Doctoral Dissertation

Context Enhancement of Recurrent Neural Network Language Models for Automatic Speech Recognition

Michael Alexander Hentschel

March 15, 2019

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Michael Alexander Hentschel

Thesis Committee:

Professor Yuji Matsumoto	(Supervisor)
Professor Satoshi Nakamura	(Co-supervisor)
Professor Hiroshi Sawada	(Co-supervisor)
Associate Professor Tomoharu Iwata	(Co-supervisor)

Context Enhancement of Recurrent Neural Network Language Models for Automatic Speech Recognition*

Michael Alexander Hentschel

Abstract

Language models are a key component of automatic speech recognition systems. In recent years, language models based on neural networks and in particular recurrent neural networks have shown significant performance improvements on traditional count-based language models. Language models calculate the probability of the next word from the word history. In the task of automatic speech recognition, various context information beside words is available. This context can be from different sources like the acoustic signal, the recent word history, or a global text topic. However, neural network language models do not specifically exploit this kind of context information. In the context of this thesis, different kinds of context information that is available to the language model in automatic speech recognition is identified. To make this information accessible to the language model, different network architectures to exploit this context are presented. Subsequently, different combinations of context information and network architectures are investigated on their effectiveness for state of the art recurrent neural network language models. The presented methods achieved improvements on state of the art recurrent neural network language models in terms of model perplexity and word error rate in rescoring.

Keywords:

Language model, Automatic speech recognition, Domain adaptation, Prosody, Context information

*Doctoral Dissertation, Graduate School of Information Science, Nara Institute of Science and Technology, March 15, 2019.

音声認識用ニューラルネットワーク言語モデルのコンテキスト強化*

Michael Alexander Hentschel

内容梗概

言語モデルは、自動音声認識システムの重要な要素である。近年、ニューラルネットワーク、特にリカレントニューラルネットワークを用いている言語モデルは、従来のカウントベース言語モデルの性能を大幅に改善した。言語モデルは、単語履歴から次の単語の確率を計算するモテルである。自動音声認識のタスクでは、単語以外に様々なコンテキスト情報が利用可能である。このコンテキストは、音響信号、最近の単語履歴、またはグローバルテキストトピックの様な異なる情報源に基づくことができる。しかし、ニューラルネットワーク言語モデルは、この種のコンテキスト情報を特に利用していない。本論文では、自動音声認識において言語モデルが利用可能な種類のコンテキスト情報を提案する。これらの情報を言語モデルで利用可能にするため、コンテクスト情報を利用するネットワークアーキテクチャを提案する。続いて、最新のリカレントニューラルネットワーク言語モデルへの有効性にに対して、コンテキスト情報とネットワークアーキテクチャとの様々な組み合わせを調査する。提示された手法は、モデルのパープレキシティとリスコアリングでのワードエラーレートで最新のリカレントニューラルネットワーク言語モデルの改善を達成した。

キーワード

言語モデル, 音声認識, ドメイン適応, 韻律, コンテキスト情報

*奈良先端科学技術大学院大学情報科学研究科 博士論文, 情報処理学専攻, 2019 年 3 月 15 日.

Contents

List of Figures

List of Tables

1. Introduction

Speech is the natural form of communication for humans. We learn a language as a child without any prior knowledge only by being exposed to it. Because it is such a natural means of communication for us, researchers and engineers have already been trying for a long time to develop technologies to enable machines recognise speech and understand language. As natural and obvious this task might seem to outstanding persons, the problem has shown to be anything but trivial.

Currently, the most successful models in speech recognition use as a statistical approach. From a statistical point of view, the problem of speech recognition can be described as finding the most likely sequence of words corresponding to an acoustic signal that we observe. As introduced in more detail in the next section, this problem can be divided into two smaller sub-problems. The first one is to model a likelihood of the feature sequence and the second one is to calculate a probability for the word sequence. The first problem is handled by the acoustic model and the second problem is handled by the language model. This makes the language model a very important part of an automatic speech recognition system. The language model is used for finding the best word sequence in a step called decoding and subsequently a stronger language model can be used in a second-pass decoding, the so-called rescoring.

1.1 Motivation

Within the scope of this thesis, language models used in automatic speech recognition will be investigated. A language model's task is to predict the probability for a sequence of words. In order to improve this prediction, different approaches have been proposed over the last decades. Early language models used linguistic knowledge and incorporated rules. These models were eventually superseded by statistical models. The parameters for these models are estimated from the relative frequencies of each term on a large data set. By using a sufficiently large data set, one might assume that reliable statistics for a language model could be estimated. However, using this discrete estimation technique has different short comings. Estimating the frequency of a single term does not provide a good

probabilistic model for a sequence of several terms. However, when estimating the relative frequencies of sequences, many of the possible sequences do not appear in the data. From this problem, many techniques on how to find a good approximation for these unseen events have emerged. Some of the basic concepts and state of the art methods will be explained in this thesis.

After these discrete models, more recently a different approach using artificial neural networks has been used for language modelling. Neural network language models first used a similar method as count-based models, that is, using a truncated history of words. However, recurrent neural networks which were designed for sequence modelling problems showed a better performance when applied to speech recognition and other tasks. Models based on neural networks have the ability to learn a common continuous-valued space that all words get projected into. This common space allows to model semantics of words. Language models based on neural networks have so far not completely replaced count-based models. Both language models showed to cover complementary information and in current state of the art speech recognition systems an interpolation between both models is used.

However, despite being the strongest language models that are currently available, neural network language models lack the ability to explicitly account for context. How context can have an effect on the word probabilities can be illustrated easily by an example. Let us consider a newspaper as our data set. From the whole newspaper, we can obtain a global estimate of word probabilities. However, a newspaper is segmented into different sections, like politics, finance, sports and so on. If we investigate the distribution of words in each of these sections, we find a different distribution depending on the section which we are in. Language models that can account for such information can give a better estimate for likely and unlikely word sequences. This will help a speech recogniser to improve the recognition.

Context information from a text category is not the only context that can be accessed in a speech recognition system. In a speech recognition system acoustic information is available in addition to the textual information. Among this whole information, some context can have a positive influence on the language model and improve the word prediction. By enabling neural network language models

to use this information, an improvement on current state of the art methods can be achieved in speech recognition.

1.2 Contributions

The focus of this thesis is on exploiting context information surrounding the current word that can be extracted in an automatic speech recognition system for language modelling. This task is two fold. First, one needs to identify useful context information which can improve word prediction and improve speech recognition results. Second, investigating network architectures that neural network language models can make effective use of this context information.

Depending on the task, there is different context which can be exploited for language modelling. The context can be visual information from images or videos. It can be information from a different language in machine translation. In a search application, we might have access to a user's location. In this thesis, language models for automatic speech recognition are investigated. Among others, we can imagine that we have access to the following information. An audio signal, that is the signal we try to recognise. Information about the speaker like the gender or a speaker ID. In case we try to recognise a longer text, we have access to previously recognised sentences. This might help us to find a local or global topic. Still, we need a way to reliably extract this information in an automated way. Creating annotations with this information in the data is time-consuming and requires expert knowledge. Therefore, the information should be extracted as a feature in an unsupervised manner. Introducing methods to achieve this objective is one part of this thesis.

Next to identifying the available information, another main part of this thesis is how to exploit the information in neural network language models. The general network architecture used in this thesis will be recurrent neural network language models. These models showed state of the art results in speech recognition. Different modifications of neural network architectures that allow to include context information as an auxiliary feature will be described. These architectures have partially been proposed, however, here a different way to use these architectures will be investigated. Other architectures have been proposed in the context of a different task that is unrelated to language modelling and in this thesis the ap-

plication to language modelling will be investigated. This thesis also investigates an integrated approach to extract context information with a neural network and adapting a language model, combining several of the investigated techniques.

1.3 Thesis Outline

The remainder of this thesis is organised as follows. Chapter 2 gives an introduction to the problem of automatic speech recognition in general and introduces the main components of a statistical automatic speech recognition system one-by-one. After this overview, the fundamentals of count-based language models and standard smoothing techniques as well as extensions will be introduced. Next to count-based models, the fundamentals of neural network language models will be explained. This includes different neural network language models architectures using feed-forward and recurrent neural networks as well as an outline of the training algorithm. After these basics, Chapter 3 introduces different kinds of context information that can be used in the context of automatic speech recognition for language modelling. In particular, prosodic features, cache models and topic features will be presented. Chapter 3 describes each kind of information and shows how features can be calculated from this information. How these features can be exploited in neural network language models is described in the subsequent Chapter 4, where different network architectures are introduced. Chapter 5 shows experimental results for different context features with different model architectures on common data sets. The experiments are evaluated in terms of perplexity and word error rate. Finally Chapter 6 provides concluding remarks and gives an outlook on future work that could not be covered in the scope of this thesis.

1.4 Publications

Parts of this thesis have been published at international conferences or domestic meetings. The investigation of prosodic features for neural network language model adaptation was published in [60]. A study of different cache extension was presented at a domestic meeting [59]. Different methods for feature-based domain adaptation of recurrent neural network language models were presented at an international conference [55, 58]. An extended study of hidden layer factorisation

for feature-based domain adaptation was published as a journal paper [57]. The unified approach for context extraction and adaptation of language models was presented at an international conference [56].

2. Statistical Language Models in Automatic Speech Recognition

This section provides a short overview of an Automatic Speech Recognition (ASR) system. First, it describes the general problem description we addressing in ASR and subsequently describe a general structure of an ASR system. Following the general description of an ASR system's components, we briefly overview the major techniques used in statistical Language Models (LM) in ASR. These LMs are M-Gram and Neural Network (NN) LMs (NNLM). The main focus of this thesis is the practical application of NNLMs in the field of ASR.

2.1 Automatic Speech Recognition

Speech is our most efficient and natural means of communication. We are able to convey complex concepts by just using speech. For machines, the task of automatically recognising and evaluating speech is commonly known as Automatic Speech Recognition (ASR). One major application of ASR is Human-Machine Interaction (HMI), where we would like to enable machines to interact with humans using their most preferred means of communication. However, building a machine that successfully performs ASR has proven to be a non-trivial task. Today's systems make use of a statistical approach as described in the following.

The general layout of an ASR system is depicted in Figure 1. The input to the system is the digitally converted representation $s[t]$ of the continuous-valued analog speech signal. This signal is recorded by one or multiple microphones and then pre-processed ($\tilde{s}[t]$). Subsequently, a sequence of feature vectors $\overline{\boldsymbol{o}} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_K]$ is extracted from the pre-processed signal. These features are used for training of the system and for testing. During system training we try to find the optimal model parameters λ and testing means that we the system on a new input it has not seen before. For the training, the features as well as a transcription of the training data are available to build the lexicon, the language model, and the acoustic model. During testing, these models are used to estimate the transcription $\hat{\overline{w}} = \hat{w}_1, \hat{w}_2, \ldots, \hat{w}_L$ for a newly extracted feature sequence.

For statistical ASR, we view the problem of speech recognition as a pattern recognition problem. Given an observed sequence of acoustic features $\overline{\boldsymbol{o}}$, an ASR

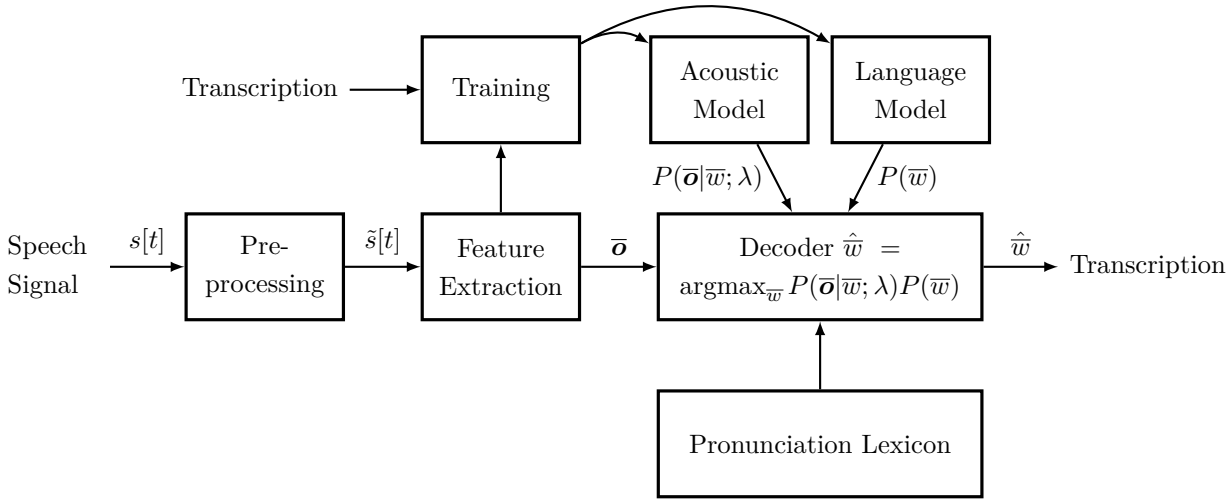


Figure 1: The general structure of an automatic speech recognition system (after [79] and [68]).

system should find the most likely word sequence $\hat{\overline{w}} = \hat{w}_1, \hat{w}_2, \ldots, \hat{w}_L$ that corresponds to the features

$$\hat{\overline{w}} = \underset{\overline{w}}{\text{argmax}}\, P(\overline{w}|\overline{\boldsymbol{o}};\lambda), \tag{1}$$

where λ denotes the statistical model of the recogniser. However, this problem is in practice intractable, so we apply Bayes' rule (e.g. [104]) to rewrite the above problem as

$$\hat{\overline{w}} = \underset{\overline{w}}{\text{argmax}}\, \frac{P(\overline{\boldsymbol{o}}|\overline{w};\lambda)P(\overline{w})}{P(\overline{\boldsymbol{o}})}. \tag{2}$$

$P(\overline{\boldsymbol{o}})$ can be neglected in this optimisation problem, because it is independent of the variable $\overline{w}$ which has to be optimised. The two remaining parts are $P(\overline{w})$ the a priori probability of a particular word sequence and $P(\overline{\boldsymbol{o}}|\overline{w};\lambda)$ the probability of observing a particular feature vector sequence given a corresponding word sequence. This allows (2) to be simplified to

$$\hat{\overline{w}} = \underset{\overline{w}}{\text{argmax}}\, P(\overline{\boldsymbol{o}}|\overline{w};\lambda)P(\overline{w}). \tag{3}$$

The above equation allows to decompose the original problem (1) into two sub-problems. For each of these sub-problems, we can estimate a statistical model from the training data. The a priori probability of a word sequence $P(\overline{w})$ is modelled by the Language Model (LM). The probability of observing a feature vector sequence given a word sequence $P(\overline{\boldsymbol{o}}|\overline{w};\lambda)$ is modelled by the Acoustic Model (AM).

Regarding the mathematical notation,the following conventions will be used in the remainder of this thesis. The matrix of feature vectors is composed as follows

$$\overline{\boldsymbol{o}} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_K], \tag{4}$$

where k is the time index in the observed sequence of K feature vectors. Vectors and scalars in the matrix are indexed with brackets in the following way

$$\overline{\boldsymbol{o}}[1] = \boldsymbol{o}_1, \tag{5}$$

$$\overline{\boldsymbol{o}}[1:4] = [\boldsymbol{o}_1, \boldsymbol{o}_2, \boldsymbol{o}_3, \boldsymbol{o}_4], \tag{6}$$

$$\overline{\boldsymbol{o}}[1,1] = \boldsymbol{o}_1(1) = o_1. \tag{7}$$

A sequence of values is denoted by the respective indices in the subscript

$$\boldsymbol{o}_{1\ldots4} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \boldsymbol{o}_3, \boldsymbol{o}_4]. \tag{8}$$

A word sequence $\overline{w}$ is usually composed of L words where without a loss of generality each word w_l is assumed to be represented by an integer ID

$$\overline{w} = w_1, w_2, \ldots, w_L. \tag{9}$$

2.1.1 Pre-processing and Feature Extraction

Natural signals usually exhibit a lot of redundancy. This redundancy leads to a high-dimensional and highly correlated signal which is not desirable for the classification task. Therefore, the main purpose of using features rather than the raw audio signal is to remove this redundancy and the dimensionality of the signal. It is well-known that in high dimensional spaces classification fails ("*the curse of dimensionality*", [9]). The reduction in dimensionality from, for example, 512 to

12 greatly reduces the computational complexity, which, for example, improves processing speed or reduces power consumption. However, this requires features that can capture the essential discriminative information that is necessary for classification and which have little correlation and redundancy.

As common features for ASR, Mel Frequency cepstral Coefficients (MFCC) [25] or Perceptual Linear Prediction (PLP) [61] are used. These features are motivated by human perception. The computation scheme for MFCCs is depicted in Figure 2. Features are commonly computed over a window length of 20-40ms with a 10ms frame shift. This window is used because the speech signal is assumed to be stationary over this short duration. The raw features usually exhibit a high speaker and gender dependency which might be undesired in the system. To remove these effects various normalising techniques exist.

A simple and common normalisation is Cepstral Mean and Variance Normalisation (CMVN) [139], which normalises the mean and variance of the features to have zero mean and unit variance. Each element of the feature vector is normalised as follows

$$\bar{\boldsymbol{o}} = (\boldsymbol{o} - \boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}, \tag{10}$$

$$\boldsymbol{\mu} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{o}_k, \tag{11}$$

$$\boldsymbol{\Sigma} = \sqrt{\frac{1}{K}\sum_{k=1}^{K}(\boldsymbol{o}_k - \boldsymbol{\mu})(\boldsymbol{o}_k - \boldsymbol{\mu})^{\mathsf{T}}}. \tag{12}$$

The input signal might include other undesired signals, such as noise and reverberation. This requires further noise and reverberation removal. In an actual system implementation, such speech enhancement methods are used in the preprocessing step. These enhancements include dereverberation [99] and denoising filters [19]. In case where multiple microphones are used for recording the signals, beamforming is a used to attenuate undesired sources and to enhance the desired signal [10].

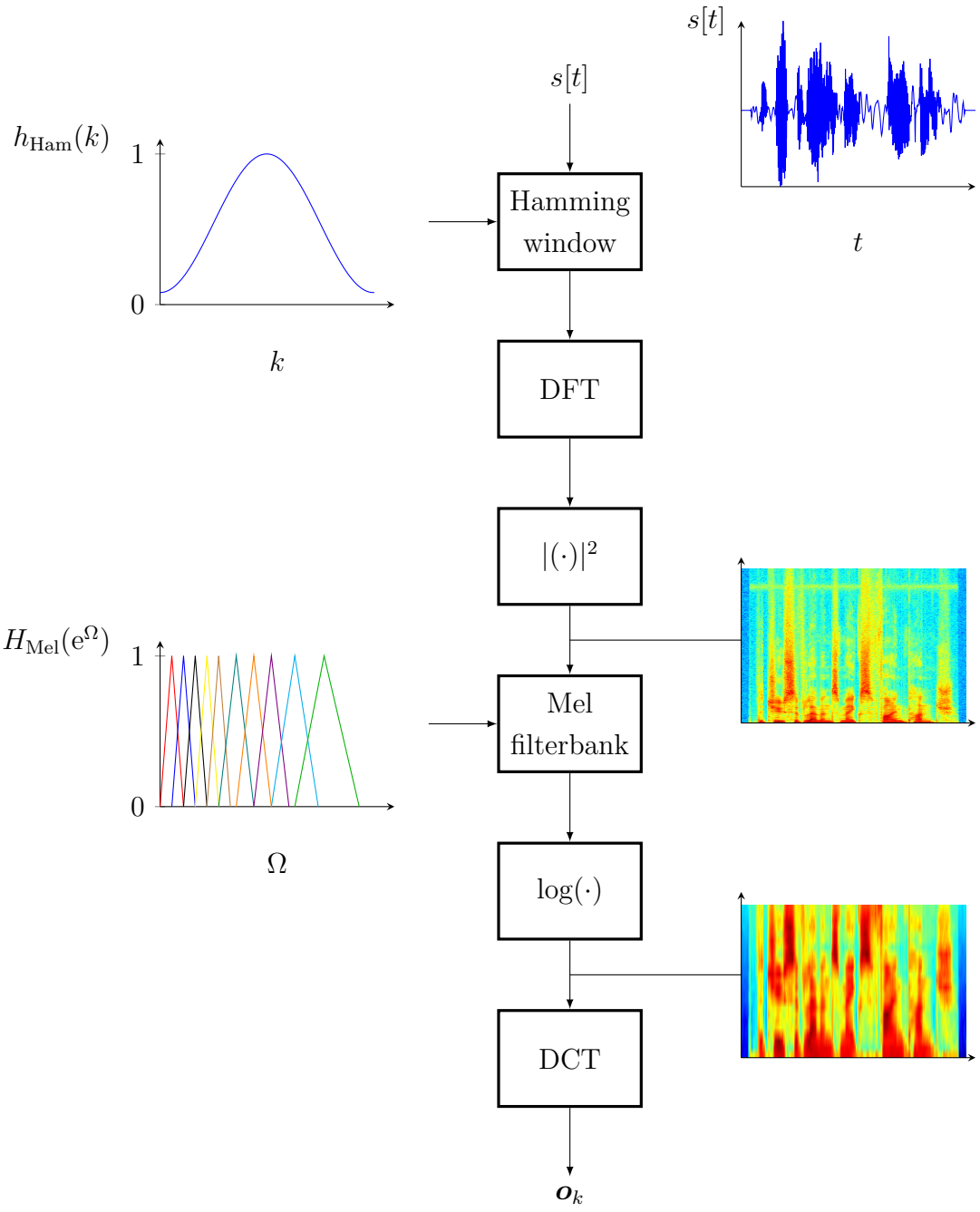


Figure 2: The computation scheme of extracting MFCC features (after [65]).

2.1.2 Acoustic Model

In statistical ASR, the AM is a statistical model for the likelihood of observing a sequence of features $\overline{\boldsymbol{o}}$ given a sequence of words $\overline{w}$ $P(\overline{\boldsymbol{o}}|\overline{w};\lambda)$. Modelling this problem on the word level has several disadvantages. One problem is that many words only appear a few times in the training data. However, it is hard to estimate reliable models for infrequent words. In addition, the system cannot recognise words that were never seen in the training data. Since words are composed of a long sequence of feature vectors, the classification problem will also be high-dimensional. Because these issues make speech recognition on the word level difficult, the problem is broken down into a smaller one.

Each word consists of sub-word units. The unit used in the AM of an ASR system is a phoneme. A phoneme is the smallest unit which carries a change in meaning. Using this sub-word unit has several advantages compared with words. First, the number of phonemes is usually by far less than the number words. Second, these units are much more frequent in the training data than individual words. Usually, the pronunciation of a phoneme is context dependent. Each phoneme has a right and a left context. This sequence consisting of three phonemes is called a triphone.

In ASR, the phonemes are modelled by Hidden Markov Models (HMM) [73, 108]. HMMs allow flexibility for different feature sequence lengths. This flexibility is desirable because the same phonene can be longer or shorter based on the way it is pronounced. This simply relates to the fact that different speakers talk at different speeds. An HMM consists of two statistical processes, where one is observable and the other one is hidden. An HMM consists of discrete states $s \in \mathcal{S}$. The acoustic observations are emitted at each state when the HMM is traversed along a sequence of states $\overline{s}$. In the AM, the underlying state is unknown and only the output, that means, feature vectors $\overline{\boldsymbol{o}}$ are observable. Using HMMs as statistical model for the AM, the probability of feature vectors can be calculated as

$$P(\overline{\boldsymbol{o}}|\overline{w};\lambda) = \sum_{\overline{s}} P(\overline{\boldsymbol{o}}, \overline{s}|\overline{w};\lambda). \tag{13}$$

The summation only has to be carried over all states $\overline{s}$ that are contained in the

word sequence. Using Bayes' rule, the above equation can be rewritten

$$P(\overline{\boldsymbol{o}}|\overline{w};\lambda) = \sum_{\overline{s}} \prod_{k=1}^{K} P(\boldsymbol{o}_k|\overline{\boldsymbol{o}}[1:k-1], \overline{s}, \overline{w};\lambda) \cdot P(s_k|\overline{\boldsymbol{o}}[1:k-1], \overline{s}[1:k-1], \overline{w};\lambda). \tag{14}$$

This can be simplified with a first-order Markov assumption, that means, the current state and feature vector only depend on the previous state

$$P(\overline{\boldsymbol{o}}|\overline{w};\lambda) = \sum_{\overline{s}} \prod_{k=1}^{K} P(\boldsymbol{o}_k|s_k, \overline{w};\lambda) \cdot P(s_k|s_{k-1}, \overline{w};\lambda). \tag{15}$$

$P(\boldsymbol{o}_k|s_k, \overline{w};\lambda)$ is the emission probability of a feature vector $\boldsymbol{o}_k$ in state s_k. $P(s_k|s_{k-1}, \overline{w};\lambda)$ is the transition probability from state s_{k-1} to the next state s_k. For the emission probability $P(\boldsymbol{o}_k|s_k, \overline{w};\lambda)$, Gaussian Mixture Models (GMM) with $\boldsymbol{\Theta} = \{c_m, \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m\}$ are used. Each mixture has a weight c_m, a mean vector $\boldsymbol{\mu}_m$, and a covariance matrix $\boldsymbol{\Sigma}_m$. The probability density to observe a feature vector $\boldsymbol{o}_k$ by GMM m is calculated as a sum over all mixture components

$$f_X(\boldsymbol{o}_k|\boldsymbol{\Theta}) = \sum_{m=1}^{M} c_m \mathcal{N}_m(\boldsymbol{o}_k|\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m). \tag{16}$$

HMM and GMM parameters can be estimated jointly using Baum-Welch algorithm [6].

An example of an HMM with three states is shown in Figure 3. The HMM is always entered in the first state, because the initial transition equals zero for any other state but state s_1. All states have an outgoing transition except the last one s_3 which means that this is the final state of the HMM (marked by the double lines). The model in Figure 3, is a linear HMM which has only transitions to the next state or a self-loop. This type of HMMs is commonly used in speech recognition. The HMMs for multiple phonemes are concatenated to form a model for a triphone. The number of all possible triphones is very high, so that we might not observe all triphones in the training data. In such case, triphones can be decomposed into monophones and the parameters among different triphones can be shared and clustered.

AMs using GMM-HMM have been used for a long time and they are still very robust for example in low-resource applications. More recently, hybrid models

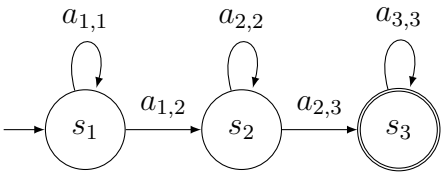


Figure 3: An example of a tri-state HMM.

which use a deep neural network (DNN) in the AM showed better performance than GMM-HMM models. In a hybrid system, the DNN is used to calculate the posterior probabilities of the HMM states from a window of acoustic features.

2.1.3 Language Model

In natural language, words rarely occur as isolated events. They are usually concatenated in a sentence and the formation of sentences is restricted by the grammar of a language. Language Models (LM) are used to encode the syntax and semantic of a language.

An LM calculates the overall probability for a sequence of words $P(\overline{w})$ of length L. The probability for a word in the sequence is a conditional probability depending on the previous words

$$P(\overline{w}) = P(w_1) \cdot P(w_2|w_1) \cdot \ldots \cdot P(w_L|\overline{w}[1, \ldots, L-1]) \tag{17}$$

$$= \prod_{l=1}^{L} P(w_l|\overline{w}[1, \ldots, l-1]). \tag{18}$$

When calculating the joint probability for every word in a sentence, the context length increases from word to word. Due to the high amount of different possible sentences it is impossible to train an LM which accounts for the whole word history in a sentence. In practice, the Markov assumption is used and the history is shortened accounting for the last $M-1$ words. These M-Grams calculate the probability of a word sequence as

$$P(\overline{w}) = \prod_{l=1}^{L} P(w_l|\overline{w}[l-M+1, \ldots, l-1]). \tag{19}$$

Here, we will briefly describe the case of $M = 3$, that is a trigram LM. A more general overview of count based LMs will be given in Chapter 2.2. The parameters

of an M-Gram are estimated from the relative frequencies of the training data. For example, the probability for a trigram LM can be estimated by

$$\hat{P}(w_l|w_{l-2}, w_{l-1}) = \frac{C(w_l, w_{l-1}, w_{l-2})}{C(w_{l-1}, w_{l-2})}, \tag{20}$$

where $C(w_l, w_{l-1}, w_{l-2})$ is the word count for the word sequence w_l, w_{l-1}, w_{l-2}. An M-Gram is thus a maximum likelihood estimate of the training data.

However, it is very unlikely that every possible combination of words exists in the training data. To make the M-Gram robust against unseen events, smoothing strategies for obtaining an estimate for these events exist. Smoothing techniques shift probability mass of frequently occurring events to unseen events. They are distinguished between interpolation and back-off. In interpolation, the probability from a lower-order M-Gram is linearly interpolated with the full M-Gram

$$\hat{P}(w_l|w_{l-2}, w_{l-1}) = \alpha(w_l|w_{l-2}, w_{l-1}) + \gamma(w_{l-1})\beta(w_l|w_{l-1}), \tag{21}$$

where $\gamma(w_{l-1})$ is a normalisation term.

Back-off, in comparison, estimates the probability of unseen events by backing-off to lower-order M-Grams

$$\hat{P}(w_l|w_{l-2}, w_{l-1}) = \begin{cases} \alpha(w_l|w_{l-2}, w_{l-1}), & \text{if } C(w_l, w_{l-1}, w_{l-2}) > 0 \\ \gamma(w_{l-1})\beta(w_l|w_{l-1}), & \text{otherwise.} \end{cases} \tag{22}$$

M-Grams estimate word probabilities in high-dimensional discrete space. This space does not consider any word similarities which lead to the development of other LMs beside M-Grams. One of these is a neural network LM (NNLM). NNLMs project all the words into a common low-dimensional continuous valued space. There are two main types of NNLMs, which are based on feed forward networks and recurrent neural networks.

2.1.4 Lexicon

As mentioned in the previous sections, the AM uses phonemes as modelling unit. However, the LM used in ASR is based on words. To estimate the probability $P(\overline{\boldsymbol{o}}|\overline{w}; \lambda)P(\overline{w})$ of the AM and the LM, a mapping from words to phonemes is needed. The lexicon provides this pronunciation mapping. According to the entries in the lexicon, HMMs for different triphones are concatenated to form the AM for a word.

2.1.5 Decoder

Given LM, AM, and lexicon, the decoder tries to find the most likely word sequence, i.e., optimising the initial problem

$$\hat{\overline{w}} = \underset{\overline{w}}{\operatorname{argmax}} P(\overline{\boldsymbol{o}}|\overline{w};\lambda)P(\overline{w}). \tag{23}$$

A solution to this problem is given by Viterbi decoding [141, 3]. An alternative is the A^* algorithm [52] where a search tree is decoded in depth-first manner. Since the search space grows exponentially, it is important to prune unpromising search hypothesises. One such pruning technique is beam search. This, however, does not guarantee anymore that the optimum solution is found.

After a first-pass decoded result is obtained, more complex models can be applied to improve the decoding result. From the decoder a list of n-best hypothesises can be obtained, which can be re-ranked using for instance NNLMs.

2.1.6 Evaluation Metric

Throughout this thesis there are two important evaluation metrics. For isolated evaluation of an LM, perplexity (PPL) is used. The PPL of a word sequence $\overline{w}$ of length L is defined as

$$\begin{aligned}
\mathrm{PPL}(\overline{w}) &= P(\overline{w})^{-\frac{1}{L}} && (24)\\
&= \left(\prod_{l=1}^{L} P(w_l|\overline{w}[1,\ldots,l-1]\right)^{-\frac{1}{L}} && (25)\\
&= 2^{-\frac{1}{L}\sum_{l=1}^{L}\log_2 P(w_l|\overline{w}[1,\ldots,l-1])}. && (26)
\end{aligned}$$

PPL can be interpreted as the average number of words the LM has to choose from at each word prediction. It thus is a metric that defines how large the decision tree is. The logarithm of base two of the PPL is equivalent to the entropy of the model

$$\begin{aligned}
H(\overline{w}) &= \log_2 \mathrm{PPL}(\overline{w}) && (27)\\
&= -\frac{1}{L}\sum_{l=1}^{L}\log_2 P(w_l|\overline{w}[1,\ldots,l-1]). && (28)
\end{aligned}$$

For evaluation of the whole ASR system, Word Error Rate (WER) is used. For an utterance of length L WER is defined as

$$\text{WER} = \frac{\text{Substitutions} + \text{Deletions} + \text{Insertions}}{L}. \tag{29}$$

It is defined for the smallest number of operations (substitutions, deletions, insertions) that is required to change the decoded utterance to the ground truth utterance. WER can be used to measure the performance of a whole ASR system by counting the mistakes in the output of the system compared to the reference. However, it is only a measure of the combined model and might not show the effect of a single component.

2.2 Count-based Language Models

Count-based LMs are an essential part of an ASR system. Their major advantage is that their parameters can be estimated at low computational cost. This allows to apply the model to large amounts of training data with a large vocabulary size. In addition, these models are an essential part for decoding. Count-based LMs can also be used jointly with NNLMs by linear interpolation to improve the performance of NNLMs. Chapter 2.1.3 already gave a brief introduction on LMs and their application in an ASR system. This section introduces count-based LMs in more detail. This includes an overview of common smoothing techniques and extensions like class-based LMs and cache LMs.

2.2.1 M-Gram Language Models

An M-Gram is an approximation for predicting a word w_l in a word sequence $\overline{w}$. The Markov assumption is applied to truncate the word history to the previous $M-1$ words. That means the conditional probability for the whole word sequence is shortened

$$P(\overline{w}) = \prod_{l=1}^{L} P(w_l|\overline{w}[1,\ldots,l-1]) \tag{30}$$

$$= \prod_{l=1}^{L} P(w_l|\overline{w}[l-M+1,\ldots,l-1]). \tag{31}$$

For notational convenience, the history of a word w_l in an M-Gram $\overline{w}[l-M+1,\ldots,l-1]$ will be denoted by $\overline{h}$ for the remainder of this section. In ASR systems, bigrams $M=2$ or trigrams $M=3$ are used for decoding. M-Grams with longer history can be used after a decoding result is obtained in a rescoring step.

