapore, Sept. 2014.

[Sundermeyer & Tüske+ 14] M. Sundermeyer, Z. Tüske, R. Schlüter, H. Ney: Lattice decoding and rescoring with long-span neural network language models. In *Interspeech*, pp. 661–665, Singapore, Sept. 2014.

[Sutskever & Martens+ 11] I. Sutskever, J. Martens, G. Hinton: Generating text with recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pp. 1017–1024, Bellevue, Washington, USA, June 2011.

[Sutskever & Vinyals+ 14] I. Sutskever, O. Vinyals, Q.V.V. Le: Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3104–3112, Montréal, QC, Canada, Dec. 2014.

[Tüske & Golik+ 14] Z. Tüske, P. Golik, R. Schlüter, H. Ney: Acoustic modeling with deep neural networks using raw time signal for LVCSR. In *Interspeech*, pp. 890–894, Singapore, Sept. 2014.

[Tüske & Nolden+ 14] Z. Tüske, D. Nolden, R. Schlüter, H. Ney: Multilingual MRASTA features for low-resource keyword search and speech recognition systems. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 7904–7908, Florence, Italy, May 2014.

[Tüske & Schlüter+ 13] Z. Tüske, R. Schlüter, H. Ney: Multilingual hierarchical MRASTA features for ASR. In *Interspeech*, pp. 2222–2226, Lyon, France, Aug. 2013.

[Vaswani & Zhao+ 13] A. Vaswani, Y. Zhao, V. Fossum, D. Chiang: Decoding with large-scale neural language models improves translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1387–1392, Seattle, WA, Oct. 2013.

[Vilar & Huck+ 10] D. Vilar, D.S.M. Huck, H. Ney: Jane: Open source hierarchical translation, extended with reordering and lexicon models. In *ACL Joint Fifth Workshop on Statistical Machine Translation and Metrics MATR*, pp. 262–270, Uppsala, Sweden, July 2010.

[Vintsyuk 71] T.K. Vintsyuk: Elementwise recognition of continuous speech composed of words from a specified dictionary. *Kibernetika*, Vol. 7, pp. 133–143, March 1971.

[Viterbi 67] A. Viterbi: Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, Vol. 13, pp. 260–269, 1967.

[Weng & Stolcke+ 98] F. Weng, A. Stolcke, A. Sankar: Efficient lattice representation and generation. In *International Conference on Spoken Language Processing (ICSLP)*, pp. 2531–2534, Sydney, NSW, Australia, Nov. 1998.

[Werbos 88] P.J. Werbos: Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, Vol. 1, No. 4, pp. 339–356, 1988.

[Werbos 90] P.J. Werbos: Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, Vol. 78, No. 10, pp. 1550–1560, Oct. 1990.

[Wessel & Schlüter$^+$ 01] F. Wessel, R. Schlüter, K. Macherey, H. Ney: Confidence measures for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, Vol. 9, No. 3, pp. 288–298, March 2001.

[Williams & Peng 90] R.J. Williams, J. Peng: An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, Vol. 2, No. 4, pp. 490–501, 1990.

[Williams & Zipser 95] R.J. Williams, D. Zipser: Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvain, D.E. Rumelhart, editors, *Backpropagation: Theory, Architectures, and Applications*, pp. 433–486. Psychology Press, Hillsdale, NJ, USA, 1995.

[Wuebker & Huck$^+$ 12] J. Wuebker, M. Huck, S. Peitz, M. Nuhn, M. Freitag, J.T. Peter, S. Mansour, H. Ney: Jane 2: Open source phrase-based and hierarchical statistical machine translation. In *International Conference on Computational Linguistics (COLING)*, pp. 483–491, Mumbai, India, Dec. 2012.

[Wuebker & Mauser$^+$ 10] J. Wuebker, A. Mauser, H. Ney: Training phrase translation models with leaving-one-out. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 475–484, Uppsala, Sweden, July 2010.

[Wuebker & Peitz$^+$ 13] J. Wuebker, S. Peitz, F. Rietig, H. Ney: Improving statistical machine translation with word class models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1377–1381, Seattle, WA, USA, Oct. 2013.

[Xu & Rudnicky 00] W. Xu, A. Rudnicky: Can artificial neural networks learn language models? In *International Conference on Spoken Language Processing (ICSLP)*, pp. 202–205, Beijing, China, Oct. 2000.

[Young 92] S.J. Young: The general use of tying in phoneme based HMM recognizers. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 1, pp. 569–572, San Francisco, CA, USA, March 1992.

[Zweig & Makarychev 13] G. Zweig, K. Makarychev: Speed regularization and optimality in word classing. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 8237–2341, Vancouver, BC, Canada, May 2013.