BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

PHD THESIS

Brno, 2017 Ing. Karel Veselý

BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

SEMI-SUPERVISED TRAINING OF DEEP NEURAL NETWORKS FOR SPEECH RECOGNITION

'SEMI-SUPERVISED' TRÉNOVÁNÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ PRO ROZPOZNÁVÁNÍ ŘEČI

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR — **Ing. KAREL VESELÝ**
AUTOR PRÁCE

SUPERVISOR — **Doc. Ing. LUKÁŠ BURGET, Ph.D.**
ŠKOLITEL

BRNO 2017

Abstract

In this thesis, we first present the theory of neural network training for the speech recognition, along with our implementation, that is available as the 'nnet1' training recipe in the Kaldi toolkit. The recipe contains RBM pre-training, mini-batch frame Cross-Entropy training and sequence-discriminative sMBR training. Then we continue with the main topic of this thesis: semi-supervised training of DNN-based ASR systems. Inspired by the literature survey and our initial experiments, we investigated several problems: First, whether the confidences are better to be calculated per-sentence, per-word or per-frame. Second, whether the confidences should be used for data-selection or data-weighting. Both approaches are compatible with the framework of weighted mini-batch SGD training. Then we tried to get better insight into confidence calibration, more precisely whether it can improve the efficiency of semi-supervised training. We also investigated how the model should be re-tuned with the correctly transcribed data. Finally, we proposed a simple recipe that avoids a grid search of hyper-parameters, and therefore is very practical for general use with any dataset. The experiments were conducted on several data-sets: for Babel Vietnamese with 10 hours of transcribed speech, the Word Error Rate (WER) was reduced by 2.5%. For Switchboard English with 14 hours of transcribed speech, the WER was reduced by 3.2%. Although we found it difficult to further improve the performance of semi-supervised training by means of enhancing the confidences, we still believe that our findings are of significant practical value: the untranscribed data are abundant and easy to obtain, and our proposed solution brings solid WER improvements and it is not difficult to replicate.

Abstrakt

V této dizertační práci nejprve prezentujeme teorii trénování neuronových sítí pro rozpoznávání řeči společně s implementací trénovacího receptu 'nnet1', který je součástí toolkitu s otevřeným kódem Kaldi. Recept se skládá z předtrénování bez učitele pomocí algoritmu RBM, trénování klasifikátoru z řečových rámců s kriteriální funkcí Cross-entropy a ze sekvenčního trénování po větách s kriteriální funkcí sMBR. Následuje hlavní téma práce, kterým je semi-supervised trénování se smíšenými daty s přepisem i bez přepisu. Inspirováni konferenčními články a úvodními experimenty jsme se zaměřili na několik otázek: Nejprve na to, zda je lepší konfidence (t.j. důvěryhodnosti automaticky získaných anotací) počítat po větách, po slovech nebo po řečových rámcích. Dále na to, zda by konfidence měly být použity pro výběr dat nebo váhování dat – oba přístupy jsou kompatibilní s trénováním pomocí metody stochastického nejstrmějšího sestupu, kde jsou gradienty řečových rámců násobeny vahou. Dále jsme se zabývali vylepšováním semi-supervised trénování pomocí kalibrace kofidencí a přístupy, jak model dále vylepšit pomocí dat se správným přepisem. Nakonec jsme navrhli jednoduchý recept, pro který není nutné časově náročné ladění hyperparametrů trénování, a který je prakticky využitelný pro různé datové sady. Experimenty probíhaly na několika sadách řečových dat: pro rozpoznávač vietnamštiny s 10 přepsanými hodinami (Babel) se chybovost snížila o 2.5%, pro angličtinu se 14 přepsanými hodinami (Switchboard) se chybovost snížila o 3.2%. Zjistili jsme, že je poměrně těžké dále vylepšit přesnost systému pomocí úprav konfidencí, zároveň jsme ale přesvědčení, že naše závěry mají značnou praktickou hodnotu: data bez přepisu je jednoduché nasbírat a naše navrhované řešení přináší dobrá zlepšení úspěšnosti a není těžké je replikovat.

Keywords

Deep neural networks, speech recognition, semi-supervised training, Kaldi, nnet1

Klíčová slova

Hluboké neuronové sítě, rozpoznávání řeči, semi-supervised trénování se smíšenými daty s přepisem i bez přepisu, Kaldi, nnet1

Reference

VESELÝ, Karel. *Semi-supervised training of Deep Neural Networks for speech recognition.* Brno, 2017. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. Ing. Lukáš Burget, Ph.D.

Semi-supervised training of Deep Neural Networks for speech recognition

Declaration

Hereby I declare that this doctoral thesis was prepared as my original author's work under the supervision of Doc. Ing. Lukáš Burget, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Karel Veselý
April 6, 2018

Acknowledgements

Here, I would like to thank all the great people I had a chance to interact with during the Ph.D. studies, they were numerous. Firstly, I would like to thank to our group leader Jan 'Honza' Černocký for all the support, as well as for creating and especially maintaining such a great speech processing group as we have in Brno University of Technology. Then, I would like to thank my supervisor Lukáš Burget for introducing me to all the details of the neural network training and also for being open to the numerous discussions of technical questions which arised on-the-way. Also, I owe a big 'Thank you!' to Daniel Povey for being such a great father of Kaldi and for allowing me to contribute the 'nnet1' code to Kaldi. Very helpful were the code reviews, suggestions and discussions. Then, I would like to thank Sanjeev Khudanpur and Hynek Heřmanský for supporting the stays at the Johns Hopkins University. I would also like to thank all the colleagues in the Speech@FIT in Brno, all the present members, former members and the international visitors. Thank you for being around, ready for a technical discussion or to spend a good time. Next, I would like to thank the people from CLSP in Johns Hopkins University and the people from numerous workshops (Kaldi, JHU, JSALT). I would also like to thank the authors who published their work about semi-supervised training in speech recognition, they were a great source of inspiration. Finally, I would like to thank our secretaries and my parents for all the care and support. Lastly, I thank all the people I might have forgotten to thank to.

Contents

Acronyms

ASR	Automatic speech recognition
WER	Word error rate (evaluation metric)
PER	Phone error rate (evaluation metric)
AM	Acoustic model
LM	Language model
WFST	Weighted finite state transducer (automaton representing a graph with costs on arcs)
GMM	Gaussian mixture model
EM	Expectation maximization (training algorithm for GMM)
NN	Neural network
DNN	Deep neural network (NN with 'several' hidden layers)
RBM	Restricted Boltzmann Machine (an auxiliary model for DNN pre-training)
DBN	Deep Belief Network (a stack of RBMs, initializiation of hidden layers)
SGD	Stochastic gradient descent (training algorithm for NNs)
CE	per-frame cross-entropy (objective function for NN training)
MMI	Maximum mutual information (objective function)
MPE	Minimum phone error (objective function)
BMMI	Boosted maximum mutual information (objective function)
sMBR	State minimum Bayes risk (objective function)
MFCC	Mel-frequency cepstral coefficients (input features)
FBANK	Log Mel-filterbank output (input features)
PLP	Perceptual linear predictive analysis (input features)
LDA	Linear discriminant analysis
MLLT	Maximum likelihood linear transform (feature projection, also called Semi-tied covariance method)
fMLLR	Feature-based maximum-likelihood linear regression (speaker adaptation technique)

Notation

t, T	time indexing
k, K	NN-output class indexing
w_i, W	single word, word sequence
s, s_i, S	tied-state, tied-state from a sequence, a sequence of tide states
$\mathcal{L}, \pi$	lattice, a path from lattice
κ	acoustic scale applied to the per-frame likelihoods from the acoustic model
ϱ	graph scale applied to graph-costs in WFST graphs, representing lattices or recognition network
$\mathbf{x}$	single data-point (vector) of input features
$\mathbf{X}$	matrix with input features (composed from single vectors)
$\mathbf{h}$	hidden vector from neural network
$\mathbf{y}$	output vector from neural network
$\mathbf{W}$	matrix with synapses from single layer of neurons
$W_{i,j}$	single element from the matrix of synapses
$\mathbf{b}$	bias vector from single layer of neurons
σ	logistic sigmoid
$\mathbf{w}$	all neural network parameters reshaped in one big vector
M	size of mini-batch
N	number of inputs from neuron or neural network
L	identifier of specific neural network layer
$\alpha_{\mathsf{src}(a_j)}$, $\beta_{\mathsf{tgt}(a_j)}$	accumulation statistics in Forward-backward algorithm, which produces lattice-posteriors $\gamma(a_j)$
$\gamma(a_j)$	posterior probability of traversing through lattice-link a_j
$\gamma(t, s)$	posterior probability of being in tied-state s at time t
$\gamma(t, \bar{s}_t)$	posterior of a particular tied-state $\bar{s}_t$ from the best path of lattice
$\gamma(q, w)$	posterior probability of word w being at position q in decoder output
$\gamma(q, \bar{w}_q)$	posterior of a particular word $\bar{w}_q$ from the best path of lattice
λ	the lattice-scale, applied to re-scale the lattice posteriors by multiplying the link scores before the forward-backward algorithm is started
α	the exponential scale, applied to the confidences which are already extracted from the lattices

Chapter 1

Introduction

Probably the most natural way of communicating non-trivial ideas between humans is by speech. It is very likely that the ability to speak was developing slowly for thousands of years when our pre-historic ancestors lived in the groups of 'hunters-gatherers'. Certainly, they needed an efficient way to communicate the ideas to achieve a common goal, be it a mammoth hunt, or taking care of youth, while other fellows were searching for food.

Since then, the way of life has changed, the culture and language have changed, and also the technology has changed. A big leap forward was the invention of a script, which allowed to 'conserve the speech'. Before that, the only way to preserve a message was to memorize it. The next huge step forward was the invention of the print. This allowed to spread the knowledge to a larger mass of people. Without this invention, both the industrial revolution and modern education would not be possible. Nowadays, we live in an electronic era, a data-gram equivalent to the content of a whole book can be transferred via Internet to the other side of our planet in a fraction of a second with almost no cost.

Our ancestors would definitely be bewildered seeing us communicating with 'computer machines'. The means of human-machine communication also evolved over time. The perforated paper-cards were replaced with keyboards and terminal screens. Later on, after inventing a computer mouse, the graphical user interface was born. The keyboard and mouse are the dominant interfaces until now. Despite their huge success, they also have some limitations. They require space, they are operated by hands and typically, the user needs to have a mental capacity to process the visual feedback from the screen.

The lack of space becomes a limiting factor in the case of small devices, like smart-phones or tablets. These are usually equipped with touch-screens, however writing on them is not comfortable. Then, a good example of a cognitively loaded person would be the driver, who would like to make a call or to control the GPS-navigation. In both cases the interaction using natural speech is a good alternative to the traditional input methods.

Other successful uses of speech recognition are in the fields that traditionally involve dictating for documentary purposes. This is the case of medicine, courts, state administrative and parliamentary talks. Speech recognition might also be interesting for companies, where it can be used to keep track of internal meetings or it can be integrated into customer care systems.

A very specific area is the on-the-fly generation of TV-captions, which is done by re-speaking the simplified content by a shadow speaker in a space with controlled acoustic conditions [Trmal et al., 2010]. The subtitles need to be generated rapidly, there is only a little time for correction of the recognition output. A little easier situation is the off-line production of video transcriptions, where processing time does not matter. The speech

recognition output can then be used to build an index and make the multimedia-document searchable for keywords, as is done in *Superlectures*[1]. This system enables textual search for video lectures, conference talks, or recordings from various events.

Lastly, speech recognition is used as a front-end of the machine translation systems, which makes the *'speech to speech translation'* possible, as it is demonstrated by the Google Translate application[2].

Nowadays, speech recognition is commonly used to search the Internet from mobile devices. The speech is also an input interface to the personal assistants like Siri, Cortana or Google Now. Slowly, speech recognition is finding its way to our home gadgets like TV-sets, light switches, etc. A recent product is the question-answering machine Amazon Echo[3]. It is equipped with a speaker and a microphone array for improved robustness on distant speech. It can be asked about news or weather, it can read a Wikipedia page or play a music on demand. Technically, it is a hardware client, while the speech recognition and the dialogue management is running in a cloud server.

It really seems that we are experiencing the golden era of speech recognition. The companies are investing into the new trendy technology, while the speech recognition is used in many ways. However, it is a result of a long evolution. Very interesting documents are the popularization videos from the historic speech projects at Carnegie Mellon University: *Hear Here*[4] (1968), *Hearsay*[5] (1973) and *Harpy*[6] (1976). At the time voice commands were used to control a robot to move simple objects like cubes or chess figures. It is very likely that the speech recognition research did also inspire the authors of science fiction literature to write about intelligent computers with voice interface: the HAL in 'Space odyssey 2001', the human-like robot C-3PO in 'Star wars', or the spaceship computer in 'Star trek'.

From the more serious history, a remarkable researcher with Czech origin was Fred Jelinek (1932-2010)[7], who founded and led a research team at IBM in 1972-1993. Inspired by Shannon's work on the information theory, he pioneered with his colleagues the use of statistical methods for speech recognition [Jelinek, 1976], which led to adoption of Hidden Markov Models. A rumor says that in '70 the computers were becoming fast enough for the standard tasks, which was not a favorable prediction for selling new computers. With speech recognition, it was clear that there will be need for new faster computers, once the technology was ready for massive use, which motivated the industrial research.

Although there was a tremendous research progress in the last few years, especially after the Deep Neural Networks were introduced, the current Automatic Speech Recognition (ASR) systems are not perfect. Although the recognition of formal talks, conversational telephone speech and meetings does have an acceptable performance ranging between 10-25% of word error rate, a big challenge remains the far-field speech recognition of informal speaking style, here the error rate ranges 35-50%. Then, an unresolved problem remains the ASR of recordings with overlapped speakers, recorded by 1 microphone. Another challenge might be the limited amount of noisy training data, the WER can increase up to 50-70%, which we observed while working on Babel program.

Another limitation of current ASR systems comes from the nature of its supervised

[1] http://www.superlectures.com/

[2] https://support.google.com/translate/answer/6142468?co=GENIE.Platform%3DDesktop&hl=en&oco=0

[3] https://www.youtube.com/watch?v=KkOCeAtKHIc

[4] https://www.youtube.com/watch?v=8MrkioDG2xU

[5] https://www.youtube.com/watch?v=McrSCrl4mUc

[6] https://www.youtube.com/watch?v=N3i6NoUZsSw

[7] https://en.wikipedia.org/wiki/Frederick_Jelinek

machine-learning. To develop an ASR system for a new language, we need to carefully transcribe at least few hours of the training recordings. Then, for the language model training, we need a text corpus, ideally with vocabulary and speaking style similar to the target domain, in which the recognizer will be deployed. We also need a linguist to design a phone-set and create a pronunciation lexicon. Finally, the acoustic conditions in target domain need to be similar to those in the training data, otherwise a mismatched acoustic model will cause performance degradation.

Fortunately, many of the problems are partially solved: The robustness is improved by various acoustic feature normalizations or by capturing the signal by microphone array and processing the multi-channel input (beam-forming, de-noising, source separation, ...). The pronunciation lexicon can be replaced by a graphemic one at the cost of small degradation in performance (typically few percents of WER). The text corpus can be prepared by cleaning downloaded web resources.

And finally, in the case of having only a little amount of the transcribed training data, we can improve the acoustic model by using the untranscribed data in the semi-supervised training, which is the main topic of this thesis.

1.1 Motivation

The state-of-the-art Automatic Speech Recognition (ASR) systems need carefully transcribed and thus expensive data to be trained on. It is therefore in the interest of the research community to search for such techniques that will help to reduce this 'cost barrier', and make the ASR technology more accessible both for commercial and non-commercial use.

The idea of semi-supervised training is to improve the system by using non-transcribed data. This is done by generating automatic transcripts along with their confidences, i.e. the probability of being correct. Then the performance is improved because of better acoustic models trained on more data. We are using a 'self-training' scenario, where a small part of data is transcribed manually. This allows us to train a 'seeding' system, which we use to generate the automatic transcripts, that we later use for the self-training.

The self-training of ASR systems has been studied extensively. However, at that time, the dominant acoustic models were GMMs (Gaussian Mixture Models) [Wessel and Ney, 2005, Wessel et al., 2001]. These are typically trained *generatively* by EM algorithm. In this model, each acoustic unit is described by a GMM, i.e. a probability density function estimated on its associated data-points. In the EM training, the models do not partition the feature space exclusively, so two acoustic units can have similar distributions. A minor part of wrong labels in the training data, may not have too bad influence on the final model.

However, the current state-of-the-art acoustic modeling is based on neural networks, which are usually trained *discriminatively* to classify feature frames into a closed set of acoustic units. Here the classification is exclusive, the posterior probability is sub-divided among the acoustic units, which makes the training potentially less robust to wrong labels.

For this reason, it is interesting to re-visit some of the older techniques and to use them as an inspiration for developing a self-training recipe for current state-of-the-art ASR systems.

1.2 Scope of the thesis

In this thesis is presented a systematic study of acoustic model self-training. The acoustic model is a feed-forward Deep Neural Network and I searched for answers to these crucial questions:

- What is the most suitable confidence granularity for the semi-supervised DNN training. Should we extract confidences: per-sentence, per-word, per-frame?
- How should we use the confidence, for data selection or for weighted training?
- What is the 'ideal confidence'? What should be its role in self-training?
- Is confidence calibration important?
- Do we need to further post-process the self-trained model by using the correctly transcribed data?
- Can we build a simple generic recipe that is applicable to different data-sets?

In the thesis I worked with feed-forward neural networks with sigmoid units. It is likely that the observations will generalize to other types of networks, although we did not particularly investigate this in detail.

The Kaldi 'nnet1' recipe

Before the experiments with semi-supervised training were started, I have developed and made publicly available the DNN training recipe *'nnet1'* as part of the toolkit *Kaldi*[8]. The design of this implementation was partially inspired by my previous project *TNet*[9]. Both tool-kits are used by other researchers in various laboratories or companies from all over the world.

The 'nnet1' recipe consists of Restricted Boltzmann Machine pre-training, the mini-batch frame classification training and the sequence-discriminnative sMBR training. The important aspects of the recipe are covered in chapter 3, which sources mainly from my own publications. This recipe represents a solid basis upon which the semi-supervised experiments are performed.

[8] http://kaldi-asr.org/doc/dnn1.html
[9] http://speech.fit.vutbr.cz/software/neural-network-trainer-tnet

1.3 Original claims

1. In this thesis is performed an extensive study of semi-supervised training of DNN in which we use confidences extracted from a single ASR system.