The conditional probability in (31) for an M-Gram is obtained by a maximum likelihood estimate on the training data. For an M-Gram with arbitrary history length M, the conditional probabilities are estimated from the relative frequencies of the word sequences in the training data

$$\hat{P}_{\mathrm{ML}}(w_l|\overline{h}) = \frac{C(w_l\overline{h})}{C(\overline{h})}. \tag{32}$$

$C(\cdot)$ denotes the count of a word sequence, and the word sequences $w_l\overline{h}$ and $\overline{h}$ are the M-Gram and its context, respectively. As the history length M increases, the observations of these M-Grams in the training data decrease. For instance, if the vocabulary size $|\mathcal{U}|$ is 10.000 words, the number of possible trigrams would be $10.000^3 = 10^{12}$. This means, many of these trigrams will not be observed in the training data and the probability estimate for these trigrams will be zero. However, if the output probability of the LM is zero for a word, according to (3) this word cannot be output by the ASR system. To obtain non-zero estimates for all M-Grams, which are not observed in the training data, smoothing techniques are applied.

2.2.2 Smoothing Techniques for M-Gram Language Models

The goal of smoothing techniques is to assign probability mass to unseen M-Grams in the training data. Depending on the smoothing technique, there are existing different ways to redistribute the probability mass. In general, we distinguish between back-off and interpolation. A smoothing technique based on back-off uses the probability estimate of a lower-order M-Gram in case its count is zero

$$\hat{P}_{\mathrm{back-off}}(w_l|\overline{h}) = \begin{cases} \alpha(w_l|\overline{h}), & \text{if } C(w_l|\overline{h}) > 0 \\ \gamma(\overline{h})\beta(w_l|\overline{w}[l-M+2,\ldots,l-1]), & \text{otherwise,} \end{cases} \tag{33}$$

where $\alpha(w_l|\overline{h})$ denotes the higher order estimate, $\gamma(\overline{h})$ is a constant such that all probabilities sum up to one, and $\beta(w_l|\overline{w}[l-M+2,\ldots,l-1])$ the lower-order M-Gram estimate. For notational convenience, this lower-order shortened history will be denoted as $\overline{g} = \overline{w}[l-M+2,\ldots,l-1])$ for the remainder of this section.

A method relying on interpolation will always interpolate the higher order estimate with the lower-order estimate

$$\hat{P}_{\text{interpolate}}(w_l|\overline{h}) = \lambda(w_l|\overline{h})\alpha(w_l|\overline{h}) + (1-\lambda(w_l|\overline{h}))\beta(w_l|\overline{g}). \tag{34}$$

$\lambda(w_l|\overline{h})$ is an interpolation parameter that again should ensure that the probabilities sum up to one. A comprehensive overview of different smoothing techniques for M-Grams was provided in [21, 46]. Here, only a few major techniques are introduced.

When word sequences only appear a few times (one to three) in the whole training set, their probability is generally greatly overestimated [46]. If a larger size training set is available, their number of appearances might still be the same, which in turn means that their probability would be much lower. It might also just happen by chance that a particular word sequence from many with the same history occurs but none of the others. For instance, this could be the case for word sequences containing weekdays, where examples with "Monday" and "Tuesday" are contained in the training data, but none for the remaining days of the week. For this reason, in the Good-Turing [45] estimation modified counts are used. In modified counts r^*, an M-Gram that occurs r times in the training data is pretended to occur $r+1$ times

$$r^* = (r+1)\frac{n_{r+1}}{n_r}, \tag{35}$$

where n_r is the number of M-Grams that occur r times. The modified counts r^* will usually be lower than the original counts r. This is according to the intention to lower the overestimate for rare M-Grams. By modifying the counts, probability mass is left over which can be attributed to unseen events. This Good-Turing estimation is used in Katz smoothing [78]. Katz smoothing uses modified counts for word sequences that occurred in the training data and it backs-off to

a lower-order M-Gram for unseen histories

$$\hat{P}_{\text{Katz}}(w_l|\overline{h}) = \begin{cases} \frac{C^*(w_l\overline{h})}{C(\overline{h})}, & \text{if } C(w_l\overline{h}) > 0 \\ \gamma(\overline{h})\hat{P}_{\text{ML}}(w_l|\overline{g}), & \text{otherwise.} \end{cases} \tag{36}$$

$C^*(\cdot)$ denotes the count for an M-Gram where the count is modified according to (35). In Katz smoothing, the modified count is divided by the true count of the context to derive the estimates of observed M-Grams. The weight $\gamma(\overline{h})$ assures that the probabilities sum up to one and the probability "mass" is proportionally redistributed from M-grams which appear in the training data to those which do not.

In Katz smoothing, the back-off probability assigns an over-estimate to words that appear usually only in certain contexts. For example, currencies like "dollar", "euro", or "yen" are likely to appear after a number, or the word "Francisco" is likely to appear after "San", but all these words are unlikely to appear after a word like "on". This problem of overestimates appears especially for words with low count and the model uses back-off in the case of low or zero counts.

Kneser-Ney smoothing [101, 82] addresses this problem by using a modified back-off distribution. For the estimates of observed events, absolute discounting [100] with a single discount paramter D is used

$$\alpha(w_l\overline{h}) = \frac{C(w_l\overline{h}) - D, 0}{C(\overline{h})}. \tag{37}$$

Kneser-Ney smoothing was motivated by marginal constraints. In general, the marginal distribution of the joint distribution $P(\overline{h}, w_l|\overline{g})$ should be identical to the given distribution $P(w_l|\overline{g})$

$$P(w_l|\overline{g}) = \sum_{\forall \overline{h}} P(\overline{h}, w_l|\overline{g}). \tag{38}$$

For instance, for a unigram and a bigram, the marginal constraints are given by

$$P(w_l) = \sum_{w_{l-1}} P(w_l, w_{l-1}), \tag{39}$$

$$P(w_l|w_{l-1}) = \sum_{w_{l-2}} P(w_l, w_{l-2}|w_{l-1}). \tag{40}$$

These marginal constraints should be fulfilled for a well-designed probability distribution, however, they are not fulfilled for most conventional backing-off estimates relying on relative frequencies. Using these marginal constraints, the back-off $\beta(w_l|\overline{g})$ can be derived as

$$\beta(w_l|\overline{g}) = \frac{C_+(\cdot, \overline{g}, w_l)}{C_+(\cdot, \overline{g}, \cdot)}, \tag{41}$$

$$C_+(\cdot, \overline{g}, w) = \sum_{w:C(\overline{g},w)>0} 1, \tag{42}$$

$$C_+(\cdot, \overline{g}, \cdot) = \sum_w C_+(\cdot, \overline{g}, w), \tag{43}$$

where $C_+(\cdot, \overline{g}, w) = |\{\overline{g}|C(w\overline{g}) > 0\}|$ denotes the number of distinct contexts $\overline{g}$ the word w can appear in. For instance, the unigram back-off $\beta(w_l)$ is given by

$$\beta(w_l) = \frac{C_+(\cdot, w_{l-1}, w_l)}{\sum_w C_+(\cdot, w_{l-1}, \cdot)}. \tag{44}$$

Combining this back-off probability (41) and the estimate for observed events using absolute discounting (37), the combined formula can be written as

$$\hat{P}_{\mathrm{KN}}(w_l|\overline{h}) = \begin{cases} \frac{C(w_l\overline{h})-d,0}{C(\overline{h})}, & \text{if } C(w_l\overline{h}) > 0 \\ \gamma(\overline{h})\frac{C_+(\cdot,\overline{g},w_l)}{C_+(\cdot,\overline{g},\cdot)}, & \text{otherwise.} \end{cases} \tag{45}$$

A modification of Kneser-Ney smoothing is modified Kneser-Ney smoothing [22]. Whereas in Kneser-Ney smoothing a single discount parameter D is used for all M-Gram counts, in modified Kneser-Ney smoothing, the discount parameters D depends on the M-Gram count. The back-off probability is unchanged from normal Kneser-Ney smoothing.

In smoothing methods using back-off, the estimate is backed-off to the lower-order M-Gram when the count of the higher order M-Gram is equal to zero. However, smoothing methods based on interpolation always interpolate the higher order model with the lower-order model. One method for interpolation is Jelinek-Mercer smoothing [74]. In cases where the higher order M-Gram cannot be estimated from the training data, the lower-order ones are used. In Jelinek-Mercer smoothing the maximum likelihood estimate (32) is interpolated with a lower-order model

$$\hat{P}_{\mathrm{JM}}(w_l|\overline{h}) = \lambda(w_l|\overline{h})\hat{P}_{\mathrm{ML}}(w_l|\overline{h}) + (1 - \lambda(w_l|\overline{h}))\hat{P}_{\mathrm{JM}}(w_l|\overline{g}). \tag{46}$$

The lower-order M-Gram in the above equation is defined recursively. The lowest-order M-Gram can thus be the unigram ($P(w_l)$) or the zero-gram, that is, the uniform distribution over the vocabulary size

$$\hat{P}_{\text{zero}}(w_l) = \frac{1}{|\mathcal{U}|}. \tag{47}$$

The interpolation coefficient can be estimated efficiently with the Baum-Welch algorithm [6, 108] on held-out data when a fixed maximum likelihood estimate is given.

2.2.3 Class-based Language Models

A drawback of count-based LMs is that these models do not learn any semantic information. Semantically similar words are treated as different events. For instance, an M-Gram LM treats all days of the week as a different word, whereas they are likely to appear in the same context. Class-based LMs [17, 47] try to mitigate this shortcoming by clustering words into semantically similar classes. A class-based LM defines a mapping from the vocabulary $\mathcal{U}$ to a set of word classes $\mathcal{C}$. The probability estimate of a word can be derived by multiplying the conditional probability of the word class C_l with the M-Gram probability given the word class

$$P_{\text{class}}(w_l|\overline{h}) = P(C_l|C_{l-1}) \cdot P(w_l|C_l). \tag{48}$$

$P(C_l|C_{l-1})$ is a bigram estimate over word classes and $P(w_l|C_l)$ gives the membership probability of w_l belonging to C_l. Using word classes showed to improve Word Error Rate (WER) in ASR for small data sets compared with using only an M-Gram LM, but the gains vanish when more training data is available [46].

2.2.4 Cache Language Models

M-Gram LMs have only a very limited history. However, there might be words in the more distant past relevant for prediction of the current word w_l. Cache LMs [83, 75, 24] were introduced to extend the short M-Gram history by a dynamic longer history. Cache LMs are like M-Grams estimated from relative frequencies. However, they use a context window of the most recent words to derive the

estimate dynamically. Cache LMs are not used as stand-alone LM. Usually, the estimate from the cache is used in a linear interpolation with a smoothed M-Gram

$$P_{\text{ngram}-\text{cache}}(w_l|\overline{h}) = \lambda P(w_l|\overline{h}) + (1-\lambda)P_{\text{cache}}(w_l|\overline{h}), \tag{49}$$

where P_{cache} is the probability estimated from relative frequencies over the cash. Experiments showed large perplexity (PPL) improvements when using cache LMs. However, improvements in ASR were harder to achieve because errors in the recognition result affect the cache [46].

2.3 Neural Network Language Models

Despite their simplicity, there are several disadvantages of count-based LMs, as pointed out in Chapter 2.2. By increasing the history length, M-Grams become increasingly sparse and their parameter size grows exponentially. In addition, count-based LMs do not consider any semantics. For these models, the singular or plural of a word, or different days of a week, which might be likely to appear in the same context, all represent different events. Because M-Grams consider words and counts as discrete events, they are known to not generalise well for unseen events.

In this section, a different paradigm for language modelling will be described. These language models are based on Neural Networks (NN). NNs showed superior performance in many tasks such as AMs for ASR, and machine translation to name just two examples. NN-based LMs (NNLM) use a continuous valued space. All words are embedded into this (usually) low dimensional space. In this space, it is possible to encode semantics of similar words. This property can be illustrated by the following example.

`The party will be on Monday.`

`My friend's birthday will be next week.`

Observing a couple of these examples in the training data will give higher probabilities to sentences with a more general pattern.

`[Event] will be [time].`

However, not all of these possible examples might be observed in the training data. For NNLMs, the two paradigms feed-forward and Recurrent Neural Network LMs (RNN-LMs) exist, which will be described in the following.

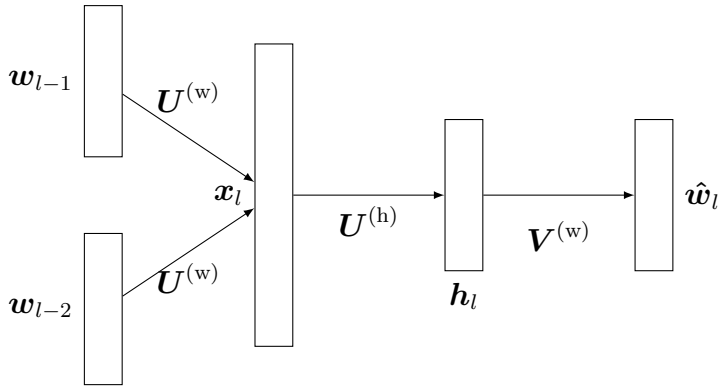


Figure 4: Trigram feed-forward neural network language model.

2.3.1 Feed-Forward Neural Network Language Model

Before the application to LMs, NNs were successfully applied to LM related tasks such as word category prediction [97]. LMs based on feed-forward NNs were first proposed by Yoshua Bengio et al. [11, 12]. However, the computational constraints were very high and there had to be made many compromises on the model. Holger Schwenk et al. [116, 117] were among the first to explore the application of feed-forward NNLMs to tasks like ASR. An overview for feed-forward NNLM was published by Holger Schwenk in [115].

Feed-forward NNLMs are similar to count based models. They use a fixed size context window of past words, which means they use a shortened word history. As with M-Grams, these models rely on the Markov assumption and model the probability of a word sequence as

$$P(\overline{w}) = \prod_{l=1}^{L} P(w_l|\overline{w}[l-M+1,\ldots,l-1]), \tag{50}$$

where M is the order of the model. The architecture of a trigram feed-forward NNLM is shown in Figure 4. The model uses as input the last two preceding words. Each word is represented by a word ID, which is encoded in a word vector $\boldsymbol{w}$. The dimensionality of this vector is the size of the vocabulary $|\mathcal{U}|$. In the word vector $\boldsymbol{w}$, all dimensions except for the active word ID are set to zero. This encoding is called a one-hot vector and it allows for efficient computation of the network input.

Subsequently, the input words $\boldsymbol{w}$ are projected into a lower-dimensional embedding space by a shared projection matrix $\boldsymbol{U}^{(\mathrm{w})}$

$$\boldsymbol{x}_l = [\boldsymbol{U}^{(\mathrm{w})}, \boldsymbol{U}^{(\mathrm{w})}][\boldsymbol{w}_{l-1}, \boldsymbol{w}_{l-2}]^{\mathrm{T}}. \tag{51}$$

For convenience, the above equation uses block matrix notation. The projections of all input words are concatenated to form the input vector $\boldsymbol{x}_l$ to the hidden layer. Feed-forward NNLMs usually use a single hidden layer with a hyperbolic tangent or sigmoid non-linearity $f(\cdot)$

$$\boldsymbol{h}_l = f(\boldsymbol{U}^{(\mathrm{h})}\boldsymbol{x}_l + \boldsymbol{b}^{(\mathrm{U,h})}). \tag{52}$$

$\boldsymbol{b}^{(\mathrm{U,h})}$ is the bias vector for the hidden layer. The prediction of word $\hat{\boldsymbol{w}}_l$ is calculated by the output layer followed by the softmax function

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\,(\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l), \tag{53}$$

where the softmax function for a vector $\boldsymbol{x}$ is defined as

$$\mathrm{softmax}\,(\boldsymbol{x}) = \frac{\mathrm{e}^{\boldsymbol{x}}}{\sum_{i=0}^{I} \mathrm{e}^{x_i}}. \tag{54}$$

The softmax function normalises the output, such that its elements sum up to one. This is required to obtain output probabilities $P(w_l|\overline{w}[l-M+1,\ldots,l-1])$ for word w_l. The size of the output layer is the size of the vocabulary $|\mathcal{U}|$. This is the computational bottle-neck, as with all NNLMs

The model learns an embedding space for the words through the projection matrix $\boldsymbol{U}^{(\mathrm{w})}$ as part of the training. In this space, semantically similar words are projected to positions close to each other in space. As shown by Tomas Mikolov et al. [91], this leads to a space where simple arithmetic operations on the projected word vectors, like plus and minus, can be used to get to similar words. For instance, subtracting the capital of Germany (Berlin) from the word vector for Germany and adding the capital for Japan (Tokyo), will lead to a vector that is very similar to the word vector for Japan

$$\boldsymbol{x}(\text{'Germany'}) - \boldsymbol{x}(\text{'Berlin'}) + \boldsymbol{x}(\text{'Tokyo'}) = \boldsymbol{x}(\text{'Japan'}). \tag{55}$$

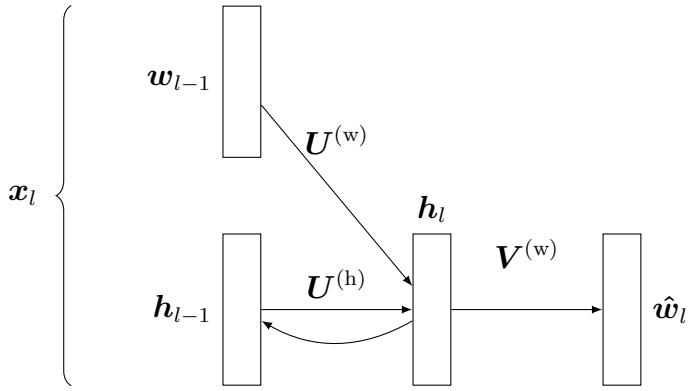


Figure 5: A vanilla RNN-LM as proposed in [92].

2.3.2 Recurrent Neural Network Language Model

Feed-forward NNLMs use a fixed input length which limits their history to the sequence length of the input. This was the motivation for using Recurrent Neural Networks (RNN) in LMs. Vanilla RNN-LMs have an Elman-type recurrent layer [36] that enables these networks in theory to use an unlimited history. However, the vanishing an exploding gradient problem [13] limits the effective history length these networks can remember. Different kinds of RNN-LM architectures were proposed [92, 128, 72], which differ in the kind of recurrent unit that is used in the network. Here, vanilla RNN-LMs based on Elman-type recurrent networks [92, 93] are discribed and LSTM-LMs where the recurrent layer is realised by long short-term memory [64, 128, 127].

The vanilla RNN-LM uses an Elman-type recurrent layer as shown in Figure 5. The input to the network is the one-hot vector $\boldsymbol{w}_{l-1}$ for the last word and the hidden layer state from the previous time step $\boldsymbol{h}_{l-1}$

$$\boldsymbol{x}_l = [\boldsymbol{w}_{l-1}^{\mathrm{T}},\ \boldsymbol{h}_{l-1}^{\mathrm{T}}]^{\mathrm{T}}. \tag{56}$$

The current hidden state $\boldsymbol{h}_l$ is calculated by multiplying the input with the corresponding transition matrices and applying a non-linear activation function $\sigma(\cdot)$

$$\begin{aligned} \boldsymbol{h}_l &= \sigma(\boldsymbol{U}\boldsymbol{x}_l) && (57)\\ &= \sigma(\boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1} + \boldsymbol{U}^{(\mathrm{h})}\boldsymbol{h}_{l-1}). && (58) \end{aligned}$$

In this case, $\sigma(\cdot)$ is the sigmoid function

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{x}}. \tag{59}$$

$\boldsymbol{U}$ is the weight matrix for the input vector, which can be decomposed into $\boldsymbol{U}^{(\mathrm{w})}$ and $\boldsymbol{U}^{(\mathrm{h})}$, the parts for the word and last hidden layer, respectively.

This structure is at first equivalent to a bigram LM because the output only depends on the current word. However, by unfolding the hidden state recursively in time, it becomes obvious that the model in theory covers an infinitely long history

$$\begin{aligned}
\boldsymbol{h}_l &= \sigma(\boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1} + \boldsymbol{U}^{(\mathrm{h})}\boldsymbol{h}_{l-1}) && (60)\\
&= \sigma(\boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1} + \boldsymbol{U}^{(\mathrm{h})}\sigma(\underbrace{\boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-2} + \boldsymbol{U}^{(\mathrm{h})}\boldsymbol{h}_{l-2}}_{=l-1})) && (61)\\
&= \sigma(\boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1} + \boldsymbol{U}^{(\mathrm{h})}\sigma(\underbrace{\boldsymbol{U}^{(\mathrm{w})}\sigma(\underbrace{\boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-3} + \boldsymbol{U}^{(\mathrm{h})}\boldsymbol{h}_{l-3}}_{=l-2}) + \boldsymbol{U}^{(\mathrm{h})}\boldsymbol{h}_{l-2}}_{=l-1})). && (62)
\end{aligned}$$

Thus, RNN-LMs are capable of calculating the probability for a word sequence on an untruncated history unlike all other models introduced so far

$$P(\overline{w}) = \prod_{l=1}^{L} P(w_l | w_{1\ldots l-1}). \tag{63}$$

The output of the network, that is the probability for word w_l, is computed by multiplying the hidden state with the output layer $\boldsymbol{V}^{(\mathrm{w})}$ and applying the softmax function

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\,(\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l). \tag{64}$$

One of the problems of all NNLMs is the large output layer and the softmax function. The softmax function normalises each element in the output vector which is a computational bottle-neck for a vector of such high dimensionality. Mikolov et al. [93] proposed to segment the output layer into two parts to speed up the computation. This extension to the original RNN-LM is shown in Figure 6. The vocabulary is sorted according to the word frequency and split into word

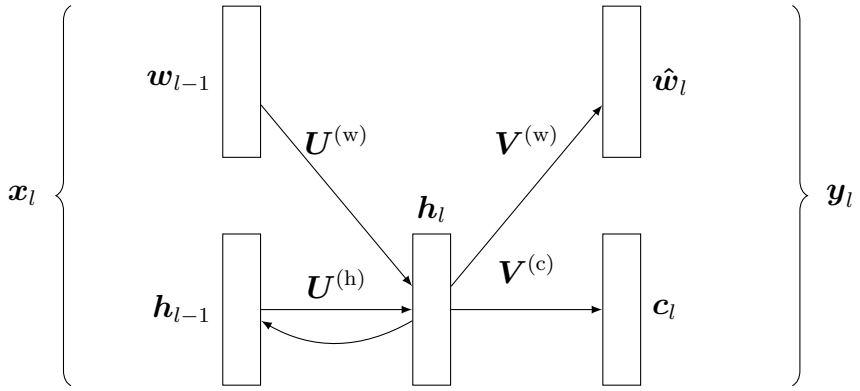


Figure 6: A vanilla RNN-LM with word classes as proposed in [93].

classes. Each word class covers words of a similar relative frequency. When calculating the output probability, the network first calculates the softmax over each class and in a second step the softmax over all words in the most-likely class C_l

$$\boldsymbol{y}_l = \mathrm{softmax}\left(\boldsymbol{V}^{(\mathrm{c})}\boldsymbol{h}_l\right) \cdot \mathrm{softmax}\left(\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l\right)\Big|_{C_l}. \tag{65}$$

Usually the number of classes is much smaller than the vocabulary size which leads to a significant reduction of computational cost at a small loss of accuracy.

Although RNN-LMs in theory can cover an infinitely long history, in practice these networks cannot be trained to exploit such a long history. Vanilla RNNs are limited by the vanishing and exploding gradient problem [13]. During network training, gradients are calculated and the error of a training criterion is back-propagated through the network. These errors can get scaled by a factor larger or smaller than one. This leads to an explosion or vanishing of the gradient over multiple time steps (see [64] for a detailed analysis). To circumvent this problem of recurrent networks, Hochreiter et al. [64] proposed a novel recurrent unit which they called Long Short-Term Memory (LSTM). In this unit the gradients are restricted to one and the LSTM cell has additional gates to enhance learning capabilities. Using these LSTM cells, Martin Sundermeyer et al. [128] proposed a modification of Tomas Mikolov et al.'s vanilla RNN-LM.

A single LSTM cell is shown in Figure 7. In the language model, multiple of these cells are used in the hidden layer. Each cell uses the following equations to

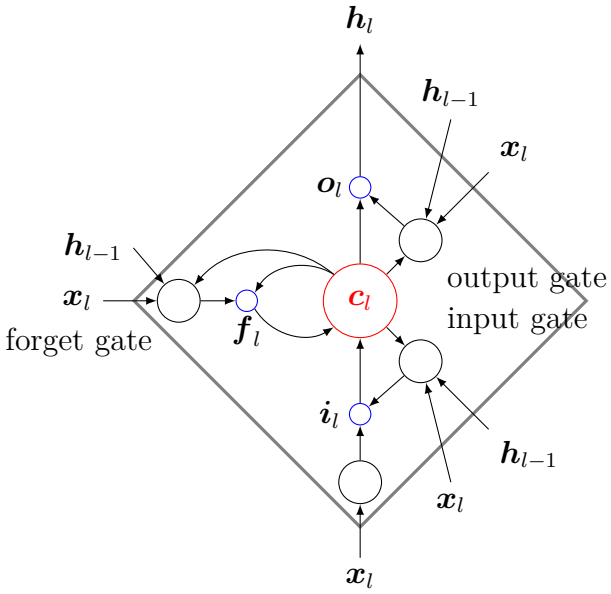


Figure 7: A long short-term memory cell [64].

calculate its cell state $\boldsymbol{c}_l$ and output $\boldsymbol{h}_l$

$$\boldsymbol{i}_l = \sigma(\boldsymbol{W}^{(\mathrm{i,w})}\boldsymbol{x}_l + \boldsymbol{W}^{(\mathrm{i,h})}\boldsymbol{h}_{l-1} + \boldsymbol{b}^{(\mathrm{i})}), \tag{66}$$

$$\boldsymbol{f}_l = \sigma(\boldsymbol{W}^{(\mathrm{f,w})}\boldsymbol{x}_l + \boldsymbol{W}^{(\mathrm{f,h})}\boldsymbol{h}_{l-1} + \boldsymbol{b}^{(\mathrm{f})}), \tag{67}$$

$$\boldsymbol{o}_l = \sigma(\boldsymbol{W}^{(\mathrm{o,w})}\boldsymbol{x}_l + \boldsymbol{W}^{(\mathrm{o,h})}\boldsymbol{h}_{l-1} + \boldsymbol{b}^{(\mathrm{o})}), \tag{68}$$

$$\boldsymbol{g}_l = \tanh(\boldsymbol{W}^{(\mathrm{g,w})}\boldsymbol{x}_l + \boldsymbol{W}^{(\mathrm{g,h})}\boldsymbol{h}_{l-1} + \boldsymbol{b}^{(\mathrm{g})}), \tag{69}$$

$$\boldsymbol{c}_l = \boldsymbol{f}_l \odot s_{l-1} + \boldsymbol{i}_l \odot \boldsymbol{g}_l, \tag{70}$$

$$\boldsymbol{h}_l = \boldsymbol{o}_l \odot \tanh(\boldsymbol{c}_l), \tag{71}$$

$\boldsymbol{i}_l$, $\boldsymbol{f}_l$ and $\boldsymbol{o}_l$ are usually named the input, forget and output gate, respectively. $\boldsymbol{W}^{(\{\mathrm{i,f,o}\},\mathrm{w})}$ and $\boldsymbol{W}^{(\{\mathrm{i,f,o}\},\mathrm{h})}$ denote the weight matrices for each gate for the word input and the previous hidden layer, respectively. $\boldsymbol{b}^{(\{\mathrm{i,f,o}\})}$ are the bias vectors for the respective gates. Since the above equations use vector notation, $\sigma(\cdot)$ is the element wise sigmoid, $\tanh(\cdot)$ is the element wise hyperbolic tangent and $\odot$ denotes an element wise multiplication.

When the recurrent layer uses LSTM memory cells instead of a vanilla RNN, in the remainder of this thesis a simplified description from the complete set of equations in 66 to 71 will be used. To distinguish the difference between vanilla

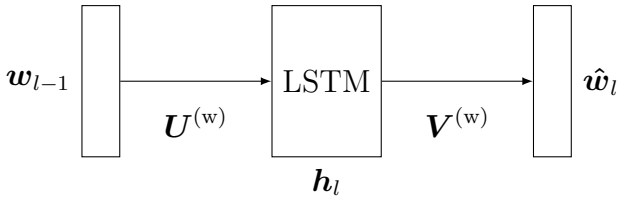


Figure 8: An LSTM-LM with long short-term memory as recurrent unit.

RNN-LMs and those using LSTMs, the latter one will be denoted as LSTM-LM. Figure 8 shows the simplified model of an LSTM-LM. The input word ID is again encoded by a one-hot vector $\boldsymbol{w}_{l-1}$. Then the input into the LSTM $\boldsymbol{x}_l$ is calculated using the word embedding matrix $\boldsymbol{U}^{(\mathrm{w})}$

$$\boldsymbol{x}_l = \boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1}. \tag{72}$$

The LSTM computations in (66) to (71) are replaced by a single operation

$$\boldsymbol{h}_l = \mathrm{lstm}(\boldsymbol{x}_l). \tag{73}$$

After the output of the recurrent layer $\boldsymbol{h}_l$, the output layer $\boldsymbol{V}^{(\mathrm{w})}$ and the softmax function follow to calculate the probability for the next word $\hat{\boldsymbol{w}}_l$

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\left(\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l\right). \tag{74}$$

2.3.3 Neural Network Language Model Training

Training of a statistical classifier involves usually two steps. First, a training criterion or optimisation function is defined. This is the output function the classifier computes. Second, the derivative of this training criterion with respect to the classifier parameters is derived and optimised by for example gradient descend. For NNLMs the training criterion is cross entropy (CE). Adapting a general notation, it is defined as ([34, p. 318])

$$CE = \sum_{l=1}^{L}\sum_{c=1}^{C} \boldsymbol{t}_{l,c}\log(\frac{\boldsymbol{t}_{l,c}}{y_{l,c}}), \tag{75}$$

where $\boldsymbol{t}_{l,c}$ is the target label and $y_{l,c}$ the actual output of the NNLM for pattern l and class c. The number of classes is in the case of language modelling the

number of words in the vocabulary $\mathcal{U}$. For language modelling, we use a maximum likelihood of the training data. In this case we assume that the training label $\boldsymbol{t}$ is one for the target word and zero for all other output words

$$\boldsymbol{t}_{l,c} = \begin{cases} 1, \text{if } w_l = w_{c_l} \\ 0, \text{otherwise.} \end{cases} \tag{76}$$

Using this assumption, the training criterion can be simplified using $y_{l,c=c_l}$ as the output of the network for the target word w_{c_l} to the following form

$$CE = -\sum_{l=1}^{L} \log y_{l,c=c_l}. \tag{77}$$

The derivative of the CE training criterion has to be taken with respect to all parameters in the network. To update the parameters β in the NN, the following update rule is then applied

$$\beta' = \beta - \eta \frac{\partial CE}{\partial \beta}, \tag{78}$$

where η is called the learning rate. To update parameters prior to the output layer, the chain rule has to be applied. This training method is commonly known as stochastic gradient descend (SGD) or error backpropagation [112] because the errors from the output are backpropagated to the input. SGD is the simplest case, but there also exist modifications where the parameter update rule is modified [33]. A general derivation of the backpropagation algorithm for NNs can be found in common literature on machine learning and pattern recognition [34, 15]. Following in this section, the parameter update will be outlined step-by-step for the case of a feed-forward NNLM. The notation in this section follows the one used in other literature [90, 126].

For NNLM training, first a training sample is presented to the the network. Then the output error, that is, the derivative of the CE is calculated with respect to the output of the network

$$\boldsymbol{e}_l^{(\mathrm{o})} = \boldsymbol{t}_l - \boldsymbol{y}_l. \tag{79}$$

$\boldsymbol{e}_l^{(\mathrm{o})}$ is the error in the output, $\boldsymbol{t}_l$ the target label, that is word w_l and $\boldsymbol{y}_l$ is the output of the NN. To update the parameters, the chain rule is applied to (79)

and the derivative is taken with respect to the parameters of the output layer $v_{jk} \in \boldsymbol{V}^{(\mathrm{w})}$

$$v'_{jk} = v_{jk} + \eta \boldsymbol{h}_{l,j} \boldsymbol{e}^{(\mathrm{o})}_{l,k}. \tag{80}$$

To update the parameters in the input layer, the derivative of the hidden state $\boldsymbol{h}_l$ is taken with respect to its input

$$\boldsymbol{e}^{(\mathrm{h})}_l = (\boldsymbol{V}^{(\mathrm{w})^\mathsf{T}} \boldsymbol{e}^{(\mathrm{o})}_l) \odot \boldsymbol{h}_l \odot (1 - \boldsymbol{h}_l). \tag{81}$$

In the above equation, $\odot$ is an element-wise multiplication of two vectors. (81) uses the derivative for the sigmoid function

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)) \tag{82}$$

Using this derivative, the following update rule can be used to obtain the updated parameters for each element in the input layer matrix $u^{(\mathrm{h})}_{i,j} \in \boldsymbol{U}^{(\mathrm{h})}$

$$u^{(\mathrm{h})'}_{i,j} = u^{(\mathrm{h})}_{i,j} + \eta \boldsymbol{x}_{l,i} \boldsymbol{e}^{(\mathrm{h})}_{l,j} \tag{83}$$

Finally, the parameters in the projection layer $u^{(\mathrm{w})}_{m,i} \in \boldsymbol{U}^{(\mathrm{w})}$ are updated by one more step of error backpropagation

$$\boldsymbol{e}^{(\mathrm{p})}_l = \boldsymbol{U}^{(\mathrm{h})^\mathsf{T}} \boldsymbol{e}^{(\mathrm{h})}_l, \tag{84}$$

$$u^{(\mathrm{w})'}_{m,i} = u^{(\mathrm{w})}_{m,i} + \eta \boldsymbol{e}^{(\mathrm{p})^\mathsf{T}}_l \boldsymbol{U}^{(\mathrm{w})} (\boldsymbol{w}_{l-1,m} + \boldsymbol{w}_{l-2,m}). \tag{85}$$

The last equation can be greatly simplified since $\boldsymbol{w}$ is zero for all elements other than the input at time $l-1$ and $l-2$. This means we do not have to update all elements in the projection layer, but only those for the active word indices.