2. I have carefully compared the scenarios in which the confidences are extracted per-sentence, per-word or per-frame. Along the way are also compared other state-of-the-art confidence measures, and it is shown that our preferred confidence is better.

3. From the results is apparent that standard 'sentence selection' approach provides only limited performance improvements. Better results are achieved either with selecting smaller units (words, frames) or from the use of weighted training with some appropriate scaling mechanism.

4. I also identified a simple rule for setting an optimal threshold in word-selection, which generalizes both for Babel Vietnamese and Switchboard English. The amount of words added in self-training is determined by word accuracy from development set. Such simple system is not far from our best recipe, which involves a time-consuming grid search over a hyper-parameter.

1.4 Structure of the thesis

This thesis is organized as follows: in chapter 2 we introduce the theoretical view on speech recognition and back-propagation algorithm. In chapter 3 we describe Kaldi 'nnet1' recipe that we implemented, it is used as experimental framework in other chapters. An important part of chapter 3 is 3.1.3 which describes sequence-discriminative training [Veselý et al., 2013a], and 3.2 dealing with acceleration of DNN training [Veselý et al., 2010].

Then, in chapter 4 are described the data-sets we worked with. In chapter 5 we start discussing semi-supervised training, in 5.2 we discuss the key design questions, and in 5.3 is a literature survey. In section 6.1 are presented our experiments from [Veselý et al., 2013b], being followed in 6.2 by unpublished approximative calibration of confidences.

In chapter 7 we perform a systematic search of best use of confidences in semi-supervised training. We experiment with per-sentence, per-word and per-frame confidences. Simultaneously, we compare data-selection and data-weighting. With the data-weighting, we also introduce calibration of confidences, trained on development data. In all setups we post-process the model by re-training with manually transcribed data.

After the 'exploration' in chapter 8, we identify a recipe that generalizes across different experimental setups. A selection of experiments from chapters 7 and 8 was published in [Veselý et al., 2017]. Finally, in chapter 9 we experimented with two-system combination for generating automatic labels, which did not bring performance improvement. The thesis is concluded by final remarks in chapter 10.

Chapter 2

Introduction to Neural Network based speech recognition

In this chapter, we introduce the theory of speech recognition, the models that are used in the recognizer and the derivation of the back-propagation algorithm for neural network training. If the reader is already familiar with these topics, he/she may consider skipping this chapter.

2.1 The history of neural networks in speech recognition

In each epoch, the modeling techniques were adapted to the possibilities of the current computing hardware. Originally, the speech recognition systems were based on vector quantization at phone center-marks [Jelinek, 1976] and a processing of strings of symbols representing the acoustic-units. Later on, the vector quantization was replaced by more demanding Gaussian Mixture Models (GMM) and the Hidden Markov Model decoder was introduced. The GMMs were later refined by using context-dependent phonemes, which made both the systems better and the models larger.

The idea of neural networks as acoustic models in ASR systems is not a question of the last decade, when it finally became popular. A lot of work has been done in the late 80's and in the first half of 90's in the laboratories at OGI, ICSI and IDIAP under supervision of Herve Bourlard and Nelson Morgan [Bourlard and Morgan, 1993]. A very interesting work focused on recurrent neural networks was done at Cambridge by Tony Robinson [Robinson, 1992]. However, the computers were slow and the acceleration of the training required to develop a specialized hardware such as SPERT-II cards [Wawrzynek et al., 1996]. At that time, better results were achieved with GMMs, which are easier to train fast in parallel on a cluster of computers.

A small *renaissance* of neural networks then came around year 2000, started by the discovery that they can be successfully used to produce input features for GMMs [Sharma et al., 2000]. Initially, in the form of post-processed NN outputs, while in 2007 [Grézl et al., 2007] it was found that even better results can be obtained with bottleneck features extracted from a narrow hidden-layer.

Finally, the big success of neural networks came after year 2010 [Hinton et al., 2012], when it was shown that the DNNs can efficiently replace the GMM acoustic models, decreasing the word error rate by 1/3 compared to a GMM model trained on the same PLP features [Seide et al., 2011b]. However, this huge boost of performance, which attracted a lot of attention, is not achieved when we replace GMMs by DNN applied on top of the

bottleneck features [Tüske et al., 2012].

Nowadays, the neural networks are the most common acoustic model in speech recognition applications. This would not be possible without an affordable way to accelerate the training either by using the computation on the graphic cards (GPGPU) [Scanzio et al., 2010, Veselý et al., 2010] or by distributing the training within a cluster of computers [Dean et al., 2012] or by a combination of both [Povey et al., 2014].

Because the neural networks have become popular, a lot of effort focused on experiments with their internal structure and the training procedure. For example, the traditional sigmoid activation functions were replaced by rectified-linear units [Glorot et al., 2010, Zeiler et al., 2013], max-out units [Goodfellow et al., 2013, Miao et al., 2013, Swietojanski et al., 2014] or its more generic form called p-norm [Zhang et al., 2014b]. Another largely discussed technique is drop-out [Wager et al., 2013, Srivastava et al., 2014], where hidden-neuron outputs get discarded randomly, which helps to avoid early co-adaptation of neurons and over-fitting.

Another branch of experiments has focused on the recurrent neural networks, which can transfer the inner state over time via recurrent connections. In the recent past, the recurrent networks became popular for language modeling [Mikolov et al., 2010], and later also for acoustic modeling as the Long Short Term Memory (LSTM) networks [Hochreiter and Schmidhuber, 1997, Gers et al., 2000, Sak et al., 2014, Senior et al., 2015].

In this thesis, we limit our research to the *feed-forward* neural networks with *sigmoid* activation function. We believe that the effectiveness of semi-supervised training is 'orthogonal' to the changes in the neural network structure, and it is very likely that the well performing techniques will generalize. At the same time, our feed-forward networks have still quite competitive performance. The recipe was successfully used in the Babel[1] evaluations, where our DNN system was the best single system in the Babelon team in the years 2013, 2014 and 2015.

2.2 The problem of speech recognition

According to the theory of Automatic Speech Recognition (ASR), the problem is to correctly recognize the sequence of words that corresponds to the 'observed' acoustic signal.

As illustrated in figure 2.1, the input is a speech signal, while the output is the recognized text. The processing is subdivided into 3 stages.

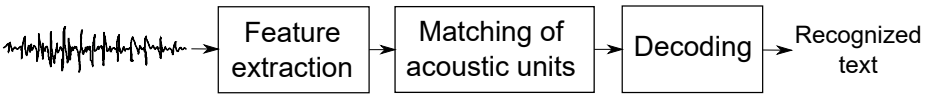


Figure 2.1: *Architecture of speech recognizer*

The purpose of **feature extraction** is to compress the waveform in a sequence of fixed-length vectors of low dimension. Usually, we extract one vector per a 10 ms step from 25 ms long chunks of speech signal, i.e. the *speech frames*. The encoding must preserve the information relevant for recognition and suppress the irrelevant information. For example, in the case of speech recognition, we try to suppress the differences across speakers, genders, dialects, microphones etc. The typical feature extraction is based on signal processing techniques such as filtering and Discrete Fourier Transform. Usually, Fourier spectrum

[1] http://www.iarpa.gov/index.php/research-programs/babel

is post-processed to obtain a representation convenient for the machine-learning models. Typically, the short-term spectrum is projected into a set of triangular filters, 'sitting' at different frequency ranges, and some further processing steps are performed. The most popular feature extraction methods for ASR are FBANKs (log-Mel filterbanks), MFCCs (Mel-frequency cepstral coefficients) and PLPs (Perceptual Linear Prediction).

In **Matching of acoustic units**, we 'convert' the features into scores of some closed set of *acoustic units*. For illustration, one can think of phonemes as units. The acoustic scores are computed by an *acoustic model*, usually a Gaussian Mixture Model (GMM) or a Deep Neural Network (DNN). Formally, each score is a likelihood $P(\mathbf{x}|s)$, i.e. the density function value for the feature vector $\mathbf{x}$ given the identity of acoustic unit s. The acoustic models in general are trained on a set of speech recordings, the training is usually supervised by manual transcriptions. Naturally, better models are obtained when more training data is used. For a poor system, we need to have at least few transcribed hours, while up to hundreds of thousands of hours are used in some companies. For the training, the feature vectors need to be assigned to acoustic units. This is not done manually, but by using a *forced-alignment* to reference transcripts with some existing model. If there is no model yet, equal lengths are assigned to all acoustic units in an utterance.

In **Decoding**, we search for the most likely word sequence $\hat{W}$ that corresponds to the 'observed' sequence of feature vectors $\mathbf{X}$. This is done by a search in a huge graph of all the possible hypothesis, where we combine the scores from the acoustic model, language model and lexicon. A typical decoding algorithm is based on two ideas: *token passing* and *beam search*. The idea of token passing is a frame-by-frame cycle advancing in time over the input features, and for the current frame we have a stack of tokens with partial recognition paths. In the next frame, each token is expanded into many new tokens with the possible continuations of the hypothesis. To avoid having too many tokens, some of them are discarded. Only the tokens with scores within some margin from the best token survive; this is the idea of beam search. This local and greedy heuristic makes the speech recognition fast enough for practical use, however it can lead to a *search error*, if a more accurate path is discarded because its token 'fell-out' of the beam. Practically, the beam-width is the distance of the scaled log likelihoods of partial recognition hypotheses, and we should make sure the beam is large enough, so that its further extension does not improve the recognition results.

2.3 Models in speech recognition and decoding

Mathematically, the decoding is formulated as finding the word string $\hat{W}$ with the maximal *a posteriori* probability given the sequence of input feature vectors $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$:

$$\hat{W} = \arg\max_W P(W|\mathbf{X}) \; . \tag{2.1}$$

By using Bayes rule we can rewrite this expression as:

$$\hat{W} = \arg\max_W \frac{P(\mathbf{X}|W)P(W)}{P(\mathbf{X})} \; , \tag{2.2}$$

where the prior term $P(W)$ corresponds to the probability of the word sequence without using any acoustic information. Practically, this score is obtained from a *language model* (e.g. an n-gram) trained on a large text corpus. Then we have the likelihood term $P(\mathbf{X}|W)$

from the *acoustic model*, which is the score of the feature vector $\mathbf{X}$ given the word sequence W. The likelihood term $P(\mathbf{X}|W)$ involves the sum over all state sequences corresponding to our word-string W, while $P(\mathbf{x}|s)$ are the likelihoods of the acoustic units in those state-sequences. The normalization term $P(X)$ can be ignored as it is constant for any word/state sequence.

For practical reasons, the formulation in Kaldi is simplified to the search of the most likely state sequence S. This is mapped to the corresponding word sequence by the mapping function 'wrds':

$$\hat{W} = \text{wrds}\left(\arg\max_{S} \frac{P(\mathbf{X}|S)P(S)}{P(\mathbf{X})}\right) . \tag{2.3}$$

In other words, instead of getting the score of a word-string by marginalizing over all its possible state-sequences, only the best state sequence is considered in the decoding process. The decoder becomes simpler and faster.

2.3.1 Language model

We begin with *language model*, which is the highest-level model in speech recognition. The role of language model is to assign a probability to any word sequence $W = (w_1, w_2, \ldots, w_M)$ from the language. In the very popular **N-gram language model**, we model the probability of a current word by considering a limited history of the previous $N-1$ words. In case of trigram model, the probability of a word w_i is $P(w_i|w_{i-1}, w_{i-2})$. The model is trained on a large text corpus, where we count the N-tuples of adjacent words (i.e. the N-grams). The maximum likelihood estimate of N-gram probability is the fraction of the actual trigram count divided by the sum of trigrams counts sharing the same history:

$$P(w_i|w_{i-1}, w_{i-2}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{\sum_{w_j} C(w_{i-2}, w_{i-1}, w_j)} . \tag{2.4}$$

The text corpus for the training should ideally cover the target area that the recognizer will be used for (similar topic, speaking style).

As there will always be valid combinations of words that do not appear in the training corpus, there is a mechanism to assign them non-zero probability, which is called *smoothing*. In this case, we *back-off* to the model of a lower order (bigram, unigram), while part of the probability mass is redistributed to the back-off model. For instance, the Kneser-Ney smoothing [Kneser and Ney, 1995] is a popular method.

The term $P(S)$ in (2.3) includes the language model scores $P(w_i|w_{i-1}, w_{i-2})$. With no or limited history available at the beginning, we use unigram score $P(w_1)$ or bigram score $P(w_2|w_1)$.

2.3.2 Lexicon

The lexicon is a mapping of words into strings of phonemes. It tells us how to build the HMM of a word and it allows sharing of acoustic units among words. The HMM of a word is built by connecting the HMMs of the acoustic units corresponding to the pronunciation in the lexicon. It can contain multiple pronunciations of the same word and their scores, for example:

```
...
camera 1 k ae m r ax
camera 0.451613 k ae m ax r ax
...
```

In our experiments, the pronunciation probabilities are re-scaled with *max-normalization* [Chen et al., 2015], the score of the most likely variant is 1, the scores of other variants are re-scaled accordingly.

If we fail to recognize a word because it is not in the lexicon, it is an *Out of vocabulary error (OOV)*. In this case, we can synthesize its pronunciation with a G2P model extracted from the existing lexicon. The two most popular tools for this task are *Sequitur* [Bisani and Ney, 2008] or *Phonetisaurus* [Novak et al., 2016].

2.3.3 Hidden Markov Model

Each acoustic unit from the pronunciation lexicon is expanded into HMM model as in figure 2.2. Because the durations of acoustic units differ and their pattern changes over time, we represent each of them by 3 state Hidden Markov Model (HMM), where the 3 states model the beginning, the middle part and the end of the acoustic unit. During the

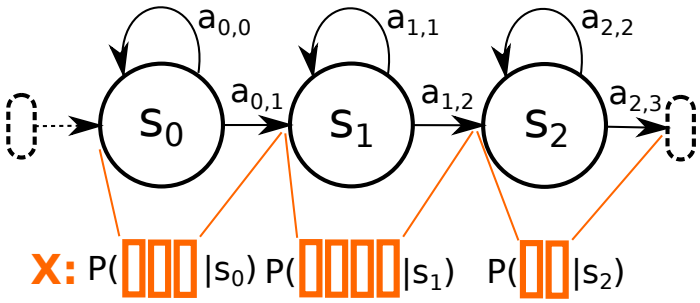


Figure 2.2: *3-state Hidden Markov Model, which models acoustic unit in ASR systems*

decoding, the token is passed through all the states, while it is allowed to stay in the state for several frames by traversing the self-loop transitions. Also note that the transitions have associated transition probabilities $a_{i,j}$, and the outgoing transition probabilities from each state sum up to 1. A score of a path through the simple HMM in figure 2.2 is the product of all the acoustic likelihoods $P(\mathbf{x}_t|s_i)$ and the transition probabilities $a_{i,j}$. In a more complicated HMM called *recognition network*, the scores on HMM-links will be a product of *transition probabilities*, *lexicon scores* and *language model scores*.

2.3.4 Hybrid acoustic model, context dependency, prior trick

As can be seen in a spectrogram (figure 2.3), speech is a continuum where one phoneme changes into another one without a clear boundary between the phonemes. Moreover, the realizations of phonemes are influenced by preceding and following phonemes, this influence is called *coarticulation*.

Inspired by this, a more precise modeling in ASR system is achieved by having context dependent phonemes, usually triphones. Each such unit is labeled as a triplet composed of the preceding, current and the following phoneme. With phone set size n, we get n^3 triphones. This can be a lot. For example, with 40 phonemes there are 64k units. Moreover,

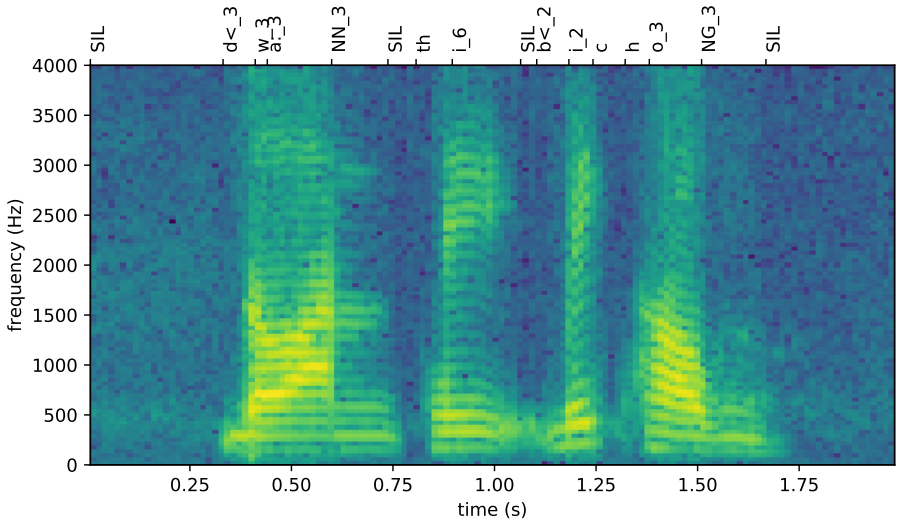


Figure 2.3: *Spectrogram of a sample Vietnamese expression. Note the phone-alignment on the top (from DNN).*

each triphone is described by a 3-state HMM, which further increases the overall number of context-dependent states. In practice, not all triphone combinations exist, and some may be very rare. Therefore, it is better to cluster the HMM-states corresponding to similar sounds into so called *tied-states*, which leads to a model that is easier to train. The tied-state clustering [Young and Woodland, 1994] is obtained by training a decision tree. It is trained by a top-down greedy splitting, where, in each step, we add a split that maximally increases the likelihood of data.

Coming back to the decoding formula (2.2), the term $P(\mathbf{X}|S)$ is given as follows:

$$P(\mathbf{X}|S) = \Pi_{t=1}^{T} P(\mathbf{x}_t|s_t) \ , \tag{2.5}$$

where s_t is the HMM state generating the feature frame $\mathbf{x}_t$.

In case of 'hybrid setup' (i.e. HMM decoding of NN outputs), the deep neural network produces posterior probabilities of tied states $P(s|\mathbf{x})$, while the formulation of *maximum a posteriori* decoding expects the acoustic scores as likelihoods $P(\mathbf{x}|s)$. To convert the posteriors into pseudo-likelihoods, we use:

$$P(\mathbf{x}|s) = \frac{P(s|\mathbf{x})}{P(s)} \ , \tag{2.6}$$

where $P(s)$ is the prior probability of acoustic unit s. The $P(s)$ can be estimated as relative frequency of s in the set of training labels [Bourlard and Morgan, 1993]. Or alternatively by marginalizing $\mathbf{x}$ from the posteriors by $P(s) = E_{\mathbf{x}}[P(s|\mathbf{x})]$ on a representative set of training data [Zhang et al., 2014b].