Usually, not only a single training example is used to update the parameters but many samples are grouped in a so-called mini-batch. This makes all update equations matrix-matrix operations. The update process is repeated for all elements in the training data. One whole sweep through the training data is called an epoch. If we train a NNLM by SGD, the model learns the training data and it is well known that NNs tend to overfit on the training data when trained for many epochs. To control the training progress, a held-out data set (validation data) is used to adjust the learning rate and the number of epochs. When the

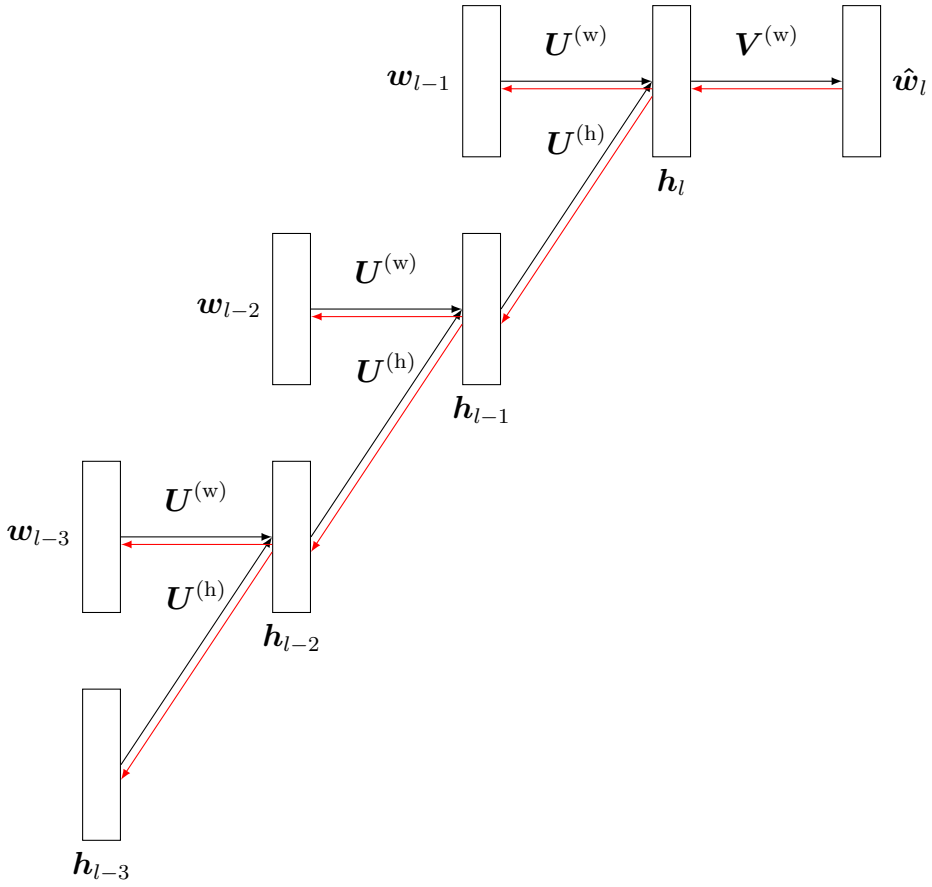


Figure 9: Training of a vanilla RNN-LM with backpropagation through time where black lines denote the forward path and red lines the backward path.

classification performance on the validation set starts to deteriorate, the learning rate is reduced and eventually training is terminated.

The parameter update equations as in (80), (83), and (85) can be derived in a similar way for the recurrent layer in vanilla RNN-LMs and the LSTM cells in LSTM-LMs [64]. However, training an RNN-LM in this manner will not make the network learn to use the history. It effectively trains the network only as a bigram LM. To make the RNN-LM learn to use its history, a recursive unfolding of the recurrent layer over time as shown in Figure 9 is required. By unfolding the recurrent layer, the network becomes equivalent to a feed-forward network with shared parameters for the hidden layer. However, in contrast to a multi-

layer feed-forward NN, an unfolded RNN-LM has an input at each hidden layer. The unfolding of the recurrent layer over time and the backpropagation of errors through time is called backpropagation through time [95, 143, 144] (BPTT). Vanilla RNN-LMs are only unfolded over a few timesteps (usually less than five) because of the vanishing gradient problem.

To train RNN-LMs, the training data has to be presented in sequential order because these models work on sequences. For mini-batch training it is thus common to have different mini-batches starting at different points in the training data. These mini-batches then proceed sequentially through the training data.

3. Context Information for Neural Network Language Models

LMs prediction the probability of a word sequence. As introduced in the previous section, M-Grams LMs and feed-forward NNLMs make this prediction using a truncated word history. RNN-LMs have the ability to cover a long history and in theory they can properly calculate the conditional probability for a word sequence as given in (18).

However, for vanilla RNN-LMs the effective history is also limited and LSTM-LMs will in practice bias their prediction on the recent word history. Besides being able to model semantic similarities among words very well, these models might not be able to cover a more global context of the underlying word sequence. A text or a presentation can be thought of to cover a general topic. For instance, a newspaper article might report news on a recent event, or someone might give a talk on current social problems and suggestions how to handle them. Further on, texts are usually structured into segments which should handle a consistent line of thought. Depending on the section, there can be several sub-topics beside the main topic. The same applies to conversations. A conversation or a meeting covers one or multiple topics.

In addition, there is also context information beside this word history and topic information in ASR. In ASR, there is also the acoustic signal s and the feature vector sequence extracted from it $\overline{\boldsymbol{o}}$. This signal can contain speaker characteristics, which could allow for adaptation to a speaker's style when combined with the LM.

In the remainder of this section, different kinds of context information are introduced as follows. In the context of ASR, there is first the information contained in the acoustic signal s, that is spectral and prosodic features. Second is local word context as it can be exploited in caches for LMs. Third is context information from topic information. Topic information is information provided on a document level. Here, a common context model called latent Dirichlet allocation which has been a very popular topic model in research will be introduced. In addition, this section introduces neural network based methods to extract topic-like information.

3.1 Prosodic Features

Prosodic features are one possible kind of information which can help to improve language modelling. Prosodic information is a way to describe the speaking style. A person's way of speaking is characterised by many variables. Men and women speak with a lower or a higher voice in general. Some speakers speak very fast and fluent, while others have a slower rhythm and make many breaks. When talking in front of a larger crowd, we tend to raise our voice so that everyone in the audience is able to hear us. Vice versa, when we want to deliver a message to someone secretly without anyone around us noting it, we are whistling.

These are just a few examples, but they illustrate the main points that prosodic information consists of: fundamental frequency (or pitch), signal power, and duration. Where duration usually refers to speaking rate or the length of syllables or pauses in between syllables. The motivation for using prosodic information for language modelling is that studies showed that humans can make use of prosody in their speech processing [122]. In addition, this information is less affected by noise and channel variations compared to other spectral features [118].

Furthermore, prosodic information is usually not captured by the AM. Features for AM can include signal power but usually do not include the fundamental frequency. Duration is not included explicitly either. Including prosodic features in the LM rather than the AM uses features on a word level compared to phone level. We might think for example that important words are stressed differently from unimportant words. Such information could be useful in rescoring to distinguish correct from incorrect hypothesises.

In the past, prosodic information was included in several approaches for various language processing tasks. An early approach for including prosody in an ASR system included the investigation of acoustic and lexical stress on a phoneme level [62]. For the resolution of ambiguities in the scoring of parsing hypothesis, [136] and [103] used automatically detected prosodic phrase boundaries.

It was also investigated if prosodic information could be modelled by statistical models. [39] investigated the explicit modelling of word durations with GMMs. The authors used feature vectors with phoneme durations for each word to train the GMMs. To make the system robust against unseen words they included back-

off monophones as well into their system. Subsequently, using hidden events, such as, sentence boundaries or disfluencies in an M-Gram LM was investigated in [124]. The speech data was annotated with sentence boundaries and disfluencies, such as, repetitions, deletions, and filled pauses. This data was then used to train a decision tree classifier. An extension to both the afore mentioned approaches was provided by [137], where an M-Gram model was introduced to predict the pause duration between words from the word context.

For the task of speech under standing, Nöth et al. [102] were the first to claim integrating prosody throughout a speech understanding system. Nöth et al. proposed a speech-to-speech system that should translate utterances from German to English. It was trained and evaluated on a corpus that was hand-labelled with specific prosodic information. In an extension to maximum entropy LMs [109, 14], Chan et al. [18] proposed to incorporate information derived from part of speech tags and accent annotations.

Recent approaches using RNN-LMs, proposed the integration of duration and frequency information for word prediction [40]. Prosodic features from a context size of several syllables were used as input to the network. A similar approach was shown by Fu et al. [38]. They used prosodic information on a word level as an additional feature for the LM. In contrast to many of the above mentioned approaches, Gangireddy et al. and Fu et al.'s approaches did not require any annotation in the data for specific prosodic features. They used prosodic features that can be calculated from the acoustic signal and did not require any special hand-labelled data. In practise, this method is preferred because annotation for prosody might not be available in many corpora and systems.

This is also the motivation for the prosodic features described in the remainder of this section. The features should be computable using tools that are available for free on the internet or their computation should follow a standard processing scheme that is known to the public. In addition, there are other requirements and restrictions to features. First, the information for the features should be extractable in an ASR system. Second, the information has to be combinable with the LM. The prosodic features introduced in the following are based on fundamental frequency and signal power.

3.1.1 Fundamental Frequency

The fundamental frequency is a basic characteristic of human speech. It varies between male and female, children and adults. The fundamental frequency or F0 is defined as "an inverse of the period of the voiced signal in each short time frame, and a voiced segment is defined as one composed of successive short time frames that contain voiced signals" [98]. From this definition, the task of estimating F0 can be divided into two subtasks, that is, deciding whether a segment contains voice or not, and estimating the F0.

Various algorithms have been developed to estimate F0. Currently, state of the art estimators are Getf0 [131], YIN [26], and SAaC [35]. The publicly available open-source speech recogniser Kaldi [107] features an estimator that builds on these state of the art algorithms [43]. In the experiments section 5.3, the method described in [98] is used for estimating F0 features.

From the F0 feature vector sequence $\overline{\boldsymbol{o}}_{\mathrm{F0}}$, the following features are calculated for each word:

1. Mean F0 (μ_{F0}) of all F0 feature frames N_l within one word w_l (f0m)

$$\mu_{\mathrm{F0}}[l] = \frac{1}{N_l} \sum_{n=n_s}^{n_e} \overline{\boldsymbol{o}}_{\mathrm{F0}}[n], \tag{86}$$

   where n_s is the corresponding start index and n_e the end index of word w_l in the F0 feature sequence.

2. The central difference of f0m between adjacent words w_{l-1} and w_{l+1}, commonly known as delta coefficient (f0d)

$$\Delta_{\mathrm{F0}}[l] = \frac{\mu_{\mathrm{F0}}[l+1] - \mu_{\mathrm{F0}}[l-1]}{2}. \tag{87}$$

The prosodic information is estimated frame-wise at a frame shift of 1 ms. To calculate the mean F0, static word boundaries cannot be used because usually a different amount of feature frames is included within the word boundaries for one word w_l. These word boundaries are estimated from a word-frame alignment obtained using a speech recogniser. From the word-frame alignment, estimated start and end times of each word can be obtained.

3.1.2 Signal Power

The signal power which is related to loudness is used a second prosodic feature in this thesis. The signal power is calculated from the speech signal $s[t]$ by taking the square of the signal value. As for F0 features, mean (pwm) and delta (pwd) features per word features w_l are calculated.

$$\mu_{\text{pw}}[l] = \frac{1}{N_l} \sum_{n=n_s}^{n_e} s[n], \tag{88}$$

$$\Delta_{\text{pw}}[l] = \frac{\mu_{\text{pw}}[l+1] - \mu_{\text{pw}}[l-1]}{2}, \tag{89}$$

where n_s is the corresponding start index and n_e the end index of word w_l in the speech signal. The speech signal in ASR is commonly an uncompressed linear PCM audio file sampled at 8 to 16 kHz. As for F0 features, the word boundaries for each word n_s and n_e are estimated from a word-frame alignment from a speech recogniser.

3.2 Cache Memory Features

As outlined in Chapter 2.3.2, RNN-LMs [92] have the ability to learn word sequences, however, their learning capability is limited by the exploding and vanishing gradient problem[13]. This was subsequently addressed in LSTM-LMs [64, 128]. However, recent research suggests, that even LSTMs do not have sufficient memory for some tasks. Neural touring machines [49] demonstraged more effective generalisaion beyond the training data compared to LSTMs. One of the tasks Graves et al. trained their networks on, the model had to learn a simple sequence copy algorithm. It showed to successfully learn this algorithm and generalising the task to sequences of arbitrary length.

Memory networks [145][125] share a similar idea. They were able to achieve a better performance on a question answering task and a lower PPL on the Penn Treebank dataset [88] compared with RNN or LSTM models. Using this idea, Tran et al. [135] proposed recurrent memory networks that combine an LSTM with the memory cell from the memory network. Tran et al. evaluated their model in terms of perplexity on datasets for three languages and a sentence completion task. Furthermore, stack-augmented RNNs [76] were investigated on the task

of learning simple algorithmic patterns. Stack-augmented RNNs are a recurrent network that has the ability to read from and write to a stack.

However, a problem of the above methods is that they are computationally expensive. For instance, during training the network has to learn the memory access mechanism. This mechanism is often realised by calculating attention vectors over the memory and the input to the network. For language modelling, simpler ways of extending neural networks with memory, such as, pointer networks [140, 89] and a continuous cache-based approach [48], showed to be effective. In pointer networks, the pointers enable the network to establish direct connections from the input to the output. This enables the network to use Out-Of-Vocabulary (OOV) words appearing on the input of the network for word prediction. In the continuous cache, the hidden state of the network and the true output label are used to calculate a probability for the predicted word from the cache.

For language modelling, there have been other approaches in recent years based on Bag-of-Words (BoW). In order to make use of the information in preceding dialogue turns, Zamora et al. [147] extended a feed forward neural network with a cache component. The cache should keep information from previous dialog turns because, for instance, a question in a dialogue can be related to the answer following it. BoW was introduced later as a cache extension for RNN-LMs [71, 50]. The BoW input was an exponentially decaying cache over the whole word history.

In this section, the continuous cache [48] will be described in more detail. Followed by a cache using BoW that is different from the one introduced in previous research [71, 50]. The BoW cache that is introduced here uses higher-order M-Gram information in addition to only unigrams. A similar idea has been proposed for M-Gram by Goodman [46].

3.2.1 Continuous Neural Cache

A novel way to implement a continuous Neural Cache (NC) was presented in [48]. The cache was proposed as an extension to an LSTM-LM. It consists of a list of tuples $(\boldsymbol{h}_{\mathrm{c},l}, \boldsymbol{w}_{\mathrm{c},l})$, where $\boldsymbol{h}_{\mathrm{c},l}$ is the hidden state of the network at time l and $\boldsymbol{w}_{\mathrm{c},l}$ the true target label the network should predict at this time step. Items in the cache are marked with a subscript c to distinguish them from the elements in

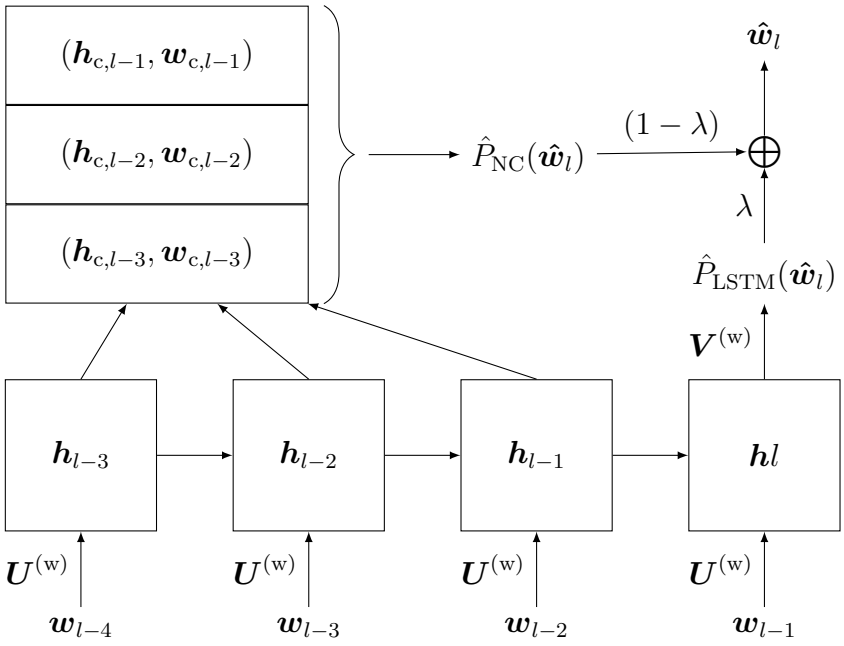


Figure 10: Continuous neural cache model as proposed by [48] visualised for three time steps.

the network. The input word vector $\boldsymbol{w}_{l-1}$ to the network is encoded as a one-hot vector and the input to the LSTM is obtained by multiplication with a linear layer $\boldsymbol{U}^{(\mathrm{w})}$. The cache and an LSTM-LM are shown with an unfolded structure over three time steps in Figure 10.

Denoting the elements in the cache by a subscript c as $(\boldsymbol{h}_{\mathrm{c},l}, \boldsymbol{w}_{\mathrm{c},l})$, the output of the cache for each possible element k in the output vector $\boldsymbol{\psi}_l$ is calculated from a cache of length N as

$$\boldsymbol{\psi}_l[k] = \sum_{n=1}^{N} \mathbb{1}_{\hat{\boldsymbol{w}}_l[k]=\boldsymbol{w}_{\mathrm{c},l-n}[k]} \mathrm{e}^{(\theta \boldsymbol{h}_l^{\mathrm{T}} \boldsymbol{h}_{\mathrm{c},l-n})}. \tag{90}$$

$\mathbb{1}_{a=b}$ is a function that evaluates to one if a is equal to b and is zero for all other cases. θ is a positive real valued parameter that can be chosen freely.

In addition, a cosine word similarity measure instead of this hard word identity can also be used.

$$\boldsymbol{\psi}_l[k] = \sum_{n=1}^{N} \left(\frac{\hat{\boldsymbol{w}}_l^{\mathrm{T}} \boldsymbol{w}_{\mathrm{c},l-n}}{||\hat{\boldsymbol{w}}_l|| ||\boldsymbol{w}_{\mathrm{c},l-n}||} \right)^2 \mathrm{e}^{(\theta \boldsymbol{h}_l^{\mathrm{T}} \boldsymbol{h}_{\mathrm{c},l-n})}$$

Using a similarity measure, also semantically similar words will be assigned some probability mass. Similar words can be expected to appear in a semantically similar context.

The exponent of the exponential calculates an inner product of two vectors. This value is high when the previous hidden state stored in the cache $\boldsymbol{h}_{\mathrm{c}}$ is similar to the current hidden state $\boldsymbol{h}_l$ of the network. By multiplying this similarity with $\mathbb{1}_{a=b}$ it is restricted only to the cases, where the previous target of the network matches the k-th word in the output. In other words, the cache can be seen as a similarity measure which tries to increase the probability of words with a similar history. To obtain a probability from the cache, we can sum over all elements in the output vector $\boldsymbol{\psi}_l$ of the cache and normalise all elements by this sum

$$\hat{P}_{\mathrm{NC}}(\hat{\boldsymbol{w}}_l) = \frac{1}{\sum_k \boldsymbol{\psi}_l[k]} \boldsymbol{\psi}_l. \tag{91}$$

As stated by the authors, an advantage of this cache architecture is that it is not required to train the LM and the cache jointly. The joint probability for $\hat{\boldsymbol{w}}_l$ is obtained by a linear interpolation of the output probability of the cache $\hat{P}_{\mathrm{NC}}(\hat{\boldsymbol{w}}_l)$ with the prediction of the network $\hat{P}_{\mathrm{LSTM}}(\hat{\boldsymbol{w}}_l)$

$$\hat{P}(w_l) = \lambda \hat{P}_{\mathrm{LSTM}}(\hat{\boldsymbol{w}}_l) + (1-\lambda)\hat{P}_{\mathrm{NC}}(\hat{\boldsymbol{w}}_l). \tag{92}$$

The cache itself does not influence the parameters of the network. Therefore, there is no increase in training time.

3.2.2 Bag-of-Words Cache

A simple form for a cache is BoW as it was used in previous research [71, 50]. A BoW can be thought of as a list that counts how often a word appears but without remembering when a word appeared. To obtain some time correspondence in the BoW cache, an exponentially decaying weight was multiplied with all words up to the current time $l-1$. The word appearing most recent had the highest weight and the word furthest in the past had the lowest weight. These weighted word vectors are summed up and input into the network.

The first approach introduced here is similar, but it does not keep a continuously updated BoW. In this BoW cache realisation a cache of N entries is used.

The output of the cache is calculated by assigning an exponentially decaying weight to each entry in the cache

$$\boldsymbol{c}_l^{(\mathrm{u})} = \sum_{n=0}^{N-1} \boldsymbol{w}_{\mathrm{cu},n} \mathrm{e}^{-n}, \tag{93}$$

where $\boldsymbol{w}_{\mathrm{cu},n}$ is the n-th vector in the cache. The first word in the cache has the highest weight and it decreases down to last word in the cache.

The cache itself can be thought of as an ordered list of the past N words. The words in the list are sorted according to their frequency of appearance. When a new word appears, it is prepended to the list. If the cache contains more than N words after prepending the new one, the last word in the list is removed from the cache. However, the list only contains each word once. If a word re-appears, it is taken from its current position and inserted at the head of the list. Organising the cache this way ensures that the most frequent words get assigned the highest weight. This type of cache is further on denoted the unigram cache because as it contains the information about the last most frequent N unigrams.

Since many high frequency words in a corpus, like 'the' or 'a', do not carry any important information these types of words are excluded from the cache. The frequency of all words in the vocabulary is estimated from the training data. Likewise, it is reasonable to assume that infrequent words in the training data are unlikely to appear often in the evaluation data. Therefore, infrequent words are also excluded from the cache.

In addition to the unigram cache, one can also build another BoW cache attached to each unigram for bigrams. Each word in the unigram cache has a list with the last M words that appeared after this word. For this bigram BoW cache the same ordering and insertion policy as for the unigrams is used. However, for the bigram cache there is no threshold for the word frequency. The calculation for the output of the bigram cache $\boldsymbol{c}_l^{(\mathrm{b})}$ is analogue to the calculation for the output of the unigram cache $\boldsymbol{c}_l^{(\mathrm{u})}$ in (93)

$$\boldsymbol{c}_l^{(\mathrm{b})} = \sum_{m=0}^{M-1} \boldsymbol{w}_{\mathrm{cb},m} \mathrm{e}^{-m}. \tag{94}$$

3.3 Text Topic Features

Texts of whatever sort can usually be grouped into larger segments that contain a similar subject. These larger segments can span from a few sentences up to several pages in a book. Depending on the subject that is addressed in such a segment, different words are more likely to appear in this segment. For example, when looking at a textbook about finances and comparing it with a textbook about signal processing, many different terms will appear depending on the book. Looking at each textbook in more particular, there will be chapters on different methods. In the book about signal processing from the example, there will be a chapter on Fourier transformation and there will also be a chapter on linear shift invariant systems. Both chapters will use different terms for some parts but the Fourier transformation is a basic method that will also be mentioned in the section on linear shift invariant systems. This short example shows that a very different vocabulary can be used depending on the subject but subjects can also share parts of the vocabulary.

Estimating such subjects is commonly known as topic models. In information retrieval, topic models are of interest to find related documents. An important scheme is Salton and McGill's term-frequency inverse-document-frequency [113]. Words which appear frequently in a document but only rarely in a set of documents are assumed to be characteristic terms for this document. An early probabilistic model to estimate topics is latent semantic indexing [28]. This was later extended to probabilistic latent semantic indexing [66]. However, the parameter size of these models grew with the number of documents. This issue was addressed in Latent Dirichlet Allocation [16] (LDA).

In addition to these probabilistic models, more recently NNs became a popular method for context representations. Convolutional NNs (CNN) were used as context representation in a sentence classification task by Kim et al. [81]. Denil et al. used CNNs for summarising documents [32]. CNNs were used for other NLP tasks as well [77]. A method for obtaining a document representation by a feed-forward NN was presented by Le et al. This method called paragraph vector was very successful and extensions of this method were subsequently proposed [20]. [84]. Lin et al. combined RNNs for word and sentence prediction in a hierarchichal RNN for document modelling [85]. Another popular and successful

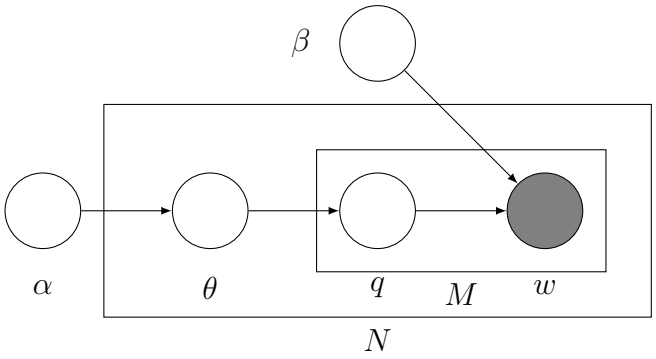


Figure 11: Graphical model representation of an LDA topic model.

context representation is the encoder-decoder framework [129]. Encoder-decoder models were very successful in machine translation [5] or in the generation of conversation responses [119] among other tasks.

The remainder of this section will briefly introduce the LDA topic model. Features calculated from an LDA topic model were used successfully for domain adaptation of LMs. In addition, a neural context representation based on a Sequence Summary Network [138] (SSN) will be described.

3.3.1 Latent Dirichlet Allocation

LDA as presented by Blei et al. [16] is a generative model for estimating topics in a collection of documents. It assumes that each document in this collection is modelled by a mixture of an underlying set of topics. These topics are modelled by a mixture over a set of topic probabilities. As a Bag-of-Words model, LDA ignores the word order in a document and provides a low-rank representation of the document, namely the number of topics. Figure 11 shos the graphical model for LDA.

LDA describes the following generative process to generate each document in a collection of N documents [16]

1. Sample the document length M from a Poisson distribution: $M \sim \text{Poisson}(\xi)$
2. Choose a multinomial distribution over topics for the document by sampling from a Dirichlet distribution parameterised by α: $\theta \sim \text{Dir}(\alpha)$

3. For each word w_m of the M words:

   - Choose a topic: $q_m \sim \text{Multinomial}(\theta)$
   - Choose a word w_m from the unigram distribution associated with the topic $p(w_m|q_m, \beta)$

LDA has the two key parameters α and β that have to be learned during model training. The first parameter α determines the shape of the Dirichlet distribution $\theta \sim \text{Dir}(\alpha)$ over the multinomial distribution Multinomial(θ) that is used for drawing each topic q_m. β directly influences the word probabilities $p(w_m|q_m, \beta)$.

In the context of LMs it is important to address the specification of a document for LDA. To train the LDA model, the training corpus has to be split into several documents. A case where this can be done easily is, for instance, when the corpus consists of individual talks which are appended to form the whole training set. If the training data should consist only of a single training file, [94] suggested to regard a chunk of several sentences as a document. However, in some cases this violates our assumption that a unit of text, which the topic distribution is calculated for, consists of a consistent topic. To generate features from the LDA topic model that can be used with the LM, [94] suggested to calculate the topic distribution for a window of past words.

An LDA topic model neglects word order and all words in a document will contribute equally when calculating a topic distribution. When combining features from an LDA topic model with an LSTM-LM, the LSTM also captures some kind of context information. The cell state $\boldsymbol{c}_l$ can be related to the context an LSTM remembers. However, this state is processed by an exponentially decaying function. That means, words further in the past have less influence on the current output than more recent words.

3.3.2 Neural Network Context Representation

The NN context representation used in this thesis uses a sequencs summary network [138] (SSN). The SSN is a feed forward NN with one or more linear layers, each followed by a non-linearity. In this thesis, rectified Linear Units (reLU) is used as non-linearity. Figure 12 shows an SSN.

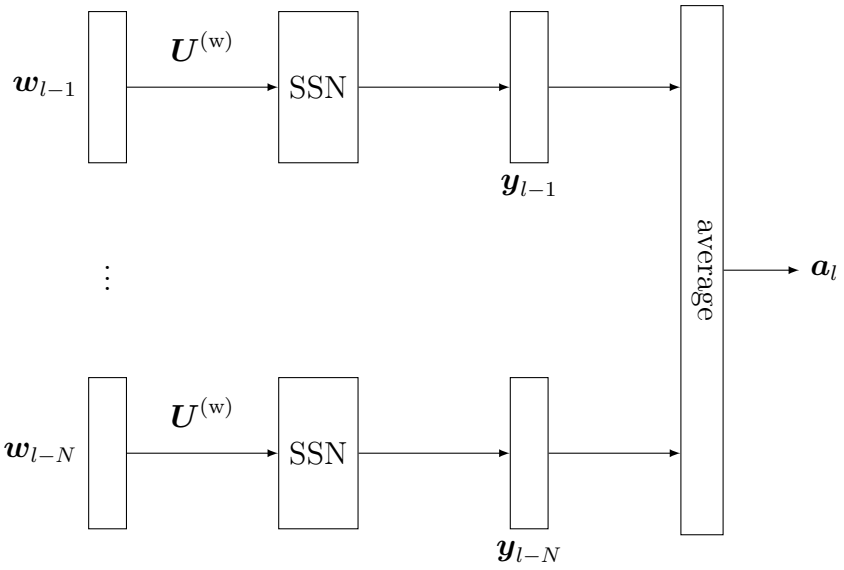


Figure 12: A sequence summary network (SSN) that calculates a summary vector from its inputs $\boldsymbol{w}_{l-1,\ldots,l-N}$.

When applied to word sequences, the input to the SSN is a context window of N words $[\boldsymbol{w}_{l-1}, \boldsymbol{w}_{l-2}, \ldots, \boldsymbol{w}_{l-N}]$. Each element in this word sequence is multiplied with a word embedding matrix $\boldsymbol{U}^{(\mathrm{w})}$ and the word embeddings $[\boldsymbol{x}_{l-1}, \boldsymbol{x}_{l-2}, \ldots, \boldsymbol{x}_{l-N}]$ are input into the SSN

$$[\boldsymbol{x}_{l-1}, \boldsymbol{x}_{l-2}, \ldots, \boldsymbol{x}_{l-N}] = [\boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1}, \boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-2}, \ldots, \boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-N}]. \tag{95}$$

The context window covers the current word w_{l-1} and the previous $N-1$ words. For each input, the SSN computes an output $[\boldsymbol{y}_{l-1}, \boldsymbol{y}_{l-2}, \ldots, \boldsymbol{y}_{l-N}]$

$$\boldsymbol{y}_{l-n} = \mathrm{SSN}(\boldsymbol{x}_{l-n}), \quad \forall n \in \{1, 2, \ldots, N\}. \tag{96}$$

All outputs are averaged over the whole context window to obtain a fixed-size vector representation of the input word sequence

$$\boldsymbol{a}_l = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{y}_{l-n}. \tag{97}$$

This context representation $\boldsymbol{a}_l$ can be used as adaptation feature for the LM.

4. Neural Network Architectures for Context Adaptation

The previous section introduced different kinds of context information which can be useful for language modelling. This section now deals with the question of how such information can be incorporated into an LM. Different ways for exploiting the context information in NNLMs will be introduced. The context information will mainly be provided as an auxiliary feature. Different ways on how this feature can be input into the NNLM and effectively improve word prediction will be described. As basic NNLM architecture, an RNN-LM is used in this thesis. As recurrent unit, mainly LSTM will be used. This section introduces different ways how a common LSTM-LM can be modified to include the context information.

4.1 Input Enhancement

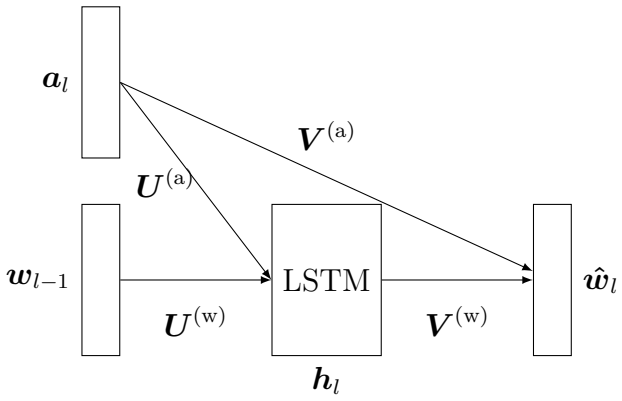


Figure 13: LSTM-LM feature-based adaptation with Context dependent LSTM-LM (contLSTM).

A simple way to use the context information is to extend the input of the network to include an auxiliary feature $\boldsymbol{a}_l$. Figure 13 shows this extension as originally proposed for RNN-LMs [94, 23] and also used with LSTM-LMs [120]. The input $\boldsymbol{x}_l$ can be written as

$$\boldsymbol{x}_l = \boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1} + \boldsymbol{U}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{U,a})}, \tag{98}$$

where $\boldsymbol{U}^{(\mathrm{a})}$ and $\boldsymbol{b}^{(\mathrm{U,a})}$ are the weight matrix and bias vector for the adaptation features, respectively. This model will be referred to as context dependent LSTM-LM (contLSTM) in the following. In addition to using the feature on the input of the network, the adaptation feature can also be connected with the output layer directly.

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\left(\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l + \boldsymbol{b}^{(\mathrm{V,w})} + \boldsymbol{V}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{V,a})}\right), \tag{99}$$

where $\boldsymbol{V}^{(\mathrm{a})}$ and $\boldsymbol{b}^{(\mathrm{V,a})}$ are the weight matrix and bias vector of the linear layer connecting the adaptation feature vector to the output. Introducing this connection is motivated by maximum-entropy LMs.

For RNN-LMs, this adaptation scheme achieved a significant PPL reduction compared with a vanilla RNN-LM when LDA features [94] or prosodic features [40] were used as auxiliary features. The PPL reduction with LDA features can be explained to some extent by the vanishing gradient problem that RNNs suffer from [13]. The long context information provided by the adaptation features can circumvent this problem. However, in combination with LSTM-LMs the relative PPL reduction achieved by this method was not as large as with vanilla RNN-LMs [120].

4.2 Cache and Memory Augmentation

4.2.1 Connected Neural Cache

In Chapter 3.2.1, a continuous NC was introduced. The output of the cache was interpolated with the output from the LM to obtain a probability estimate for the combined model in (92). Here, a way to connect the NC directly with the NN is presented. Figure 14 shows the combination of the continuous NC and the LSTM-LM. The cache is connected at two points. A linear layer with transition matrix $\boldsymbol{U}^{(\mathrm{nc})}$ connecting the output from the cache $\boldsymbol{h}^{(\mathrm{nc})}$ to the input of the LSTM is added. The input to the LSTM can thus be formulated as

$$\boldsymbol{x}_l = \boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1} + \boldsymbol{U}^{(\mathrm{nc})}\boldsymbol{h}_l^{(\mathrm{nc})}. \tag{100}$$

The cache is also directly connected to the softmax by a linear layer $\boldsymbol{V}^{(\mathrm{nc})}$

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\left(\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l + \boldsymbol{V}^{(\mathrm{nc})}\boldsymbol{h}_l^{(\mathrm{nc})}\right). \tag{101}$$

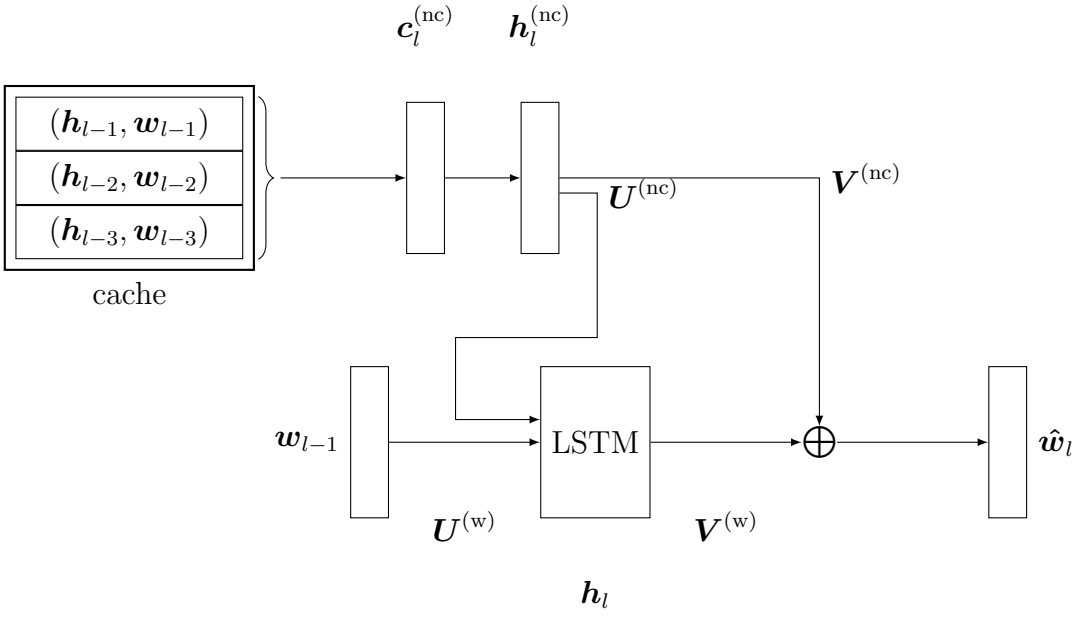


Figure 14: Continuous connected neural cache with an LSTM-LM.

The continuous NC in Chapter 3.2.1 uses only a single interpolation factor for all words. By connecting the NC to the network with linear layers, the network can learn interpolation weights for each word through error backpropagation. During training, the parameters of the network can be jointly optimised with the cache. This should enable the network to learn better interpolation weights for each individual word.

Calculating the output of the cache can be computationally expensive. Therefore, the cache is combined with a pre-trained LM and only the weights in the linear layers for the cache $\boldsymbol{U}^{(\mathrm{nc})}$ and $\boldsymbol{V}^{(\mathrm{nc})}$ are trained. The parameters of the base model are left unchanged except for the linear layer before the softmax $\boldsymbol{V}^{(\mathrm{w})}$.

4.2.2 Bag-of-Words Cache

Chapter 3.2.2 introduced a cache that was a list of the last N most frequent unigrams. For each unigram, it could hold the last M most frequent bigrams. This cache can be connected via linear layers to an LSTM-LM in the same manner

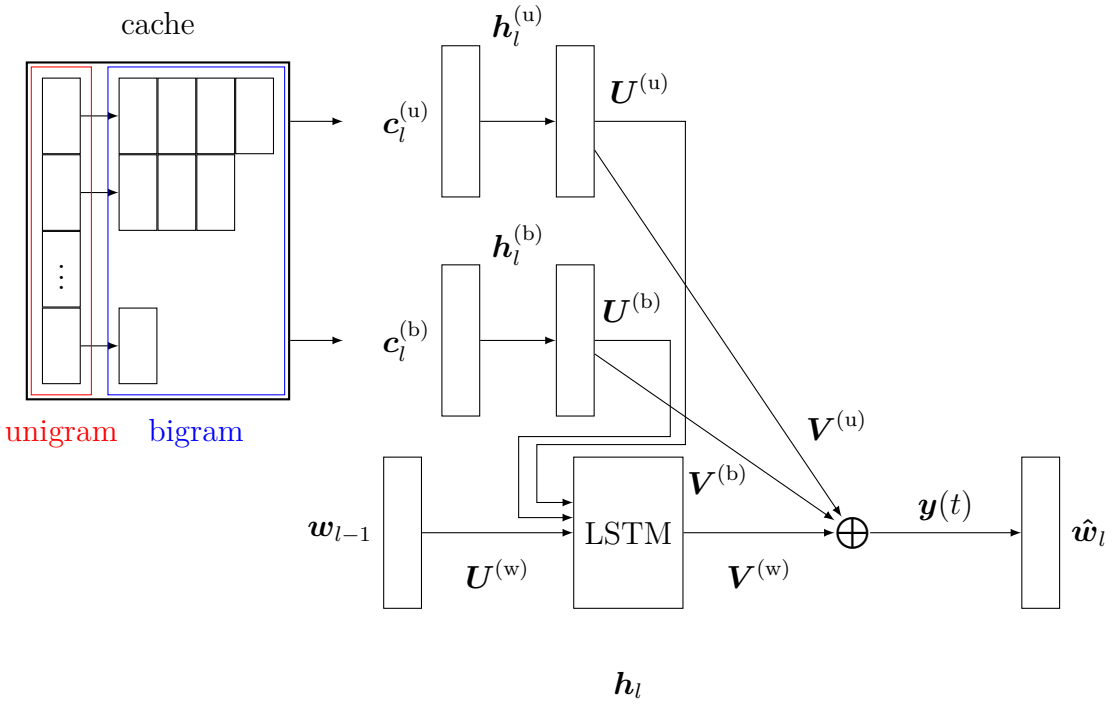


Figure 15: BoW cache extension for an LSTM-LM.

as the continuous NC in Chapter 4.2.1.

This connection is shown in Figure 15. The unigram cache is highlighted in red and the bigram cache is highlighted in blue in Figure 15. The output of both caches is a vector of the vocabulary size. These vectors are compressed to the same size as the number of LSTM units by a linear layer with a subsequent non-linearity. Here, reLU was used as non-linearity. This dimensionality reduction is used to reduce the computational complexity. The number of hidden units in the LSTM is usually significantly smaller than the number of words in the vocabulary $\mathcal{U}$. With $\boldsymbol{h}_l^{(\mathrm{u})}$ and $\boldsymbol{h}_l^{(\mathrm{b})}$ denoting the outputs of the hidden layer for the unigram and bigram cache, respectively, the input to the LSTM can then be written as

$$\boldsymbol{h}_l = \boldsymbol{U}^{(\mathrm{w})}\boldsymbol{w}_{l-1} + \boldsymbol{U}^{(\mathrm{u})}\boldsymbol{h}_l^{(\mathrm{u})} + \boldsymbol{U}^{(\mathrm{b})}\boldsymbol{h}_l^{(\mathrm{b})}, \tag{102}$$

where $\boldsymbol{U}^{(\mathrm{u})}$ and $\boldsymbol{U}^{(\mathrm{b})}$ are a linear layer connecting the vectors to the LSTM.

Both outputs of the hidden layer $\boldsymbol{h}_l^{(\mathrm{u})}$ and $\boldsymbol{h}_l^{(\mathrm{b})}$ are also added to the output of the LSTM before the softmax. A linear transformation is applied on both vectors and the input to the softmax can be written as

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\left(\boldsymbol{V}\boldsymbol{h}_l + \boldsymbol{V}^{(\mathrm{u})}\boldsymbol{h}_l^{(\mathrm{u})} + \boldsymbol{V}^{(\mathrm{b})}\boldsymbol{h}_l^{(\mathrm{b})}\right). \tag{103}$$

The direct connection of the cache output to the network output is inspired by several previous research. [50] proposed using a direct connection when using an RNN-LM. To a further extend, it is also related to the idea of highway networks [121] and residual connections [53], which have proven useful when training deeper networks.

4.3 Domain Adaptation Architectures

This section introduces different network structures for domain adaptation of LMs. Usually, LMs are trained on general-domain text data covering a large variety of domains. However, domain specific language models achieve lower PPL and WER in ASR compared with general-domain LMs. Therefore, the adaptation of general-domain LMs to specific topics has been an ongoing research interest. For M-Gram LMs, a comprehensive overview of adaptation techniques was provided in [110] and [8]. M-Gram adaptation with topic model information

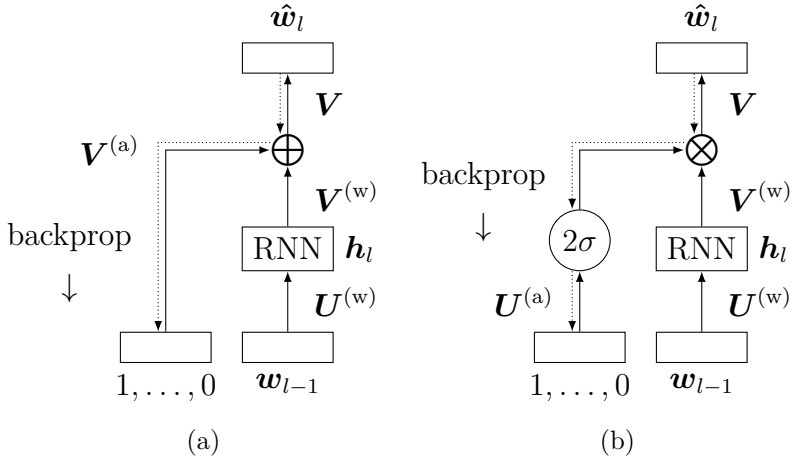


Figure 16: Examples for model-based domain adaptation using (a) a linear hidden network and (b) learning hidden unit contributions. The inputs to the adaptation layers are a one-hot vector encoding the domain information and the weights for this one-hot vector are learned in re-training by error backpropagation.

such as LDA features was presented in [132, 54, 86, 142, 51]. There were also domain-adaptation approaches for other LM types, such as maximum entropy LMs [109, 14, 31, 80, 2].

Commonly, two paradigms for domain adaptation of NNLMs are distinguished, that is, model-based and feature-based domain adaptation. In model-based adaptation, the parameters in the network are adapted with in-domain data in a two-step process. First, a general LM is trained, where often an adaptation layer is inserted into the network. In the second step, the weights in this adaptation layer are updated using in-domain data. Model-based adaptation has been used for feed-forward NNLMs [105, 1] and RNN-LMs [133]. Recently, model-based adaptation with a Linear Hidden Network [42, 7] (LHN) was proposed. Figure 16 (a) shows model-based adaptation with an LHN. An LHN adds a linear hidden layer in the network without a subsequent non-linearity. As input to this linear layer, the authors used the output of the RNN and a one-hot label encoding the domain. The weights in this linear layer are learned during re-training.

A slightly different approach for model-based adaptation uses a gating mechanism. In AM adaptation, a method called Learning Hidden Unit Contributions

[130, 114] (LHUC) has been previously proposed. In this case, the speaker adaptation data is used to apply a gating mechanism to the hidden units of an NN-based AM. In [41], the application of LHUC to RNN-LMs was investigated. This model is shown in Figure 16 (b). The authors applied the concept of LHUC to the output of the hidden layer and the adaptation weights were learned from in-domain data. The authors showed improvements for PPL and N-best rescoring.

However, with little adaptation data, model-based adaptation can be a problem because the adapted models are prone to overfitting [41]. In addition, model-based adaptation requires domain labels throughout the whole corpus. Creating this annotation is expensive.

Compared with model-based domain adaptation, feature-based domain adaptation has the advantage that it does not require domain labels in the corpus. In feature-based adaptation, an adaptation feature such as topic information from LDA [16] is used. The LM is trained with the adaptation feature to adapt its activations in the network to the topic information. Usually, the input of the network is extended by additional domain specific features, such as the network structure presented in Chapter 4.1. Many methods have been proposed, where these features act as a domain dependent bias. For RNN-LMs, the first proposed approach was a context dependent RNN-LM [94], which used LDA features to allow the network to exploit information from a context window of the current word. This method has also shown to be successful for RNN-LMs [23] on multi-domain broadcast data in the MGB Challenge [7]. The authors showed that feature-based domain adaptation outperforms model-based adaptation in the context of the MGB Challenge. Domain adaptation has mainly been proposed using vanilla RNN-LMs. Soutner et al. [120] proposed the domain adaptation of an LSTM-LM, using the same adaptation mechanism as for context dependent RNN-LMs [94]. A feature-based adaptation mechanism using a gating on the word vectors was proposed in [148]. The authors combined information from in-domain and general-domain word vectors.

In the remainder of this section, three different approaches for feature-based domain adaptation of NNLMs are presented. The first one is a feature-based realisation of an LHN (fLHN). The second one is feature-based LHUC (fLHUC). The third one is factorised hidden layers. RNN-LMs have been the main target

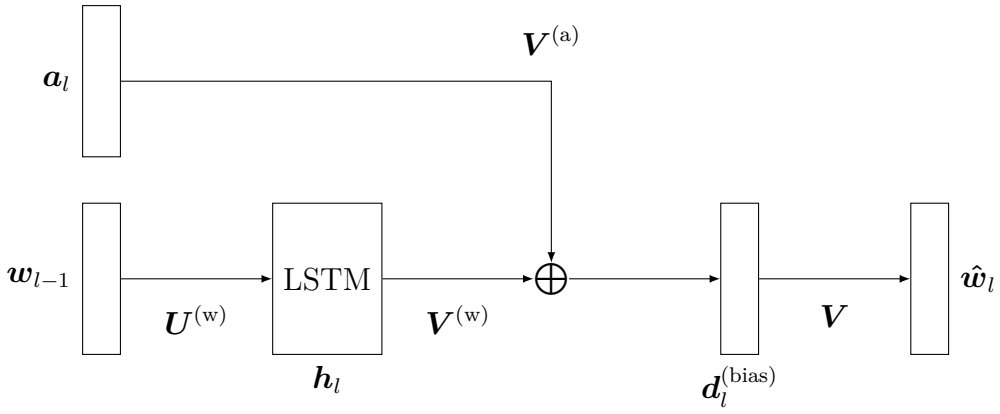


Figure 17: LSTM-LM feature-based model adaptation with Linear Hidden Network (fLHN-LSTM).

of domain adaptation techniques presented in the literature, but LSTM-LMs are currently the state-of-the-art. For this reason, LSTM will be used as recurrent unit in all NNLMs. In addition, all compared adaptation methods work in an unsupervised manner, that means, these methods do not require any domain label in the training data. This setting is more relevant in practice because LMs can be trained on large text corpora of many million words. It would be costly and time intensive to annotate such large corpora by humans.

4.3.1 Feature-based Linear Hidden Network

Recently, domain adaptation with a linear hidden network (LHN) was proposed as model-based adaptation method for vanilla RNN-LMs [27]. Since the focus of this thesis is on feature-based adaptation methods, an LHN adaptation layer is used with LDA topic features instead of a domain label as input. To distinguish the feature-based approach from the model-based approach, this is denoted by fLHN-LSTM. Figure 17 shows the network structure for an fLHN-LSTM. The LHN introduces an additional linear layer between the LSTM and the output layer. Since the output $\boldsymbol{d}_l^{(\mathrm{bias})}$ of this intermediate linear layer is not followed by a non-linearity, it is called a linear hidden network.

The LHN takes two inputs. First, the LDA features $\boldsymbol{a}_l$ that are transformed by a linear layer with weight matrix $\boldsymbol{V}^{(\mathrm{a})}$ and bias vector $\boldsymbol{b}^{(\mathrm{V,a})}$. Second, the

output of the LSTM $\boldsymbol{h}_l$ that is also transformed by a linear layer with weight matrix $\boldsymbol{V}^{(\mathrm{w})}$ and bias $\boldsymbol{b}^{(\mathrm{V,w})}$

$$\boldsymbol{d}_l^{(\mathrm{bias})} = \boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l + \boldsymbol{b}^{(\mathrm{V,w})} + \boldsymbol{V}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{V,a})}. \tag{104}$$

From the above equation, it becomes visible that the LHN introduces a topic dependent bias term ($\boldsymbol{V}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{V,a})}$). After the LHN follows a linear layer $\boldsymbol{V}$ and the softmax function to calculate the probability for the current word $\hat{\boldsymbol{w}}_l$

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\left(\boldsymbol{V}\boldsymbol{d}_l^{(\mathrm{bias})} + \boldsymbol{b}^{(\mathrm{V})}\right). \tag{105}$$

The LDA features are input to the LHN during network training and evaluation.

4.3.2 Feature-based Learning Hidden Unit Contributions

LHUC was first introduced for AM adaptation in ASR. It was also applied as model-based adaptation method to vanilla RNN-LMs [41] and it showed to reduce PPL and WER compared with a vanilla RNN-LM. Here, LHUC is introduced as a feature-based adaptation method. The adaptation weights used in LHUC are calculated from auxiliary features in a similar way to what has been proposed as subspace LHUC [114]. For NNLM domain adaptation, LHUC is used in a similar scheme to fLHN-LSTM, as shown in Figure 18. Feature-based LM domain adaptation with LHUC will be in the following denoted by fLHUC-LSTM.

The adaptation parameters in fLHUC are calculated from the LDA features. The features are multiplied with a linear layer, followed by a sigmoid non-linearity and an amplification by two

$$\boldsymbol{h}_l^{(\mathrm{a})} = 2\sigma(\boldsymbol{U}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{U,a})}), \tag{106}$$

where $\boldsymbol{U}^{(\mathrm{a})}$ and $\boldsymbol{b}^{(\mathrm{U,a})}$ are the weight matrix and bias vector of the linear layer connecting the LDA feature vector. The amplification of the sigmoid function is introduced to compensate for the reduced amplitude of some nodes. The multiplication with the sigmoid non-linearity sets some of the outputs of the LSTM to zero. To keep the amplitude of the activations in the output layer on approximately the same level, in [130, 114] it was suggested to multiply the output of the sigmoid non-linearity by two. The adaptation parameters $\boldsymbol{h}_l^{(\mathrm{a})}$ are used as a

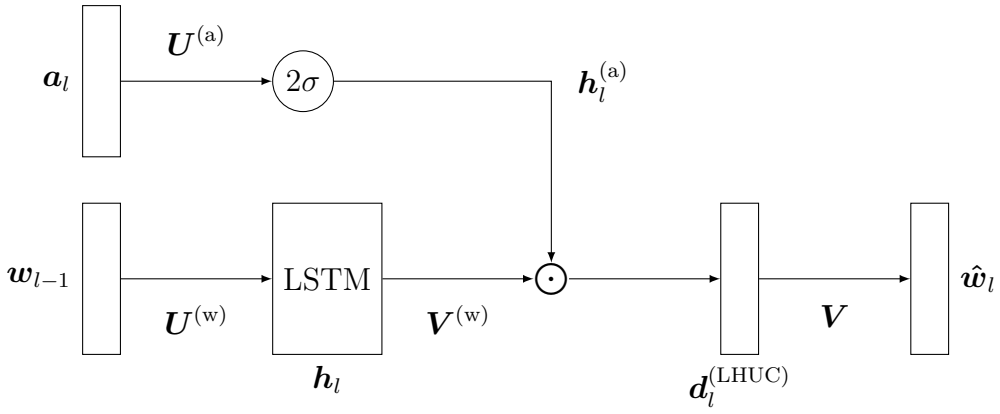


Figure 18: LSTM-LM feature-based domain adaptation with Learning Hidden Unit Contributions (LHUC, fLHUC-LSTM).

gate for the output of the LSTM

$$\boldsymbol{d}_l^{(\mathrm{LHUC})} = (\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l + \boldsymbol{b}^{(\mathrm{V,w})}) \odot \boldsymbol{h}_l^{(\mathrm{a})}, \tag{107}$$

where $\odot$ denotes an element-wise multiplication of two vectors. The adaptation parameters $\boldsymbol{h}_l^{(\mathrm{a})}$ have values between $[0, 2]$ and it can be interpreted as a context dependent gating of the units in the adaptation layer. The output layer and the softmax function follow after the output of the adaptation layer

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\,(\boldsymbol{V}\boldsymbol{d}_l^{(\mathrm{LHUC})} + \boldsymbol{b}^{(\mathrm{V})}). \tag{108}$$

The paradigm mainly used for NNLM domain adaptation in the literature is bias-based adaptation. The model shown in Figure 19 combines bias-based adaptation and context dependent gating with fLHUC. The model is a combination of the models described in Chapter 4.3.1 and this section. The model uses an addition of both adaptation layers before output layer

$$\boldsymbol{d}_l^{(\mathrm{LHUC})} = (\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l + \boldsymbol{b}^{(\mathrm{V,w})}) \odot \boldsymbol{h}_l^{(\mathrm{a})}, \tag{109}$$

$$\boldsymbol{d}_l^{(\mathrm{bias})} = \boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l + \boldsymbol{b}^{(\mathrm{V,w})} + \boldsymbol{V}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{V,a})}, \tag{110}$$

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\,(\boldsymbol{V}(\boldsymbol{d}_l^{(\mathrm{LHUC})} + \boldsymbol{d}_l^{(\mathrm{bias})}) + \boldsymbol{b}^{(\mathrm{V})}). \tag{111}$$

Bias adaptation has been shown to be effective in model-based and feature-based domain adaptation methods and with fLHUCB we can also make use of this adaptation mechanism. As a result, by using fLHUCB we can combine the advantages of both bias adaptation and fLHUC-LSTM in a single model.

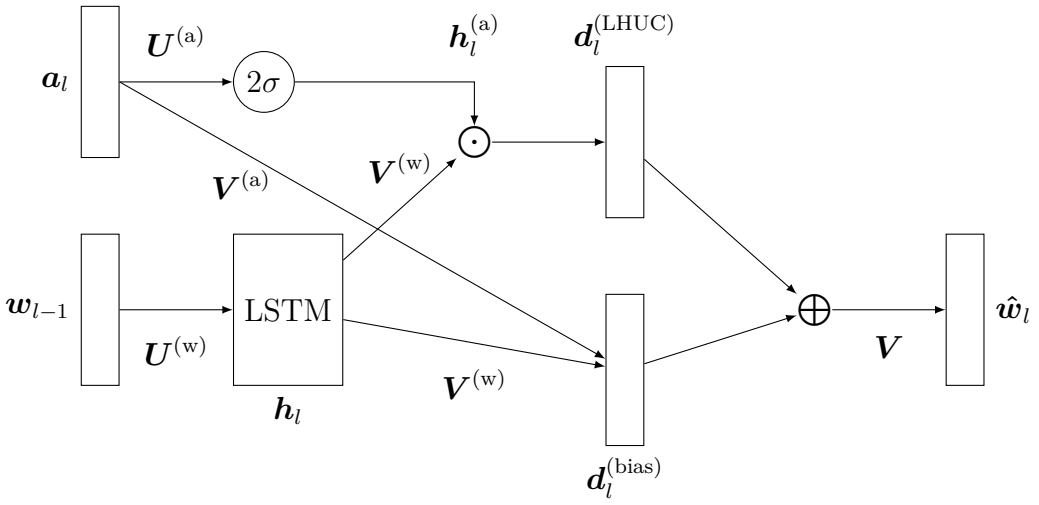


Figure 19: LSTM-LM domain adaptation with fLHUC and bias adaptation (fLHUCB).

4.3.3 Feature-based Hidden Layer Factorisation

The feature-based domain adaptation methods introduced so far usually add a bias to the network input or output. This bias depends on the topic feature or a (given) domain label. The factorised hidden layer-based domain adaptation that is introduced here is fundamentally different from other approaches. The output layer is factorised into multiple layers so-called factors, where each factor is weighted by a factor weight. This can also be seen as a linear combination of different subspaces.

It is also similar to a linear combination of multiple domain dependent LMs that share a common hidden state. In comparison with a simpler model combination, hidden layer factorisation can use a much simplified training scheme. To train a model combination or mixture of experts, the following training process is required:

1. Train baseline general-domain LSTM-LM on the whole training set
2. Train an LDA topic model
3. Calculate topic distribution for each document (talk, lecture, conversation etc.) in training, validation and test set

4. Cluster all documents according to the topic distribution

5. Create one LM for each cluster by re-training the baseline LSTM-LM on each document cluster

6. Optimise interpolation weights of all cluster-LSTM-LMs on the validation set

For the re-training (step (5)), the user has to decide if all weight matrices in the LM should be retrained or only certain weight matrices. Hidden layer factorisation greatly simplifies this training process, because (1) no document clustering is required, (2) no re-training of a baseline model is required, and (3) the optimisation of interpolation weights is done automatically during training. The interpolation weights are the factor weights that are calculated from an auxiliary network. This auxiliary network is trained jointly with the main network by standard error backpropagation. The input of this auxiliary network is a vector of LDA features. Factorised hidden layer adaptation has been successfully applied to acoustic model adaptation [29] and speaker aware beamforming [149].

Figure 20 shows an LSTM-LM with domain adaptation using factorised hidden layers. This network architecture is hereafter denoted by factLSTM. In factLSTM, the output of the LSTM is used as an input to N linear layers with corresponding weight matrix $\boldsymbol{L}_n^{(\mathrm{w})}$ and bias $\boldsymbol{b}_n^{(\mathrm{L,w})}$. These layers are called factors. The size of each factor is the number of hidden units times the vocabulary size. That means, each factor has the same size as the output layer. Each factor is weighted by a factor weight γ_n

$$\boldsymbol{z}_l = \sum_{n=1}^{N} \underbrace{\gamma_n (\boldsymbol{L}_n^{(\mathrm{w})} \boldsymbol{h}_l + \boldsymbol{b}_n^{(\mathrm{L,w})})}_{=\boldsymbol{z}_n} . \tag{112}$$

The sum of all factors is used as input to the softmax function

$$\hat{\boldsymbol{w}}_l = \mathrm{softmax}\left(\boldsymbol{z}_l\right). \tag{113}$$

As with the fLHN-LSTM and fLHUC-LSTM, there is no non-linearity after the factors. There is only a multiplication with a factor weight before calculating the probability for the current word $\hat{\boldsymbol{w}}_l$.

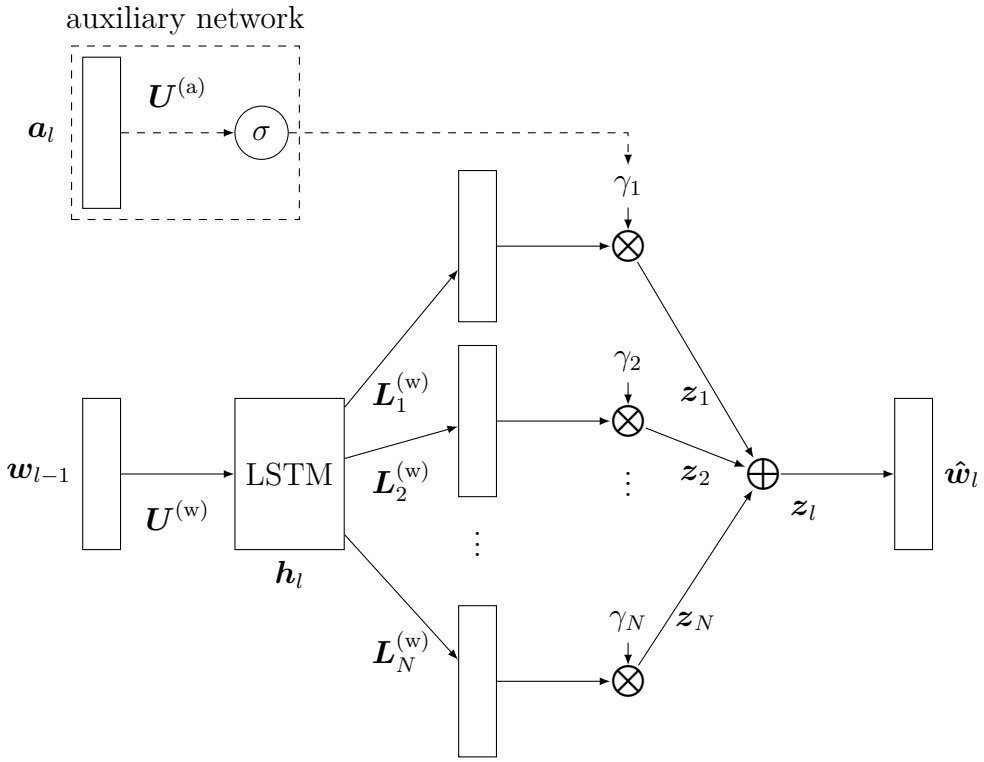


Figure 20: LSTM-LM feature-based model adaptation with factorised hidden layers (factLSTM).

To calculate each factor weight γ_n, an auxiliary network is used. The input to the auxiliary network are the topic features calculated from an LDA topic model. This auxiliary network can be of arbitrary depth. Here, a feed-forward network with a single hidden layer followed by a sigmoid non-linearity is used

$$\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \dots \gamma_n, \dots, \gamma_N]^{\mathrm{T}} = \sigma(\boldsymbol{U}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{U,a})}), \tag{114}$$

where $\boldsymbol{U}^{(\mathrm{a})}$ and $\boldsymbol{b}^{(\mathrm{U,a})}$ are the weight matrix and bias vector for the linear layer. The parameters of the auxiliary network can be trained jointly with the main network by error backpropagation, as shown in [30]. This means the auxiliary network and the main network do not have to be trained in separate training steps.

The two architectures factLSTM and fLHUC-LSTMare related to each other. In fLHUC-LSTM, one weight is multiplied with one node, whereas in factLSTM one weight is multiplied with one hidden layer. Comparing both methods,

fLHUC-LSTM has the advantage that it needs less parameters, because it does not require multiple output layers. However, factLSTM has the advantage that individual factors can cover more domain specific information while others can cover general-domain information.

4.4 Unified Framework for Context Extraction and Feature-based Adaptation

The training of LMs with feature-based domain adaptation requires several steps. First, an LDA topic model has to be trained independently from the LM as outlined in Chapter 3.3.1. This requires text pre-processing and the segmentation of the training data into documents. However, the training data might not always have this segmentation. Second, training the LDA topic model follows a very different scheme from extracting the features. The LDA topic model is trained on a set of documents, whereas the features are calculated from a sliding window over the input text. In addition, the LDA features are not optimised for LM adaptation.

Facing this scenario, it is desirable to have a more integrated approach for feature-based domain adaptation. This is the motivation for UniFA, a Unified framework for Feature-based domain Adaptation. UniFA is a combined approach for training the context representation and the LM jointly in a single training step. It does not require any text pre-processing and the training data can be used in the same form for training the LM and the context representation. The model learns to extract the context features itself to improve word prediction.

To obtain a context representation in this framework, an SSN as introduced in Chapter 3.3.2 is used. The SSN learns to extract a context representation from a fixed-size window of past words. This context representation is used in an adaptation layer to calculate the adaptation parameters. An adaptation layer in inserted in the LM before the output layer. In UniFA, this adaptation layer is realised by fLHUC as introduced in Chapter 4.3.2. In comparison with conventional LDA feature-based adaptation, the advantage of using such approach is that the context representation and the LM can be trained jointly by standard error backpropagation in a single training step. In contrast to an LDA topic

model, the approach does not require any pre-processing of the text data. The SSN and the LM can both be trained using the same data.

The adaptation framework UniFA as shown in Figure 21 consists of two main parts:

1. A context extractor network, that learns a fixed-length context representation from a window of past words.

2. An adaptation layer, where the feature extractor network's output is used to adapt the LSTM cells' output.

For the context representation, a context extractor network based on an SSN shown in Figure 21 (a) is used. SSNs were developed for speaker adaptation of acoustic models but they are also suitable for summarising context of word vectors. There are usually other more common methods for context representations in natural language processing, but the SSN is computationally very efficient. Because it consists of a (shallow) feed-forward network, the output can easily be computed in parallel for the whole context window on a GPU.

More common methods for context representations in natural language processing include convolutional neural networks [81, 77], or vector based representations like paragraph vector [84] and derivatives thereof [20]. Paragraph vector, despite being a successful method, is not suited for UniFA because it requires training a document matrix for each document. A row in this matrix serves as representation for each document and the matrix has to be extended for unknown documents in the evaluation set. Another popular and successful context representation is the encoder-decoder framework [129], which showed to be very successful in machine translation [5] or in the generation of conversation responses [119], among other tasks. However for this application, such encoder architecture would be computationally very expensive because the LSTM-based encoder has to be run over a long (possibly a few hundred words) context window at each word prediction.

A recent approach for LM domain adaptation which is related to UniFA was presented by Irie et al [70]. It uses a mixture of pre-trained LSTM-LMs where the weight for each LSTM-LM is determined by a mixer network. This mixer network is represented by another LSTM. However, there are several differences

with UniFA. First, the method introduced by Irie et al. is not a pure feature-based domain adaptation method because it requires domain information during pre-training. Second, it is a multi-step training process whereas UniFA is a single-step training process.

The SSN was already described in more detail in Chapter 3.3.2. Therefore, a detailed description is omitted here. For the application in UniFA, it is important to note that an SSN can be trained jointly with the network it is attached to by error backpropagation. In [138], this was shown for speaker adaptation of an AM. That means, it fulfils the requirement that the context representation is jointly trainable with the main network. The context representation $\boldsymbol{a}_l$ that is the output of the SSN is used as adaptation feature for the LM.

The context adaptation layer in UniFA is realised by fLHUC. As described in Chapter 4.3.2, the adaptation layer is inserted after the LSTM before the output layer. The context representation $\boldsymbol{a}_l$ calculated by the context extractor network is used as adaptation feature for the fLHUC as follows

$$\boldsymbol{h}_l^{(\mathrm{a})} = 2\sigma(\boldsymbol{U}^{(\mathrm{a})}\boldsymbol{a}_l + \boldsymbol{b}^{(\mathrm{U,a})}). \tag{115}$$

$\boldsymbol{U}^{(\mathrm{a})}$ and $\boldsymbol{b}^{(\mathrm{U,a})}$ are the weight matrix and bias vector of the linear layer for the output of the SSN. This linear layer is necessary to match the output dimensionality of the context extractor network and the adaptation layer. As described in Chapter 4.3.2, fLHUC applies a gating function to the output of the LSTM using an adaptation parameter

$$\boldsymbol{d}_l^{(\mathrm{LHUC})} = (\boldsymbol{V}^{(\mathrm{w})}\boldsymbol{h}_l + \boldsymbol{b}^{(\mathrm{V,w})}) \odot \boldsymbol{h}_l^{(\mathrm{a})}. \tag{116}$$

When combining the output of the SSN with the adaptation layer, it was very helpful to use a normalisation of the context features. In UniFA, layer normalisation [4] is applied to the input of the sigmoid function. This normalisation is necessary to reduce the output of the SSN to a value range that works well with the input range of the sigmoid function. The sigmoid function has a range from $[-4, 4]$ where backpropagation works well. Using a sigmoid non-linearity in the SSN could be a way to remove the normalisation function but experiments showed that the gradients are not backpropagated properly in this case and the SSN does not learn a context representation.