2.3.5 Compiling the recognition network

The decoder in Kaldi is based on WFST transducers [Mohri et al., 2002], and it uses a statically pre-compiled recognition network. The *weights* in WFST are defined with *semiring* $\mathbb{K}$, which is an abstract algebra formed from a carrier set K, additive operator $\oplus$

and multiplicative operator $\otimes$. The elements of carrier set K can be real numbers, this is used for recognition network. Or the elements can be tuples containing various types, which is used to represent Kaldi lattices. The WFST T over a semiring $\mathbb{K}$ is defined as 8-tuple:

$$T = (\Sigma, \Delta, \mathcal{Q}, \mathcal{I}, \mathcal{F}, E, \lambda, \rho) \ , \tag{2.7}$$

where:

- Σ is finite set of input symbols,
- Δ is finite set of output symbols,
- $\mathcal{Q}$ is finite set of states,
- $\mathcal{I}$ is finite set of initial states, $\mathcal{I} \subseteq \mathcal{Q}$,
- $\mathcal{F}$ is finite set of final states, $\mathcal{F} \subseteq \mathcal{Q}$,
- E is finite set of transitions, $E \subseteq \mathcal{Q} \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times K \times \mathcal{Q}$, where ϵ is an 'empty' symbol,
- λ is assignment of weights to initial states $\mathcal{I} \to K$,
- ρ is assignment of weights to final states $\mathcal{F} \to K$,

The finite set E is a description links in WFST. Each link has its source state $q_1 \in \mathcal{Q}$, input symbol from $(\Sigma \cup \{\epsilon\})$, output symbol from $(\Delta \cup \{\epsilon\})$, an element from carrier set of semiring $\mathbb{K}$ (i.e. the 'weight') and a target state $q_2 \in \mathcal{Q}$.

The theory of WFSTs describes following non-trivial operations: *composition*, *determinization*, *weight pushing* and *minimization*. These are formally introduced in [Mohri, 2004].

The *recognition network* is built by a composition of 4 graphs: $H \circ C \circ L \circ G$, where the operator '$\circ$' denotes WFST composition. In the composition $\circ$, the output symbols from the graph on the left are 'matched' with the input symbols of the graph on the right side, and the weights are combined appropriately.

The grammar **G** is represented as a WFST acceptor with word symbols and LM scores on links (in acceptor the input and output symbols are the same). The standard ARPA N-gram model can be converted to WFST format in which each distinct N-gram history will have its WFST state. Back-off states are also present in the WFST graph.

The lexicon **L** is represented as a transducer with phoneme input symbols and word output symbols. The eventual homophones and the words pronounced the same as a prefix of another word have appended disambiguation symbol after last phoneme, which is necessary to have a determinizable $L \circ G$ composition.

The graph **C** encodes the context dependency of phonemes: its input symbols are context dependent phonemes, while the output symbols are the context independent phonemes that were used in the lexicon. We generate all the possible contexts and the composition $C \circ L \circ G$ 'selects' the existing combinations, which implicitly introduces *cross-word context dependency* [Mohri et al., 1998]. Only the silence 'phones' are always context independent.

The **H** transducer defines the HMM topologies for triphones, which consist of three states connected by arcs with transition probabilities (see figure 2.2). The WFST graph contains HMMs for all the triphones, connected in parallel by union. The output symbols are triphones. In Kaldi, the input symbols are the *transition-id's*. Each *transition id* identifies the phone, its particular HMM state and the corresponding probability distribution by its *pdf-id*. The *pdf-id* corresponds to a GMM, NN output, or other source of 'acoustic likelihoods'. The translation of *transition-id* to *pdf-id* is done outside of recognition network in *transition model*. The transition model is usually embedded into the acoustic model file.

The preparation of HCLG decoding graph in Kaldi is expressed as follows:

$$HCLG = \text{asl}(\min(\text{rds}(\det(H \circ \min(\det(C \circ \min(\det(L \circ G)))))))) \tag{2.8}$$

Note that each composition '$\circ$' is followed by determinization det() and minimization min() steps. After the last composition, the disambiguation symbols are removed by rds() operator, and the HMM state self-loops are added into the graph by asl() operator.

2.3.6 Rewriting the decoding formula

To make the decoding formula (2.3) coherent with the implementation in the Kaldi decoder, we rewrite it as:

$$\hat{W} = \text{wrds}(\arg\max_{S} P_{AM}(\mathbf{X}|S)^{\kappa} P_G(S)^{\varrho}\ ), \tag{2.9}$$

where the κ is the *'acoustic scale'* applied as exponent to the acoustic likelihoods $P_{AM}(\mathbf{X}|S)$ and ϱ is the *'language model scale'* applied to the *graph scores* $P_G(S)$ (note that $P_G(S)$ contains the language model scores $P(W)$). The scales κ, ϱ are typically tuned to provide the best recognition performance on a development set. Usually, ϱ is fixed to 1.0 and the κ is tuned in range $[0.05, 0.1]$.

Scaling down the acoustic score by a small κ can be interpreted intuitively as follows: the duration of acoustic units usually spans over several time-steps of input features, and in the decoding, we multiply the likelihoods of all those frames. By exponentiating the scores with a small $\kappa = 1/K$, we get the K-th root of the acoustic score, which can be seen as the smooth version of taking the original unscaled scores from every K-th frame. From empirical experience, it seems that the optimal K is usually not too far from the average duration of a syllable, which is 156.2 ms [Kuwabara, 1996].

The graph score $P_G(S)$ consists of all scores on state sequence S except for the acoustic ones: it involves language model scores $P_{LM}(w_i|w_{i-1}, w_{i-2})$, the HMM transition probabilities $a_{s_t,s_{t+1}}$ and optionally the scores of pronunciation variants from lexicon $P_{LEX}(w_{i,v})$:

$$P_G(S) = \prod_{t=1}^{T-1} a_{s_t,s_{t+1}} \prod_{i=1}^{|\text{wrds}(S)|} P_{LEX}(w_{i,v})\, P_{LM}(w_i|w_{i-1}, w_{i-2}) \tag{2.10}$$

where $S = (s_1, s_2, \ldots, s_T)$ and $\text{wrds}(S) = (w_1, w_2, \ldots, w_M)$.

When implemented in decoders, the scores are usually accumulated in log-domain, to easily represent very small positive numbers arising from multiplication of many probabilities.

2.3.7 Lattices in WFST format

As we will be generating confidences based on lattice-posteriors, we will describe them together with some characteristics arising from the 'exact lattice' generation [Povey et al., 2012]. The lattice is a graph for representing alternative hypothesis of speech recognition that is in Kaldi represented by two types of WFSTs, which can be mutually converted.

The type `Lattice` is a trellis with per-frame arcs, here one HMM transition corresponds to one arc in the lattice. The input symbol is *transition-id*, output symbol is *word-id* or ϵ,

and the WFST weight is a tuple of acoustic score and graph score. This type is used mostly as an internal representation in the C++ code.

For storing lattices there is the type `CompactLattice`, it is a 'deterministic acyclic weighted acceptor' in which one arc corresponds to one word. Here, the input and output symbols are identical words, and the WFST 'weight' consists of: acoustic score, graph score and a sequence of transition-id's that is obtained over the word's duration.

Due to the determinization algorithm described in 'Generating exact lattices' [Povey et al., 2012], each distinct word sequence is present in lattice only once (i.e. lattice is deterministic), and with its best score. As a side effect of weight pushing, the positioning of scores and word-boundaries on a lattice path is not always properly synchronized in time with the original signal. The timing can be fixed by Kaldi tool `lattice-align-words-lexicon`.

2.3.8 Forward-backward algorithm for lattice

The forward-backward algorithm is used later for obtaining per-frame confidences of tied-states from `Lattice`. To get this, we first need to compute the *'responsibility'* $\gamma(a_j)$ representing a conditional probability of being in lattice-arc a_j, given some lattice $\mathcal{L}$. The identity of lattice-ark a_j encodes implicitly timing by length of any path leading to the arc, as in `Lattice`, each arc corresponds to one HMM transition. The *tied-state* of the arc is identified from *transition-id* in its input symbol.

The lattice link a_j is defined in Kaldi type `Lattice` by following elements:

$\mathsf{src}(a_j)$	source state in lattice from which the arc points out,
$\mathsf{tgt}(a_j)$	target state in lattice into which the arc is pointing,
$S_{input}(a_j)$	input symbol (transition id),
$S_{output}(a_j)$	output symbol (word id),
$(P_G(a_j), P_{AM}(a_j))$	WFST weight, tuple consisting of graph score and acoustic score (already scaled with graph scale ϱ and acoustic scale κ).

The posterior probability $\gamma(a_j)$ is then the *total probability* of crossing the arc a_j, computed as a ratio of score-sum of all paths that cross the arc (illustrated in figure 2.4) over the score-sum α_Ω of all paths in the lattice:

$$\gamma(a_j) = \frac{\alpha_{\mathsf{src}(a_j)}\, a_j\, \beta_{\mathsf{tgt}(a_j)}}{\alpha_\Omega} \,, \tag{2.11}$$

where $\alpha_{src(a_j)}$ are forward statistics computed as the sum of scores on all paths π in sub-lattice $\mathcal{L}_{\mathcal{I},\mathsf{src}(a_j)}$ that spans between initial state $\mathcal{I}$ and the source state of our arc a_j:

$$\alpha_{\mathsf{src}(a_j)} = \sum_{\pi \in \mathcal{L}_{\mathcal{I},\mathsf{src}(a_j)}} P(\pi) \,, \tag{2.12}$$

where a_j corresponds to the WFST weights on our link a_j:

$$a_j = P_G(a_j) P_{AM}(a_j) \,, \tag{2.13}$$

and where $\beta_{tgt(a_j)}$ are backward statistics computed as the sum of scores on all paths π in sub-lattice $\mathcal{L}_{\mathsf{tgt}(a_j),\Omega}$ that spans between the target state of our arc a_j and the final super-state Ω[2]:

[2]Because a WFST lattice can have more final states, we added final super-state Ω (according to WFST definition on page 16, the initial and final states have weights, other states have no weights).

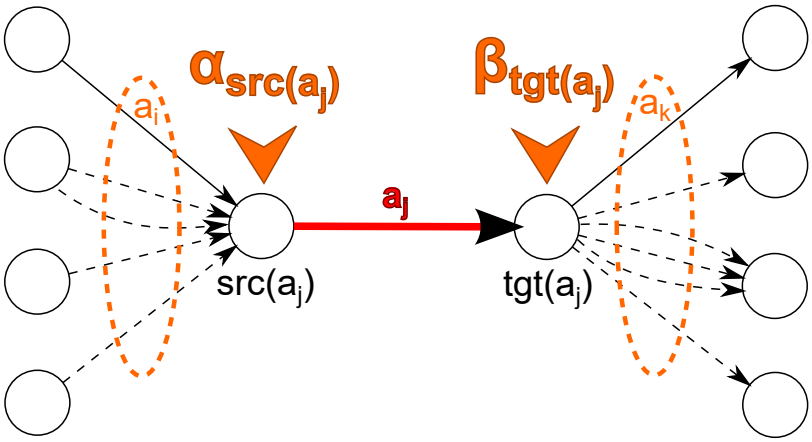


Figure 2.4: *Calculation of an arc-posterior $\gamma(a_j)$ in a lattice. Dynamic programming is used to obtain the statistics $\alpha_{\mathsf{src}(a_j)}, \beta_{\mathsf{tgt}(a_j)}$ calculated for all nodes in the lattice. The statistics represent the scores from all partial-paths leading into/from arc a_j.*

$$\beta_{\mathsf{tgt}(a_j)} = \sum_{\pi \in \mathcal{L}_{\mathsf{tgt}(a_j),\Omega}} P(\pi) \ . \tag{2.14}$$

The efficient calculation of statistics $\alpha_{\mathsf{src}(a_j)}$, $\beta_{\mathsf{tgt}(a_j)}$ is done for each state in the lattice by means of *dynamic programming* according to recursions, which sum over all incoming arcs a_i or outgoing arcs a_k:

$$\alpha_{\mathsf{src}(a_j)} = \sum_{a_i \in \{a_i | \mathsf{tgt}(a_i) = \mathsf{src}(a_j)\}} \alpha_{\mathsf{src}(a_i)} P_G(a_i) P_{AM}(a_i) \ , \tag{2.15}$$

$$\beta_{\mathsf{tgt}(a_j)} = \sum_{a_k \in \{a_k | \mathsf{src}(a_k) = \mathsf{tgt}(a_j)\}} \beta_{\mathsf{tgt}(a_k)} P_G(a_k) P_{AM}(a_k) \ . \tag{2.16}$$

Recall that, with Kaldi type `Lattice`, the duration of each arc is exactly one data-point of input features, hence the time info can be computed from number of traversed arcs. The initial conditions for recursions are $\alpha_{\mathcal{I}} = 1$ for initial state $\mathcal{I}$, and $\beta_{\Omega} = 1$ for terminal super-state Ω. The corresponding final alpha is:

$$\alpha_{\Omega} = \sum_{f \in \mathcal{F}} \alpha_f \, P_G(f) \tag{2.17}$$

where $\mathcal{F}$ is set of all final states in the lattice, these have state weights $P_G(f)$. The α_{Ω} is our normalizer term from (2.11).

A similar algorithm is used in Baum-Welch training in HTK-book [Young et al., 2002]. However, Baum-Welch algorithm considers a trellis of many HMM paths through a word, while in our algorithm, we consider only the best HMM path of each word, as we use 'exact lattice' generation [Povey et al., 2012]. Other difference is that we defined the algorithm for calculating posteriors of arcs, while the original HTK definition of Forward-Backward algorithm produced posteriors of HMM-states.

For the semi-supervised experiments we are primarily interested in posterior probability of *acoustic units* (tied-states) denoted as $\gamma(t, s)$, meaning a posterior of tied-state s at time t.

We use *transition model* to convert the posteriors of arcs $\gamma(a_j)$ to the posteriors of tied-states $\gamma(t, s)$. If several arcs are mapped to same tied-state at time t, we sum the posterior probabilities of all such arcs.

In later chapters (for example 6.2), we will use lattice-scale λ. Its purpose is to have a control over the 'uncertainty/sharpness' of the resulting lattice-posteriors. In our practical implementation, we use λ to simultaneously multiply the *acoustic scale* κ and *graphs scale* ϱ. This effectively exponentiates the scores of whole lattice-paths, making them closer or farther from the score of the best path, which translates to a change of 'sharpness' of posteriors probabilities.

2.4 The Backpropagation training algorithm

The backpropagation algorithm is the standard algorithm for neural network training. It can be found in the literature [Bishop, 2007], but we would like to explain it in our own illustrative way.

The principle of backpropagation algorithm is shown on the computation of the update (gradient) for a neural network with 1 hidden layer, while the extension to deeper networks is simple. We consider a network with sigmoid non-linearity and the classification output layer with softmax. The gradient is computed for single datapoint represented by an input feature vector $\mathbf{x}$ and its target vector with 1-of-K encoding $\mathbf{t} = [0\ 1\ 0\ \ldots\ 0]^T$, where the element '1' identifies the NN output of the 'correct' class.

Neural network as a feed-forward function

At first, we will define the neural network as a structured function with trainable parameters. Being a function, it maps the multidimensional inputs to the outputs. It is organized into 'layers', and the typical feed-forward network consists of several alternating linear and non-linear transformations. The smallest processing unit of a neural network is one *neuron*:

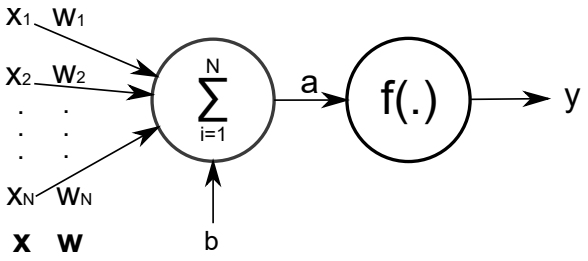


Figure 2.5: *Functional scheme of one neuron*

As illustrated in figure 2.5, the neuron has vector of inputs $\mathbf{x}$, and trainable parameters $\mathbf{w}$ and b. Functionally, it computes the activation a as a weighted combination of the inputs $\mathbf{x}^T\mathbf{w}$ plus bias term b. The output of the neuron y is obtained by transforming *activation* a with a differentiable non-linear *activation function* $f(.)$, which can be defined in many ways. The single neuron with logistic sigmoid activation function is capable of binary classification. The *neural network* is then composed of neurons going both into the width (parallel neurons in single layer) and the depth (serially connected layers of neurons).

Now let's return to our example network. As mentioned above, the activation of j-th

neuron in the first layer is given as:

$$a_j = \mathbf{w}_j^T \mathbf{x} + b_j, \tag{2.18}$$

each input feature x_i has its weight $w_{i,j}$, and a bias b_j is added. For convenience, we can group all the neurons in the first layer and form a weight matrix $\mathbf{W}^{(1)}$, in which j-th row is our weight vector $\mathbf{w}_j$. The biases are grouped into a vector $\mathbf{b}^{(1)}$. Then, the activation vector for all neurons in first layer is:

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \ . \tag{2.19}$$

Here we should emphasize that the input vector $\mathbf{x}$ is the same for all neurons in the first layer.

The key element which gives the neural network higher representative power than simple logistic regression is the nonlinearity in the hidden layers. Without the non-linearities, we could simply multiply all the weight matrices together to obtain the multi-class logistic regression with no hidden layers. A very popular nonlinearity is *logistic sigmoid*, which converts the numbers from interval $(-\infty, \infty)$ to probability-like numbers $[0, 1]$. The sigmoid is denoted with sigma σ :

$$h_j^{(1)} = \sigma(a_j^{(1)}) = \frac{1}{1 + \exp(-a_j^{(1)})} \tag{2.20}$$

The output of the sigmoid is the *hidden vector* $\mathbf{h}^{(1)}$, it can be seen as an intermediate encoding of the input features, on the way towards the output of the neural network. Usually we are not interested in the actual values there, which is why we call the layers 'hidden'. It is important that we can calculate the output of the neural network from these values.

Analogically to (2.19), the hidden vector $\mathbf{h}^{(1)}$ is transformed with another affine transform to the second layer activations $\mathbf{a}^{(2)}$:

$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)} \ . \tag{2.21}$$

From these, the posterior probabilities of classes y_i are computed using the softmax function:

$$y_i = \frac{\exp(a_i^{(2)})}{\sum_j \exp(a_j^{(2)})}, \quad y_i \in \mathcal{R}_0^+, \ \sum_i y_i = 1 \tag{2.22}$$

As illustrated in (2.22), the softmax function ensures that we always obtain positive quantities which sum-up to one.