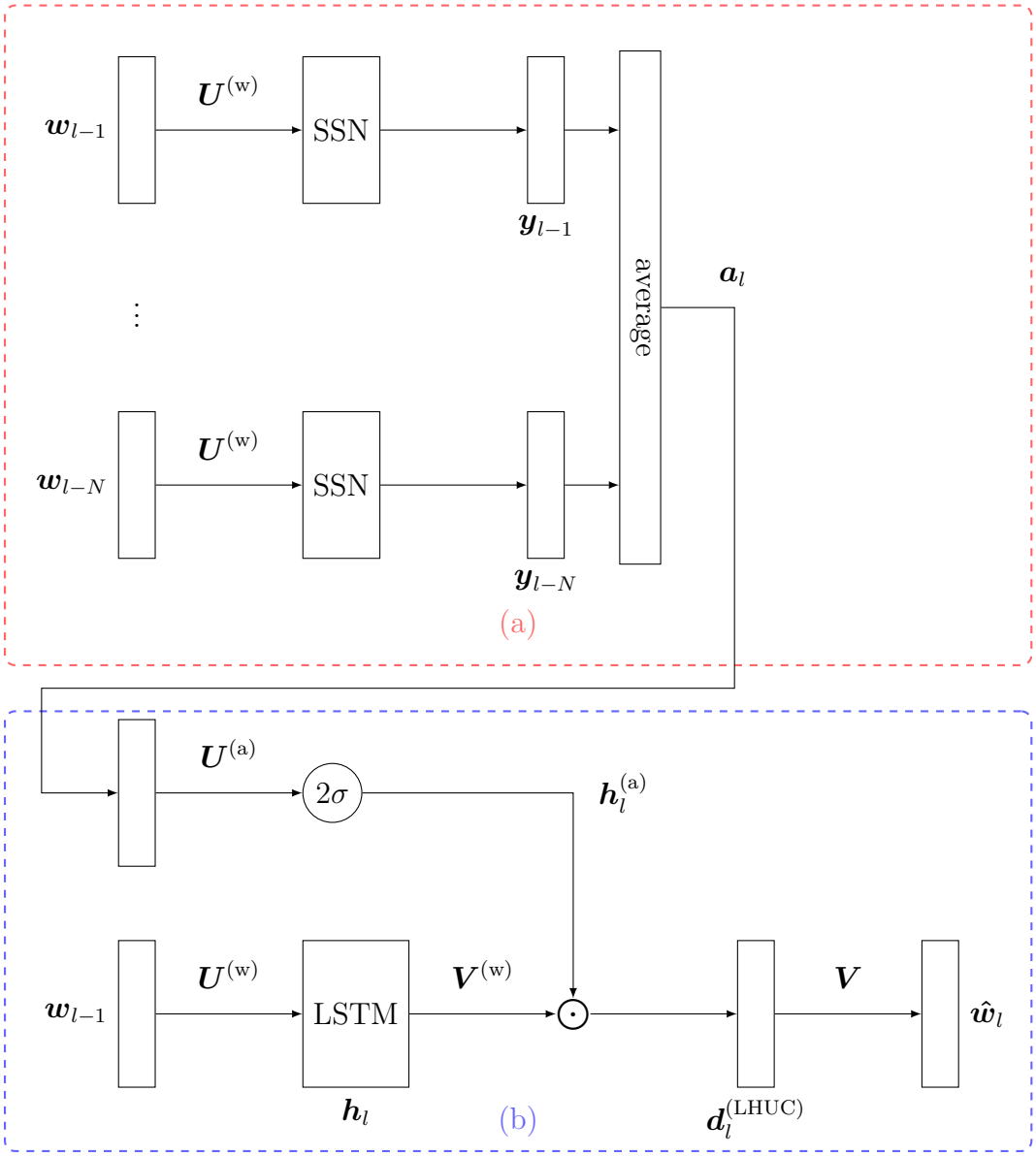


Figure 21: UniFA adaptation framework with (a) the sequence summary network (SSN) based context extractor network, and (b) LSTM-LM domain adaptation with fLHUC.

5. Experiments and Discussion

This section covers the experimental results for the methods introduced within this thesis. First, an overview of the different datasets that were used for conducting the experiments will be given. Second, the general experimental settings will be explained. This covers the toolkits used for implementation of the models as well as the toolkits used for the speech recognition experiments. Following these explanations will be the experimental results and discussion for different models. For each set of experiments, the detailed experimental settings and model training parameters are outlined before the analysis of the results.

5.1 Datasets

The experiments were conducted on four different data sets. These data sets consisted of written language and spontaneous speech corpora. The written language corpora comprised news articles and the spontaneous speech corpora were mostly lecture corpora. Here more details on each data set is given.

Table 1: Overview of the Penn Treebank data set.

Set	Sentences	Words	Vocabulary/OOV
Training	42068	887521	10000
Validation	3370	70390	—
Test	3761	78669	—

Table 1 shows the details for the Penn Treebank [88] (PTB) data set. It is a small data set with under 1M words in the training set. The whole data set consists of news articles from the Wall Street Journal. It has been commonly used as a benchmark task for LMs. The vocabulary was thresholded at 10K words and in the available pre-processed versions[1] all OOVs in the validation and test sets are already replaced by OOV tokens. Therefore, no OOV rate can be given.

Table 2 shows the details for the MIT OpenCourseWare [44] (MIT-OCW) data set. It consists of lectures and talks from the MIT OpenCourseWare website

[1] `http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz` or `https://github.com/tomsercu/lstm/tree/master/data`

Table 2: Overview of the MIT OpenCourseWare data set. The OOVs are given with respect to the LM training set.

Set	Sentences	Words	Vocabulary/OOV
LM Training	417611	6153008	47448
AM Training	56306	1149602	27096
Validation	3460	23720	112
Test	6989	75050	253

Table 3: Overview for the TED Talks and TED-LIUM data sets.

Set	Sentences	Words	Vocabulary/OOV
TED Talks Training	665070	5112647	72239
TED Talks Validation	2125	17344	1
TED Talks Test	3464	26704	4
TED-LIUM Training	56803	1545951	32656
TED-LIUM Validation	507	17783	240
TED-LIUM Test	1155	27500	230

and MIT World website. The training data for the LM is a combination of MIT lectures (1.3M words), data from the Switchboard corpus (3.1M words) and another lecture corpus (1.7M words) (MICASE[2]). The data from the lecture corpora are suited to each other, but the data from the Switchboard corpus does not match the lecture data well. The smaller AM training set has about 1.1M words which corresponds to roughly 100h of transcribed speech.

Table 3 shows the details for the TED talks data set and the TED-LIUM [111] speech recognition task. The TED talks data set consists of the original subtitles while for TED-LIUM the validation set and test set use re-transcribed talks. The TED talks set uses subtitles from more talks compared with TED-LIUM to train a better LM.

Table 4 shows the details for the Corpus of Spontaneous Japanese [87] (CSJ). As ASR task for CSJ, the publicly available task [96] was used. CSJ is the largest of all corpora used in the experiments. It consists of Japanese lectures

[2] `http://www.hti.umich.edu/m/micase`

Table 4: Overview of the Corpus of Spontaneous Japanese data set.

Set	Sentences	Words	Vocabulary/OOV
Training	430672	7721240	70901
Heldout	10000	234145	—
Test 1	1272	27651	96
Test 2	1292	28424	106
Test 3	1385	18283	92

and presentations. The training data available for training the AM is about 500h of speech. The same data was also used to train the LM. The first 10K utterances in the training set were used as validation set for LM training. CSJ does not have a distinguished validation set. There are only three test sets. Test set number three was used to determine the interpolation parameters for the LMs in rescoring.

5.2 Common Experimental Settings

Following the introduction of the data sets, this section gives an overview of the different toolkits used for the experiments. To train the LMs (M-Gram and NNLM) different toolkits were used depending on the model. The SRILM toolkit [123] was used to train M-Gram LMs. It is a commonly used toolkit for M-Gram LMs that contains a wide functionality to train different LMs with different smoothing techniques and also allows to use dynamic cache LMs. For the NNLMs, two different toolkits were used. In case of vanilla RNN-LMs, Mikolov et al.'s RNNLM toolkit [92] was used as base for the implementation. For the experiments with LSTM-LMs, a self-implemented LM toolkit based on the deep-learning toolkit Chainer [134] was used. The training and evaluation of these models were based on the PTB example available online[3].

As speech recogniser, two different systems were used. The first ASR system was the NTT SOLON [67] speech recogniser. This recogniser is a proprietary GMM-HMM-based ASR system. The second speech recogniser was the open-source speech recognition toolkit Kaldi [107]. This recogniser allows to train

[3] `https://github.com/chainer/chainer/tree/master/examples/ptb`

DNN-HMM hybrid models and offers a large set of evaluation tasks. The tasks and baseline results are publicly available.

The training parameters vary from experimental set to experimental set because the toolkits can be different depending on the experimental set. Therefore, the detailed parameters are given with each experimental set.

Depending on the experiment, the following other tools were also used. To estimate F0 contours of a speech signal, the proprietary estimator from Nakatani et al. was used [98]. This estimator calculates F0 values and an estimate if a speech sequence is voiced or unvoiced. To train an LDA topic model and calculate topic features, the scikit learn [106] machine learning toolkit was used.

5.3 Re-training of Prosodically-enhanced RNN-LMs

This section summarises the experimental results for RNN-LMs enhanced with prosodic features. The key point of these experiments was to address the imbalance that can occur when an LM should be trained on both text and acoustic data. It is not for every corpus the case that there are existing acoustic data for the all available text data. In the experiments, retraining of RNN-LMs, rather than training an LM from scratch is investigated. Using this method, PPL and WER reductions for N-best rescoring on the MIT-OCW lecture corpus were achieved over conventional RNN-LMs that only rely on textual information.

5.3.1 Experimental Setup

The experiments were conducted on the MIT-OCW corpus [44] consisting of MIT lectures. The corpus consists of an LM training set and an AM training set. The LM training set has approximately 6M words and a vocabulary of 47K words. The AM training set has approximately 1M words and a vocabulary size of 27K words. Both datasets have partially non-overlapping vocabulary. Some words are included in the smaller AM training set but not in the LM training set and vice versa. In addition, there is a validation set and a test set. Figure 22 illustrates the details of the MIT-OCW corpus in more detail. Furthermore, these numbers nicely show the imbalance of text and transcribed training data.

The implementation for the RNN-LM used in the experiments was based

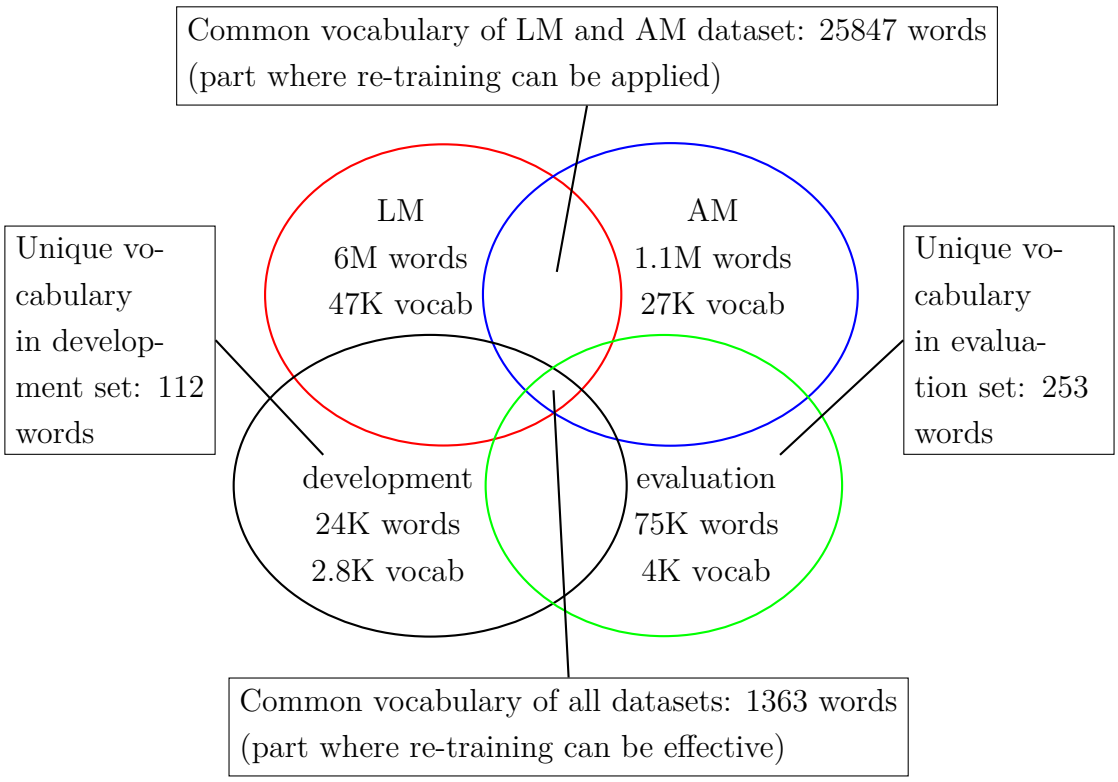


Figure 22: An overview of the different datasets in the MIT-OCW corpus.

on Mikolov's RNN-LM toolkit [92]. However, modifications were made to the implementation to include the auxiliary features. The model used the architecture as described in Chapter 4.1 but without the direct connection from the auxiliary features to the output layer. In addition, an option to train for a minimum number of epochs was added. Training was continued with the final learning rate until the minimum number of epochs was reached. To prepare the baseline network for re-training, it was already extended at its initialisation to include the auxiliary features. During training of the baseline model, this additional input was always set to zero. However, there is no particular need to include the inputs in training of the base model. By extending the input transition matrix by additional columns for the auxiliary features, this addition can also be made to a fully-trained network.

The RNN-LM used for all experiments had a hidden layer with 300 nodes and 250 classes on the output. The speech recogniser for the ASR experiments on N-best rescoring was NTT SOLON. A trigram LM with Kneser-Ney smoothing [82] was estimated using the SRILM toolkit [123] and used for interpolation with the RNN-LM probability. The trigram had a PPL of 199 on the test data.

F0 features were estimated using the method from [98]. The features were calculated from a 42 ms window and the frame shift was 1 ms. Subsequently, these features were combined with a word-frame alignment estimated with NTT SOLON speech recogniser on the transcribed data for the training, validation and test data. During N-best rescoring, the word-frame alignment was estimated with the 100-best list produced by the speech recogniser.

For F0 features, logarithmic and linear scale are often used. In the experiments, logarithmic f0d was used, which was normalised to zero mean and unit variance with the global mean and variance estimated on the training set. A f0d feature was calculated from the voiced and unvoiced F0 frames within a word boundary. For f0m, linear scale was used and the features were normalised per utterance. One feature was only calculated from the voiced frames (as estimated by [98]) within a word boundary.

The signal power was calculated per sample and the feature for one word was the average of all samples within the boundaries of this word. The same feature types as for F0, that means, average per word (pwm) and difference between words

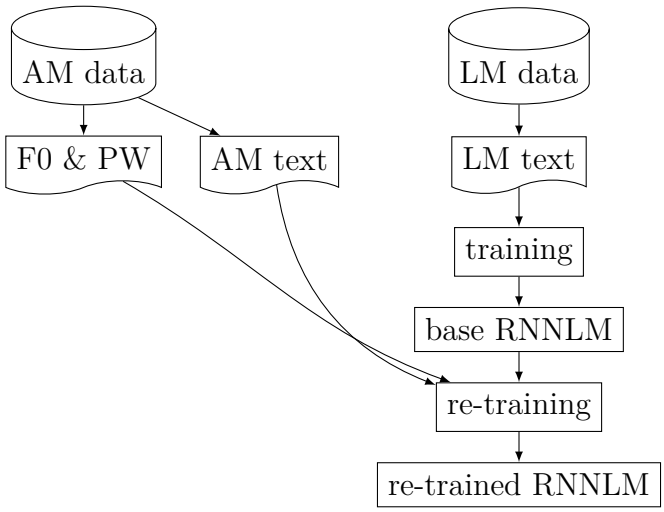


Figure 23: Outline of our proposed training method.

(pwd) were used. For both pwm as well as pwd logarithmic scale features were used. All features were normalised to zero mean and unit variance. pwd features were normalised with the global mean and the global variance estimated from the training set. pwm features were normalised per utterance. For calculating the logarithmic signal power, always a small epsilon (10^{-10}) was added to cope with zero signal power.

The re-training scheme is outlined in Figure 23. In the first step, a base RNN-LM was trained on the 6M LM training set. Second, this model was re-trained with the text and prosodic features from the AM training set. The base RNN-LM was trained on the LM training data for 16 epochs with a start learning rate of 0.1. The re-trained models were re-trained for 6 epochs and re-training started with a learning rate of 0.000024. Throughout the experiments, all parameters (e.g. feature computation method) were chosen based on the validation set. The results given in the following are for the test set. The validation set results can be found along with additional results in Appendix C.

5.3.2 Perplexity Evaluation

Table 5 shows the PPLs for the different models. Re-training with text data only results in a relative PPL reduction of 7 % for the network and 1 % after

Table 5: PPL results for RNNLMs trained on textual and prosodic features for the test data of MIT-OCW.

Model	PPL Net	PPL Net+3-gram
LM text only	163.79	150.12
Re-training on AM text	152.50	148.22
+ f0d	149.27	145.52
+ f0m	150.28	146.51
+ both f0	145.40	142.84
+ pwd	146.89	143.77
+ pwm	148.25	145.29
+ both pw	141.92	139.95
+ f0d and pwd	146.09	142.94
AM text only	203.87	164.10
+ f0d	186.68	151.34

interpolation with a trigram LM.

Re-training the base LM with different F0 features, the best result was obtained with delta features. The PPL was reduced by 9 % compared with the base model and 2 % compared with the network re-trained using text only. The PPL reduction was 3 % after trigram interpolation. Signal power features also improved on the baseline. The largest PPL reduction among all prosodic features was 10 % and 4 % on the baseline and text re-training, respectively, with pwd.

In addition to individual features, also combinations of features were investigated. Table 5 shows the results for a combination of F0 features, power features and the best F0 (f0d) and power feature (pwd), respectively. Among all these combinations, combining signal power features gave the largest PPL reduction, that is, 13 % and 7 % reduction on the baseline and the model re-trained on text only, respectively.

As additional information, Table 5 also provides results using the conventional method, that means, training an LM jointly on text and acoustic information from the beginning. However, these results cannot be directly compared to the results from re-training because there are significant differences in the models

Table 6: WER results for 100-best rescoring of MIT-OCW.

Model	WER
LM text only	24.6
Re-training on AM text	24.5
+f0d	24.4
+f0m	24.6
+both f0	24.5
+pwd	24.7
+pwm	24.6
+both pw	24.7
+f0d and pwd	24.6
AM text only	25.7
+ f0d	25.9

(vocabulary size, training data size etc.). The results are shown in the last two rows of Table 5. f0d was used as example for prosodic information. With f0d PPL reduced 8 % compared with a model trained on AM text data only.

5.3.3 N-best Rescoring

After PPL evaluation, N-best rescoring experiments were conducted on the 100-best list obtained using NTT SOLON speech recogniser. Table 6 shows the WER for the baseline model, re-training with text only and re-training with prosodic features. All RNN-LM scores were interpolated with the score from a 3-gram LM (interpolation weight 0.8 for the RNN-LM).

The baseline for a vanilla RNN-LM resulted in a WER of 24.6 %. Re-training with the AM training text resulted in a WER reduction of 0.1 %. The highest WER improvement was achieved with f0d features. f0d features reduced the WER about 1 % relative or 0.2 % absolute compared with the baseline. pwd features gave the largest reduction in the PPL evaluation, but they did not lead to an improvement of WER. This was consistent during all experiments for different hyperparameters with pwd features.

Since the training objective of the LM (minimising word prediction error) and

the evaluation target in rescoring (minimising errors between recognition result and ground truth) are different, an improvement of the LM in terms of PPL does not necessarily have to lead to a reduction in WER. Comparing the rescoring results of the baseline RNN-LM and the model re-trained with pwd features, there were only small differences in the selected hypothesis. These differences often affected the beginning of a sentence where fill words were inserted, or other positions in the sentence where functional words were inserted or misrecognised for the wrong word. Further on, comparing the mean and variance of all features of each word in the vocabulary for training, test, and rescoring, the statistics of the features for rescoring were closer to the values of the training data than the test data. The errors in the recognition hypothesises might have introduced errors in the word-frame alignment which lead to feature sequences that were more likely for the LM but these hypothesises contained more errors. A definite answer to this problem can, however, not be given because it would require the comparison of all hypothesis for all utterances. In addition, the recurrent state of the RNN-LM could also influence the result because it could carry over wrong information to subsequent utterances. Considering the hidden state of the RNN-LM in this analysis is difficult, because it is hard to analyse.

As for the PPL results, Table 6 also shows rescoring results for the conventional method as additional information. Using the same two models as for PPL evaluation, the baseline WER was 25.7 % for the model trained on AM text data only. The WER increased by 0.2 % when additional f0d features were used. Although the results cannot be compared directly, for these experimental conditions re-training showed to be more effective than the conventional combined training.

5.4 Cache Extensions for LSTM-LMs

This section presents the experimental results for cache extensions of LSTM-LMs. The different cache architectures as introduced in Chapter 3.2 and Chapter 4.2 will be evaluated. For this analysis, PPL and WER experiments were conducted on the MIT-OCW lecture corpus.

5.4.1 Experimental Setup

For the experiments, PPL evaluation and N-best rescoring on the MIT-OCW lecture corpus [44] were conducted. The corpus had about 6M words and the vocabulary size is 47K. The vocabulary was not truncated. The corpus has about 100h of transcribed speech.

As speech recogniser, the GMM-HMM based NTT SOLON speech recogniser was used. The 1-best result for this speech recogniser had a WER of 26.7%. The LSTM-LMs were implemented using the deep learning toolkit chainer [134]. All LMs were trained with mini-batches of length 128 and truncated backpropagation through time after 20 words. The initial learning rate was set to 0.1 and was reduced by a factor of 1.3 after the sixth epoch in every epoch. As optimiser AdaGrad [33] was used and all models were trained for 16 epochs. Dropout was used on all linear layers during training.

For the networks with connected NC, the NC was connected to an LSTM-LM that was pre-trained for 16 epochs. In the combined training with the cache, only the connections in the linear layer before the softmax of the base LSTM-LM and the linear layers of the cache were updated for 5 epochs with a learning rate of 0.1. The learning rate decreased by a factor of 1.3 every epoch.

To calculate the word error rate in N-best rescoring, the log probability from the LSTM-LM for the whole utterance was interpolated with a trigram probability and the acoustic model score. The trigram LM was trained using the SRILM toolkit [123] and used Kneser-Ney [82] smoothing. The trigram itself had a PPL of 199 on the test data.

5.4.2 Perplexity Evaluation

Table 7 shows the PPL for an LSTM-LM baseline and the compared cache architectures for the validation and test data. A trigram had a PPL of 199 on the test data. The LSTM-LM baseline had a PPL of 147.03.

Using the continuous cache from Chapter 3.2.1, the PPL was reduced by roughly 30% on the test data to 103.42. This result corresponds to the numbers reported in [48]. However, connecting the cache directly to the softmax layer did not result in any PPL reduction. Looking at the training logs, the PPL remained almost unchanged during training. This suggests that the network was not able

Table 7: PPL results for validation and test data of MIT-OCW.

model	val	test
LSTM	175.24	147.03
NC	125.58	103.42
Connected NC	213.12	189.60
Connected NC (wsim)	213.89	190.79
BOW unigram	183.07	149.02
BOW uni + bigrams	178.81	149.79

Table 8: WER results for 100-best rescoring on MIT-OCW.

model	WER
LSTM	24.3
NC	24.2
Connected NC	25.2
Connected NC (wsim)	25.3
BOW unigram	31.5
BOW uni + bigram	31.7

to learn good parameters for the interpolation with the cache in this architecture. Using a word similarity (wsim) instead of equality for the word vector did not change the result.

For the BoW cache, two scenarios were investigated. In the first case, only the unigram information $\boldsymbol{h}_l^{(\mathrm{u})}$ was used. In the second case, both unigram $\boldsymbol{h}_l^{(\mathrm{u})}$ and bigram $\boldsymbol{h}_l^{(\mathrm{b})}$ information was used. In both cases, the PPL increased slightly compared with the LSTM baseline.

5.4.3 N-best Rescoring

Table 8 shows the results for n-best rescoring on the 100-best list obtained from SOLON speech recogniser [67]. Without rescoring, the speech recogniser achieved a WER of 26.7%. For the baseline system in 100-best rescoring with an LSTM-LM, the WER was 24.3%.

Using the continuous cache, although a significant PPL reduction was achieved,

the WER only improved by 0.1% (absolute) on the LSTM-LM. During PPL evaluation, the cache can be filled with the ground-truth next-word information. However, in rescoring, the LM can only use recognition hypothesises. These hypothesises contain most likely recognition errors. In addition, to achieve a low WER not the most likely utterance from an LM point of view might be the best one. The hypothesis which has the lowest number of errors compared with the transcription will achieve the lowest WER. This hypothesis can be very unlikely from an LM point of view. For example, a hypothesis with multiple word insertions and replacements might get a higher LM score than a single utterance with just one word substitution, because the former one is more likely for the LM. Furthermore, the cache output zero in many cases and the capability of the cache to predict OOVs was irrelevant for rescoring.

As expected from the PPL results, the neural connected continuous cache did not achieve any improvements in WER on the baseline. Using the BoW cache, the WER increased significantly. The results were even worse than those after the first pass only. Such a behaviour was not to be expected from the PPL evaluation where performance dropped only slightly. So far, there is not any explanation why the WER increased by such high number.

5.5 Feature Based Domain Adaptation

This section presents the results for feature-based domain adaptation of LSTM-LMs with conventional LDA features. The network architectures as described in Chapter 4.1 and 4.3 were investigated in the experiments.

5.5.1 Experimental Setup

For the experiments, three different datasets were used. First is the well-known Penn Treebank [88] (PTB), which has roughly 0.9M words in the training set and a vocabulary size of 10K. The dataset consists of articles from the Wall Street Journal covering different topics, such as, politics and finance. The standard segmentation, that is, sections 0-20 as training, 21-22 as validation and 23-24 as test set was used.

The second corpus consists of TED talks and the TED-LIUM corpus [111] for

Table 9: Comparison of subtitle and TED-LIUM test sets.

	sentence length (words)				Vocabulary	length
	min	max	mean	var	size	(words)
subtitle	2	19	9	8.88	3638	30168
TED-LIUM	2	122	25	209.39	3568	28655

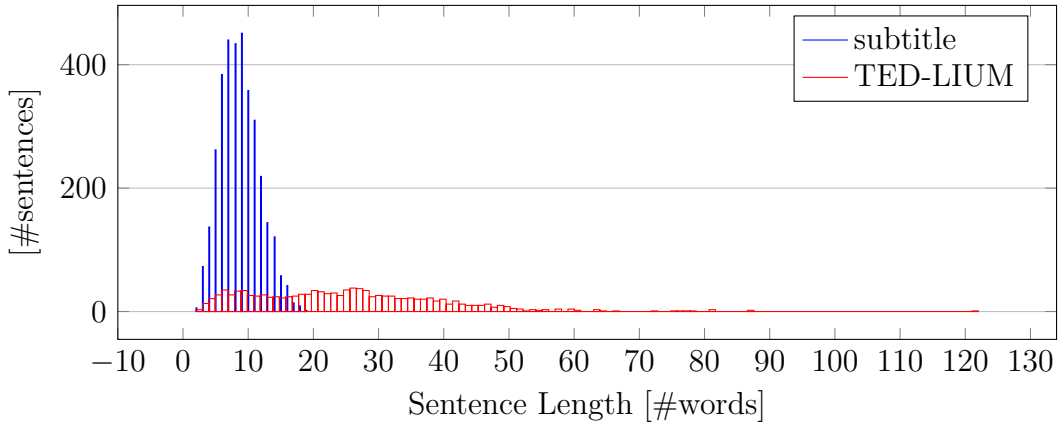


Figure 24: Histogram of sentence lengths in subtitle and TED-LIUM evaluation sets.

the ASR experiments. For the ASR system, the standard Kaldi [107] recipe was used. The training data provided in TED-LIUM is small for an LM. To have a larger training set, subtitles from additional TED talks were crawled. The final LM training set consisted of 2494 talks and with a total size of 5.1M tokens and a vocabulary size of 73K words. All words appearing only once in the training data were replaced by an unknown token. This resulted in an effective vocabulary size of 43K words which was used to train the NNLMs.

From the original subtitles, a subtitle validation and test set were generated in the same way as the 5.1M word training set. The validation and test sets used the same data as in the IWSLT 2011 evaluation campaign [37]. The results reported in the following will be for the subtitle test set and TED-LIUM's test set.

The TED-LIUM validation and test set were re-transcribed from the original talks to have verbatim transcriptions. Rousseau et al. wanted to have verba-

tim transcriptions to accurately tune the AM during training. However, the re-transcription introduced a mismatch with the subtitle sets. Some key-figures about the subtitle-based and the TED-LIUM test set are summarised in Table 9. Furthermore, Figure 24 shows a histogram of the different sentence lengths. The sentence length for both evaluation sets is very different. Because the TED-LIUM validation and evaluation sets have longer sentences on average, the probability of the end of sentence symbol is reduced compared with the subtitle-based sets.

The third dataset was the Corpus of Spontaneous Japanese (CSJ). As training data for the LMs, a training set of approximately 8.2M words was used. The vocabulary size was 71K and 44K after keeping only every word which appeared at least twice. For the rescoring experiments with CSJ, the ASR system from the publicly available Kaldi recipe was used. The validation set for LM and AM were the first 10K utterances of the training data provided in the CSJ corpus. The implementation for the acoustic model was the same as for the experiments with TED talks, that is Karel's DNN. The acoustic model was a feed-forward DNN. As with TED talks, the speech recogniser was trained without any sequence discriminative training.

For training of the LDA model, two different schemes depending on the dataset. For PTB, the same scheme as used by Mikolov et al. [93] was applied. That means, the training set was divided into chunks of ten utterances and each of these chunks was regarded as a single document. For the TED talks and CSJ, this segmentation was not necessary because the datasets consist of different talks and each talk could be used as a separate document.

Before training the LDA topic model, a list of common stop words as well as high and low frequency words were removed. This pre-processing was only applied to train the LDA topic model and to compute the LDA features for the different datasets. However, this processing was not applied to the text which was used to train the LMs. The LDA implementation for the experiments was the one provided in scikit-learn [106].

For calculation of the LDA features, a sliding window covering the previous 50 words in case of PTB and 200 words in case of TED and CSJ was used. The LDA features extracted from this context window represented the topic distribution over this sliding window. Than means, the LDA features were not static features

for each talk and instead represented dynamic, context dependent features. For N-best rescoring, the LDA features were calculated from the N-best list. The features were not calculated on the ground-truth data. In case the sliding window extended over the current utterance, the context for the one-best recognition result was kept over proceeding utterances. In a similarly way, the state of the LSTM for the one-best hypothesis was kept across sentences.

As common part to all NNLMs, the LSTM-LM used a single layer LSTM with 300 units. The adaptation layer for fLHUC-LSTM and fLHN-LSTM also had 300 units. For PTB and CSJ, all networks were trained for 20 epochs. For TED, the number of training epochs was optimised for each model on the validation set WER. The mini-batchsize and backpropagation through time length were tuned on PTB to give a good compromise of training time and PPL. From the results, the mini-batchsize was set to 128 and the backpropagation through time length was set to 20 words. The initial learning rate was 0.1 and the AdaGrad [33] optimiser was used. Gradients were clipped to an L2-norm of five. If the PPL improvement on the validation set data was less than 0.1% within one epoch, the learning rate was multiplied by 0.5. In all models, dropout [63] with a dropout ratio of 50% was used. As for [27], in fLHN-LSTM, the weight matrix of the linear layer connecting the LSTM and the domain adaptation layer ($\boldsymbol{V}^{(\mathrm{w})}$) was initialised with the identity matrix.

5.5.2 Penn Treebank Results

Because PTB is a small data set, it was used to test different parameter settings for further experiments. The vocabulary size is also very low which allows to train different types of factLSTM with a large number of factors and to investigate the effect of a larger number of factors on the PPL. A further point of interest was the influence of the number of LDA topics on the results.

Table 10 shows the validation set results for different LM architectures. The topic features for these models were calculated from topic models with 30 to 60 topics. The PPL of an LSTM-LM baseline was 105.66. contLSTM did not show any improvement on an LSTM-LM. This is in contradiction to the result from Soutner et al. [120], but the authors used only 20 LSTM units and here 300 LSTM units were used in the model. In case of a small number of hidden

Table 10: PPL on the validation set of PTB for different numbers of factorised hidden layers versus different LDA dimensions (LSTM-LM 105.66). The number in brackets with factLSTM gives the number of factors used.

LDA topics / Model	30	40	50	60
contLSTM	118.72	117.83	122.72	121.74
fLHN-LSTM	103.99	105.59	107.16	105.76
fLHUC-LSTM	103.35	104.01	104.76	104.80
fLHUCB	106.81	109.70	108.93	107.55
factLSTM(5)	105.06	105.55	105.93	106.08
factLSTM(10)	102.15	102.11	102.81	101.02
factLSTM(20)	102.92	102.80	101.54	101.06
factLSTM(30)	101.36	102.64	102.24	100.91
factLSTM(40)	103.75	102.69	101.75	**100.69**

units, the network might be able to use the LDA features as a context memory. fLHN-LSTM and fLHUC-LSTM were able to reduce the PPL compared with the LSTM-LM baseline in some cases. fLHUCB did not show any PPL reduction on the LSTM-LM baseline.

The performance of factLSTM, depends on the number of LDA topics as well as the number of factors. For the same number of factors, the general trend showed a lower PPL with an increasing LDA size. When the LDA size was kept constant and the number of factors was increased, the PPL was also reduced. From the investigated configurations, the combination of the highest number of topics and the highest number of factors had the lowest PPL. However, choosing a large number of factors (larger than 10) did not result in much further PPL reduction on the PTB dataset. This might be due to the small size of PTB. With a small number of factors, the PPL is not much reduced compared with the LSTM-LM baseline.

Table 11 shows the PPL results for the validation and the test set obtained with the best model from each architecture. The trigram LM uses Kneser-Ney [82] smoothing and was estimated using the SRILM toolkit [123]. However, all

Table 11: PPLs for baseline, best fLHN-LSTM and factLSTM model on the validation and test set of PTB.

Model	LDA topics	val	test
trigram	—	182.16	171.68
LSTM	—	105.66	98.94
contLSTM	40	117.83	109.00
fLHN-LSTM	30	103.99	97.42
fLHUC-LSTM	30	103.35	95.90
fLHUCB	30	106.81	99.61
factLSTM(40)	60	**100.69**	**94.99**

PPLs shown for the NNLMs were obtained without trigram interpolation. For contLSTM, 40 LDA topics were used. For fLHN-LSTM, fLHUC-LSTM, and fLHUCB 30 LDA topics were used, and for factLSTM 60 topics and 40 factors were used. The number of parameters were around 6.7M for contLSTM, around 6.5M for fLHN-LSTM, fLHUC-LSTM, and fLHUCB, and around 123M for factLSTM. The baseline LSTM-LM had around 6.4M parameters. The training time of LSTM-LM, contLSTM, fLHN-LSTM, fLHUC-LSTM, and fLHUCB was about 1h. factLSTM took 12h to train. contLSTM again had the overall highest PPL among all NNLMs. fLHN-LSTM had a 1% lower PPL compared with the LSTM-LM baseline (relative improvement). fLHUC-LSTM had a 2% and 3% lower PPL than the LSTM-LM baseline on the validation and test set, respectively. fLHUCB had a slightly higher PPL compared with the baseline and the individual models. Overall, out of all domain adaptation methods, factLSTM achieved the highest relative PPL reduction compared with the LSTM-LM baseline. factLSTM improved 5% on the validation and 4% on the test set.

5.5.3 TED Talk Results

On the TED talks data set, each model was trained until convergence. The PPL of the validation set over 80 epochs is shown in Figure 25. LSTM-LM reached its minimum after roughly 40 epochs. contLSTM had a slightly lower PPL compared with LSTM-LM. fLHUC-LSTM had almost the same convergence

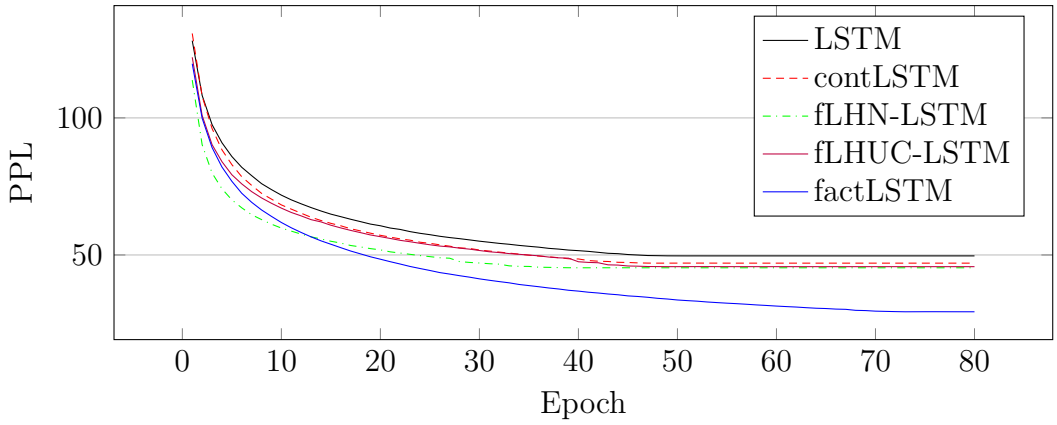


Figure 25: Convergence on the validation set over 80 epochs for different models on TED dataset.

as contLSTM. Both reached the minimum PPL on the validation set around 50 epochs. fLHN-LSTM converged after around 40 epochs to its minimum and convergence appeared to be faster than for LSTM-LM. factLSTM had the lowest PPL of all models after 20 epochs and reached the lowest PPL of all models at 80 epochs. The converged models had the lowest PPLs on the subtitle validation and test sets. However, PPL is not directly related with WER after rescoring. To find the model with the lowest WER on TED-LIUM, the WER for 100-best rescoring was evaluated every five epochs around the point where the PPL converged. From these results, the model with the lowest WER on the validation set was selected for each model architecture. These models were used for the PPL and WER analysis in this section.

Table 12 shows the results for the TED-LIUM dataset. For the TED talks, PPL results for the trigram LM were obtained with the model that is distributed with the Kaldi recipe [146]. The PPLs for the NNLMs are, as for the PTB results, without trigram interpolation. The trigram was used to interpolate LM and AM scores in 100-best rescoring. The corresponding interpolation weights for each LM and the trigram were optimised on the TED-LIUM validation set. The interpolation weight of trigram and NNLMs was mainly between $[0.9, 1]$ for the NNLM. Therefore, the WER after rescoring was for the main part determined by the NNLM. All NNLMs with domain adaptation used LDA features from 50

Table 12: PPL and WER for our own subtitle based test set and TED-LIUM with 50 LDA topics, a 200-word window size and factLSTMwith 15 factors. The trigram result represents the 1-best result and the results for the neural network LMs are for 100-best rescoring.

Model	Training	Test PPL		WER[%]	
	Epochs	subtitle	TED-LIUM	val	test
trigram	—	156.41	222.05	16.3	15.1
LSTM	40	51.98	156.29	14.2	12.1
contLSTM	45	47.34	165.69	14.5	12.3
fLHN-LSTM	40	44.72	127.99	14.1	12.1
fLHUC-LSTM	40	47.69	144.70	14.0	11.9
fLHUCB	40	38.65	133.46	13.9	11.8
factLSTM	70	**31.74**	**101.04**	**13.8**	**11.6**

topics and a window size of 200 words.

The PPL for the trigram is considerably higher than for the NNLMs on the subtitle and TED-LIUM test sets. This can be explained by the fact that the trigram is trained on out-of-domain data. A trigram trained on the subtitle training set had a considerably lower PPL of 106.58 on the subtitle test set. However, the trigram trained on out-of-domain data was used for the ASR system and consequently the results for this trigram are provided here. The baseline LSTM-LM reduced the trigram's PPL by 67% on the subtitle test set and 30% on the TED-LIUM test set. The WER decreased by 20% after rescoring. The baseline LSTM-LM had approximately 26M parameters and training for 40 epochs took approximately 4.5h.

contLSTM improved on an LSTM-LM on the subtitle test set. However, on the TED-LIUM test set, the PPL was higher than for the LSTM-LM. After 100-best rescoring, the WER also increased compared with the baseline. The number of model parameters increased by 2M to 28M and the training time increased by around 2h. fLHN-LSTM had a 14% and 18% lower PPL compared with an LSTM-LM for the subtitle and TED-LIUM test sets, respectively. In 100-best rescoring, the WER showed slight improvement on the LSTM-LM baseline for the validation set, but it was equal on the test set. fLHUC-LSTM had a slightly

higher PPL than fLHN-LSTM on both test sets but it was lower than the LSTM-LM baseline. The WER showed a slight but not significant reduction on the baseline. fLHN-LSTM and fLHUC-LSTM had 105K more parameters than an LSTM-LM, but the training time was about the same.

The combination of bias and gating based domain adaptation in fLHUCB reduced PPL by 26% and 15% compared with the LSTM baseline. For the TED-LIUM test set, the PPL was between that of fLHN-LSTM and fLHUC-LSTM. This shows again the different characteristics of training and test data. The relative WER reduction was 2.5% compared with the LSTM-LM baseline on the test set. This shows that both methods have complementary information that can help to improve the performance.

For factLSTM, 15 factors were used due to the limitation of GPU memory size. factLSTM had 207M parameters and the training time was 67h for 70 epochs. This method did increase the number of model parameters as well as the training time compared with the other methods, however, no particular optimisation of the implementation was done which speeds up the computation on GPUs. On the TED-LIUM test set, this method showed significantly lower PPL compared with all compared methods, namely 39% lower than an LSTM-LM and 21% lower than an fLHN-LSTM. After 100-best rescoring, a 3% relative WER reduced reduction compared with an LSTM-LM was achieved.

A matched-pair significance test showed a significant difference of factLSTM compared with the LSTM-LM baseline at a significance level of $p = 0.1\%$. All remaining methods did not show a significant improvement on the baseline. These results show that feature-based adaptation can improve both PPL and ASR rescoring performance. factLSTM achieved superior performance in general compared with other approaches for exploiting auxiliary features. This is, however, at the expense of using a larger amount of parameters.

5.5.4 CSJ Results

CSJ consists of a training set and three test sets, but there is no dedicated validation set. For parameter tuning for 100-best rescoring, test 3 was selected as a validation set. The window size and the number of LDA topics was the same as for TED talks. Table 13 shows the PPL and WER for all three test sets.

Table 13: PPL and WER for CSJ using topic features from 50 LDA topics and a 200-word window size. The trigram result is the 1-best result and the NNLM results are for 100-best rescoring.

Model	Test 1		Test 2		Test 3	
	PPL	WER[%]	PPL	WER[%]	PPL	WER[%]
trigram	82.45	12.26	89.17	9.34	94.89	12.22
LSTM	40.52	10.71	41.79	8.08	41.01	10.49
contLSTM	42.66	10.72	43.66	8.17	43.30	10.67
fLHN-LSTM	39.66	10.59	41.07	7.94	41.29	10.63
fLHUC-LSTM	38.90	10.62	40.40	7.93	39.94	10.38
fLHUCB	38.79	**10.55**	39.37	**7.85**	**39.74**	**10.36**
factLSTM(15)	**38.61**	10.63	**39.34**	7.95	39.87	10.45

Using an LSTM-LM reduced the trigram PPL by more than 50% and led to an improvement in 100-best rescoring.

Feature-based domain adaptation achieved on average a relative PPL reduction of 3% compared with the LSTM-LM baseline. This results was much lower than the relative reduction with the TED talks, but the baseline LSTM-LM already had a very low PPL for CSJ. fLHN-LSTM reduced the WER around 1% relative to the LSTM-LM in test 1 and test 2. fLHUC-LSTM reduced the PPL by 3% to 4%. The relative WER reduction was between 1% to 2%. It achieved a consistent WER reduction across all test sets compared with the LSTM-LM baseline.

Using fLHUCB, the PPL was reduced by 3% to 6% relative to the LSTM-LM baseline. WER was reduced relative by 1% and 3% in test 1 and test 2, respectively. Compared with the 1-best decoding result obtained with a trigram LM, the WER was reduced 14%, 16% and 15% for test 1, test 2, and test 3, respectively. The CSJ data are much better matched between training and test data than the TED talks dataset. The numbers show that in this case fLHUCB achieved a consistent improvement over fLHN-LSTM or fLHUC-LSTM.

factLSTM achieved the lowest PPL on test 1 and test 2 among all NNLMs but the WER did not match this result. Unfortunately, factLSTM did not achieve a similar performance as on TED talks. The PPL and WER were only slightly

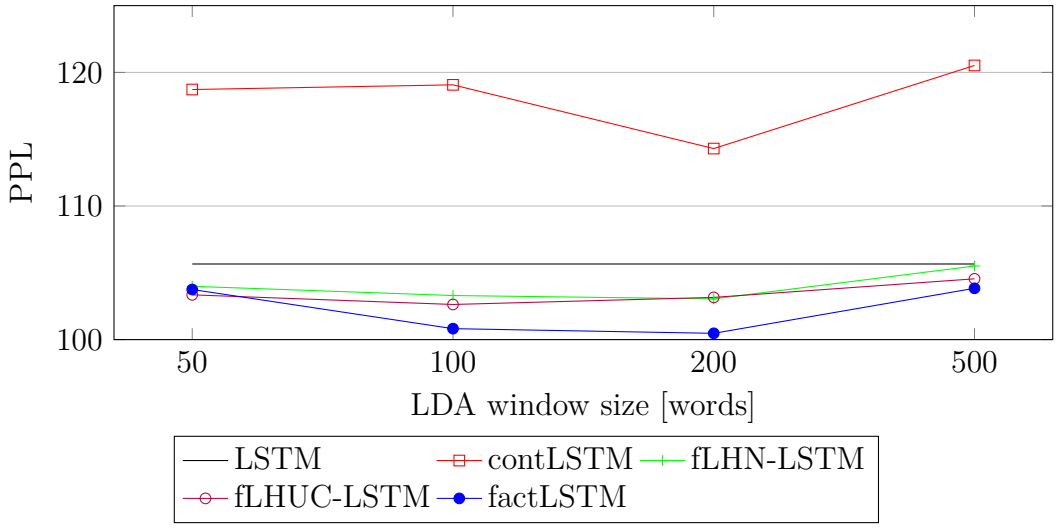


Figure 26: Comparison of different context window sizes versus validation PPL for PTB with LDA features from 30 topics.

reduced compared with the LSTM-LM baseline.

5.5.5 Discussion

Following the PPL and WER results, this section provides further discussion on the effect of LDA features on NNLMs. At first, the features themselves are discussed. As mentioned in Chapter 3.3.1, the LSTM itself captures context information in its cell state. However, it is valid to assume that an LDA topic model can capture a different kind of context information. To investigate under which circumstances the information provided by the LDA features can have a beneficial effect on the model performance, experiments with varying context window lengths were performed.

Figures 26 and 27 show the validation set PPL for PTB and TED, respectively. In these two figures, the PPL for different context window lengths for extracting the LDA features is shown. For PTB, the window length was between 50 to 500 words and the topic model size was fixed to 30 topics. For TED, a window length of 50 to 1000 words and 50 topics were used because the training data for TED is larger than for PTB. For the ASR results in Chapter 5.5.3, the number of training epochs was optimised for each model to obtain the lowest possible

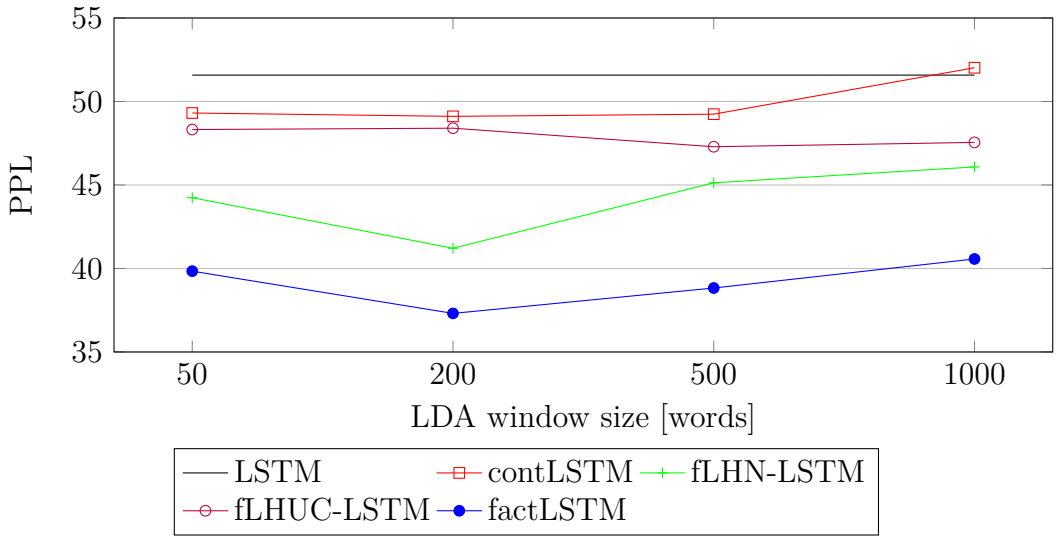


Figure 27: Comparison of different context window sizes versus PPL for the Ted talks subtitle validation set with features from 50 LDA topics.

WER. However, here model PPL is the major focus and consequently all NNLMs compared in Figure 27 were trained for 40 epochs.

For PTB, the trend shows that all models reached a minimum PPL with a window size between 100 to 200 words. Using a smaller or a longer window, the PPL of domain-adapted models usually increased. This result indicates that shorter windows are well covered by the LSTM itself and longer windows contain too general information. In case of TED, a context window size around 200 words gave a good result as well. In general, the LMs using domain adaptation improved on both corpora on a baseline LSTM-LM. This result suggests that the LDA features contain information, that is not captured in the LSTM.

A general remark regarding the topic features themselves. During the experiments, it showed that the model performance is highly dependent on the quality of the topic model and the features estimated from it. Estimating a good LDA topic model will improve the feature quality and this is important for model performance. It is reasonable to assume that all models can benefit from better topic features that might be derived from other topic models than LDA. Comparing the training set sizes of TED talks and PTB, in case of TED talks the amount of data is roughly five fold the amount of data for PTB. The increased data size might

Table 14: N-best hypothesis comparison for selected utterance from TED-LIUM test set.

REF:	this really solves the problem i've got a picture here of a place in kentucky this is the left over the ninety nine percent where they've taken out the PART THEY BURN NOW so IT'S called depleted uranium that would power the U S
1-best:	this really solves the problem i've got a picture here HAVE a place in kentucky this is the left over the ninety nine percent where they've taken out the **** **** PARTY BERNAU so **** called depleted uranium that would power the * U.S.
LSTM-LM:	this really solves the problem i've got a picture here of a place in kentucky this is the left over the ninety nine percent where they've taken out the **** **** PARTY BERNAU so **** called depleted uranium that would power the * U.S.
factLSTM:	this really solves the problem i've got a picture here of a place in kentucky this is the left over the ninety nine percent where they've taken out the part they burn now so it's called depleted uranium that would power the * U.S.

have helped estimating a better LDA topic model. The features calculated from this model were more effective for the LMs and lead to a higher PPL reduction.

In addition to analysis of the features, a more in-depth analysis of the recognition results before and after rescoring is provided at this point. Comparing the results for different model architectures, the NNLMs had less errors with functional words. In the 1-best decoding result, these words were often left out or misrecognised. In cases where an utterance contained a large number of errors, rescoring with the NNLMs did not reduce the number of errors considerably. In this case, the output from the speech recogniser contained too many errors. It was not possible to bring up hypothesises containing less errors in the output because they might not have been contained in the 100-best list. In some cases, a relation between the corrected error and a topic could be established. One such example is shown in Table 14. It shows the ground-truth utterance, the result after 1-best decoding, rescoring with LSTM-LM, and rescoring with factLSTM.

Table 15: N-best hypothesis comparison for several utterances from CSJ for LSTM-LM and factLSTM.

ID	S00F0019 0174028 0176082
REF:	あのコンクリートとか木でできた
LSTM-LM:	あのコンクリートとか霧で来た
factLSTM:	あのコンクリートとか切りできた
ID	A01M0097 0181518 0188807
REF:	音調指令えー第三す声一個飛びますけれどもは一個の負の指令えー第二声については
LSTM-LM:	音調指令えー大賛成行くと言いますけれどもは一個の負の指令えー第二声については
factLSTM:	音調指令えー大賛成以降と言いますけれどもは一個の負の指令えー第二声については
ID	A01F0001 0663553 0670466
REF:	アンプフィルターを通しＤＡＴで録音しディバイダーで変換してえパソコン取り込むのは先程と一緒です
LSTM-LM:	アンプフィルターを通しＤＡＴで録音シーディば間で変換してえーパソコン取り込むのは先程と一緒です
factLSTM:	アンプフィルターを通しＤＡＴで録音シーディバイザーで変換してえーパソコン取り込むのは先程と一緒です

A close investigation of the probabilities that each NNLM assigned to the different hypothesises for one utterance showed that each model preferred the hypothesises in a different way. There was no constant offset in the probabilities between different models. This leads to the conclusion that each model architecture makes different use of the information provided by the LDA features.

Different from English, Japanese has many homonyms. Topic information might be helpful to select the correct homonym depending on the context. Table 15 shows different rescoring results from CSJ. The comparison shows the true reference transcription, the result for a conventional LSTM-LM and the result for factLSTM. The different LMs sometimes selected different homonyms but in most cases a relation to a topic is not visible.

Table 16: Test PPL on TED talks for fLHN-LSTMwith different LHN sizes after training for 70 epochs.

LHN size	300	600	1200
Parameters	26M	39M	65M
Training time	9h	11h	16h
subtitle PPL	45.25	41.98	42.79
TED-LIUM PPL	133.67	131.50	135.04

Figure 28 shows an analysis of the factor weights for factLSTM. Figure 28 (a) shows a principal component analysis (PCA) plot of the LDA features for all talks in the TED subtitle based test set. Each colour represents one talk and one dot corresponds to one feature. The plot reveals that the LDA features are mainly distributed along three axes. The PCA of the factor weights for the same talks is shown in Figure 28 (b). Each dot in the figure represents a factor weight. The distribution is very different from the LDA feature plot. The network learns a mapping of the features that helps to improve the prediction for the next word. In particular, some factor weights for different talks appear to be mapped to similar regions in the PCA space.

In addition to the PCA plots, Figure 29 (a) and (b) show the LDA features and corresponding factor weights for part of the test set. The figures show distinct regions of high weights for the LDA features and high factor weights that correspond to each other. Interesting is the region highlighted in red in Figure 29 (a) and (b). In this interval, there is no distinct LDA feature with high intensity but factor one exhibits a high activation. The same factor keeps a high activation after word index 16000 which can be linked to one topic with high activation in this region. This is a good illustration how the auxiliary network learns to combine different topic features into a single factor weight. The corresponding factor can then learn common information among these different topics.

factLSTM has significantly more parameters than the other models. It is therefore necessary to investigate if an increased the number of parameters in compared methods can lead to a similar PPL reduction. As compared method fLHN-LSTM was investigated. To increase the number of parameters, the size of the adaptation layer was increased. Table 16 shows PPL results for fLHN-LSTMs

Table 17: Comparison of subtitle and TED-LIUM test sets.

	sentence length (words)		
	min	max	mean
subtitle	2	19	9
TED-LIUM	2	122	25

with different LHN sizes after 70 epochs for the subtitle test set of TED talks. The table also shows the number of parameters and the training time for each model. The PPL did not change considerably, even if the number of nodes in the LHN was increased four fold. As comparison, factLSTM had a PPL of 31.74 for the subtitle test set and 101.04 for the TED-LIUM test set. The model had 207M parameters and the training time for 70 epochs was 67h. These numbers confirm that not only the increased number of parameters may cause the performance improvement of factLSTM.

5.6 Unified Context Extraction and Adaptation Framework

This section compares conventional LDA feature-based domain adaptation of LSTM-LMs with a unified framework for joint context extraction and model adaptation as described in Chapter 4.4. Experimental results for PPL analysis and rescoring will be given. In addition, some further discussion on the context representation and model adaptation parameters will be provided.

5.6.1 Experimental Setup

For the experiments with UniFA, the TED talks dataset and CSJ were used. For TED talks, the LM training set consisted of subtitles from 2494 TED talks. The validation and test sets were composed of subtitles as well. The training set had approximately 5.1M tokens and a vocabulary size of 43K words, where every word appearing only once was mapped to the OOV token.

For ASR experiments, a speech recogniser based on the standard TED-LIUM Kaldi recipe [111, 107] was used. The speech recogniser in the recipe is a DNN-HMM hybrid model using a feed-forward DNN AM without any sequence discriminative training. The validation and test sets in the TED-LIUM recipe and

in the subtitle set contained the same talks, but TED-LIUM uses re-transcribed talks. This re-transcription introduced some mismatch with the text data that was used for training the LMs. The major difference is the sentence length in the subtitle and TED-LIUM test sets as shown in Table 17. By re-transcribing the talks, the sentence length in TED-LIUM increased compared with the original subtitles.

The experiments on CSJ used also the publicly available Kaldi recipe [96]. The implementation for the acoustic model was the same as for the experiments with TED talks, namely Karel's DNN. The AM was a DNN speech recogniser without any sequence discriminative training. The training data for the LMs were the same as those used for the LMs in the CSJ recipe. The training set had approximately 8.2M words. The vocabulary size was 71K and became 44K after retaining only words that appeared at least twice. The validation set used during model training was the same as that used for training the LMs in Kaldi's CSJ recipe, namely the first 10K utterances of the training data.

All NNLMs in the experiments used state of the art LSTMs as recurrent units. The recurrent layer had 300 LSTM cells. The optimiser for model training was AdaGrad [33] and the starting learning rate was 0.1. The gradients were clipped to an L2 norm of 5. Further on, standard backpropagation through time for 20 time steps and a mini-batch size of 128 was used. The networks were regularised by dropout [63] with a dropout ratio of 50%. As mentioned in Chapter 4.4, layer normalisation [4] was used for combining the SSN with the fLHUC adaptation layer. All methods were implemented with the open-source toolkit chainer [134].

In the experiments, fLHUC was compared with two different methods to derive the adaptation features. In both cases, the same fLHUC adaptation layer [58] was used but the adaptation parameters were derived in a different way. The first method was conventional LDA feature-based domain adaptation (fLHUC-LSTM). In this case, the LDA implementation from Scikit-learn [106] was used to train the topic model and to calculate the features. The number of LDA topics was set to 50. The topic model was trained by splitting the subtitle training set into individual talks. For training and evaluation of the LMs, the LDA features were calculated from a fixed-size sliding window. The second method uses an SSN to extract the context features.

Table 18: PPL for subtitle and TED-LIUM validation and test set. The number in brackets denotes the context window size.

Model	Subtitle PPL		TED-LIUM PPL	
	val	test	val	test
LSTM-LM	51.58	51.98	209.34	156.29
fLHUC-LSTM (50)	48.32	48.56	226.48	154.15
fLHUC-LSTM (100)	47.47	47.44	188.33	139.12
fLHUC-LSTM (200)	46.76	46.98	173.51	135.68
UniFA (50)	**35.74**	**36.82**	**144.09**	**120.14**
UniFA (100)	38.27	37.66	165.55	129.21
UniFA (200)	37.18	37.82	168.79	135.34

For N-best rescoring, the LDA features and the context features for UniFA were calculated from the 100-best list. That means, recognition errors in the hypothesis can have an effect on the context representation of subsequent utterances if an utterance is shorter than the context window.

5.6.2 Perplexity Results

First, the PPL results are compared. In Table 18 the results for TED talks are shown. All PPLs for NNLMs were obtained without M-Gram interpolation and show therefore a fair comparison of the different adaptation mechanisms. As baseline, the PPL of an LSTM-LM without any domain adaptation is given.

As comparison to UniFA, conventional LDA feature-based domain adaptation with LDA features calculated from a window size of 50, 100, and 200 words was used. The adaptation parameters in fLHUC were calculated from the adaptation features. fLHUC-LSTM showed slight improvements on the LSTM-LM for the subtitle set and TED-LIUM for longer context window lengths. On TED-LIUM, a window size of 200 words led to a 23% and 12% PPL reduction for the validation set and test set compared with a 50-word window size.

The SSN in UniFA had 300 units and used a single hidden layer. For UniFA, the same window sizes as with fLHUC-LSTM were used. Comparing the PPL with other methods, UniFA achieved a significantly lower PPL on the subtitle set

Table 19: PPL results for CSJ. The number in brackets denotes the context window size.

Model	Heldout	Test 1	Test 2	Test 3
LSTM	37.88	40.52	41.79	41.01
fLHUC-LSTM (50)	36.64	39.39	40.79	40.15
fLHUC-LSTM (100)	36.44	39.03	**40.13**	40.19
fLHUC-LSTM (200)	**36.16**	**38.90**	40.40	39.94
UniFA (50)	36.89	39.98	40.70	40.28
UniFA (100)	36.72	39.52	40.79	**39.38**
UniFA (200)	36.65	39.48	40.61	39.60

and TED-LIUM. The PPL reduction ranged from 26% to 19% on the subtitle set. Especially for small context window sizes, UniFA showed a high PPL reduction compared with fLHUC-LSTM. On TED-LIUM, it consistently outperformed the LSTM-LM baseline and could further improve on fLHUC-LSTM.

Table 19 shows the PPL results for CSJ. The models using context adaptation with LDA features consistently achieved lower PPLs compared with the baseline LSTM-LM. As for TED talks, the model using the longest context window size for the context representation had the lowest PPL.

For the experiments on CSJ, an the SSN with 300 units and two hidden layers was used in UniFA. The UniFA models achieved a comparable PPL as fLHUC-LSTM on CSJ. There was not a large improvement on fLHUC-LSTM as with TED talks. However, all models consistently improved on the LSTM-LM.

5.6.3 Rescoring Results

Another very important metric for LMs used in ASR systems is their improvement of the recognition result after rescoring in terms of WER. For this purpose, Kaldi's TED-LIUM task was used for 100-best rescoring. The 1-best result was obtained with a trigram LM that is part of the TED-LIUM recipe [146]. The trigram was trained on different data than TED talks. Table 20 shows the WER for the 1-best decoding result and after 100-best rescoring for all models. The baseline LSTM-LM achieved already a great WER reduction compared with the

Table 20: WER after 100-best rescoring for TED-LIUM. The number in brackets is the context window size.

Model	val WER[%]	test WER[%]
1-best	16.3	15.1
LSTM-LM	14.2	12.1
fLHUC-LSTM (50)	14.3	12.2
fLHUC-LSTM (100)	14.0	12.2
fLHUC-LSTM (200)	14.0	11.9
UniFA (50)	13.8	**11.8**
UniFA (100)	**13.7**	12.0
UniFA (200)	13.9	12.0

1-best result. For fLHUC-LSTM, the ASR results were analogue to the PPL results. With a short context window, fLHUC-LSTM did not show a lower WER than the LSTM-LM. The method could only improve on the baseline with a larger context window.

UniFA showed an improvement on an LSTM-LM across all context window sizes. The validation set WER was in all cases better than for fLHUC-LSTM and it only fell behind fLHUC-LSTM for the longest window size of 200 words. A matched-pair significance test showed a significant improvement of UniFA(50) on an LSTM-LM at a significance level $p < 5\%$.

As Table 9 shows, for TED-LIUM the average utterance length is 25 words. This corresponds exactly to half the window length of the best UniFA model. LMs do not benefit much from a very large context window in rescoring because there might be many recognition errors from preceding utterances in the context window. However, these errors seem not to harm the model at shorter context windows and the extractor network can successfully provide additional (short-term topic) information.

Table 21 shows the WER after 100 best rescoring on CSJ. Since CSJ does not have a dedicated validation set, the interpolation parameters for the trigram LM and the NNLMs were optimised on test 3. The obtained parameters were applied to the remaining test sets as well. Rescoring with an LSTM-LM lead to a relative WER reduction of 13 to 14% compared with the 1-best result. fLHUC-

Table 21: WER after 100-best rescoring for CSJ. The number in brackets denotes the context window size.

Model	Test 1 WER[%]	Test 2 WER[%]	Test 3 WER[%]
1-best	12.26	9.34	12.22
LSTM	10.71	8.08	10.49
fLHUC-LSTM (50)	10.70	7.91	10.49
fLHUC-LSTM (100)	10.63	**7.85**	10.34
fLHUC-LSTM (200)	10.62	7.93	10.38
UniFA (50)	10.65	7.90	10.44
UniFA (100)	**10.60**	**7.85**	10.32
UniFA (200)	10.65	7.99	**10.29**

LSTM slightly improved on the LSTM-LM baseline result. UniFA achieved a comparable or slightly lower WER as fLHUC-LSTM.

5.6.4 Analysis of fLHUC Adaptation Parameters

In addition to comparing each model's effectiveness in ASR, further investigation of the adaptation parameters learned by each model is provided. Basis for this comparison are the best models with LDA features and SSN from Chapter 5.6.3. TED talks was used as data set. Figure 30 shows a visualisation for the fLHUC adaptation parameters before the sigmoid function when learned from LDA features and from the SSN with different window lengths. As data for the comparison, talks six, seven and eight in the TED talks subtitle test set were used. Figure 30 (a) and (b) show the LDA features for a 200-word window and the adaptation parameters the network learned to extracted from them, respectively. The LDA features show three distinct topics with high activation for each of the talks. The adaptation parameters have values around zero for most of the nodes in the adaptation layer, which means that the node passes through its input. However, certain nodes show either high (red horizontal lines) or low (blue horizontal lines) activation depending on the active LDA topic. This means that these nodes amplify or attenuate their input, respectively. This result also nicely illustrates the effectiveness of fLHUC. Some LSTM output nodes get am-

plified, while others get attenuated depending on the active topic. This means that certain words will see a higher activation in the output layer depending on the context which should correspond to a higher probability of these words. This is successful realisation of a context dependent LM because depending on the context some words will be more likely than others.

Figure 30 (c) shows the adaptation parameters when learned to extract with the SSN from a context window of 50 words[4]. Similar to the adaptation parameters learned from LDA features, there are also some nodes in the adaptation layer that receive constant high or low activations during each talk. This suggests that the SSN learns to capture a global topic-like context spanning several thousand words. However, the adaptation parameters from the SSN appear noisier compared with the ones learned from the LDA features. This means that the SSN output changes more frequently depending on the context compared with LDA features. This suggests that in addition to the long-term context the SSN can also capture more local context. This leads to a more frequent regulation of each node in the adaptation layer. As the experimental results showed, LDA features with shorter context windows were unable to capture these local topic changes but it was important for reducing PPL and WER.

5.6.5 Combination of UniFA with Conventional LDA Feature-based Domain Adaptation

The comparison of fLHUC-LSTM and UniFA showed different tendencies. fLHUC-LSTM had better performance with long context windows whereas UniFA had better performance for short context windows. This might come from the different training schemes. The LDA topic model was trained on larger document units compared with the SSN (whole talk vs. 50–200 word context window). However, for feature calculation, both methods used the same context window. This means both should have produced a feature from the same amount of context information. It is possible that the feature from the SSN was only an average of the context window for longer window sizes because the network could not learn to extract any specific information from these windows. It is thus worthwhile investigating a combination of both LDA features and the SSN. (115) was modified

[4]A threshold was to values around minus two to improve the visualisation.

Table 22: PPL for subtitle and TED-LIUM validation and test set. The number in brackets denotes the context window size.

Model	Subtitle PPL		TED-LIUM PPL	
	val	test	val	test
LSTM-LM	51.58	51.98	209.34	156.29
fLHUC-LSTM (200)	46.76	46.98	173.51	135.68
UniFA (50)	35.74	36.82	**144.09**	120.14
UniFA (100)	38.27	37.66	165.55	129.21
UniFA (200)	37.18	37.82	168.79	135.34
UniFA (50) + LDA (200)	**34.73**	**35.73**	153.23	**117.66**
UniFA (100) + LDA (200)	37.14	37.85	153.23	125.43
UniFA (200) + LDA (200)	42.94	42.93	156.28	133.03

in such way that the input to the sigmoid function was the output of the SSN as well as the LDA features after a transformation by a linear layer.

PPL results for a combination of LDA features with UniFA framework are shown in Table 22. The combination of both context representations showed a slight reduction of PPL. Especially on the TED-LIUM set, the models improved using the additional information from the LDA features. This suggests that the LDA features captured some complementary information from the SSN. In addition, the LDA features might have contained more invariant information compared with the context from the SSN.

Table 23 shows the results for rescoring for the combination of both LDA and SSN. For UniFA with a 50 word context window, the LDA features from the 200 word context window did not help to further reduce the WER. For UniFA with longer context windows (100 and 200 words), the WER was closer to the result of fLHUC-LSTM.