To illustrate that our example neural network is one big structured function, we show the forward-propagation formula in which we compute the output $\mathbf{y}$ from the input vector $\mathbf{x}$:

$$\mathbf{y} = \text{softmax}\left(\mathbf{W}^{(2)}\ \sigma\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right) \tag{2.23}$$

An alternative look at formula (2.23) is shown in figure 2.6, where the nodes represent individual neurons and arcs the interconnections. The first layer of neurons produces the hidden vector $\mathbf{h}$, while the second layer produces the output $\mathbf{y}$. The trainable parameters are weight matrices $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$ and bias vectors $\mathbf{b}^{(1)}, \mathbf{b}^{(2)}$. In this example, all the functions we used are differentiable, so we can calculate the partial derivatives for the backpropagation

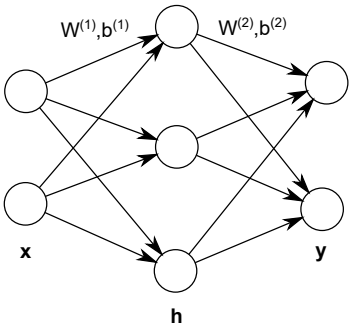


Figure 2.6: *Structure of neural network, the nodes represent neuron outputs.*

algorithm. In fact, the functions are also smooth, which is however not strictly required for the training algorithm.

A related interesting question is: What is the set of functions that can be represented by a neural network? The studies from Cybenko [Cybenko, 1989] and Barron [Barron, 1993] show that theoretically, they can approximate any continuous function with arbitrary precision, if the neural network with one hidden layer has sufficient number of sigmoid neurons. Although this work does not say how to train them, the statement is very promising.

From the practical experience, it is commonly agreed that it is advantageous to use many hidden layers, which is the origin of the very frequently used term *'Deep Neural Networks'*. In our recipes, we usually use 6 hidden layers. A popular explanation is that with more layers we better smooth out the feature variability that is irrelevant to the classification task, and that the layers near the network output encode more complex features. However, this intuitive view is not easy to be validated practically.

Loss function, multi-class cross-entropy

Before we can derive the backpropagation training, we need to define a loss function to optimize. For the classification of n-th data-point into 1-of-K mutually exclusive classes, the natural loss function is the multi-class cross-entropy (CE):

$$E_n(\mathbf{w}) = -\sum_{k=1}^{K} t_k \ln y_k \ . \tag{2.24}$$

Recall that vector $\mathbf{t}$ has 1-of-K encoding $\mathbf{t} = [\,0\ 1\ 0\ \ldots\ 0\,]^T$, so that the sum picks-up the k-th element, which corresponds to the sole non-zero target. The loss value is always positive and reaches zero minimum, if the posterior vector $\mathbf{y}$ exactly matches the target vector $\mathbf{t}$. (note that in this section we denote time-id with subscript n to avoid confusion with training labels $\mathbf{t}$, in other chapters time is denoted with t subscript)

In the more general case, when the 'soft' probabilistic targets are used (the 1-of-K encoding of $\mathbf{t}$ is replaced by a vector with positive values, which sum-up to one), it is convenient to replace the cross-entropy with KL-divergence:

$$E_n(\mathbf{w}) = D_{KL}(\mathbf{t}||\mathbf{y}) = -\sum_{k=1}^{K} t_k \ln \frac{y_k}{t_k} \tag{2.25}$$

the difference is that the KL-divergence subtracts the entropy of the target labels and reaches its zero minimum when $\mathbf{y} = \mathbf{t}$, while the cross-entropy would have a non-zero value

if $t_k \notin \{0, 1\}$. With targets $\mathbf{t}$ in 1-of-K encoding, both functions become the same. The derivatives with respect to neural network outputs are the same for both functions, and in Kaldi 'nnet1' we implemented (2.25).

When training on a data-set composed of T data-points, the overall loss is the sum of the per-frame values:

$$E = \sum_{n=1}^{T} E_n \ . \tag{2.26}$$

An interesting value for the log-prints is the per-frame average $\bar{E} = E/T$, which can be converted to the geometrical-average posterior value of the correct class by $\bar{p}_{corr} = \exp(-\bar{E})$, if we assume to have the 1-of-K targets.

Stochastic gradient descent, update rule

The most popular training algorithm for neural networks is *Stochastic Gradient Descent* (SGD). The other frequent term 'backpropagation' refers to the way how the SGD gradient is computed from the neural network.

The idea behind *gradient descent* training is to greedily minimize the loss by doing small parameter steps in the direction of the opposite gradient, i.e. the direction of steepest descent of the loss.

Stochastic Gradient Descent training is a gradient descent, in which the model is updated on-line after processing a single or a small group of randomly selected training data-points. It is possible that, while the loss decreases on some samples, we can see a loss increase for other samples. The overall trend 'steers' the model towards the regions, where the loss is low. Hence, the training progress is noisier and more 'exploratory', and this lowers the risk of converging to a poor local minimum.

Due to practical reasons, we usually do mini-batch SGD training, in which we calculate the gradients for M data-points together (default $M = 256$). The data-points are grouped into matrices where, for example, the matrix-vector multiplication in (2.23) becomes matrix-matrix multiplication. This can better employ hardware by reusing data elements in cache and better caching accelerates the passes through training data (i.e. *epochs*) considerably.

The update formula for the mini-batch SGD is:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \sum_{n=1}^{M} \nabla E_n(\mathbf{w}^{(\tau)}) \ , \tag{2.27}$$

where $\mathbf{w}^{(\tau)}$ is a vector with all the trainable parameters, and $\nabla E_n(\mathbf{w}^{(\tau)})$ is the gradient of E_n w.r.t. $\mathbf{w}^{(\tau)}$ calculated on a single data-point $\mathbf{x}_n$. Note that we sum the gradients from M data-points, and that η is *learning-rate*, i.e. a scalar controlling step-size during training. The suitable learning-rate value needs to be tuned carefully and depends on many factors (network-type, non-linearity type, loss type, size of mini-batch, per-utterance training, etc). By the nature of SGD training, the updates of the parameters are 'noisy' with a 'correct' global trend. Therefore, we can see the learning-rate as the temperature of simulated annealing. By controlled 'cooling' (i.e. decreasing) of the learning-rate, we can reduce the exploration while approaching the end of the training and let the network converge to a better solution.

Despite that SGD is a greedy first-order optimization method, it is known to converge fast on the data-sets containing many similar training examples [Bishop, 2007]. The final

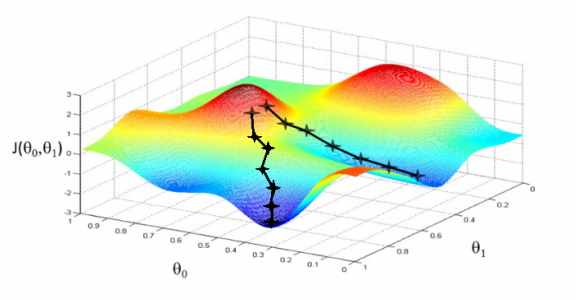


Figure 2.7: *Gradient descent, converging to different local optima depending on initialization.*

set of parameters depends on the initialization (illustrated in figure 2.7), and there typically exist many weight permutations with identical functionality (swapped neurons). Hence, it is a good practice to try several initializations to see the eventual differences in performance.

The size of mini-batch M should not be too small as it leads to a less efficient caching (training run-time is slower), nor too large as it reduces the number of updates per epoch (the overall progress of the training would become slower, more epochs will be needed) a good default value is 256 data-points.

Calculating the gradient by backpropagation

Now let's derive the gradient $\nabla E_n(\mathbf{w}^{(\tau)})$ for a single data-point n from a mini-batch (the gradient of mini-batch is a sum of the data-point gradients). We extensively use chain rule to obtain Jacobian of function composition $f \circ g$:

$$J_{f\circ g}(\mathbf{a}) = J_f(g(\mathbf{a}))J_g(\mathbf{a}) \;, \tag{2.28}$$

where f is the outer and g the inner function, while both functions are multi-dimensional. The composition of Jacobians is linear, it is the product of two Jacobian matrices $J_f J_g$. In J_g the row index corresponds to individual output of g, the column index corresponds to input of g. The rule (2.28) can be used recursively, so that the Jacobian of the innermost function appears on the right of the series of matrix multiplications:

$$J_{f_1\circ f_2\circ f_3}(\mathbf{a}) = J_{f_1}(f_2(f_3(\mathbf{a})))\; J_{f_2}(f_3(\mathbf{a}))\; J_{f_3}(\mathbf{a}) \;, \tag{2.29}$$

The reader might notice that we will consistently use the 'numerator layout convention' of matrix calculus, in which the derivative keeps the orientation of numerator and flips the orientation of denominator. The layout of $\frac{\partial y}{\partial \mathbf{x}}$ is according to $\mathbf{x}^T$, layout of $\frac{\partial y}{\partial \mathbf{X}}$ is according to $\mathbf{X}^T$. And the layout of Jacobian matrix $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is according to orientation of $\mathbf{y}$ and $\mathbf{x}^T$.

Derivative of output activations: Now let's apply the chain rule to gradient computation. The Jacobian of the multi-class cross-entropy E_n (2.24) w.r.t. neural network outputs $\mathbf{y}$ is a row vector:

$$\frac{\partial E_n}{\partial \mathbf{y}} = \left(-\operatorname{diag}(\mathbf{y})^{-1}\,\mathbf{t}\right)^T \;, \tag{2.30}$$

and the Jacobian of softmax function is a symmetric matrix:

$$\mathbf{J}_{\text{softmax}} = \frac{\partial \mathbf{y}}{\partial \mathbf{a}^{(2)}} = \operatorname{diag}(\mathbf{y}) - \mathbf{y}\mathbf{y}^T \;. \tag{2.31}$$

By using the chain rule 2.28, we obtain the result:

$$\frac{\partial E_n}{\partial \mathbf{a}^{(2)}} = \left(-\text{diag}(\mathbf{y})^{-1}\,\mathbf{t}\right)^T \left(\text{diag}(\mathbf{y}) - \mathbf{y}\mathbf{y}^T\right) = (\mathbf{y}-\mathbf{t})^T \tag{2.32}$$

It is not a coincidence that the derivative simplifies: the softmax is the inverse canonical link function for the cross entropy loss. The same expression $(\mathbf{y}-\mathbf{t})^T$ appears also for other couples: the logistic sigmoid and two-class cross-entropy, and also for 'identity' activation and the mean square error.

Gradient of the parameters in the second layer: To calculate the gradient of loss E_n w.r.t trainable parameters from the second layer $\mathbf{W}^{(2)}$ $\mathbf{b}^{(2)}$, we begin with their derivatives from affine transform (2.21):

$$\frac{\partial a_j^{(2)}}{\partial W_{i,j}^{(2)}} = h_i^{(1)} \quad \frac{\partial a_j^{(2)}}{\partial b_j^{(2)}} = 1\ , \tag{2.33}$$

By using these in the chain rule, we get the following elements in gradient of loss:

$$\frac{\partial E_n}{\partial W_{i,j}^{(2)}} = \frac{\partial E_n}{\partial a_j^{(2)}} \frac{\partial a_j^{(2)}}{\partial W_{i,j}^{(2)}} = (y_j - t_j)\, h_i^{(1)} \tag{2.34}$$

$$\frac{\partial E_n}{\partial b_j^{(2)}} = \frac{\partial E_n}{\partial a_j^{(2)}} \frac{\partial a_j^{(2)}}{\partial b_j^{(2)}} = (y_j - t_j)\, 1 \tag{2.35}$$

The same rewritten for the whole weight matrix and bias vector:

$$\frac{\partial E_n}{\partial \mathbf{W}^{(2)}} = \mathbf{h}^{(1)}(\mathbf{y}-\mathbf{t})^T \quad \frac{\partial E_n}{\partial \mathbf{b}^{(2)}} = (\mathbf{y}-\mathbf{t})^T \tag{2.36}$$

Note that according to 'numerator layout convention', the derivatives have transposed orientation compared to model parameters $\mathbf{W}^{(2)}$ $\mathbf{b}^{(2)}$.

Gradient of the parameters in the first layer: To calculate the gradient of trainable parameters from the first layer $\mathbf{W}^{(1)}$ $\mathbf{b}^{(1)}$, we need the Jacobian of loss derived w.r.t. activations $\mathbf{a}^{(1)}$. Recall that $\mathbf{a}^{(2)}$ is computed from $\mathbf{a}^{(1)}$ by: $\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\sigma(\mathbf{a}^{(1)}) + \mathbf{b}^{(2)}$, which is a composite function created from the element-wise logistic sigmoid (2.20) and the affine transform (2.21). Hence by substituting into the chain rule (2.29) we get:

$$\frac{\partial E_n}{\partial \mathbf{a}^{(1)}} = \frac{\partial E_n}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}}\, \mathbf{J}_\sigma\ , \tag{2.37}$$

where the outer Jacobian is the already existing expression (2.32), the middle Jacobian comes from the second layer affine transform (2.21), and the innermost Jacobian $\mathbf{J}_\sigma$ corresponds to activation function σ (2.20). The Jacobian $\mathbf{J}_\sigma$ is, for our logistic sigmoid, equal to:

$$\mathbf{J}_\sigma = \text{diag}(\mathbf{h}^{(1)})\ \text{diag}(1 - \mathbf{h}^{(1)})\ , \tag{2.38}$$

and the middle Jacobian can be rewritten as:

$$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial}{\partial \mathbf{h}^{(1)}} \left[\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right] = \mathbf{W}^{(2)}\ , \tag{2.39}$$

hence we can rewrite the desired Jacobian of loss as:

$$\frac{\partial E_n}{\partial \mathbf{a}^{(1)}} = \frac{\partial E_n}{\partial \mathbf{a}^{(2)}} \, \mathbf{W}^{(2)} \, \mathbf{J}_\sigma \; . \tag{2.40}$$

Then, analogically to (2.33), the partial derivatives of parameters $\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$ from the first affine transform (2.19) are:

$$\frac{\partial a_j^{(1)}}{\partial W_{i,j}^{(1)}} = x_i \quad \frac{\partial a_j^{(1)}}{\partial b_j^{(1)}} = 1 \; , \tag{2.41}$$

so we can finally substitute (2.32) into (2.40) and apply the chain rule together with (2.41) to get the gradients:

$$\frac{\partial E_n}{\partial \mathbf{W}^{(1)}} = \mathbf{x} \left[(\mathbf{y} - \mathbf{t})^T \, \mathbf{W}^{(2)} \, \mathbf{J}_\sigma \right] \quad \frac{\partial E_n}{\partial \mathbf{b}^{(1)}} = (\mathbf{y} - \mathbf{t})^T \, \mathbf{W}^{(2)} \, \mathbf{J}_\sigma \; . \tag{2.42}$$

Assembling the gradient for update rule: In update rule (2.27), the gradient $\nabla E_n(\mathbf{w}^{(\tau)})$ is represented as a single vector of partial derivatives w.r.t. all model parameters $\mathbf{w}^{(\tau)}$. We reshape the partial derivatives accordingly:

$$\nabla E_n(\mathbf{w}^{(\tau)}) = \text{vec}\left(\frac{\partial E_n}{\partial \mathbf{W}^{(2)}}, \frac{\partial E_n}{\partial \mathbf{b}^{(2)}}, \frac{\partial E_n}{\partial \mathbf{W}^{(1)}}, \frac{\partial E_n}{\partial \mathbf{b}^{(1)}} \right) , \tag{2.43}$$

where the operation vec(.) concatenates the elements from all its arguments into a single vector.

Backpropagation, extension to L hidden layers

The idea of backpropagation is well illustrated in formula (2.40): the error derivatives w.r.t. activations in second layer are recomputed into derivatives w.r.t. activations in preceding layer. To support an arbitrary number of hidden layers, we can re-write this formula into a generic version which converts the loss derivative from L-th layer to $(L-1)$th layer:

$$\frac{\partial E_n}{\partial \mathbf{a}^{(L-1)}} = \frac{\partial E_n}{\partial \mathbf{a}^{(L)}} \, \mathbf{W}^{(L)} \, \mathbf{J}_\sigma \; . \tag{2.44}$$

We use the formula iteratively from the last layer towards the first one. This computation pattern of back-propagating the error derivatives is the origin of the term *'back-propagation'*. The gradients are then computed as:

$$\frac{\partial E_n}{\partial \mathbf{W}^{(L)}} = \mathbf{h}^{(L-1)} \, \frac{\partial E_n}{\partial \mathbf{a}^{(L)}} \; , \quad \frac{\partial E_n}{\partial \mathbf{b}^{(L)}} = \frac{\partial E_n}{\partial \mathbf{a}^{(L)}} \; , \tag{2.45}$$

where $\mathbf{a}^{(L)}$ is the activation of the current layer and the $\mathbf{h}^{(L-1)}$ is the non-linearity output from the previous layer, which becomes the input vector $\mathbf{x}$ for the first layer. Note that the Jacobian $\frac{\partial E_n}{\partial \mathbf{a}^{(L)}}$ is a row vector, as we use 'numerator layout convention'.

Alternative activation functions

In general, other activation functions can be used in NN by changing its forward-propagation formula and the corresponding Jacobian for the back-propagation.

In our example case, we would replace the logistic sigmoid (2.20) and its Jacobian (2.38). The popular alternatives are Hyperbolic tangent (Tanh), Rectified linear units (ReLU) [Glorot et al., 2010, Zeiler et al., 2013], Maximum output (MaxOut) [Goodfellow et al., 2013, Miao et al., 2013, Swietojanski et al., 2014], p-norm [Zhang et al., 2014b] or Identity function. The Identity function (i.e. the linear activation function $\mathbf{y} = \mathbf{x}$) is useful for a 'narrow' bottleneck layer inside a neural network [Veselý et al., 2011] or for factorizing output layer [Sainath et al., 2013].

Relation to logistic regression

If we compare the gradient of the last layer (2.36) with the form of gradient for multi-class logistic regression in Bishop [Bishop, 2007, page 209, eq. (4.109)], we see that they are identical. Therefore, it is meaningful to perceive the neural network as a logistic regression, in which the non-linear basis functions (i.e. the feature transformations) are parametrized by the hidden layers of neurons. The difference is that for neural networks, the basis functions are learned together with the classifier. In the case of the logistic regression, the non-linear basis functions are typically pre-defined and fixed. Or, alternatively, we can see logistic regression as a neural network with no hidden layer.

At the same time, the logistic regression can perfectly classify only simpler problems, where the classes are linearly separable. From this, we can infer that the hidden layers should learn such feature representations, that become close to be linearly separable in the last hidden layer. However, the 'quality of separation' depends on the data-set, the similarity of classes and the overall difficulty of the task to be learned.

At this point, we end our introduction to the neural network based speech recognition. The reader should already have an idea about the general structure of the system and the principle of back-propagation training.

Chapter 3

Kaldi 'nnet1' DNN training recipe

After the introduction, we proceed with the description of the DNN training recipe that we implemented into the open-source toolkit Kaldi. In this chapter, we first cover the general experimental setup and continue with sections describing RBM pre-training, mini-batch frame cross-entropy training and sequence-discriminative sMBR training. The chapter ends with notes on the scalability of the mini-batch training.

Acoustic units

As mentioned earlier, the DNN acoustic model provides the posterior probabilities $P(c|\mathbf{x})$ for a closed set of acoustic units. The typical acoustic units are the clustered states from 3-state HMMs, which model the context-dependent (CD) phonemes. Such units are often referred to as *tied-states*, *CD-states* or *senones*. The clustering is determined by a decision tree, which is built by a greedy rule that adopts the splits with the best increase of the data likelihood. Depending on the size of training set there are usually thousands of CD-states.

The neural network is trained to classify them exclusively, the Softmax function from equation (2.22) in the output layer ensures that the posteriors of all the acoustic units are non-negative and sum to one. For decoding, the posteriors are converted to pseudo-likelihoods $P(\mathbf{x}|c)$ by dividing them with priors $P(c)$ as illustrated earlier in equation (2.6).

Input features

There are many ways how to prepare the input features for a neural network, and new feature extraction methods are published at every conference. Usually, the feature vectors on NN input cover time period 150-300ms, and they are assembled from short-term feature vectors. The short-term features are typically computed from 25ms frames of speech extracted with 10ms steps.

The features we use are the PLP-fMLLR speaker-adapted features. To produce them, we need an initial GMM-HMM system, which is used to estimate the speaker specific linear transform fMLLR.

The feature extraction pipeline in figure 3.1 shows that we begin from the PLP+pitch short-term features. Then, there are two stages, the first with a GMM model and the second with DNN model.

The GMM-HMM features are obtained by splicing 9 frames of the 'short-term' feature vectors (4 on each side from the 'current' frame). The short-term features are 13-dimensional PLPs (including C0) extended by 3 Kaldi-pitch features [Ghahremani et al., 2014] (probability of voicing, pitch, delta pitch), which are both mean-variance normalized by CMVN.

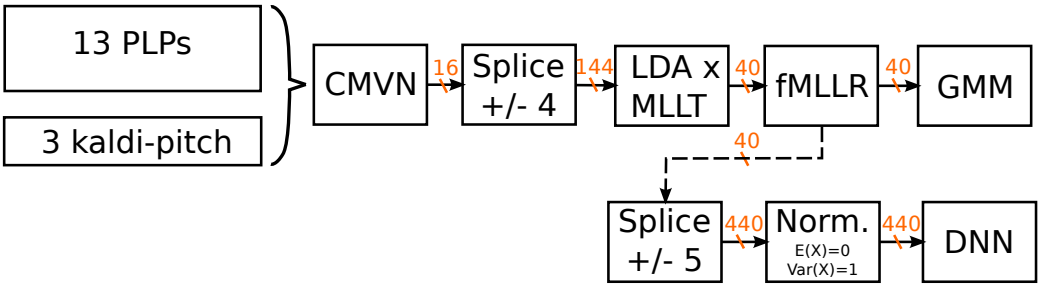


Figure 3.1: *Input features of the DNN.*

Our telephone speech has 8kHz sampling frequency, so the PLPs are computed from the frequency range 125-3800Hz and we use dithering. The spliced features were projected down to 40 dimensions using a global LDA-MLLT linear transform and per-speaker fMLLR linear transform. The fMLLR transform is obtained with a GMM trained in the adapted feature-space. For test data, the transformation is obtained by the multi-pass decoding with the GMM model.

Then, for the DNN input, we splice 11 frames of the 40-dimensional fMLLR features (5 on each side of the current frame), and we rescale them globally to have zero mean and unit variance. The total dimension of DNN input is 11 x 40 = 440.

This setup is used in most of the experiments, the exceptions are described locally in the text. Eventually, we can obtain a small improvement in recognition accuracy by replacing the LDA+MLLT+fMLLR linear transform with a non-linear bottleneck network and fMLLR. However this would make the analysis of semi-supervised DNN training more difficult, so we keep using this simpler setup.

NN topology, initialization

The typical neural network we use in this thesis has 6 hidden layers of 2048 Sigmoid neurons. The neural network has 440 inputs (the spliced fMLLR features) and thousands of Softmax outputs (for example for Vietnamese 4599 outputs).

There are three ways to initialize a neural network, either with a) small random numbers or by b) unsupervised pre-training with RBMs or by c) supervised discriminative pre-training. In our recipe, we **use the RBM pre-trainig**, which allows us to use an *'universal'* topology and obtain good results for many training sets (except the really small ones for which we need to use smaller network).

Data randomization, mini-batch training

The idea of Stochastic Gradient Descent training (SGD) is to randomly draw samples from the distribution of the training data and perform small updates of model parameters in the opposite direction of the gradient of a loss function, which decreases the loss.

Having a training data-set of finite size, the *'sampling'* is usually understood as randomizing the order in which the data is used for training. Hence, we typically shuffle the list of training sentences. Then, for the mini-batch training as introduced in equation (2.27) and also for the RBM pre-training, there is an additional frame-level shuffling mechanism inside the training tools. We can use the fixed 'random' order for all the epochs, usually it has little effect on the final results and the experiments become replicable.

For the recurrent networks, we need a different approach. We need to keep the continuity

of the sentence, so we cannot shuffle the speech frames. Instead, we can process several sentences in parallel, which accelerates the training as more frames are processed in one step.

3.1 Training the DNN acoustic model

The training recipe consists of RBM pre-training, frame classification training with cross-entropy loss function and sequence discriminative training with sMBR loss function. All the three steps are described in the following sections.

3.1.1 Pre-training with Deep Belief Network (Restricted Boltzmann Machines)

The Deep Belief Networks (DBN) were a hot topic in the DNN based speech recognition in 2010. The model and its theory were developed in the laboratories of Geoffrey Hinton [Hinton et al., 2006] and Yoshua Bengio [Bengio et al., 2007]. We can see the DBN pre-training as one of the regularization methods as it both reduces over-fitting and improves the results, when compared to the randomly initialized network. A great source of DBN related information is the 'Practical guide' from Geoff Hinton [Hinton, 2012], which we used as a basis for our implementation.

Deep belief nets are probabilistic generative models that are composed of multiple layers of stochastic, latent variables. The latent variables typically have binary values and are often called hidden units or feature detectors. The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. The states of the units in the lowest layer represent a data vector.

Scholarpedia[1]

According to this definition, DBN could be used to generate speech data. However, we typically use it for the unsupervised initialization of the hidden layer parameters. It learns the structure of the data before we start the supervised training, in which only the output layer is initialized randomly. The theoretical backgrounds of generative DBN and discriminative DNN are very different, however the neurons are organized in a 'compatible' way, which allows us to import them. Briefly, the RBMs (i.e. the 'layers' in DBN) are trained by lowering the 'energy' with the Contrastive Divergence algorithm [Hinton et al., 2006] based on Gibbs sampling.

Thanks to pre-training a DBN, the over-fitting is less of a problem, which is particularly helpful for small training databases. For larger databases, we still obtain a small improvement, as illustrated in table 3.1 in which we always see a gain from the DBN pre-training. The WER improvement for Switchboard is 0.3%, the AMI IHM improved by 0.4% as we recently published in [Veselý et al., 2016, table 2]. The large gain of 1.7% for Babel Vietnamese LimitedLP condition is caused both by smaller amount of training data and higher error rate.

Other benefit of DBN pre-training is that it aleviates the *gradient vanishing/explosion problem* [Hochreiter, 1991], which can easily happen if the random initialization of the deep

[1] http://www.scholarpedia.org/article/Deep_belief_networks

Table 3.1: *Comparing randomly initialized NN and DBN pre-training. The topology of 6 sigmoid hidden layers, 2048 neurons each.*

	Training data	Random init.	DBN pre-training
Switchboard (eval2000)	300 hours	19.0	18.7
AMI IHM (dev)	77 hours	26.4	26.0
AMI IHM (eval)	77 hours	27.0	26.6
Vietnamese (dev)	10 hours	62.5	60.8

model is not done carefully. The DBN pre-training is done layer-wise, while each time the top-layer is seen as a Restricted Boltzmann Machine (RBM). The layers below have 'fixed' parameters, and only the top layer is being trained.

Restricted Boltzmann Machine The *Boltzmann Machine* is a stochastic undirected graphical model, in which each bidirectional link has a trainable weight. For now, we are considering to have binary nodes, with two possible states $\{0, 1\}$. The model was introduced in 1985 by Geoffrey Hinton and Terry Sejnowski [Ackley et al., 1985]. *Restricted Boltzmann Machine* is a special form of a *Boltzmann Machine* with nodes organized as a bipartite graph. The division into two disjoint sets of 'visible' and 'hidden' nodes (i.e. random variables) simplifies the inference in the model, because the nodes within each set are conditionally independent given the other set. The RBM was originally invented under the name Harmonium by Paul Smolensky in 1986 [Smolensky, 1986], at the time with a different training method.

The model is defined by *joint probability* that is assigned to particular values of 'visible' and 'hidden' random variables that are stored in visible vector $\mathbf{v}$ and hidden vector $\mathbf{h}$ as follows:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z_{h,v}} e^{-E(\mathbf{v},\mathbf{h})} \tag{3.1}$$

where $Z_{h,v} = \sum_{\mathbf{h}} \sum_{\mathbf{v}} e^{-E(\mathbf{v},\mathbf{h})}$ is the normalizing partition function. For our RBM with binary nodes, the *energy function* is defined as:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h}\,, \tag{3.2}$$

which assigns 'energy' to each vector-pair $(\mathbf{v}, \mathbf{h})$, given the model parameters. This energy function also defines the graphical model of RBM shown in figure 3.2. The parameters to optimize are the matrix with the bidirectional weights $\mathbf{W}$ and the bias vectors $\mathbf{a}$, $\mathbf{b}$. Now let's illustrate how simple the inference in the model is. If we know the values of the visible

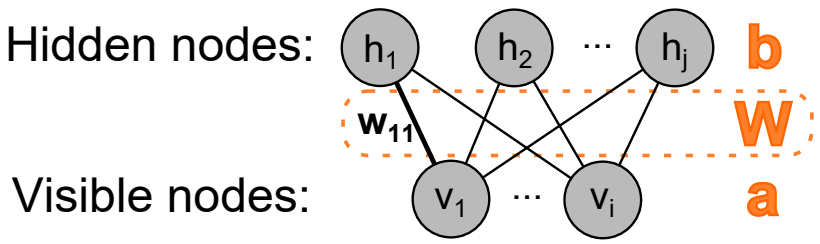


Figure 3.2: *Structure of RBM*

random variables $\mathbf{v}$, the conditional probability that binary hidden unit h_i is set to one is expressed as:

$$p(h_j = 1|\mathbf{v}) = \sigma\left(\left[\mathbf{W}\mathbf{v} + \mathbf{b}\right]_j\right), \tag{3.3}$$

where $[.]_j$ selects j-th element from vector in brackets and σ is logistic sigmoid $y = \frac{1}{1+exp(-x)}$. Having the Bernoulli conditional probabilities $p(h_i = 1|\mathbf{v})$, we can sample them to get the vector of values $\mathbf{h_S}$. Then, given $\mathbf{h_S}$, the conditional probability that the binary visible nodes are set to one is analogical:

$$p(v_i = 1|\mathbf{h_S}) = \sigma\left(\left[\mathbf{W}^T\mathbf{h_S} + \mathbf{a}\right]_i\right), \tag{3.4}$$

both steps are used in the Contrastive Divergence training algorithm.

Contrastive Divergence training In the training, we use the input features (or the output of previous layer) as the visible vectors $\mathbf{v}$. The true likelihood of the observed data $\mathbf{v}$ would be:

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v}',\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')}}, \tag{3.5}$$

which is not tractable. On the other hand, the derivative of the log-likelihood with respect to the bidirectional weight is simple:

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{i,j}} = \langle v_i h_j \rangle_{data} - \langle v_i' h_j' \rangle_{model}, \tag{3.6}$$

where the angle brackets denote the expectations under the distributions denoted by the subscript. As we can see, the derivative has two terms related to the likelihood function (3.5). The first term increases the numerator, by lowering the energy of configurations with the fixed features $\mathbf{v}$. The second term decreases the denominator by increasing the energy of the 'other likely data' generated by the model. The steepest ascent update rule for weights is then:

$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle_{data} - \langle v_i' h_j' \rangle_{model}), \tag{3.7}$$

where η is the learning rate. Getting the first term $\langle v_i h_j \rangle_{data}$ is trivial as $\mathbf{v}$ are the 'observed' values of visible variables and h_j is the conditional probability $p(h_j = 1|\mathbf{v})$ from equation (3.3). Using the conditional probability $p(h_j = 1|\mathbf{v})$ in expectation $\langle . \rangle_{data}$ is better than using samples $\mathbf{h_S}$, as it already is the expected value from a large population of $\{\mathbf{h_S}|\mathbf{v}\}$. Obtaining the expectation $\left\langle v_i' h_j' \right\rangle_{model}$ is more difficult. For this we should run Gibbs sampling with many steps.

In the Contrastive Divergence training (CD-1), the Gibbs sampling is *truncated to one step*, which makes the algorithm only poorly approximate maximum likelihood training. Still, it is good enough to train our RBMs and the training is fast [Hinton et al., 2006].

The single-step Gibbs sampling with one data-point is illustrated in algorithm 1, which we also visualize in figure 3.3. We take the 'visible' vector $\mathbf{v}$, and using by equation (3.3), we calculate the vector of conditional probabilities $\mathbf{h_p}$. From $\mathbf{h_p}$ we sample a vector of binary values $\mathbf{h_S}$. With this, using equation (3.4), we calculate vector of conditional probabilities of values in 'data reconstruction' $\mathbf{v_p'}$. From $\mathbf{v_p'}$ we directly infer the vector of conditional probabilities of values in hidden nodes $\mathbf{h_p'}$ by using equation 3.3. The outputs $\mathbf{v}, \mathbf{h_p}, \mathbf{v_p'}, \mathbf{h_p'}$ are then used in the expectations from update formula (3.7).

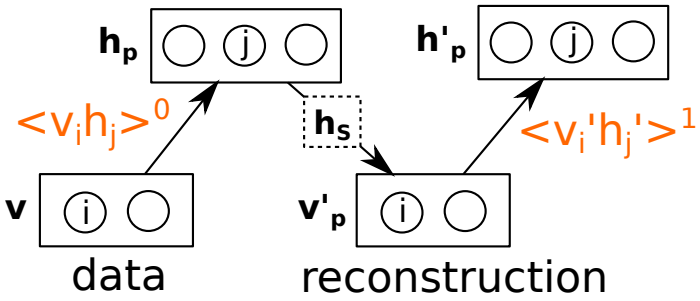


Figure 3.3: *Contrastive divergence, single-step Gibbs sampling*

Algorithm 1 Single step of Gibbs sampling in RBM training

function GIBBSSTEP($\mathbf{W}, \mathbf{a}, \mathbf{b}$)
  $\mathbf{v} \leftarrow$ get visible vector from mini-batch
  $\mathbf{h_p} \leftarrow \{h_j | h_j = \sigma([\mathbf{W}\mathbf{v} + b]_j)\}$ ▷ get conditional probabilities $p(h_j = 1|\mathbf{v})$
  $\mathbf{h_S} \leftarrow \mathsf{sample}(\mathbf{h_p})$ ▷ convert probabilities to binary values.
  $\mathbf{v'_p} \leftarrow \{v_i | v_i = \sigma([\mathbf{W}^T\mathbf{h_S} + a]_i)\}$ ▷ get cond. prob. of 'data reconstruction'
  $\mathbf{h'_p} \leftarrow \{h_j | h_j = \sigma([\mathbf{W}\mathbf{v'_p} + b]_j)\}$ ▷ get cond. prob. from 'expected reconstruction'
  return $(\mathbf{v}, \mathbf{h_p}, \mathbf{v'_p}, \mathbf{h'_p})$
end function

In the single-step Gibbs sampling, we have the 'positive statistics' $\langle v_i h_j \rangle^0$ from its 0-th step, which correspond to $\langle . \rangle_{data}$. From its first step, we have the 'negative statistics' $\langle v_i h_j \rangle^1$ which approximate the expectation $\langle . \rangle_{model}$. Each model update is done with expectations $\langle . \rangle$ computed over a mini-batch of 100 data-points, and the SGD training has *momentum* applied to all RBM parameters $\mathbf{W}, \mathbf{a}, \mathbf{b}$, while *weight decay* is applied only to the weight matrix $\mathbf{W}$. The respective update formulas are:

$$\Delta\mathbf{W}^\tau = \mu\Delta\mathbf{W}^{\tau-1} + \eta\left(\left\langle \mathbf{v}[\mathbf{h_p}]^T \right\rangle^0 - \left\langle \mathbf{v'_p}[\mathbf{h'_p}]^T \right\rangle^1 - \lambda\mathbf{W}^{\tau-1}\right) \tag{3.8}$$

$$\Delta\mathbf{a}^\tau = \mu\Delta\mathbf{a}^{\tau-1} + \eta\left(\langle \mathbf{v} \rangle^0 - \left\langle \mathbf{v'_p} \right\rangle^1\right) , \tag{3.9}$$

$$\Delta\mathbf{b}^\tau = \mu\Delta\mathbf{b}^{\tau-1} + \eta\left(\langle \mathbf{h_p} \rangle^0 - \left\langle \mathbf{h'_p} \right\rangle^1\right) , \tag{3.10}$$

$$\{\mathbf{W}, \mathbf{a}, \mathbf{b}\}^{\tau+1} = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}^\tau + \{\Delta\mathbf{W}, \Delta\mathbf{a}, \Delta\mathbf{b}\}^\tau , \tag{3.11}$$

where μ is momentum constant (default 0.9), η learning rate (default 0.4) and λ weight decay constant (default 0.0002).

Also note that all the statistics are computed from the 'soft' probabilities rather than the 'hard' binary values. This reduces the discretization noise and supports faster learning. However, sampling of $\mathbf{h_p}$ into binary values is essential, it introduces information bottleneck, and the vector $\mathbf{h_S}$ can encode at most n bits.