5.6.6 Combination of UniFA with factLSTM

The UniFA framework uses fLHUC as adaptation layer. However, the framework is not limited to fLHUC. A combination with other adaptation layers is also possible. As an example of such modification, here, the combination with

Table 23: WER after 100-best rescoring for TED-LIUM. The number in brackets is the context window size.

Model	val WER [%]	test WER [%]
1-best	16.3	15.1
LSTM-LM	14.2	12.1
fLHUC-LSTM (200)	14.0	11.9
UniFA (50)	13.8	**11.8**
UniFA (100)	**13.7**	12.0
UniFA (200)	13.9	12.0
UniFA (50) + LDA (200)	13.9	11.9
UniFA (100) + LDA (200)	13.9	12.0
UniFA (200) + LDA (200)	13.8	11.9

factLSTM is shown. factLSTM showed the best results of all adaptation layers with conventional LDA in Chapter 5.5. When combining factLSTM with UniFA, the factor weights are calculated from the output of the SSN.

Table 24 shows PPL results for conventional factLSTM with 10 and 15 factors for the TED talk dataset. Increasing the number of factors showed to consistently improve the results. The last line in Table 24 shows the results for the UniFA framework using factLSTM and 10 factors. These PPL results are close to factLSTM with 15 factors.

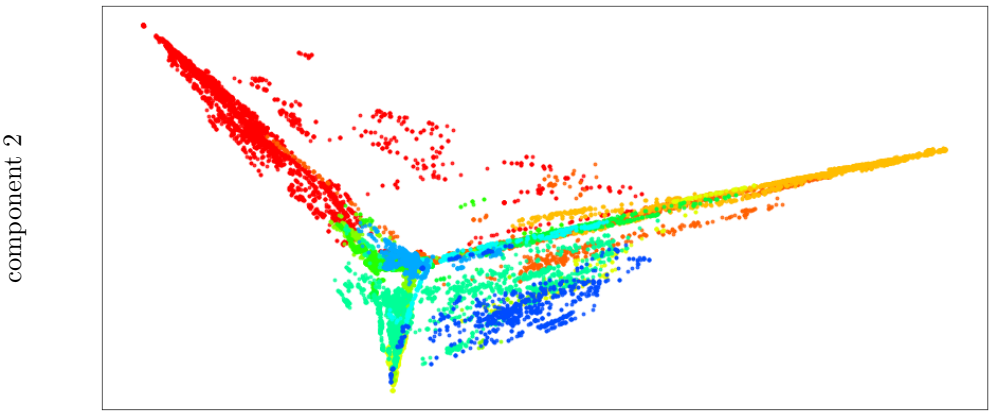
Table 25 shows the WER after 100-best rescoring. The WER after rescoring of UniFA with hidden layer factorisation is on the same level as for factLSTM with conventional LDA features and 15 factors. This means that a model with significantly less parameters can be used compared with LDA feature-based factLSTM. For the TED talk dataset a reduction of five factors reduces the number of model parameters by 64.5M. The additional number of parameters introduced by the SSN is approximately 92K. For the overall number of model parameters, the additional parameters introduced by the SSN is negligible.

Table 24: PPL results for TED-LIUM. LDA features from 200 word sliding window and 50 topics. SSN with 50 word sliding window.

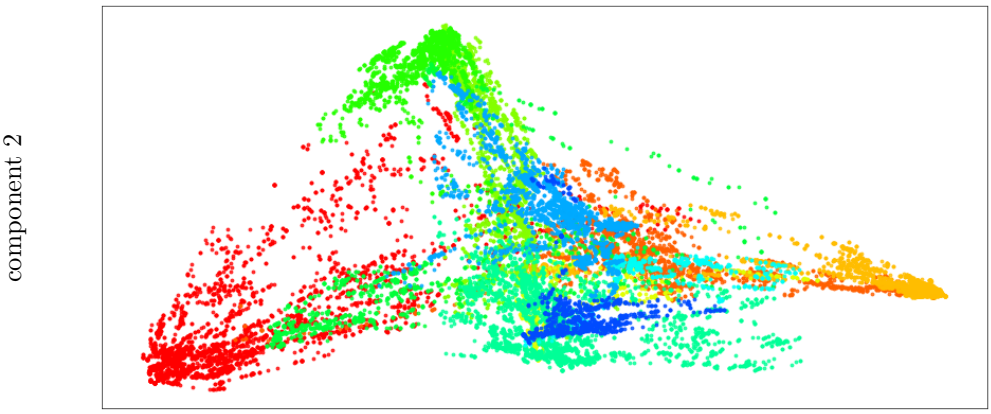
Model	Subtitle PPL		TED-LIUM PPL	
	val	test	val	test
trigram	179.39	156.41	251.89	222.05
LSTM-LM	51.58	51.98	209.34	156.29
factLSTM (10)	36.84	37.92	135.76	109.78
factLSTM (15)	29.50	**31.74**	**129.98**	**101.04**
UniFA + factLSTM (10)	**27.99**	31.80	131.68	109.81

Table 25: WER results for TED-LIUM. LDA features from 200 word sliding window and 50 topics. SSN with 50 word sliding window.

Model	val WER [%]	test WER [%]
1-best	16.3	15.1
LSTM-LM	14.2	12.1
factLSTM (10)	13.9	11.7
factLSTM (15)	**13.8**	**11.6**
UniFA + factLSTM (10)	**13.8**	**11.6**

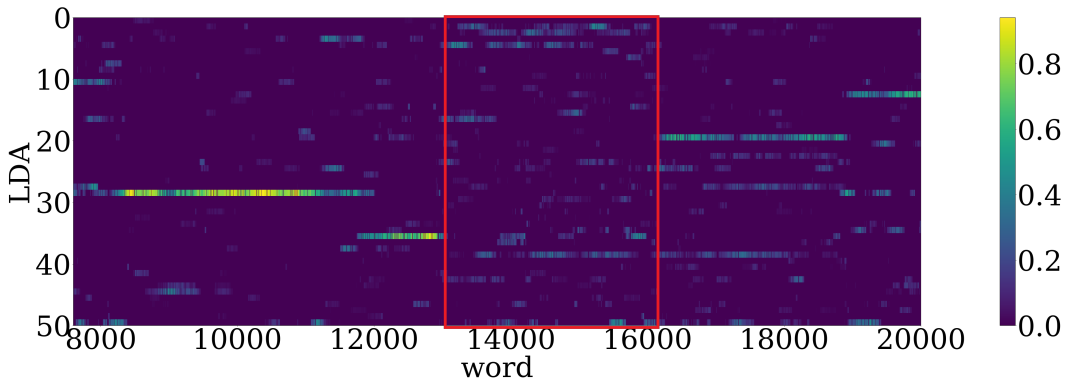


(a)

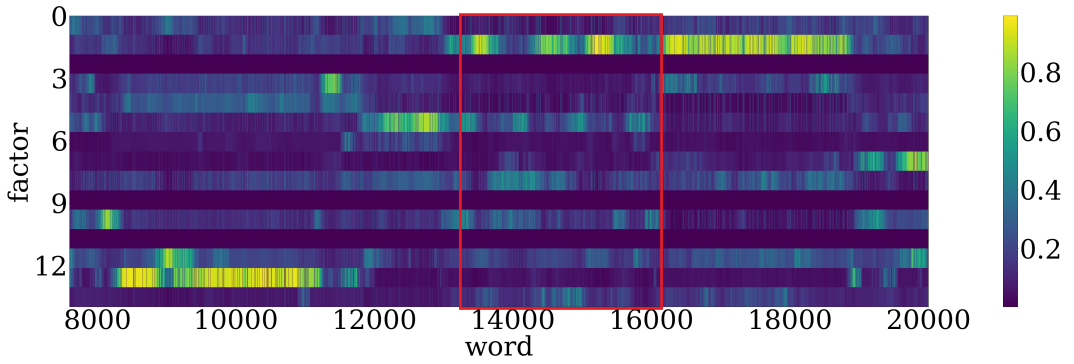


(b)

Figure 28: PCA plots of LDA features (a) and factor weights (b) for the subtitle TED talks test set. Each colour represents a different talk in the test set.



(a)



(b)

Figure 29: Comparison of LDA features (a) and factor weights (b) for part of the test set (x axis corresponds to word index).

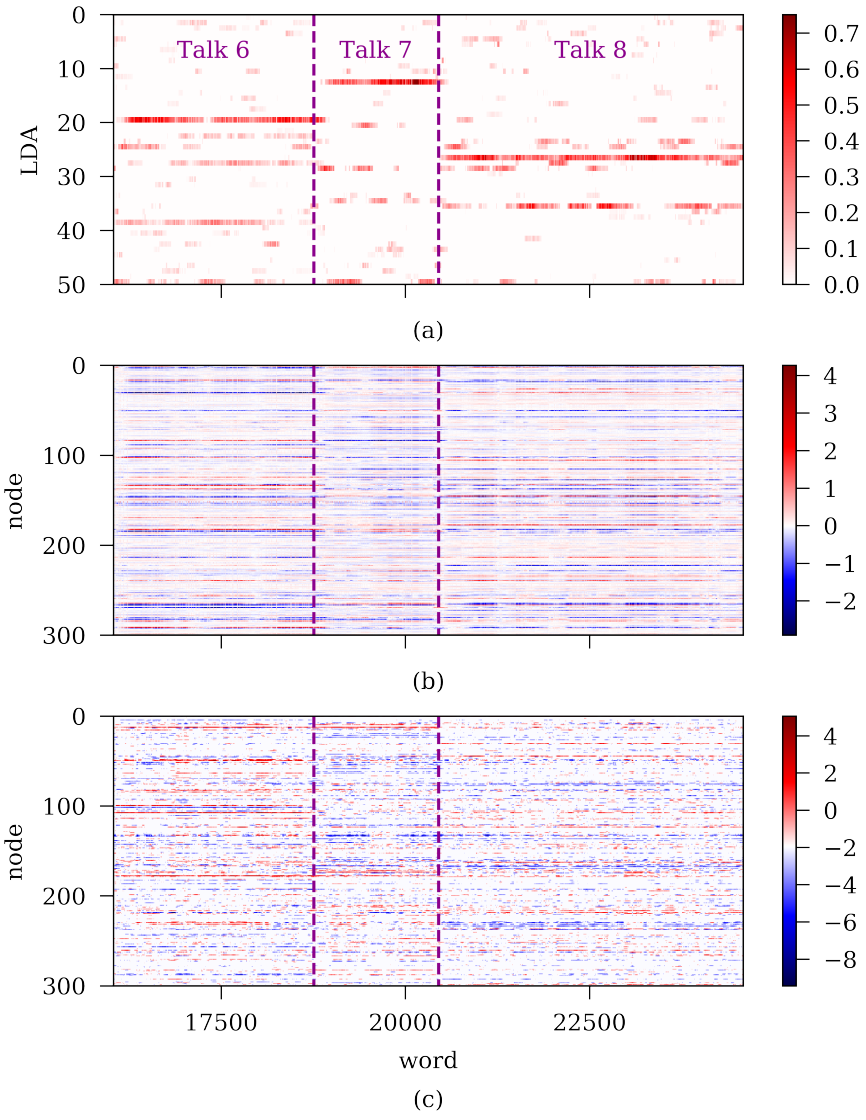


Figure 30: Visualisation of (a) LDA features for 200-word window size, fLHUC adaptation parameters before the sigmoid function from (b) LDA features from 200-word window size and (c) SSN(50) for talks six, seven and eight in the subtitle test set.

6. Conclusion and Outlook

Within the scope of this thesis, different kinds of context information and neural network architectures for exploiting context features were investigated. In the experiments, always a combination of a specific type of context feature along with a model structure was investigated. The conclusions are given in the following for each combination individually. After the conclusion from the experimental results, an outlook on future work follows in the remainder of this section.

6.1 Usage of Prosodic Features for Language Models

The usage of prosodic information was investigated in a re-training scheme for an RNN-LM. The investigated method could improve the PPL of a base model (re-)trained on text data only. Comparing the results of the re-training with prior research, a similar reduction of PPL and WER was achieved. However, the benefit of the approach investigated here is that it requires less transcribed audio data. The approach allows to train an LM with a large vocabulary on text data and adapt the weights of the part where prosodic information is available.

On the negative side, features that lead to a high PPL reduction did not always result in a WER improvement. There might have been errors with the word-frame alignment which deteriorate the quality of the features. However, so far no definite answer can be given as to why prosodic information was not helpful to reduce WER in many cases. Preliminary experiments on CSJ showed the same tendency as the results on MIT-OCW but a more in-depth analysis would require a case-by-case analysis of the rescoring results and this is left for the moment for future work.

In addition, MIT-OCW was a task very well suited for this training scheme. The LM training data consists of lot of out-of-domain training data. Re-training with the AM training data resulted already in a PPL reduction. This means, model-based domain adaptation helped to improve the LM.

The benefit of the re-training scheme is, that by using only a small amount of transcribed audio data, the performance of an LM trained on text data only can be improved. In addition, the method does not require any specific annotations in the corpus, for instance, prosody or part of speech tags, because the features

used throughout the experiments can be calculated directly from the audio data.

6.2 Cache Extensions for Language Models

Two conceptually different cache extensions using a continuous neural cache and BoW cache were evaluated when connected to LSTMs. However, in the experiments the effect of the cache on the LSTM was not beneficial. The continuous neural cache was successful in reducing the PPL but did not show a significant benefit in rescoring. With the BoW cache, although PPL evaluation was only slightly worse compared to the baseline, during rescoring the cache severely degraded the WER. Overall, the conclusion was that the investigated cache implementations were not effective for the dataset in the experiments and on the task of rescoring in ASR.

6.3 Feature-based Domain Adaptation of Language Models

A comparison and discussion of different feature-based domain adaptation techniques for NNLMs was provided. Feature-based methods have in comparison to model-based methods the advantage that they can be applied in an unsupervised manner. In practice, such methods are preferable because they do not require any domain information in the training, validation and test data which can only be made by experts. The LDA topic model used in the comparison can also be trained in an unsupervised manner.

The compared methods use different strategies. There is domain adaptation via an additive bias, a multiplicative gating and a factorisation of the output layer. Bias-based adaptation as used in fLHN-LSTM was originally proposed for model-based adaptation, but as the experiments showed it can also be successfully applied in feature-based adaptation. It improved considerably on an LSTM-LM baseline on three different data sets. Simple bias adaptation with contLSTM did not show to improve on an LSTM-LM in the experiments.

Model adaptation with fLHUC-LSTM improved on the baseline LSTM. As feature-based domain adaptation approach, fLHUC achieved a larger relative reduction in WER than in previous research as model-based adaptation. The method successfully benefited from topic information provided by an LDA topic

model and consistently improved on the baseline LSTM-LM. fLHUCB, which is a simple approach for combining fLHUC with bias adaptation, showed that it can improve on both individual methods. When the training and test conditions are matched and sufficient data is available, this method performed better than domain adaptation with a bias or fLHUC. This shows that bias and gating have complementary information that can help to improve performance.

On most of the datasets used in the experiments, factLSTM showed superior results compared with other methods for feature-based domain adaptation. The advantage of fLHN-LSTM, fLHUC-LSTM, and fLHUCB over factLSTM is the smaller number of parameters. In addition the convergence speed during model training was also faster. But none of models was able to achieve a similar PPL or WER as factLSTM on the PTB and TED talks data sets.

The last investigated domain adaptation method was a unified framework for feature-based LM domain adaptation based on an SSN and fLHUC. The results on a dataset of TED talks showed PPL reductions compared with a baseline LSTM-LM and feature-based domain adaptation with LDA features. In the rescoring experiments on the TED-LIUM dataset, UniFA consistently improved on an LSTM-LM baseline and showed lower WER than conventional feature-based adaptation in all but one cases.

Investigating the adaptation parameters that the network learns to extract from LDA features or the SSN, there were similar patterns in both cases but the SSN learned to extract more short term context. Capturing short-term changes in the topic was helpful to reduce PPL and WER. This was also a good visualisation for the effectiveness of domain adaptation. Long-term high or low activations of individual nodes in the adaptation layer correspond to a global topic. These distinct network activations show that the network learns to adjust the output distribution for certain topics.

6.4 Future Work

Out of all investigated context information, domain adaptation of NNLM with topic features was the most successful approach. The experimental results showed a higher consistency compared with results from prosodic features and were clearly superior to the cache-based methods. However, in this thesis the use

of prosodic information was only investigated as model-based adaptation for an RNN-LM. Joint training with prosodic features from scratch is equivalent to feature-based adaptation. This should be investigated with the network architectures that showed effective for feature-based domain adaptation.

In addition, the effect of prosodic information and topic information was only investigated in isolation. A joint investigation of both features should be conducted. Such investigation can also include the use of an SSN to extract prosodic information from the audio signal itself. Such extraction has the advantage that no feature-engineering is required to select prosodic features.

Regarding the results from UniFA, an improvement on the current context representation learned by the SSN has to be considered. So far, longer context window sizes were not helpful to further improve the results. The LDA features were superior when a context window length of 200 words was used. Here, a pretraining of the SSN for instance as an auto-encoder could be helpful and should be investigated.

Finally, a general remark on the application of context information to rescoring. Learning a more robust context representation has to be considered. Across different experiments, in case where large PPL reductions were achieved, the result could not always be carried over to large WER improvements. This has different reasons. The LMs are trained on the ground truth data and all features used for training are also calculated on this data. Increasing the robustness could be achieved by introducing distortions in the training data. A possible approach how these distortions could be generated was presented in [69]. The training objective of the LM is maximisation of the next word probability whereas the objective in rescoring should be to find the hypothesis that has the least amount of errors with respect to an unknown target phrase. Another possible way for improvement could in this respect be training the LM with the objective of WER reduction and not PPL reduction.

Acknowledgements

I would have not been able to accomplish this thesis without the continuous support of many people that I would like to express my sincere appreciation to. First and foremost, I would like to thank my family for their support during this adventure.

I would like to thank Prof. Yuji Matsumoto along with all other assistant professors and the students in the Computational Linguistics Laboratory at NAIST. I am very glad that you welcomed me in your laboratory and I enjoyed very much the support of other PhD students and helping each other along the way. It was always a relief to see that all of us had the same problems for some part and we were able to support each other.

Next are Dr. Tomohiro Nakatani, Dr. Atsunori Ogawa, Dr. Marc Delcroix and Dr. Tomoharu Iwata from NTT Communication Science Laboratories. They were co-authors on my publications and gave me many fruitful advise for my research. I am also very thankful that I could use the infrastructure at NTT Communication Science Laboratires for my research.

I would like to thank all the people I met in Nara as fellow interns of NTT Communication Science Laboratories during my PhD. This includes the following people Hendrik Meutzner, Ferenc Kazinczi, Christian Hümmer, Juan Azcarreta, Kateřina Žmolíková, Lukas Drude, Christopher Schmura, Thilo von Neumann, Austin Windsor and Aswin Shanmugam Subramanian. Many of them became very good friends I still keep regular contact with. Among these people, a special thanks goes to Ferenc and Juan for proof reading my thesis.

Michael Hentschel
Nara, Japan
March 15, 2019

References

[1] Tanel Alumäe. Multi-domain Neural Network Language Model. In *INTERSPEECH*, volume 13, pages 2182–2186, 2013.

[2] Tanel Alumäe and Mikko Kurimo. Domain Adaptation of Maximum Entropy Language Models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 301–306, 2010.

[3] Xavier L. Aubert. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech and Language*, 16(1):89 – 114, 2002.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

[6] L. E. Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

[7] Peter Bell, Mark JF Gales, Thomas Hain, Jonathan Kilgour, Pierre Lanchantin, Xunying Liu, Andrew McParland, Steve Renals, Oscar Saz, Mirjam Wester, et al. The MGB challenge: Evaluating multi-genre broadcast media recognition. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 687–693. IEEE, 2015.

[8] Jerome R Bellegarda. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42(1):93–108, 2004.

[9] Richard Bellman and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[10] Jacob Benesty, Shoji Makino, and Jingdong Chen, editors. *Speech Enhancement*. Springer-Verlag, Berlin Heidelberg, 2005.

[11] Yoshua Bengio and Réjean Ducharme. A neural probabilistic language model. In *Proceedings of Advances in Neural Information Processing Systems*, volume 13, pages 932–938, 2001.

[12] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[14] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71, 1996.

[15] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science & Business Media, 2006.

[16] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[17] Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479, 1992.

[18] Oscar Chan and Roberto Togneri. Prosodic Features for a Maximum Entropy Language Model. In *INTERSPEECH*, pages 1858–1861, 2006.

[19] Jingdong Chen, Jacob Benesty, Yiteng Arden Huang, and Eric J Diethorn. Fundamentals of Noise Reduction. In Jacob Benesty, M. Mohan Sondhi, and Yiteng Huang, editors, *Springer Handbook of Speech Processing*, pages 843–872. Springer-Verlag, Berlin Heidelberg, 2008.

[20] Minmin Chen. Efficient vector representation for documents through corruption. In *International Conference on Learning Representations (ICLR)*, 2017.

[21] Stanley F. Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 310–318, 1996.

[22] Stanley F. Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modelingmpirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–394, 1999.

[23] Xie Chen, Tian Tan, Xunying Liu, Pierre Lanchantin, Moquan Wan, Mark JF Gales, and Philip C Woodland. Recurrent Neural Network Language Model Adaptation for Multi-Genre Broadcast Speech Recognition. In *INTERSPEECH*, pages 3511–3515, 2015.

[24] P. Clarkson and A. J. Robinson. Language model adaptation using mixtures and an exponentially decaying cache. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 799–802, 1997.

[25] Steven Davis and Paul Mermelstein. Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, August 1980.

[26] Alain de Cheveigné and Hideki Kawahara. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.

[27] Salil Deena, Madina Hasan, Mortaza Doulaty, Oscar Saz, and Thomas Hain. Combining Feature and Model-Based Adaptation of RNNLMs for Multi-Genre Broadcast Speech Recognition. In *INTERSPEECH*, pages 2343–2347, 2016.

[28] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[29] Marc Delcroix, Keisuke Kinoshita, Atsunori Ogawa, Christian Hümmer, and Tomohiro Nakatani. Context Adaptive Neural Network Based Acoustic Models for Rapid Adaptation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(5):895–908, May 2018.

[30] Marc Delcroix, Keisuke Kinoshita, Chengzhu Yu, Atsunori Ogawa, Takuya Yoshioka, and Tomohiro Nakatani. Context adaptive deep neural networks for fast acoustic model adaptation in noisy conditions. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5270–5274, 2016.

[31] Stephen Della Pietra, Vincent Della Pietra, Robert L. Mercer, and Salim Roukos. Adaptive language modeling using minimum discriminant estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 633–636, 1992.

[32] Misha Denil, Alban Demiraj, Nal Kalchbrenner, Phil Blunsom, and Nando de Freitas. Modelling, Visualising and Summarising Documents with a Single Convolutional Neural Network. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[33] John Duchi, Elad Hazan, and Yoram Singer. Adative Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[34] Richard O. Duda, Peter E. Hart, and David G. Strok. *Pattern Classification, Second Edition*. John Wiley and Sons, 2001.

[35] Daniel PW Ellis and Byunk Suk Lee. Noise Robust Pitch Tracking by Subband Autocorrelation Classification. In *INTERSPEECH*, 2012.

[36] Jefferey Elman. Finding Structure in Time. *Cognitive Science*, 14:179–211, 1990.

[37] Marcello Federico, Luisa Bentivogli, Paul Michael, and Stüker Sebastian. Overview of the IWSLT 2011 evaluation campaign. In *International Workshop on Spoken Language Translation (IWSLT)*, 2011.

[38] Tong Fu, Yang Han, Xiangang Li, Yi Liu, and Xihong Wu. Integrating Prosodic Infromation into Recurrrent Neural Network Language Model for Speech Recognition. In *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1194–1197. IEEE, 2015.

[39] Venkata Ramana Rao Gadde. Modeling word durations. In *INTERSPEECH*, pages 601–604, 2000.

[40] Siva Reddy Gangireddy, Steve Renals, Yoshihiko Nankaku, and Akinobu Lee. Prosodically-enhanced Recurrent Neural Network Language Models. In *INTERSPEECH*, pages 2390–2394, 2015.

[41] Siva Reddy Gangireddy, Pawel Swietojanski, Peter Bell, and Steve Renals. Unsupervised Adaptation of Recurrent Neural Network Language Models. In *INTERSPEECH*, pages 2333–2337, 2016.

[42] Roberto Gemello, Franco Mana, Stefano Scanzio, Pietro Laface, and Renato De Mori. Linear hidden transformations for adaptation of hybrid ANN/HMM models. *Speech Communication*, 49(10):827–835, 2007.

[43] Pegah Ghahremani, Bagher BabaAli, Daniel Povey, Korbinian Riedhammer, Jan Trmal, and Sanjeev Khudanpur. A pitch extraction algorithm tuned for automatic speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2494–2498, 2014.

[44] James Glass, Timothy J. Hazen, Scott Cyphers, Igor Malioutov, David Huynh, and Regina Barzilay. Recent Progress in the MIT Spoken Lecture Processing Project. In *INTERSPEECH*, pages 2553–2556, 2007.

[45] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3 and 4):237–264, 1953.

[46] Joshua T. Goodman. A bit of progress in language modeling. *Computer Speech and Language*, 15(4):403–434, 2001.

[47] Joshua T. Goodman. Classes for fast maximum entropy training. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 561–564, May 2001.

[48] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *International Conference on Learning Representations (ICLR)*, 2017.

[49] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

[50] Md Akmal Haidar and Mikko Kurimo. Recurrent Neural Network Language Model With Incremental Updated Context Information Generated Using Bag-of-Words Representation. In *INTERSPEECH*, pages 3504–3508, 2016.

[51] Md Akmal Haidar and Douglas O'Shaughnessy. Topic n-gram count language model adaptation for speech recognition. In *Spoken Language Technology Workshop (SLT)*, pages 165–169. IEEE, 2012.

[52] Peter. E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[54] Aaron Heidel, Hung-an Chang, Lin-shan Lee, et al. Language model adaptation using latent dirichlet allocation and an efficient topic inference algorithm. In *INTERSPEECH*, pages 2361–2364, 2007.

[55] Michael Hentschel, Marc Delcroix, Atsunori Ogawa, Tomoharu Iwata, and Tomohiro Nakatani. Factorised Hidden Layer Based Domain Adaptation for Recurrent Neural Network Language Models. In *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1940–1944, Honolulu, Hawaii, 12–15 November 2018.

[56] Michael Hentschel, Marc Delcroix, Atsunori Ogawa, Tomoharu Iwata, and Tomohiro Nakatani. A Unified Framework for Feature-based Domain Adaptation of Neural Network Language Models. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton , UK, 12–17 May 2019.

[57] Michael Hentschel, Marc Delcroix, Atsunori Ogawa, Tomoharu Iwata, and Tomohiro Nakatani. Feature Based Domain Adaptation for Neural Network Language Models with Factorised Hidden Layers. *IEICE Transactions on Information and Systems*, 102(3), March 2019.

[58] Michael Hentschel, Marc Delcroix, Atsunori Ogawa, and Tomohiro Nakatani. Feature Based Learning Hidden Unit Contributions for Domain Adaptation of RNN-LMs. In *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1692–1696, Honolulu, Hawaii, 12–15 November 2018.

[59] Michael Hentschel, Atsunori Ogawa, Marc Delcroix, and Tomohiro Nakatani. Evaluation of Various Cache Extensions for LSTM Based Language Models. In *Special Interest Group Spoken Language Processing Research Meeting*, number 116. Information Processing Society of Japan, Osaka, May 2017.

[60] Michael Hentschel, Atsunori Ogawa, Marc Delcroix, Tomohiro Nakatani, and Yuji Matsumoto. Exploiting Imbalanced Textual and Acoustic Data for Training Prosodically-enhanced RNNLMs. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 618–621, Kuala Lumpur, Malaysia, 12–15 December 2017.

[61] Hynek Hermansky. Perceptual linear predictive (PLP) analysis of speech. *The Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[62] James L. Hieronymus, David McKelvie, and Fergus McInnes. Use of acoustic sentence level and lexical stress in HSMM speech recognition. In *1992 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 225–227. IEEE, 1992.

[63] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[64] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[65] Christian Hofmann. Evaluation of a novel hmm training concept for reverberation-robust asr. SIM Project Work, 2010.

[66] Thomas Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the Twenty-Second Annual International SIGIR Conference*, pages 50–57, 1999.

[67] Takaaki Hori. NTT speech recognizer with outlook on the next generation: SOLON. In *Proc. NTT Workshop on Communication Scene Analysis, 2004*, 2004.

[68] Takaaki Hori and Atsushi Nakamura. *Speech Recognition Algorithms Using Weighted Finite-State Transducers*. Morgan & Claypool Publishers, 1st edition, 2013.

[69] Yinghui Huang, Abhinav Sethy, Kartik Audhkhasi, and Bhuvana Ramabhadran. Whole sentence neural language model. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6089–6093, 2018.

[70] Kazuki Irie, Shankar Kumar, Michael Nirschl, and Hank Liao. RADMM: recurrent adaptive mixture model with applications to domain robust language modeling. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6079–6083, 2018.

[71] Kazuki Irie, Ralf Schlüter, and Herman Ney. Bag-of-Words Input for Long History Representation in Neural Network-based Language Models for Speech Recognition. In *INTERSPEECH*, pages 2371–2375, 2015.

[72] Kazuki Irie, Zoltán Tüske, Tamar Alkhouli, Ralf Schlüter, and Hermann Ney. LSTM, GRU, Highway and a Bit of Attention: An Empirical Overview

for Language Modeling in Speech Recognition. In *INTERSPEECH*, pages 3519–3523, 2016.

[73] Frederick Jelinek. Continuous Speech Recognition by Statistical Methods. *Proceedings of the IEEE*, 64(4):532–556, April 1976.

[74] Frederick Jelinek and Robert L Mercer. Interpolated estimation of markov source parameters from sparse data. In *Workshop on Pattern Recognition in Practice*, pages 381–397, 1980.

[75] Frederick Jelinek, Bernard Mérialdo, Salim Roukos, and Martin Strauss. A Dynamic Language Model for Speech Recognition. In *Workshop on Speech and Natural Language*, HLT 1991, pages 293–295, 1991.

[76] Armand Joulin and Tomas Mikolov. Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 190–198, 2015.

[77] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A Convolutional Neural Network for Modelling Sentences. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1, pages 655–665, 2014.

[78] Slava 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*, 35(3):400–401, 1987.

[79] Walter Kellermann. *Lecture notes in Signal Processing for Speech and Audio*. Friedrich-Alexander-Universität Erlangen-Nürnberg.

[80] Sanjeev Khudanpur and Jun Wu. Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. *Computer Speech and Language*, 14(4):355–372, 2000.

[81] Yoon Kim. Convolutional Neural Networks for Sentence Classification. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[82] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 181–184, 1995.

[83] Roland Kuhn and Renato De Mori. A Cache-Based Natural Language Model for Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, 1990.

[84] Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *International Conference on Machine Learning (ICML)*, pages 1188–1196, 2014.

[85] Rui Lin, Shujie Liu, Muyun Yang, Mu Li, Ming Zhou, and Sheng Li. Hierarchical recurrent neural network for document modeling. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 899–907, 2015.

[86] Yang Liu and Feifan Liu. Unsupervised language model adaptation via topic modeling based on named entity hypotheses. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4921–4924, 2008.

[87] Kikuo Maekawa. Corpus of spontaneous Japanese: Its design and evaluation. In *ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition*, 2003.

[88] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[89] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. *arXiv preprint arXiv:1609.07843*, 2016.

[90] Thomas Mikolov. *Statistical Language Models based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

[91] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[92] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048, 2010.

[93] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

[94] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. In *Spoken Language Technology Workshop (SLT)*, volume 12, pages 234–239. IEEE, 2012.

[95] M. Minsky and S. Papert. Perceptrons: An Introduction to Computational Geometry. *MIT Press*, 1969.

[96] Takafumi Moriya, Tomohiro Tanaka, Takahiro Shinozaki, Shinji Watanabe, and Kevin Duh. Automation of system building for state-of-the-art large vocabulary speech recognition using evolution strategy. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 610–616. IEEE, 2015.

[97] Masami Nakamura and Kiyohiro Shikano. A study of English word category prediction based on neutral networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages 731–734, May 1989.

[98] Tomohiro Nakatani, Shigeaki Amano, Toshio Irino, Kentaro Ishizuka, and Tadahisa Kondo. A method for fundamental frequency estimation and voicing decision: Application to infant utterances recorded in real acoustical environments. *Speech Communication*, 50(3):203–214, 2008.

[99] Patrick A. Naylor and Nikolay D. Gaubitch. *Speech Dereverberation*. Springer Verlag, London, 2010.

[100] Hermann Ney and Ute Essen. On smoothing techniques for bigram-based natural language modelling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 825–829, 1991.

[101] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8(1):1 – 38, 1994.

[102] Elmar Nöth, Anton Batliner, Andreas Kießling, Ralf Kompe, and Heinrich Nieman. VERBMOBIL: The Use of Prosody in the Linguistic Components of a Speech Understanding System. *IEEE Transactions on Speech and Audio Processing*, 8(5):519–532, 2000.

[103] Mari Ostendorf, Colin W Wightman, and Nanette M Veilleux. Parse scoring with prosodic information: an analysis/synthesis approach. *Computer Speech and Language*, 7(3):193–210, 1993.

[104] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables, and Stochastic Processes, Fourth Edition*. McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill, New York, 2002.

[105] Junho Park, Xunying Liu, Mark JF Gales, and Phil C Woodland. Improved Neural Network Based Language Modelling and Adaptation. In *INTERSPEECH*, pages 1041–1044, 2010.

[106] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[107] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi Speech Recognition Toolkit. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, December 2011.

[108] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[109] Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10:187–228, 1996.

[110] Ronald Rosenfeld. Two Decades of Statistical Language Modeling: Where Do We Go from Here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000.

[111] Anthony Rousseau, Paul Deléglise, and Yannick Esteve. TED-LIUM: an Automatic Speech Recognition dedicated corpus. In *LREC*, pages 125–129, 2012.

[112] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[113] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[114] Lahiru Samarakoon and Khe Chai Sim. Subspace LHUC for fast adaptation of deep neural network acoustic models. In *INTERSPEECH*, pages 1593–1597, 2016.

[115] Holger Schwenk. Continuous space language models. *Computer Speech and Language*, 21(3):492–518, 2007.

[116] Holger Schwenk and Jean-Luc Gauvain. Connectionist language modeling for large vocabulary continuous speech recognition. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 765–768, 2002.

[117] Holger Schwenk and Jean-Luc Gauvain. Training neural network language models on very large corpora. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 201–208, 2005.