Monitoring progress, stopping, initialization While training, we monitor the squared distance between the input features and the reconstruction. Even though this is not a correct objective function, we can use it as a sanity check. We should see a significant decrease of the distance at the beginning of the training and a long and slowly decreasing plateau in

the later stage of training. As a rule of a thumb, we stop the training after processing 100 hours of data, while for smaller data-sets, we swipe through the data-set several times.

The matrix $\mathbf{W}$ is initialized from $\mathcal{N}(0, 0.01)$. The initial hidden bias $\mathbf{b}$ is zero vector and the initial visible bias $\mathbf{a}$ is set to logit of expected output from the previous layer.

Gaussian RBM The first RBM is special, up to now we have considered RBMs with binary input random variables on the input, while the typical ASR features are Gaussian random variables. To reflect this, the visible nodes in the first RBM have to represent Gaussian random variables, and the reconstruction formula (3.4) produces a distribution:

$$p(v_i|\mathbf{h}) = \mathcal{N}([\mathbf{W}^T\mathbf{h} + \mathbf{a}]_i, 1) \ . \tag{3.12}$$

Again, we do not sample it, but we use its expected value $\mathbf{W}^T\mathbf{h} + \mathbf{a}$ as the reconstruction vector. We also have to make sure that the visible vectors contain features normalized to unit variance. The training with Gaussian input is less stable, so learning-rate is lowered to 0.01 and we perform two times more swipes through the data. We also implemented a stabilization method in which we compare the standard deviation of the features $\mathbf{v}$ and its reconstruction $\mathbf{v}'$ within a mini-batch. When the reconstructed data have more than 2x higher standard deviation than original features, it is a sign of oncoming weight-explosion. In order to prevent it, we scale down the model parameters $\mathbf{W}, \mathbf{a}, \mathbf{b}$ to achieve comparable standard deviations. We also reset the update buffers $\{\Delta\mathbf{W}, \Delta\mathbf{a}, \Delta\mathbf{b}\}^\tau$ from eq. (3.11) to zero and the learning rate is reduced temporarily. This ad-hoc stabilization method was never published, so we cannot reference it.

Bug which became a 'feature' As a last RBM related topic, we would like to describe a trick we found by serendipity. Originally, we intended to gradually increase the momentum μ in (3.8), (3.9), (3.10) from 0.5 to 0.9, while simultaneously reducing the learning rate as $\eta' = \eta\,(1-\mu)$. However, by accident, the momentum μ was fixed from the beginning to the maximum $\mu = 0.9$, and the learning rate η' was gradually reducing as we originally intended from 0.5 to 0.1. Hence, the effective learning rate $\eta_{eff.} = \frac{\eta'}{(1-\mu)}$ was decreasing, rather than being constant. In the subsequent experiment we realized that this bug was important for achieving the WER improvements from the RBM pre-training, so we kept it in the implementation.

3.1.2 Frame classification mini-batch training

After the pre-training of the DBN, we append to it a randomly initialized output layer and continue with the frame classification training with *multi-class cross-entropy* (CE) loss function from eq. (2.25) (actually, the cross-entropy is replaced with KL-divergence, which has the same derivative and even loss value in case of 1-of-K targets).

Although we have already described the mathematical core of the mini-batch frame classification training in section 2.4, some practical parts were not yet covered and will be presented in this section. As mentioned earlier, the idea behind mini-batch stochastic gradient descent is to reduce the value of a loss function by updating the model parameters with small noisy steps, which are taken in the direction in which the loss decreases the most according its first order derivative (i.e. the opposite gradient). The gradient is each time computed from a small group of randomly selected data-points (i.e. the mini-batch). The individual updates are noisy, while we assume that the overall trend of the updates will 'steer' the model in a good direction.

This supervised learning trains the model to classify the speech-frames (data-points) into the correct classes (usually triphone states). For input vector $\mathbf{x}$, it provides its posterior probability $p(s|\mathbf{x})$. Each data-point is considered as an independent classification trial with an equal weight, regardless of the prior frequency of the classes.

Supervision

For a given sentence, the reference state-sequence is obtained by forced-alignment of some existing model to the reference transcription. The existing model can be a GMM-HMM or a DNN-HMM. In the forced-alignment, the decoding graph is compiled from the reference transcription, and a decoder is used to determine the state-sequence that fits best the acoustic observation given the model.

By training from the DNN alignment instead of the GMM alignment, we usually obtain a small performance gain, also the frame accuracy is usually better. However, the improvement nearly vanishes after the sequence-discriminative training and the recipe with GMM alignment is simpler.

In the case of untranscribed data, the alignment gets replaced by the best-path hypothesis from the decoder.

Avoiding over-fitting

Over-fitting is a very common problem in the frame-classification discriminative training. The model with millions of parameters has a capacity to memorize the training data instead of learning the smooth boundary that will generalize well on unseen data. We should be alert if we see a big performance gap between the training data and held-out data, which usually happens for small training databases.

- **Early stopping,** to avoid over-fitting we use the early stopping algorithm: we monitor the loss function value obtained on the held-out set (usually 10% of the training data), and we accept only the models where the loss decreases. For historical reasons, we sometimes incorrectly call the held-out set as a 'cross validation' set, but we are not doing the K-fold cross-validation as one might expect.

- **Learning rate annealing,** simultaneously, we apply a learning rate scheduling algorithm inspired by *simulated annealing*, where the learning rate represents the temperature. The learning rate is on its initial value, as long as the relative loss improvement from an epoch is larger than 1%. Then, for each next epoch, the learning rate is halved, which helps to converge the training as the 'temperature' decreases. The training ends if the relative improvement of loss falls below 0.1%. This scheme is a generalization of the 'New-bob' [2] learning rate scheduling, in which the decisions were made from absolute improvements of the frame accuracies.

  The concept of 'temperature' applied to the Stochastic Gradient Descent represents the level of jittering of the noisy exploration in the parameter space, which arises from optimizing the loss locally from the mini-batches. By lowering the learning rate, the velocity gets smaller, and the loss value for the held-out set decreases dramatically, because the model becomes less specialized to the recently seen data. With a very small level of jittering, the model parameters converge to a local minimum.

[2] Originally mentioned in: http://www1.icsi.berkeley.edu/Speech/faq/nn-train.html

- **Regularization,** in the past, we experimented with L1 and L2 regularization, these were originally intended to be used for the batch methods. When applied to mini-batch SGD, the regularizer term is added to the loss of each mini-batch. In an unpublished experiment with the small database TIMIT where over-fitting evidently happens, our observation was that although the loss of held-out data decreased, this improvement did not translate into the Phone error rate (PER) reduction, which discouraged us from its further use.

  Another regularization method is Dropout [Srivastava et al., 2014], in which the outputs of neurons are randomly discarded. This introduces noise that prevents the early co-adaptation of the neurons. This is particularly helpful with Rectified linear units [Zeiler et al., 2013], which, in our experience, overfit more strongly than sigmoid networks. The dropout rate can be reduced abruptly or gradually with an annealing algorithm [Rennie et al., 2014].

When we can't use mini-batch,

Not all the network types are 'compatible' with the random selection of individual datapoints to form a mini-batch. For example, the Sequence summary NN [Veselý et al., 2016] can be elegantly trained with per-sentence updates, while for the recurrent neural networks, including LSTMs and BLSTMs, we need to prepare the data in multiple streams so that the adjacent rows in the feature matrix come from different sentences. For these cases, we have specialized 'multi-stream' training binaries in the Kaldi 'nnet1'.

Special topologies,

the 'nnet1' models are not limited to a simple sequence of components. For example, the `<ParallelComponent>` supports parallel processing of several nested neural networks. Or the `<SimpleSentenceAveragingComponent>` allows to summarize a sentence by averaging the outputs from the preceding component [Veselý et al., 2016]. Somewhat more complicated components are those which implement the recurrent LSTM and BLSTM layers [Graves et al., 2013]. The detailed discussion of these is however outside of scope of this thesis.

Monitoring the progress of training

During the training, we are typically watching the progress of the loss function value, measured both on training set and held-out set. The loss value on the training set is usually lower than on the held-out set. The learning rate annealing (i.e. halving) usually starts around the 4th epoch, and the loss value on the held-out always decreases. With reduced learning-rate we may experience a mild increase of loss value for the training set, as we stop over-fitting on consecutive mini-batches constructed from same buffer of utterances. The loss value of training set is computed with model that changes parameters on-the-go.

3.1.3 Sequence-discriminative training, sMBR

This section is based on [Veselý et al., 2013a], where we studied the sequence-discriminative DNN training with various objective functions.

Neural networks (NNs) for speech recognition are typically trained to classify individual frames based on a cross-entropy criterion, equation (2.24). Speech recognition, however, is

inherently a sequence classification problem. As such, speech recognizers using the Gaussian mixture model (GMM) as the emission density of an HMM achieve the state-of-the-art performance when trained using the sequence-discriminative criteria like maximum mutual information (MMI) [Bahl et al., 1986], boosted MMI (BMMI) [Povey et al., 2008], minimum phone error (MPE) [Povey, 2003] or minimum Bayes risk (MBR) [Kaiser et al., 2000, Gibson and Hain, 2006, Povey and Kingsbury, 2007]. It is possible to efficiently estimate the parameters based on any of these criteria using the statistics collected from lattices [Povey, 2003].

The theory for sequence-discriminative training of neural networks was also developed in the early literature [Bridle and Dodd, 1991, Krogh and Riis, 1999]. In fact, the 'clamped' and 'free' posteriors described in [Bridle and Dodd, 1991] are the same as the numerator and denominator occupancies used in discriminative training of GMM-HMM systems [Povey, 2003]. The idea to use this lattice-based framework for sequence-discriminative training of NNs was explored in [Kingsbury, 2009]. It was shown that the sequence-discriminative training can improve upon networks trained using the cross-entropy. Subsequent results reported in [Wang and Sim, 2011, Kingsbury et al., 2012, Jaitly et al., 2012] have also shown consistent gains from sequence-discriminative training of NNs. However, there is some disagreement about which of the criteria is suitable: [Kingsbury, 2009, Kingsbury et al., 2012] suggest using a state-level minimum Bayes risk (sMBR) criterion, while [Wang and Sim, 2011] finds MMI to work better than MPE, and [Jaitly et al., 2012] only provide results using MMI.

Needless to say, such empirical observations depend on the choice of the dataset and specific details of the implementation. In our work, we presented a comparison of the different training criteria for DNNs on the standard 300-hour Switchboard conversational telephone speech task, which has also been used in [Seide et al., 2011a, Kingsbury et al., 2012].

The networks are trained to optimize a given training objective function using the standard *error backpropagation* procedure [Rumelhart et al., 1986], and the optimization is done through stochastic gradient descent (SGD). For any given objective, the important quantity to calculate is its gradient with respect to the activations at the output layer, replacing the formula (2.32) on page 25. The gradients for all the parameters of the network can be derived from this one quantity based on the back-propagation procedure described in section 2.4.

Maximum mutual information, MMI

The MMI criterion used in ASR [Bahl et al., 1986] is the mutual information between the distributions of the observation and word *sequences*. With $\mathbf{O}_u = \{\mathbf{o}_{u1}, \ldots, \mathbf{o}_{uT_u}\}$ as the sequence of all observations, and W_u as the reference word-sequence for utterance u, the MMI criterion is:

$$\mathcal{F}_{MMI} = \sum_u \log \frac{p(\mathbf{O}_u|S_u)^\kappa P(W_u)}{\sum_W p(\mathbf{O}_u|S)^\kappa P(W)}, \tag{3.13}$$

where $S_u = \{s_{u1}, \ldots, s_{uT_u}\}$ is the sequence of states corresponding to W_u; and κ is the acoustic scaling factor. The sum in the denominator should be evaluated over all possible word-sequences W, but practically, it is computed from all paths through a *denominator lattice* generated for utterance u. Differentiating (3.13) w.r.t. the log-likelihood $\log p(\mathbf{o}_{ut}|r)$

for state r, we get:

$$\begin{aligned}\frac{\partial \mathcal{F}_{MMI}}{\partial \log p(\mathbf{o}_{ut}|r)} &= \kappa\delta_{r;s_{ut}} - \frac{\kappa \sum_{W:s_t=r} p(\mathbf{O}_u|S)^\kappa P(W)}{\sum_W p(\mathbf{O}_u|S)^\kappa P(W)}, \\ &= \kappa(\delta_{r;s_{ut}} - \gamma_{ut}^{DEN}(r)),\end{aligned} \tag{3.14}$$

where $\delta_{r;s_{ut}}$ is the Kronecker delta function, which equals 1 for state r at reference state sequence s_{ut}, and $\gamma_{ut}^{DEN}(r)$ is the posterior probability of being in state r at time t, computed over the denominator lattices for utterance u. The required gradient w.r.t. the activations is obtained as:

$$\begin{aligned}\frac{\partial \mathcal{F}_{MMI}}{\partial a_{ut}(s)} &= \sum_r \frac{\partial \mathcal{F}_{MMI}}{\partial \log p(\mathbf{o}_{ut}|r)} \frac{\partial \log p(\mathbf{o}_{ut}|r)}{\partial a_{ut}(s)}, \\ &= \kappa(\delta_{s;s_{ut}} - \gamma_{ut}^{DEN}(s)).\end{aligned} \tag{3.15}$$

Note that, in this work, we have assumed that the reference state labels are obtained through a forced alignment of the acoustics with the word transcript. More generally, one may use forward-backward over the word reference to obtain the numerator occupancies $\gamma_{ut}^{NUM}(s)$ instead of using $\delta_{s;s_{ut}}$ in equation (3.15).

Minimum phone error, MPE / State minimum Bayes risk, sMBR

While minimizing $\mathcal{F}_{CE}$ (2.24) minimizes expected frame-error, maximizing $\mathcal{F}_{MMI}$ minimizes expected sentence error. The MBR family of objectives are explicitly designed to minimize the expected error corresponding to different granularity of labels [Gibson and Hain, 2006]:

$$\mathcal{F}_{MBR} = \sum_u \frac{\sum_W p(\mathbf{O}_u|S)^\kappa P(W) A(W, W_u)}{\sum_{W'} p(\mathbf{O}_u|S)^\kappa P(W')}, \tag{3.16}$$

where $A(W, W_u)$ is the raw accuracy, representing the number of correct phone labels (for MPE) or state labels (for sMBR). The raw accuracy is counted for a path from some word sequence W that is compared with a path from the reference transcripts W_u. By differentiating (3.16) w.r.t. $\log p(\mathbf{o}_{ut}|r)$, we get:

$$\begin{aligned}\frac{\partial \mathcal{F}_{MBR}}{\partial \log p(\mathbf{o}_{ut}|r)} &= \kappa\, \gamma_{ut}^{DEN}(r) \left\{ \bar{A}_u(s_t = r) - \bar{A}_u \right\}, \\ &= \kappa\, \gamma_{ut}^{MBR}(r),\end{aligned}$$

where $\bar{A}_u(s_t = r)$ is the average accuracy of all paths in the lattice for utterance u that pass through state r at time t; $\bar{A}_u$ is the average accuracy of all paths in the lattice; and $\gamma_{ut}^{MBR}(r)$ is the MBR 'posterior' as defined for approximate MPE in [Povey, 2003]. Like before for $\mathcal{F}_{MMI}$, we get:

$$\frac{\partial \mathcal{F}_{MBR}}{\partial a_{ut}(s)} = \kappa \gamma_{ut}^{MBR}(s). \tag{3.17}$$

Experiments

We report experiments on the 300 hour Switchboard conversational telephone speech task. Specifically, we use Switchboard-1 Release 2 (LDC97S62) as the training set, together with

the Mississippi State transcripts[3] and the 30K-word lexicon released with those transcripts. The lexicon contains pronunciations for all words and word fragments in the training data. We use the Hub5 '00 (LDC2002S09) data as the development set and Hub5 '01 (LDC2002S13) data as a separate test set. It is worth pointing out that the Hub5 '00 data contain 20 conversations from Switchboard (SWBD) and 20 conversations from CallHome English (CHE). The CallHome data tends to be harder to recognize, partly due to a greater proportion of foreign-accented speech and more informal speaking style.

The acoustic models were trained on features similar to those described in figure 3.1. The PLPs were replaced by MFCCs, there were no pitch features and the first splicing was done with +/- 3 frames around the central frame. The details are described in [Veselý et al., 2013a].

The baseline GMM-HMM systems are trained on the LDA+STC+FMLLR, the models trained on the full 300 hour training set contain 8859 tied triphone states and 200K Gaussians. The leaves of the phonetic decision tree used for the GMM-HMM system correspond to the output units of the respective DNNs.

The DNNs are trained on the same LDA+STC+FMLLR features as the GMM-HMM baseline, except that the features are globally normalized to have zero mean and unit variance. The FMLLR transforms are the same as those estimated for the GMM-HMM system during training and testing. The network trained on the full 300 hour training set has 7 layers (that is, 6 hidden layers), where each hidden layer has 2048 neurons, the DNN has 8859 output units. The input to the network is an 11-frame (5 frames on each side of the current frame) context window of the 40 dimensional features. The DNN is initialized with stacked restricted Boltzmann machines (RBMs) that are pretrained in a greedy layerwise fashion [Hinton et al., 2006], following the recipe from the previous section 3.1.1.

Sequence-discriminative training of DNNs Just like with GMM-HMM systems, sequence-discriminative training of DNNs starts from a set of alignments and lattices that are generated by decoding the training data with a unigram LM. For each training condition, the alignments and lattices are generated using the corresponding DNN trained using frame-by-frame cross-entropy training. The cross-entropy trained models are also used as the starting point for the sequence-discriminative training.

Through initial benchmarking experiments with MMI as the objective function, we found 1e-5 to be a suitable learning rate[4] and that an exponentially decaying learning rate provided no gains.

Figure 3.4 shows the results with MMI trained DNNs on Hub5 '00. The horizontal line (`CE realign`) shows the results with CE training when starting with alignments from a CE trained DNN instead of the alignments from a GMM system. This accounts for about half of the improvements from MMI. We find the MMI objective to overfit after 2 iterations (the WER increase on the red curve). A detailed analysis revealed anomalous objective and gradient values for utterances where the reference hypothesis is missing from the lattice. This may be caused by search errors or by a poor match of the acoustics to the model or even by errors in the reference transcription. However, only in the first of these cases (i.e. when there are search errors on the training data), it is reasonable to explicitly add the reference to the lattice. As a result, we decided to remove such frames from the gradient

[3] Available from: http://www.isip.piconepress.com/

[4] In our implementation, the gradients are not scaled by the acoustic scale κ, but its effect is subsumed in the learning rate. So, with $\kappa = 0.1$, the effective learning rate is 1e-4.

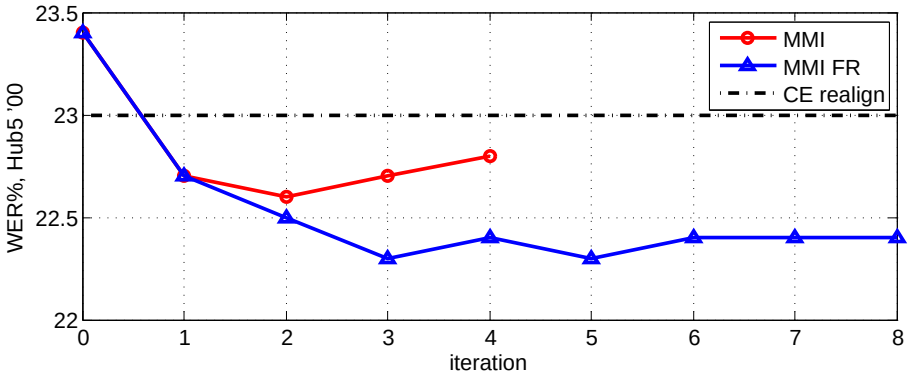


Figure 3.4: *Hub5 '00: DNNs trained with MMI on 110h set, with and without frame rejection (FR).*

computation, which reduces the amount of training data by 2.5%. The results in figure 3.4 show that this frame rejection (FR) heuristic leads to more stable learning. Nearly all of the reduction in errors is on the CallHome part, which is more mismatched to the training data.

Next, comparing the different sequence-discriminative criteria in figure 3.5, we do not find significant differences. A learning rate of 1e-5 was also found to work well for these other criteria. In figure 3.6, we compare the results when the lattices are regenerated after the first epoch. We see that regenerating lattices provides a small gain. However, this is computationally expensive and regenerating lattices after the second epoch did not produce any further gains.

Finally, table 3.2 summarizes the results of the different systems trained on the entire 300 hour training set. The results are presented on both the development set (Hub5 '00) and the test set (Hub5 '01) and their respective subsets. We see that the CE trained DNN models are better than the discriminatively trained GMM BMMI models. Then, the use of the sequence-discriminative training criteria (incl. lattice re-generation after first epoch) led to performance improvements within the range of $1.2 - 1.8\%$, and a little better results were achieved with the sMBR objective. The sMBR training was subsequently adopted as the default sequence-discriminative objective in the 'nnet1' training recipes in Kaldi.

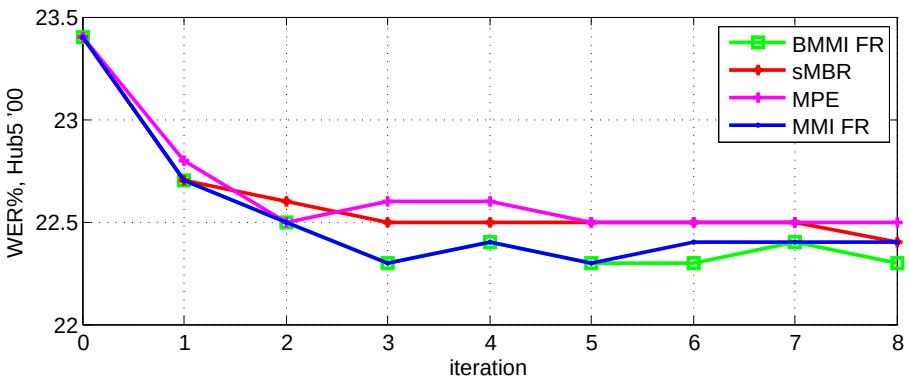


Figure 3.5: *Hub5 '00: DNNs trained on 110h set, various criteria.*

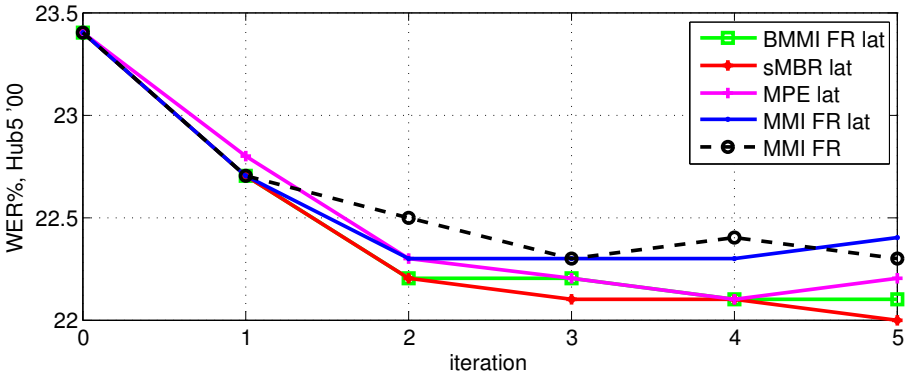


Figure 3.6: *Hub5 '00, lattice regeneration after first epoch (indicated by '`lat`' suffix).*

Table 3.2: *Results (% WER) of the DNNs trained on the full 300 hour training set using different criteria. The input features are always the same: MFCCs transformed by LDA+MLLT+fMLLR.*

	Hub5 eval'00			Hub5 eval'01			
System	SWB	CHE	Total	SWB	SWB2P3	SWB-Cell	Total
GMM	21.2	36.4	28.8	-	-	-	-
GMM BMMI	18.6	33.0	25.8	18.9	24.5	30.1	24.6
DNN CE	14.2	25.7	20.0	14.5	19.0	25.3	19.8
DNN MMI	12.9	24.6	18.8	13.3	17.8	23.7	18.4
DNN sMBR	12.6	24.1	**18.4**	13.0	17.7	22.9	**18.0**
DNN MPE	12.9	24.1	18.5	13.2	17.7	23.4	18.2
DNN BMMI	12.9	24.5	18.7	13.2	17.8	23.5	18.3

Based on its formulation, the sMBR training is more robust to the annotation errors than MMI. The sMBR gradient improves scores for lattice-paths where the approximate-accuracy is better than the average. If all the paths in the lattice are wrong, the accuracy of all the paths is the same, also the gradient is zero and the badly annotated sentence is not deleterious. The sMBR accuracy function was later modified to map the silence states to a single class, which improves the stability of the training.

3.2 Accelerating the DNN training

The acceleration of NN training was the main topic of my Master thesis [Veselý, 2010], which was later summarized in [Veselý et al., 2010]. Although already six years passed and the project TNet was abandoned, the gained experience was important for designing the 'nnet1' training tools in Kaldi.

3.2.1 NN training implementations

SNet - distributed client-server training on CPU

The first neural network training tool implemented at BUT was SNet. Inspired by the at-the-time very popular training tool QuickNet[5], the author Stanislav Kontár used a different approach to the OOP design of the neural network. The single class representing the whole network was replaced by a collection of simple 'component' classes and a container class which assembled the network from the components. This modular design allowed to construct complex on-the-fly feature expansions, which could even contain nested neural networks.

The implemented mini-batch SGD training used BLAS library for fast matrix multiplication. Further acceleration was achieved by data parallelization: The data-set was split among workers, while a parameter server was used to synchronize the model by collecting the gradients and sending the updated model parameters. The model updates could be synchronous or asynchronous.

Synchronous model updates mimicked the non-parallel training, while each mini-batch was split among all the workers. The parameters were updated only after all the workers completed their portion of the mini-batch.

Asynchronous model updates, in this case the parameters were updated immediately after any worker completed its chunk of the data. It happened that the gradients were computed from slightly outdated weights. Asynchronous training was used for example in Hogwild [Recht et al., 2011], however in our experience the asynchronous updates were causing performance degradations. This made the method unpractical despite its good scalability.

In the synchronous training, the workers were idle while synchronizing the model with the server over the relatively slow TCP-IP channel. Additional delay was caused by waiting for all the workers to complete its portion of the mini-batch. All these factors combined led to a rather poor scalability, as will be seen later on the blue curve in figure 3.7.

TNet - multi-threaded training on CPU

The next generation NN training tool was my TNet[6]. It implemented the same 'data-parallel' scheme as SNet, while the parameter synchronization overhead was reduced by using multi-threading. All threads share the same address space of memory, so no data transfers are needed. The training is synchronized by two 'barriers' which separate the gradient computation and model synchronization. The model synchronization is also multi-threaded, making each thread responsible for updating a disjoint part of the weight matrices. For better caching, we used a single instance of model parameters for all the worker threads.

The TNet is written in C++. We designed an abstract interface `Component` with the descendant classes implementing forward-propagation function and the transformation by its Jacobian necessary for back-propagation. For components with trainable parameters, we have a special interface `UpdatableComponent`, which also requires to specify the update function. These abstract classes are present also in Kaldi 'nnet1'. Again, we used the BLAS library, specifically the GotoBLAS.

[5] http://www1.icsi.berkeley.edu/Speech/qn.html
[6] http://speech.fit.vutbr.cz/software/neural-network-trainer-tnet

TNet - GPU training, CUDA

An acceleration technique complementary to the 'data parallelism' is the use of GPU. Our code uses a single GPU, which typically has hundreds or even thousands of programmable computational units. Also, the GPU memory is more than 10x faster than the RAM of the 'host' computer. The fast memory and high parallelism accelerate the training considerably, while each new generation of GPUs boosts the performance.

TNet uses separate code for CPU and GPU tools as well as for the underlying libraries representing the matrices and neural networks; some code is shared (I/O functions, logging, etc).

During the training, most of the time is spent on matrix multiplications, which is implemented efficiently in the CUBLAS library (part of the CUDA toolkit distributed with the 'nvcc' compiler). For other operations, we implemented CUDA kernels (activation functions, loss function, data shuffling, derivatives). Each kernel is defined by its thread function and the grid and block dimensions. These produce the logical coordinates for which the threads are executed. The kernels are called one after another from the 'host' process, and the CUDA library is responsible for mapping the kernels to the multi-processors of the GPU. Typically, we compile the assembly for several GPU architectures and we have a good experience with the high-end gaming cards, which are almost as fast as the much more expensive Teslas in the 32bit float computations.

A separate GPU code is also used in the C++ library that implements matrices, vectors and related operations, this is compiled with standard `gcc`. Only the kernels are compiled with the CUDA compiler `nvcc`, while the interfacing with the C++ code is done via ANSI C wrapper functions.

Kaldi - nnet1, GPU training, CPU forwarding

Much of the code design from TNet translated into Kaldi 'nnet1'. Again, there is a separate 'cudamatrix' library for CUDA vectors, matrices and related operations. However, in Kaldi, a single binary can compute both on GPU or CPU, as specified on the command line. To allow this, the `CuVector` or `CuMatrix` contains a 'shadow' `Vector` or `Matrix` for the CPU computation, and each method contains an 'if' testing if the GPU is active. Kaldi can be also compiled without the CUDA code, which is supported by conditional compilation with `#ifdef` macros.

In kaldi 'nnet1', the tools are more flexible than were in TNet. There are several training tools, and it is possible to look into the network and print statistics of activations, derivatives and gradients, which are useful for debugging. There are also other tools for manipulating the networks (initializing, converting, cutting, concatenating, pre-training). In the training scripts, the network is first defined with a human-readable prototype, to be later initialized with a binary tool. For more information, please see the documentation: http://kaldi-asr.org/doc/dnn1.html

3.2.2 Speed measurements

The speed measurements with SNet, the multi-threaded TNet and GPU TNet were done in 2012. The performance of the GPU TNet was evaluated with the following HW setup: Desktop PC with 1x Intel Core2Duo E8400 3.0GHz, 2GB RAM and the NVidia GTX 285 GPU with 240 shaders at 1.476GHz. The performances of SNet and CPU TNet were evaluated on a Blade server HS22 with 2x Intel Xeon X5675 3.07GHz 12MB cache, 24 GB RAM.

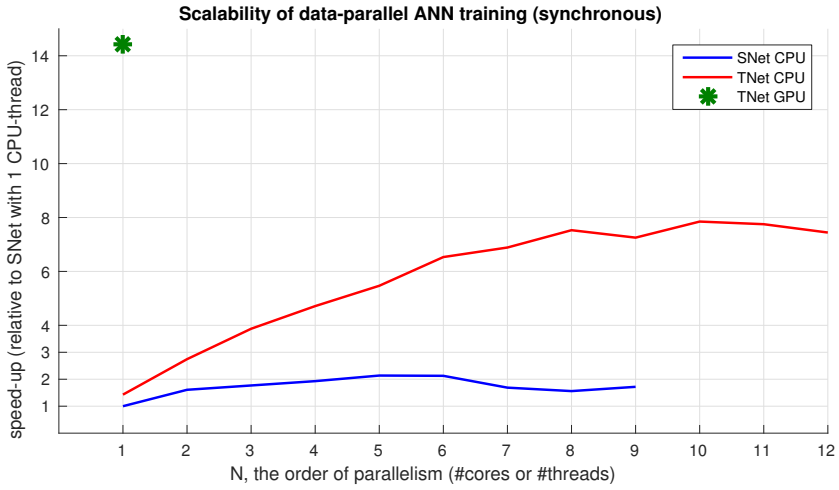


Figure 3.7: *Comparison of parallelization speedups on the example task.*

The example 3-layer MLP with 730k parameters was trained on 4000 sentences from the AMI corpus. We used an on-the-fly feature transformation, which consisted of temporal-splicing followed by a Hamming window and Discrete Cosine Transform applied to the temporal trajectory of each feature. The mini-batch size was fixed to 512 frames, and we measured the time taken by training with 4000 sentences. As a reference time, we decided to use the single-core CPU training with SNet. In the data parallel CPU training, we split the mini-batch among the computational units, each worker processes $512/N$ data-points at once, where N is the order of parallelism.
In figure 3.7 we see the 'arch-shaped' speed-up curves for both the CPU tools: SNet (blue) and TNet (red). The non-matching values for 1 worker are caused by using the ATLAS library for SNet and the faster GotoBLAS for TNet. With SNet, the peak performance was achieved with 5 nodes, while for TNet with 10 nodes. With too many workers, the performance decreases due to the increased overhead. By comparing the peaks, we see that the parallel TNet is **3.6x** faster then parallel SNet. The scalability of TNet is better because we removed the communication overhead that was present in SNet. However, an even larger speed-up can be achieved by training on a GPU (green star in figure 3.7), which is **1.8x** faster than CPU TNet at its peak performance. The GPU training is **14x** faster then the run-time of the single-core SNet.

In 2016, we compared again the training speeds of 1 CPU-core and 1 GPU with the current hardware. This time, we used Kaldi 'nnet1' training of a DNN with 8 million parameters on 10k sentences from AMI corpus. The GPU model was GTX980 and the CPU was Intel Xeon E5-2670. We used the OpenBLAS library for CPU training.

From table 3.3, we see that the speedup from using a GPU instead of 1 CPU-core is much higher than what we measured in 2012 (60x vs. 14x). This practically shows that the

Table 3.3: *Comparing the speeds of NN training with 1 CPU-core and 1 GPU. The reported time is the average duration of 13 epochs with 10k sentences.*

CPU-core	GPU	Ratio
264 min	4.4 min	60x

'computation capabilities' of GPUs grew faster than those of CPUs. It is true, that we did the comparison with slightly different conditions (different NN topology, front-end, training toolkit), on the other hand the typical neural networks we train now are larger than those in the past.

3.2.3 Other implementations in the literature

From the literature, we would like to mention the study of Asynchronous SGD method called 'Hogwild!' [Recht et al., 2011], which was later scaled up by Google as training tool 'Downpour SGD' [Dean et al., 2012]. From our experience, the Asynchronous SGD causes performance hit, while this is compensated by being able to process larger amounts of training data.

Another interesting option is to replace SGD by a Hessian Free algorithm (second order method), and use a supercomputer with very fast Infiniband communication [Sainath et al., 2014]. For universities, it is usually possible to get the access to such machine through a public bidding for the computation hours on super-computers.

Or we can use the pre-conditioning of NN parameters and synchronize the models by occasional weight averaging, which was proposed in the 'Natural Gradient' method [Povey et al., 2014]. In the same time the pre-conditioning requires additional computation, so each node processes the same amount of training data 3x slower compared to using a single GPU. On the other hand this distributed training scales linearly, which makes it very attractive for training with 10 and more GPUs. The cost of synchronizing model parameters can be reduced by using MPI [Su and Chen, 2015], which allows more frequent model synchronizations.

Another family of implementations is considering back-propagation algorithm as instance of reverse-mode automatic differentiation, for example Theano[7]. The benefit is that the user specifies the forward-propagation function, while the back-propagation is derived automatically. Usually the automatic solution creates a computational graph, which needs to be further optimized with non-trivial heuristics. This approach is also used in Torch[8], TensorFlow[9] or Kaldi 'nnet3'[10].

3.3 Summary

At this point we end the description of 'nnet1' recipe that we implemented in Kaldi. The 'nnet1' recipe will serve as experimental environment in the later chapters of this thesis. Of course, it would not be hard to find numerous ways how to extend the 'nnet1' recipe, for example by multi-GPU training. However, the time of any human being is limited, so we reserve it as a future direction of research for the time after the graduation.

Not all of my publications were included in this section. If you are interested in more of my work, please follow the link to the full publication list: http://www.fit.vutbr.cz/~iveselyk/pubs.php The semi-supervised training will be discussed in next chapters.

[7] http://deeplearning.net/software/theano/
[8] http://torch.ch/
[9] https://www.tensorflow.org/
[10] http://kaldi-asr.org/doc/dnn3.html

Chapter 4

Data-sets

In this chapter we provide a brief description of the databases we will later use for the experiments with the semi-supervised DNN training: Babel Vietnamese, some other Babel languages (Assamese, Bengali, Haiti, Lao, Zulu) and Switchboard English. We also mention some details about the experimental setups: language models, lexicons, phone-sets, OOV rates.

4.1 Babel Vietnamese

Most of the experiments with semi-supervised training were done with the Vietnamese dataset[1] as provided within the IARPA Babel program, release babel107b-v0.7. The training data consist of a large portion of conversational telephone speech and a small part of prompted speech. For training, we used both types of data. The development set consists of conversational speech only. The data come from various telephone channels: landlines, different kinds of cellphones, or phones embedded in vehicles. The sampling rate is 8000 Hz.

Two scenarios are defined – Full Language Pack (FullLP), in which all the collected data is transcribed; and Limited Language Pack (LimitedLP), in which only a subset of the data is transcribed, while the remaining part of the FullLP data can be used as 'untranscribed' data for the semi-supervised training.