[118] Elizabeth Shriberg, Andreas Stolcke, Dilek Hakkani-Tür, and Gökhan Tür. Prosody-Based Automatic Segmentation of Speech into Sentences and Topics. *Speech Communication*, 32(1):127–154, 2000.

[119] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A Neural Network Approach to Context-Sensitive Generation of Conversational Responses. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 196–205, 2015.

[120] Daniel Soutner and Luděk Müller. Application of LSTM Neural Networks in Language Modelling. In *International Conference on Text, Speech and Dialogue*, pages 105–112. Springer, 2013.

[121] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway Networks. *arXiv preprint arXiv:1505.00387*, 2015.

[122] Karsten Steinhauer, Kai Alter, and Angela D Friederici. Brain potentials indicate immediate use of prosodic cues in natural speech processing. *Nature Neuroscience*, 2(2):191–196, 1999.

[123] Andreas Stolcke. SRILM – an extensible language modeling toolkit. In *International Conference on Speech and Language Processing*, pages 901–904, 2002.

[124] Andreas Stolcke, Elizabeth Shriberg, Dilek Z. Hakkani-Tür, and Gökhan Tür. Modeling the prosody of hidden events for improved word recognition. In *EUROSPEECH*, pages 307–310, 1999.

[125] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-To-End Memory Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2440–2448, 2015.

[126] Martin Sundermeyer. *Improvements in Language and Translation Modeling*. PhD thesis, RWTH Aachen University, 2016.

[127] Martin Sundermeyer, Hermann Ney, and Ralf Schlüter. From Feedforward to Recurrent LSTM Neural Networks for Language Modeling. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 23(3):517–529, 2015.

[128] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM Neural Networks for Language Modeling. In *INTERSPEECH*, pages 194–197, 2012.

[129] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 3104–3112, 2014.

[130] Pawel Swietojanski and Steve Renals. Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *Spoken Language Technology Workshop (SLT)*, pages 171–176. IEEE, 2014.

[131] David Talkin. A Robust Algorithm for Pitch Tracking (RAPT). In W. Bastiaan Kleijn and Kuldip Paliwal, editors, *Speech Coding and Synthesis*, pages 495–518. Elsevier Science Inc., New York, 1995.

[132] Yik-Cheung Tam and Tanja Schultz. Dynamic Language Model Adaptation using Variational Bayes Inference. In *INTERSPEECH*, pages 5–8, 2005.

[133] Ottokar Tilk and Tanel Alumäe. Multi-Domain Recurrent Neural Network Language Model for Medical Speech Recognition. In *Baltic HLT*, pages 149–152, 2014.

[134] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a Next-Generation Open Source Framework for Deep Learning. In *Workshop on Machine Learning Systems (LearningSys) in the Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[135] Ke Tran, Arianna Bisazza, and Christof Monz. Recurrent Memory Networks for Language Modeling. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 321–331, San Diego, California, June 2016. Association for Computational Linguistics.

[136] Nanette M. Veilleux, Mari Ostendorf, and Colin W. Wightman. Parse Scoring With Prosodic Information. In *1993 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1993.

[137] Dimitra Vergyri, Andreas Stolcke, Venkata Ramana Rao Gadde, Luciana Ferrer, and Elizabeth Shriberg. Prosodic knowledge sources for automatic speech recognition. In *2003 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 208–211, 2003.

[138] Karel Veselỳ, Shinji Watanabe, Katerina Žmolíková, Martin Karafiát, Lukáš Burget, and Jan Honza Černockỳ. Sequence summarizing neural network for speaker adaptation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5315–5319. IEEE, 2016.

[139] Olli Viikki and Kari Laurila. Cepstral domain segmental feature vector normalization for noise robust speech recognition. *Speech Communication*, 25(1):133–147, 1998.

[140] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2692–2700, 2015.

[141] Andrew Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[142] Shinji Watanabe, Tomoharu Iwata, Takaaki Hori, Atsushi Sako, and Yasuo Ariki. Topic tracking language model for speech recognition. *Computer Speech and Language*, 25(2):440–461, 2011.

[143] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[144] Paul J. Werbos. Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[145] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory Networks. *International Conference on Learning Representations (ICLR)*, 2015.

[146] Will Williams, Niranjani Prasad, David Mrva, Tom Ash, and Tony Robinson. Scaling recurrent neural network language models. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5391–5395. IEEE, 2015.

[147] Francisco Zamora-Martínez, S Espana-Boquera, MJ Castro-Bleda, and Renato De-Mori. Cache neural network language models based on long-distance dependencies for a spoken dialog system. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4993–4996, 2012.

[148] Jian Zhang, Xiaofeng Wu, Andy Way, and Qun Liu. Fast Gated Neural Domain Adaptation: Language Model as a Case Study. In *International Conference on Computational Linguistics (COLING)*, pages 1386–1397, 2016.

[149] Katerina Zmolikova, Marc Delcroix, Keisuke Kinoshita, Higuchi Takuya, Atsunori Ogawa, and Tomohiro Nakatani. Speaker-aware neural network based beamformer for speaker extraction in speech mixtures. In *INTERSPEECH*, pages 2655–2659, 2017.

List of Major Publications

Journal Papers

- Michael Hentschel, Marc Delcroix, Atsunori Ogawa, Tomoharu Iwata, and Tomohiro Nakatani. Feature Based Domain Adaptation for Neural Network Language Models with Factorised Hidden Layers. IEICE Transactions on Information and Systems, 102(3), March 2019.

Papers at International Conferences

- Michael Hentschel, Atsunori Ogawa, Marc Delcroix, Tomohiro Nakatani, and Yuj Matsumoto. Exploiting Imbalanced Textual and Acoustic Data for Training Prosodically-enhanced RNNLMs. In 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pages 618–621, Kuala Lumpur, Malaysia, 12–15 December 2017.

- Michael Hentschel, Marc Delcroix, Atsunori Ogawa, and Tomohiro Nakatani. Feature Based Learning Hidden Unit Contributions for Domain Adaptation of RNN-LMs. In 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pages 1692–1696, Honolulu, Hawaii, 12–15 November 2018.

- Michael Hentschel, Marc Delcroix, Atsunori Ogawa, Tomoharu Iwata, and Tomohiro Nakatani. Factorised Hidden Layer Based Domain Adaptation for Recurrent Neural Network Language Models. In 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pages 1940–1944, Honolulu, Hawaii, 12–15 November 2018.

- Michael Hentschel, Marc Delcroix, Atsunori Ogawa, Tomoharu Iwata, and Tomohiro Nakatani. A Unified Framework for Feature-based Domain Adaptation of Neural Network Language Models. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019.

Papers at Domestic Research Meetings

- Michael Hentschel, Atsunori Ogawa, Marc Delcroix, and Tomohiro Nakatani. Evaluation of Various Cache Extensions for LSTM Based Language Models. In Special Interest Group Spoken Language Processing Research Meeting, number 116, Information Processing Society of Japan, Osaka, May 2017.

Appendix

A. List of Abbreviations

ASR	Automatic Speech Recognition
DNN	Deep Neural Network
PLP	Perceptual Linear Prediction
MFCC	Mel Frequency Cepstral Coefficients
CMVN	Cepstral Mean and Variance Normalisation
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
EM	Expectation Maximisation
DFT	Discrete Fourier Transform
DCT	Discrete Cosine Transform
WER	Word Error Rate
PPL	Perplexity
AM	Acoustic Model
LM	Language Model
NN	Neural Network
NNLM	Neural Network Language Model
RNN	Recurrent Neural Network
RNN-LM	Recurrent Neural Network Language Model
LSTM	Long Short-Term Memory
LSTM-LM	Long Short-term Memory recurrent neural network Language Model
OOV	Out-Of-Vocabulary
BoW	Bag-of-Words
NC	Neural Cache
LDA	Latent Dirichlet Allocation
CNN	Convolutional Neural Network
NLP	Natural Language Processing
HMI	Human-Machine Interaction
LHN	Linear Hidden Network

fLHN	Feature-based Linear Hidden Network
LHUC	Learning Hidden Unit Contributions
fLHUC	Feature-based Learning Hidden Unit Contributions
PCA	Principal Component Analysis
fLHN-LSTM	Feature-based Linear Hidden Network Long Short-Term Memory language model
factLSTM	Factorised hidden layer Long Short-Term Memory language model
contLSTM	Context dependent Long Short-Term Memory language model
fLHUC-LSTM	Feature-based Learning Hidden Unit Contributions Long Short-Term Memory language model
fLHUCB	Feature-based Learning Hidden Unit Contributions Long Short-Term Memory language model with Bias adaptation
SSN	Sequence Summary Network
UniFA	Unified framework for Feature-based domain Adaptation of neural network language models

B. List of Mathematical Symbols and Notation

B.1 List of Mathematical Operators

$\text{argmax}(\cdot)$	The argmax function
$\text{softmax}(\cdot)$	The softmax function
$\sigma(\cdot)$	The sigmoid function
$(\cdot)^{\text{T}}$	Transpose of a matrix
$C(\cdot)$	The count function for the number of occurrences of a word sequence
$\text{lstm}(\cdot)$	An LSTM cell
$\mathbb{1}_{a=b}$	A function that evaluates to one if a is equal to b and zero for all other cases
$\mathcal{N}$	The normal distribution

e	The exponential function

B.2 List of Mathematical Symbols

$s[t]$	The time-discrete input signal to the speech recogniser
$\tilde{s}[t]$	The pre-processed time-discrete input signal
Ω	The circular frequency
$\mathcal{U}$	The vocabulary of the language model
w	A word from the word sequence
$\overline{w}$	A word sequence
$\boldsymbol{o}$	A Feature vector extracted from the input signal
$\overline{\boldsymbol{o}}$	A sequence of feature vectors
o	An element of the feature vector (that is one dimension)
D	Discount parameter for Kneser-Ney Smoothing
$\overline{h}$	The history for an M-Gram
$\overline{g}$	The shortened history for back-off
$\boldsymbol{w}$	A vector of words
C	The symbol for word classes
$\mathcal{C}$	The set of word classes
r	The count of an M-Gram
r^*	The modified count for the Good-Turing formula
λ	An HMM model
$\boldsymbol{\Theta}$	A GMM model
c	The mixture weight in a GMM
$\boldsymbol{\mu}$	Mean vector of a GMM
$\boldsymbol{\Sigma}$	Covariance matrix of a GMM
$\boldsymbol{\Phi}$	A set of parameters
Ω	class label
$\mathcal{S}$	The set of states of an HMM
s	A state of an HMM
$\overline{s}$	A sequence of states in an HMM
CE	the Cross Entropy training criterion
$\boldsymbol{e}$	The output error vector
$\boldsymbol{t}$	The training target label

η	The learning rate for error backpropagation
$\boldsymbol{i}_l$	The LSTM input gate
$\boldsymbol{f}_l$	The LSTM forget gate
$\boldsymbol{o}_l$	The LSTM output gate
$\boldsymbol{c}_l$	The LSTM cell state
$\boldsymbol{W}$	The weight matrix for gates in an LSTM
$\boldsymbol{b}$	A bias vector
$\boldsymbol{x}_l$	The input vector to NNLM
$\boldsymbol{w}_{l-1}$	The input word vector
$\boldsymbol{a}_l$	An auxiliary feature vector
$\boldsymbol{h}_l$	The hidden layer of an NNLM
$\boldsymbol{h}_l^{(\mathrm{a})}$	LHUC gating weights
$\boldsymbol{y}_l$	The input to the softmax function in an NNLM
$\hat{\boldsymbol{w}}_l$	The output word vector of an NNLM
$\boldsymbol{c}_l$	The output class vector for class-based NNLMs
$\boldsymbol{U}$	The transition matrix from the input to the hidden layer
$\boldsymbol{U}^{(\mathrm{w})}$	The word part of the transition matrix from the input to the hidden layer
$\boldsymbol{U}^{(\mathrm{h})}$	The hidden layer part of transition matrix from the input to the hidden layer in RNN-LMs
$\boldsymbol{b}^{(\mathrm{U,h})}$	The bias vector for the hidden-to-hidden input layer
$\boldsymbol{U}^{(\mathrm{a})}$	The input layer for auxiliary features
$\boldsymbol{b}^{(\mathrm{U,a})}$	The input layer bias for auxiliary features
$\boldsymbol{V}$	The output layer
$\boldsymbol{b}^{(\mathrm{V})}$	The bias vector for the output layer
$\boldsymbol{V}^{(\mathrm{w})}$	The word part of the transition matrix from the hidden layer to the output layer
$\boldsymbol{b}^{(\mathrm{V,w})}$	The bias vector for the word part for the hidden-to-output layer
$\boldsymbol{V}^{(\mathrm{c})}$	The class part of transition matrix from hidden-to-output layer
$\boldsymbol{V}^{(\mathrm{a})}$	The transition matrix for auxiliary features from the input to the output or adaptation layer

$\boldsymbol{b}^{(\mathrm{V,a})}$	The bias vector for auxiliary features from the input to the output or adaptation layer
$\boldsymbol{d}^{(\mathrm{LHUC})}$	The fLHUC adaptation layer output
$\boldsymbol{d}^{(\mathrm{bias})}$	The fLHN adaptation layer output
$\boldsymbol{\psi}_l$	The output vector of the neural cache
$\boldsymbol{c}_l^{(\mathrm{nc})}$	The output vector with word probabilities from the neural cache
$\boldsymbol{c}_l^{(\mathrm{u})}$	The output vector of the unigram cache
$\boldsymbol{c}_l^{(\mathrm{b})}$	The output vector of the bigram cache
$\boldsymbol{h}_l^{(\mathrm{nc})}$	The projected output vector of the neural cache
$\boldsymbol{h}_l^{(\mathrm{u})}$	The projected output vector of the unigram cache
$\boldsymbol{h}_l^{(\mathrm{b})}$	The projected output vector of the bigram cache
$\boldsymbol{U}^{(\mathrm{nc})}$	The transition matrix from the neural cache to the NNLM input
$\boldsymbol{U}^{(\mathrm{u})}$	The transition matrix from the unigram cache to the NNLM input
$\boldsymbol{U}^{(\mathrm{b})}$	The transition matrix from the bigram cache to the NNLM input
$\boldsymbol{V}^{(\mathrm{nc})}$	The transition matrix from the neural cache to the NNLM output
$\boldsymbol{V}^{(\mathrm{u})}$	The transition matrix from the unigram cache to the NNLM output
$\boldsymbol{V}^{(\mathrm{b})}$	The transition matrix from the bigram cache to the NNLM output
γ	The factor weight for factLSTM
$\boldsymbol{\gamma}$	The vector of all factor weights
$\boldsymbol{z}$	The output of a factorised layer
$\boldsymbol{L}^{(\mathrm{w})}$	The transition matrix of a factorised layer
$\boldsymbol{b}^{(\mathrm{L,w})}$	The bias vector of a factorised layer

C. Additional Perplexity Results for Prosodically-enhanced Recurrent Neural Network Language Models

In Chapter 3.1, prosodic information from fundamental frequency (F0) and signal power was introduced. Subsequently, Chapter 5.3 showed the experimental results on the MIT-OCW corpus when this information is applied to the re-training of RNN-LMs. This section shows the full list of prosodic features that were investigated. This includes additional features from F0 and signal power. Additional results for MIT-OCW with these prosodic features are also presented. In addition to model re-training, this section also shows the full list of results for models trained from scratch on the smaller MIT-OCW AM training set.

C.1 F0 Features

Chapter 3.1.1 introduced the calculation of mean and delta features from the F0 feature sequence $\overline{\boldsymbol{o}}_{\mathrm{F0}}$ were introduced. In addition, further features were calculated from the F0 feature sequence. Chapter 3.1.1 introduced delta coefficients as they are common in ASR systems. This includes:

1. Acceleration coefficients (f0a)

$$\Delta^2_{\mathrm{F0}}[l] = \frac{\Delta_{\mathrm{F0}}[l+1] - \Delta_{\mathrm{F0}}[l-1]}{2}. \tag{117}$$

   This is the second order derivative or often also called delta-delta coefficients.

2. The variance (σ_{F0}) for all F0 features frames N_l within one word w_l (f0v)

$$\sigma_{\mathrm{F0}}[l] = \frac{1}{N_l} \sum_{n=n_s}^{n_e} \overline{\boldsymbol{o}}_{\mathrm{F0}}[n] - \mu_{\mathrm{F0}}[l], \tag{118}$$

   where n_s is the corresponding start index and n_e the end index of word w_l in the F0 feature sequence.

3. The mean F0 (f0u) for all feature frames N_L of an utterance

$$\mu_{\text{F0,utt}}[l] = \frac{1}{N_L} \sum_{n=n_s}^{n_e} \overline{\boldsymbol{o}}_{\text{F0}}[n]. \tag{119}$$

   For utterance level mean f0u, the same feature is used for all words in one utterance.

Chapter 5.3 described a specific way to normalise f0m and f0d features in the experiments. In addition, other ways for normalisation were investigated. This includes:

- Linear and logarithmic scale of all features
- Using the voiced/unvoiced decision of the F0 estimator [98] to calculate the features for one word. If the decision was used, only those F0 feature frames that were considered as voiced were used to calculate the prosodic features.
- Normalising features with the global mean and variance or with the mean and variance per utterance to unit variance and zero mean.

For the F0 estimator, two implementations were available. An older version of the implementation that used a weak smoothing and a newer implementation that used a strong smoothing for the F0 values. Both lead to slightly different results. The old implementation is hereafter denoted as NakOld and the newer implementation is denoted as NakNew.

Tables 29 and 30 show the full set of PPL results for re-training or RNN-LMs with prosodic features on the MIT-OCW corpus. Table 29 shows the results for the validation set and Table 30 shows the results for the test set.

In addition to the re-training scheme as described in Chapter 5.3.1, all features were also investigated when trained from scratch only on the smaller MIT-OCW AM training set. Tables 31 and 32 show the PPLs for these models. As mentioned in Chapter 5.3.2, these PPLs cannot be directly compared with the models trained on the LM training set because of the difference in vocabulary size.

Feature	extractor	voiced	lin/global	lin/utt	log/global	log/utt
f0a	NakNew	no	177.33	183.39	170.64	183.48
		yes	183.60	183.50	175.74	183.35
	NakOld	no	175.70	183.25	170.92	183.34
		yes	183.81	183.13	174.18	183.38
f0d	NakNew	no	178.02	183.11	170.74	183.04
		yes	185.42	183.20	178.50	183.36
	NakOld	no	183.08	182.73	171.37	183.14
		yes	184.09	182.16	177.48	182.85
f0m	NakNew	no	183.40	183.15	183.82	183.51
		yes	182.44	181.98	183.49	182.68
	NakOld	no	181.89	181.61	181.53	180.90
		yes	180.67	180.36	182.91	181.86
f0u	NakNew	no	183.62	183.43	184.02	183.43
		yes	184.08	183.43	183.97	183.43
	NakOld	no	183.62	183.43	183.50	183.43
		yes	183.91	183.43	183.73	183.43
f0v	NakNew	no	183.78	183.52	183.40	183.47
		yes	183.33	183.51	183.58	183.53
	NakOld	no	183.52	182.41	182.53	182.33
		yes	183.65	183.07	183.03	183.14

Table 29: All PPL results for MIT-OCW with prosodic F0 features for the validation set.

Feature	extractor	voiced	lin/global	lin/utt	log/global	log/utt
f0a	NakNew	no	150.62	151.87	149.34	151.69
		yes	150.35	152.24	150.65	152.15
	NakOld	no	148.50	151.72	147.85	151.54
		yes	151.47	151.88	150.17	151.79
f0d	NakNew	no	149.26	152.06	149.27	152.19
		yes	151.27	150.83	149.64	152.03
	NakOld	no	151.17	149.51	148.45	150.55
		yes	150.17	149.95	149.34	151.58
f0m	NakNew	no	150.76	150.61	152.43	152.28
		yes	150.79	149.34	152.23	150.21
	NakOld	no	151.30	150.09	150.70	148.06
		yes	150.17	150.28	152.11	151.09
f0u	NakNew	no	151.14	152.40	152.92	152.40
		yes	152.67	152.40	152.72	152.40
	NakOld	no	152.35	152.40	152.24	152.40
		yes	152.58	152.40	152.75	152.40
f0v	NakNew	no	152.45	152.39	152.74	152.40
		yes	150.82	152.36	152.32	152.14
	NakOld	no	152.03	150.00	148.94	150.92
		yes	152.35	152.39	151.73	151.51

Table 30: All PPL results for MIT-OCW with prosodic F0 features for the test set.

Feature	extractor	voiced	lin/global	lin/utt	log/global	log/utt
f0a	NakNew	no	198.27	202.65	201.70	200.31
		yes	240.84	230.25	246.59	223.04
	NakOld	no	218.15	239.48	218.66	237.26
		yes	208.25	225.25	203.12	219.70
f0d	NakNew	no	209.20	200.74	204.75	231.64
		yes	202.78	202.51	217.17	199.56
	NakOld	no	204.10	198.53	337.84	201.85
		yes	201.72	200.91	200.95	200.48
f0m	NakNew	no	194.48	207.65	197.17	198.64
		yes	198.28	225.75	203.95	214.42
	NakOld	no	201.01	215.73	201.86	219.80
		yes	211.86	237.21	206.48	224.37
f0u	NakNew	no	196.48	204.20	196.27	204.20
		yes	196.19	204.20	206.22	204.20
	NakOld	no	213.27	204.20	200.00	204.20
		yes	201.53	204.20	260.77	204.20
f0v	NakNew	no	252.32	208.92	244.88	241.65
		yes	203.81	214.81	207.71	203.33
	NakOld	no	202.25	198.13	233.37	203.89
		yes	223.70	200.97	198.98	207.69

Table 31: All PPL results for the MIT-OCW validation set when training on the AM set with prosodic F0 features.

Feature	extractor	voiced	lin/global	lin/utt	log/global	log/utt
f0a	NakNew	no	209.24	205.63	203.01	208.84
		yes	210.70	213.57	224.64	209.38
	NakOld	no	211.18	211.24	211.47	214.11
		yes	205.68	208.72	214.97	212.41
f0d	NakNew	no	217.76	201.00	202.93	213.02
		yes	198.67	198.65	211.54	197.73
	NakOld	no	202.02	201.28	281.62	200.08
		yes	199.45	199.34	197.66	199.64
f0m	NakNew	no	214.72	221.81	218.41	216.18
		yes	203.93	214.93	217.71	209.75
	NakOld	no	200.24	207.80	209.23	212.42
		yes	216.22	211.29	221.89	218.72
f0u	NakNew	no	216.89	203.87	212.30	203.87
		yes	212.54	203.87	209.73	203.87
	NakOld	no	208.26	203.87	209.00	203.87
		yes	199.15	203.87	220.88	203.87
f0v	NakNew	no	220.81	203.90	222.22	213.60
		yes	202.69	204.74	205.20	202.18
	NakOld	no	201.67	201.86	204.43	201.10
		yes	208.32	201.10	200.47	204.75

Table 32: All PPL results for the MIT-OCW test set when training on the AM set with prosodic F0 features.

C.2 PW Features

Chapter 3.1.2 introduced mean and delta features for signal power. In addition, further prosodic features were derived from signal power in a similar way to F0:

1. Acceleration coefficients (pwa) or delta-delta coefficients

$$\Delta^2_{\mathrm{pw}}[l] = \frac{\Delta_{\mathrm{pw}}[l+1] - \Delta_{\mathrm{pw}}[l-1]}{2}. \tag{120}$$

2. The variance (σ_{pw}) of the signal power within frames N_l corresponding to word w_l (pwv)

$$\sigma_{\mathrm{pw}}[l] = \frac{1}{N_l} \sum_{n=n_s}^{n_e} s[n] - \mu_{\mathrm{pw}}[l], \tag{121}$$

   where n_s is the corresponding start index and n_e the end index of word w_l in the acoustic signal.

3. The mean signal power (pwu) for all feature frames N_L of an utterance

$$\mu_{\mathrm{pw,utt}}[l] = \frac{1}{N_L} \sum_{n=n_s}^{n_e} s[n]. \tag{122}$$

   For utterance level mean pwu, the same feature is used for all words in one utterance.

For the experiments in Chapter 5.3, pwm and pwd were logarithmic scale features with different normalisations. In addition, for all power features linear were also investigated. The normalisation to zero mean and unit variance was done per utterance or with the global mean and variance.

Tables 33 and 34 show the PPL results for all prosodic signal power features on the validation and on the test set of MIT-OCW. Table 33 shows the results for the validation set for re-training with prosodic features. Table 34 shows the PPL results after re-training on the test set.

In addition to the re-training scheme as described in Chapter 5.3.1, all features were also investigated when trained from scratch only on the smaller MIT-OCW AM training set. Tables 35 and 36 show the PPLs for these models.

Feature	lin/global	lin/utt	log/global	log/utt
pwa	183.22	183.38	170.85	183.24
pwd	183.81	183.17	170.20	181.57
pwm	183.16	181.73	181.81	178.30
pwu	183.55	183.43	183.54	183.43
pwv	183.36	182.63	182.99	181.79

Table 33: All PPL results for MIT-OCW with prosodic PW features for the validation set.

Feature	lin/global	lin/utt	log/global	log/utt
pwa	152.45	152.09	148.74	152.01
pwd	152.12	151.50	146.89	149.52
pwm	152.30	151.38	149.77	148.25
pwu	152.34	152.40	152.41	152.40
pwv	152.40	151.81	152.23	151.80

Table 34: All PPL results for MIT-OCW with prosodic PW features for the test set.

Feature	lin/global	lin/utt	log/global	log/utt
pwa	245.46	275.96	346.44	198.43
pwd	207.28	201.56	245.89	194.84
pwm	201.24	204.87	203.22	191.26
pwu	202.46	204.20	194.81	204.20
pwv	202.69	200.17	202.54	205.09

Table 35: All PPL results for the MIT-OCW validation set when training on the AM training set with PW features.

Feature	lin/global	lin/utt	log/global	log/utt
pwa	224.76	222.59	424.17	209.04
pwd	207.02	202.32	263.54	209.19
pwm	198.56	202.27	223.18	210.24
pwu	201.44	203.87	210.14	203.87
pwv	200.31	201.05	210.74	206.66

Table 36: All PPL results for the MIT-OCW test set when training on the AM training set with PW features.

D. Additional Results for Domain Adaptation with a Unified Framework for Context Extraction and Adaptation

In addition to the PPL results for UniFA in Chapter 5.6.2 and the rescoring results in Chapter 5.6.3, this section provides PPL and WER results for additional configurations of the sequence summary network (SSN) in UniFA. This section contains results for TED talks and CSJ.

D.1 Perplexity Results

Tables 37, 38, and 39 show the PPL results on TED talks for UniFA with an SSN with 100, 300 and 500 nodes, respectively. The context window size for a single hidden layer SSN varied from 1 to 200. For two hidden layers the context window size was between 50 and 200.

Tables 40, 41, and 39 show the PPL results on CSJ for UniFA with an SSN with 100, 300 and 500 nodes, respectively. The context window size for the SSN varied from 50 to 200 and one or two hidden layer were used in the SSN.

D.2 Rescoring Results

Tables 43, 44, and 45 show the WER results after 100-best rescoring on TED talks for UniFA with an SSN with 100, 300 and 500 nodes, respectively. The

Table 37: PPL for subtitle and TED-LIUM validation and test set for UniFA with 100 nodes in the SSN.

SSN layer	context	Subtitle PPL		TED-LIUM PPL	
		val	test	val	test
LSTM-LM	—	51.58	51.98	209.34	156.29
UniFA 1	1	39.17	40.75	167.17	133.23
UniFA 1	5	35.42	38.31	151.68	124.53
UniFA 1	10	36.19	38.05	190.45	136.78
UniFA 1	25	36.43	38.87	157.90	119.34
UniFA 1	50	36.06	37.90	169.62	130.90
UniFA 1	100	36.12	37.37	151.55	123.58
UniFA 1	200	42.95	42.37	144.17	126.53
UniFA 2	50	42.34	43.17	170.95	138.58
UniFA 2	100	40.90	41.97	144.33	124.25
UniFA 2	200	44.23	44.70	151.44	131.76

context window size for a single hidden layer SSN varied from 1 to 200. For two hidden layers the context window size was between 50 and 200.

Tables 46, 47, and 48 show the WER results after 100-best rescoring on CSJ for UniFA with an SSN with 100, 300 and 500 nodes, respectively. The context window size for the SSN varied from 50 to 200 and one or two hidden layer were used in the SSN.

Table 38: PPL for subtitle and TED-LIUM validation and test set for UniFA with 300 nodes in the SSN.

SSN layer	context	Subtitle PPL		TED-LIUM PPL	
		val	test	val	test
LSTM-LM	—	51.58	51.98	209.34	156.29
UniFA 1	1	36.94	38.93	302.91	194.42
UniFA 1	5	31.73	35.00	150.72	124.48
UniFA 1	10	33.33	35.90	259.29	144.86
UniFA 1	25	33.35	35.45	136.74	112.66
UniFA 2	50	41.78	43.09	146.79	126.06
UniFA 2	100	44.94	45.62	141.05	123.91
UniFA 2	200	43.74	43.76	147.20	124.60

Table 39: PPL for subtitle and TED-LIUM validation and test set for UniFA with 500 nodes in the SSN.

SSN layer	context	Subtitle PPL		TED-LIUM PPL	
		val	test	val	test
LSTM-LM	—	51.58	51.98	209.34	156.29
UniFA 1	1	34.86	36.99	163.07	134.93
UniFA 1	5	30.41	33.37	158.76	125.16
UniFA 1	10	31.67	34.58	149.41	115.88
UniFA 1	25	33.89	35.44	171.16	139.20
UniFA 1	50	32.98	34.99	149.06	124.06
UniFA 1	100	35.40	36.25	164.21	131.26
UniFA 1	200	39.26	39.35	155.11	129.95
UniFA 2	50	37.71	38.85	148.65	120.91
UniFA 2	100	43.91	44.62	153.62	128.33
UniFA 2	200	41.81	41.87	139.49	120.10

Table 40: PPL for CSJ with 100 units in the SSN.

SSN layer	context	Heldout	Test 1	Test 2	Test 3
LSTM-LM	—	37.88	40.52	41.79	41.01
UniFA 1	50	36.85	39.95	41.00	40.37
UniFA 1	100	36.22	39.58	40.47	39.52
UniFA 1	200	36.63	39.71	40.56	39.42
UniFA 2	50	36.77	39.75	40.46	40.24
UniFA 2	100	36.83	39.91	40.54	40.03
UniFA 2	200	36.39	39.08	40.22	39.68

Table 41: PPL for CSJ with 300 units in the SSN.

SSN layer	context	Heldout	Test 1	Test 2	Test 3
LSTM-LM	—	37.88	40.52	41.79	41.01
UniFA 1	50	37.34	40.25	41.18	40.78
UniFA 1	100	35.94	39.06	40.13	39.07
UniFA 1	200	36.29	39.52	40.20	39.67

Table 42: PPL for CSJ with 500 units in the SSN.

SSN layer	context	Heldout	Test 1	Test 2	Test 3
LSTM-LM	—	37.88	40.52	41.79	41.01
UniFA 1	50	36.95	40.09	41.04	40.13
UniFA 1	100	36.29	39.15	40.44	39.22
UniFA 1	200	36.24	39.40	40.21	39.49
UniFA 2	50	36.76	39.90	40.88	40.10
UniFA 2	100	36.29	39.45	40.21	39.25
UniFA 2	200	36.74	39.56	40.47	39.68

Table 43: WER after 100-best rescoring for TED-LIUM for UniFA with 100 nodes in the SSN.

SSN layer	context	val WER[%]	test WER[%]
1-best	—	16.3	15.1
LSTM-LM	—	14.2	12.1
UniFA 1	1	14.1	11.9
UniFA 1	5	13.9	11.8
UniFA 1	10	14.1	11.9
UniFA 1	25	13.9	11.9
UniFA 1	50	14.0	12.1
UniFA 1	100	13.9	12.0
UniFA 1	200	13.9	12.2
UniFA 2	50	13.9	12.1
UniFA 2	100	14.0	12.1
UniFA 2	200	14.0	12.1

Table 44: WER after 100-best rescoring for TED-LIUM for UniFA with 300 nodes in the SSN.

SSN layer	context	val WER[%]	test WER[%]
1-best	—	16.3	15.1
LSTM-LM	—	14.2	12.1
UniFA 1	1	25.5	28.8
UniFA 1	5	15.7	16.8
UniFA 1	10	14.0	11.9
UniFA 1	25	13.8	11.9
UniFA 2	50	13.8	11.9
UniFA 2	100	13.8	12.0
UniFA 2	200	13.9	12.1

Table 45: WER after 100-best rescoring for TED-LIUM for UniFA with 500 nodes in the SSN.

SSN layer	context	val WER[%]	test WER[%]
1-best	—	16.3	15.1
LSTM-LM	—	14.2	12.1
UniFA 1	1	24.7	50.1
UniFA 1	5	13.8	16.8
UniFA 1	10	13.9	11.7
UniFA 1	25	13.9	11.8
UniFA 1	50	13.9	11.7
UniFA 1	100	13.8	11.9
UniFA 1	200	14.0	12.0
UniFA 2	50	13.8	12.1
UniFA 2	100	13.9	12.2
UniFA 2	200	13.9	12.1

Table 46: WER after 100-best rescoring for CSJ and UniFA with 100 nodes in the SSN.

SSN layer	context	Test 1	Test 2	Test 3
1-best	—	12.26	9.34	12.22
LSTM-LM	—	10.71	8.08	10.49
UniFA 1	50	10.70	8.03	10.39
UniFA 1	100	10.66	7.81	10.43
UniFA 1	200	10.66	7.96	10.31
UniFA 2	50	10.55	7.92	10.34
UniFA 2	100	10.57	8.01	10.34
UniFA 2	200	10.60	8.02	10.28

Table 47: WER after 100-best rescoring for CSJ and UniFA with 300 nodes in the SSN.

SSN layer	context	Test 1	Test 2	Test 3
1-best	—	12.26	9.34	12.22
LSTM-LM	—	10.71	8.08	10.49
UniFA 1	50	10.65	7.94	10.47
UniFA 1	100	10.60	7.93	10.33
UniFA 1	200	10.68	7.99	10.37

Table 48: WER after 100-best rescoring for CSJ and UniFA with 500 nodes in the SSN.

SSN layer	context	Test 1	Test 2	Test 3
1-best	—	12.26	9.34	12.22
LSTM-LM	—	10.71	8.08	10.49
UniFA 1	50	10.65	7.91	10.43
UniFA 1	100	10.59	7.96	10.29
UniFA 1	200	10.63	7.87	10.33
UniFA 2	50	10.72	7.96	10.44
UniFA 2	100	10.60	7.93	10.34
UniFA 2	200	10.70	7.90	10.34