The overview of the data (i.e. numbers of speakers and amounts of speech data after re-segmenting) is in table 4.1. We generated our segmentation, using our own MLP-based Voice activity detection (VAD) with Viterbi smoothing [Ng et al., 2012]. The speech segments were extended by 300 milliseconds on both ends.

The provided Vietnamese lexicon uses 54 phonemes. There are 25 consonants and 29 vowels, while for Vietnamese, we distinguish 6 tones.

The corpus is composed of 4 dialects, the pronunciation of some graphemes is different

[1] Collected by Appen Butler Hill: http://www.appenbutlerhill.com

Table 4.1: *Data analysis, numbers of speakers, amounts of annotated speech data after resegmentation by VAD*

Dataset	FullLP	LimitedLP	dev
speakers	991	121	120
size in hours (reseg.)	84.8	10.8	9.8

between dialects, a single grapheme can have 2-3 different vocalizations. Also, some of the phonemes can be translated into graphemes in several different ways.

For the purpose of ASR training, the phone set consists of 29 phonemes, which are marked with six different tones. The under-represented phones were merged manually. For the triphone-tree clustering, we introduced a 'position in a word' feature, which leads to the final phone-set with 350 items. We allow state sharing across phonemes.

The original syllabic lexicon provided by Appen was modified by reducing the number of pronunciation variants. The FullLP lexicon contains 6k syllables and the LimitedLP lexicon contains 3k syllables.

The ASR outputs are syllables, which is natural for Vietnamese and which conveniently avoids eventual errors from inconsistent word-segmentation. Also, there are no phonological processes that cross syllable boundaries, such as consonantal assimilation, tone sandhi or wordlevel stress. The consequence is that the OOV rate is very small, 0.21% for the FullLP condition and 1.19% for the LimitedLP condition (LimitedLP is used in semi-supervised training).

We used a trigram language model with Kneser-Ney smoothing built on the syllabic training transcripts, with 100k 3-grams and 200k 2-grams for FullLP, and with 12k 3-grams and 47k 2-grams for LimitedLP.

4.2 Other Babel languages

In section 6.2, we will report experiments on other Babel languages: Assamese, Bengali, Haiti, Lao and Zulu[2], which were provided in the second year of IARPA Babel program. The training data consist of conversational telephone speech with a small amount of prompted speech, both types of data were used for training. The development set consists of conversational speech only. The data come from various telephone channels: landlines, different kinds of cellphones, or phones embedded in vehicles. The data were pre-processed to have 8kHz sampling rate.

We focus on the Limited Language Pack (LimitedLP) condition, which was the primary condition in the second year of the Babel program. In the LimitedLP condition we have 10 hours of transcribed speech and 70 hours of 'untranscribed' speech, which are in fact transcribed for another condition. The dev set for each language contains around 7 hours of speech.

The Appen phone sets were modified by merging some under-represented phones. For the triphone-tree clustering, we introduced a 'position in a word' feature, which increases

[2]Collected by: http://www.appenbutlerhill.com

Table 4.2: *Per-language system characteristics*

	Assamese	Bengali	Haiti	Lao	Zulu
# phones	171	181	143	434	264
train lexicon size	10k	11k	6k	6k	21k
dev OOV rate	8%	9%	4%	2%	22%
# 1grams	9k	10k	5k	4k	15k
# 2grams	46k	51k	40k	38k	45k
# 3grams	4k	5k	9k	9k	3k

the size of phone set 4x. Some languages have special features: Lao (6 tones) and Zulu (word stress). In table 4.2 we see the phone-set sizes for all the languages. We allow tied-states to be shared across phonemes in the decision tree clustering.

The Appen lexicon was extended by generating pronunciations of missing words from the training data using Sequitur G2P. The lexicon sizes are in table 4.2 together with the out-of-vocabulary (OOV) rate measured on the dev data. We used trigram language model with Kneser-Ney smoothing built on the training transcripts, the model sizes are also in table 4.2.

4.3 Switchboard

The Switchboard database consists of Conversational Telephone Speech. The training set is Switchboard-1 Release 2 (LDC97S62), a collection of about 2,400 two-sided telephone conversations among 543 speakers (302 male, 241 female) from all areas of the United States. We used the Mississippi State transcripts and lexicon. The language model is built by interpolating two 3-gram language models trained on Switchboard and Fisher transcripts respectively.

Semi-supervised experiments For the semi-supervised experiments, the LM was built purely on the Fisher transcripts. Note that we generate automatic transcripts for the training data, so the true transcripts of the Switchboard data have to be removed from the LM corpus.

For the semi-supervised experiments, we split the Kaldi `'train_100k_nodup'` 100hour set, from which we randomly selected 186 conversation sides as the transcribed set (14 hours), while the remaining 1165 conversation sides (96 hours) are the untranscribed data.

Evaluation set: Hub5-2000 (eval2000) The evaluation set consists of: a) 20 conversations from the CallHome corpus, b) 20 conversations that were collected for the Switchboard Corpus but not included in the original release. Most of the speakers in these conversations, appeared in the released Switchboard Corpus for the training.

In this thesis we report the performance on both subsets together, while the conference articles from other laboratories usually report results for the b) subset. For more information see: http://www.itl.nist.gov/iad/mig/tests/ctr/2000/h5_2000_v1.3.html

Chapter 5

Semi-supervised training

This chapter is a gentle introduction to the semi-supervised training. It explains the basic pattern of improving the system with unlabeled data. We give an overview of the main design questions that we address in the following chapters. We also show the principle of frame-weighted mini-batch SGD training. The chapter is closed with a survey of the relevant literature.

The practical value of semi-supervised training is that it allows us to build better systems with an inexpensive untranscribed data, while we need to find a way how to use such data efficiently.

5.1 Definition

The *semi-supervised learning* is a type of supervised learning, where both the labeled and the unlabeled data are used. The goal of semi-supervised learning is to improve the system performance by adding the unlabeled data into the training process (compared to the case when only the labeled data are used).

We are using the heuristic approach called *self-learning*. In this case, a 'seed system' is built on the labeled data. The 'seed system' is then used to guess the labels for the unlabeled data. Next, a new system is built using the augmented dataset, while we typically add only the data where we are confident about the guessed labels.

Applied to ASR, we focus on semi-supervised training of Deep Neural Network acoustic models, which was not yet studied extensively. The process of semi-supervised system building is shown in figure 5.1, and is described as follows: We use the transcribed data to train the seed system. Then we generate the automatic transcripts and their confidences for the untranscribed data by decoding it with the seed system. The data with more reliable automatic transcripts are selected for the system re-training, where the confidences can be calculated in many ways. Lastly, the process of decoding and re-training can be iterated until no further improvements are obtained. This was done for GMM-HMMs in [Wessel and Ney, 2005].

Relation to active learning

Another technique related to semi-supervised training is *Active Learning* [Cohn et al., 1994, Yu et al., 2010c]. For both techniques, we have the labeled and unlabeled data-sets, while in the case of active learning, we use the initial system to select the problematic data for

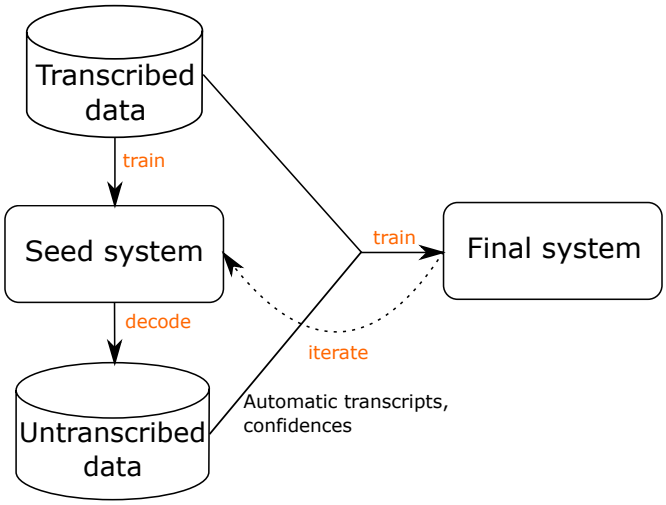


Figure 5.1: *General paradigm of Semi-supervised training for ASR.*

manual annotation that is done by an expert. Active learning is particularly interesting for companies running speech-based services, which serve them as a source of new in-domain training data. We do not consider using active learning in this thesis.

Word-error-rate recovery

As the measure of success for the semi-supervised training, we use the *'Word error rate recovery'* R_{wer}, defined in [Novotney and Schwartz, 2009] as the ratio:

$$R_{wer} = \frac{\Delta\text{WER}_{sst}}{\Delta\text{WER}_{oracle}} , \tag{5.1}$$

in which ΔWER_{sst} is the WER improvement from the semi-supervised training while $\Delta\text{WER}_{oracle}$ is the WER improvement obtained when the correct transcripts are used. The idea of R_{wer} is to quantify the achieved proportion from the possible improvement, which allows us to compare the efficiency of the semi-supervised training starting from different initial systems.

However, the ultimate measure for ASR performance is the *Word error rate*, defined as:

$$\text{WER} = \frac{\#Sub + \#Del + \#Ins}{\#Ref} \cdot 100 \tag{5.2}$$

where $\#Sub$ is number of substitution errors, $\#Del$ number of deletions, $\#Ins$ number of insertions and $\#Ref$ is the number of words in the reference transcription. The WER is based on Levenshtein distance of reference word sequence and the hypothesis (i.e. the recognized text).

5.2 The key questions of semi-supervised DNN training

When thinking about the semi-supervised training for the DNN models, we first aim to identify the questions, which help us establish the search space for finding a good semi-supervised recipe.

5.2.1 Granularity of confidence units

The first question is: „What should be the size of the unit for which we calculate the confidence?“ It can be a *sentence*, a *word* or a *feature-frame* (i.e. the smallest unit).

Per-word confidence $c_{\bar{w}_q}$

There are many methods how to extract the per-word confidences. For example, C_{max} [Wessel et al., 2001] is based on summing the posteriors of lattice-links that both correspond to the same word and overlap in time. Within the word-link, we take the value from such time-slice for which the sum of posteriors is the highest. However, this seems to be less necessary because we use 'exact lattice' generation [Povey et al., 2012]. The lattice is 'deterministic': each distinct word string is present in lattice only once and with its best score.

The method that we are using in our experiments is the calculation of statistics $\gamma(q, w)$ taken from the Minimum Bayes Risk (MBR) decoding [Xu et al., 2011, section 7.1]. The quantity $\gamma(q, w)$ is the posterior probability of the word symbol w being aligned with the position q in a word sequence, given lattice $\mathcal{L}$. In our case, we purposely fix word sequence to be from the best path in lattice $\bar{\pi}$: $\bar{W} = \text{wrds}(\bar{\pi}) = (\bar{w}_1, \bar{w}_2, \ldots, \bar{w}_M)$, and the confidence score is $c_{\bar{w}_q} = \gamma(q, \bar{w}_q)$. This MBR confidence is the default word confidence implemented in Kaldi.

Yet another method to obtain word-confidence is based on the averaging of the neural network log-posteriors selected with the one-best state-sequence corresponding to the recognized word [Zhang et al., 2014a].

Ideally, a well calibrated word-confidence should correspond to the probability that the word is correctly recognized. The experiments with per-word confidences are in section 7.3.

Per-sentence confidence c_{sent}

In works of other authors, the sentence confidences are usually computed as arithmetic mean of the per-word confidences [Novotney et al., 2009, Novotney and Schwartz, 2009, Thomas et al., 2013, Zhang et al., 2014a]. It is better to think about it as the estimate of the *word accuracy* in the sentence, rather than the correctness of the whole sentence. The supporting arguments for this interpretation are:

- a long sentence with one incorrect word can still be valuable for SST
- the word accuracy is closely related to the word error rate, which is the main evaluation metric for ASR systems

The experiments with per-sentence confidences are in section 7.2.

Per-frame confidence $c_{\bar{s}_t}$

In our work [Veselý et al., 2013b], we advocated for using frame-level confidence to do the frame-selection in mini-batch SGD training.

The per-frame confidence $c_{\bar{s}_t}$ is taken from the lattice posterior $\gamma(t, s)$, which is obtained by the forward-backward algorithm (see section 2.3.8). The posterior $\gamma(t, s)$ corresponds to the probability of being at time t in the tied-state s. Supposing that we have a sequence

of tied-states for the best-path $\bar{S} = (\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_N)$, the confidence value for frame t is extracted with its associated state $\bar{s}_t$ as:

$$c_{\bar{s}_t} = \gamma(t, \bar{s}_t) \tag{5.3}$$

The forward-backward algorithm over a lattice is the same as for calculating the denominator posteriors in MMI discriminative training, which was discussed in chapter 3.1.3.

An ideally calibrated frame-confidence should correspond to the probability that the frame-label $\bar{s}_t$ is correct. The experiments with per-frame confidences are in sections: 6.1 – frame selection, 6.2 – frame weighting, and 7.4 – detailed analysis, exploring of calibration.

What is best? It is very hard to predict which of the three types of confidence will be more useful. In some situations, it might be better to use the less specific sentence-confidences, while with the per-word and per-frame confidences we can locally decide about processing sub-chunks of utterances, which should be good as well. Any guess at this point would be a pure speculation, and we will search for the answer experimentally.

5.2.2 The concept of 'ideal' confidence

By its nature, an optimally calibrated confidence corresponds to the probability that the label is correct. For word-confidence, it is the probability that the recognized word matches its reference. In the case of frame-confidence, it is the probability that the hypothesized tied-state is the same as the element in the forced-alignment.

Only the per-sentence confidence is an exception following a different pattern. Usually we are not interested in the probability that the whole sentence is correct. Instead, we can replace the 'ideal' confidence value by the *word accuracy* in the sentence.

This applies to confidences in general, however, when used in semi-supervised training, it is not clear if these 'ideally calibrated' confidences also lead to the best results. It can be the case that additional processing of confidences is beneficial.

5.2.3 Calibration of confidences

The 'raw' confidences usually require further processing to become closer to the 'ideal' ones.

Calibrating confidences with logistic regression

The classical approach to calibrating confidences is to train a *binary logistic regression*. This requires a development set with transcribed data for training the model parameters. In [Novotney et al., 2009, Huang et al., 2013], logistic regression is used to calibrate the word-confidences, while analogically the calibration can be trained also for the frame-confidences. Logistic regression produces calibrated confidence, which corresponds to the probability that the label is correct. The experiments are in sections: 7.2.2 – per-sentence confidences, 7.3.2 – per-word confidences, and 7.4.3 – per-frame confidences.

Approximate calibration without training a model

If the confidences are values from the interval $(0, 1)$, we can warp them by exponential scaling $c' = c^{\alpha}$. With this approach, one can approximately calibrate the confidence by looking at a suitable scatter plot. Or the profile of calibrated confidence can be changes by

trying several values of α in a grid search experiment. The exponential scale then becomes a model hyper-parameter, ideally we would like to find a value that will generalize to other data-sets, if such value exists. The scale α is tuned in many experiments in chapter 7 and later.

Yet another approach to approximate the calibration is re-scaling the lattice weights (P_G, P_{AM}) with λ, which modifies the 'sharpness' of resulting posteriors (explained in section 2.3.8 on page 20). This technique was used mainly in section 6.2. However, later, in table 7.16 we see the best results with 'default' $\lambda = 1$, this suggested to stop using lattice-scale λ.

5.2.4 Use of confidences in SGD

Selecting data by confidence

The simplest approach is to select the automatically transcribed training data by setting a threshold on the confidence, or alternatively by setting the target fraction of data to accept. The more reliable data are accepted (higher confidence), the less reliable data are discarded. For data selection, we don't need calibration of confidences. What matters is the ordering of data according to confidences, the actual values of confidence are not important.

Confidence-weighted training

An alternative approach is to weight the data by the confidence, where the weights c_n are applied to the NN gradients of the individual data-points ∇E_n. The update rule of mini-batch SGD (2.27) is slightly modified to:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \sum_{n=1}^{M} c_n \nabla E_n(\mathbf{w}^{(\tau)}) \ . \tag{5.4}$$

The simple way to implement the gradient scaling is to use c_n to scale the partial derivative of loss function E_n w.r.t. network output $\mathbf{y}$, that we had in equation (2.30):

$$\frac{\partial E'_n}{\partial \mathbf{y}} = c_n \frac{\partial E_n}{\partial \mathbf{y}} \ . \tag{5.5}$$

The re-scaled loss derivative is then back-propagated into the network. We can do this because the back-propagation is a series of matrix multiplications with the Jacobian matrices corresponding to NN components. Therefore, scaling the loss derivative is equivalent to scaling of the gradient itself.

We can also see the weighted training as a generalization of the data selection. In data selection, the weights are constrained to be strictly binary (1 = include, 0 = discard). In weighted training, we can include the data-points partially, in the sense that the network will have a chance to learn the input pattern, while giving it lower importance in calculating the overall loss.

Again, it is not clear if the 'ideally calibrated confidence' is also the best weight for the mini-batch SGD training. It is very likely that the 'ideal confidence' is a good starting point, from which we should try going further and transform it in some convenient way, for example by the exponential scaling with α.

The approaches of the 'data selection' or 'data weighting' are explored in almost all later chapters.

5.3 Summary of published works

In this summary, we mention the important works related to the semi-supervised DNN training. The articles are organized into categories, while often we could assign them to more than one category.

5.3.1 Semi-supervised training of GMM-HMM systems

Self-training with confidence

The self-training of ASR systems was studied in journal article [Wessel and Ney, 2005]. In the proposed training scenario, the per-word confidences are used to select the frames corresponding to words for which the confidence is higher than a threshold. The GMM-HMM model is trained by Maximum-likelihood EM algorithm in which the alignment of automatic labels changes during the training.

The article demonstrates that word-confidences are helpful in semi-supervised training starting from the seed systems trained on 1.2 – 5.6 hours of transcribed data. Also, the iterative self-training is shown as helpful when starting from 1.2 hour seed system.

Our comment: The situation might be different with neural network training, which are trained to classify frames. In the case of GMM a few wrongly labeled data-points have low effect to the overall shape of the distribution, while for SGD, the wrongly labeled data can be more harmful. The data-sets in Wessel's article have similar sizes as ours (5.6h transcribed and 66h untranscribed; our setup has 10h transcribed and 74h untranscribed). The article uses dynamic alignment, while in our setup, we work with a fixed best path from lattice.

The role of language model

[Novotney et al., 2009, Novotney and Schwartz, 2009] is provide a different perspective to self-training of HMM-GMM systems. The authors observed that a strong language model improves the WER recovery, while using a weak LM is still sufficient to improve the acoustic model by self-training. The WER recovery ranged between 40% - 80%, depending on the size of the transcribed and untranscribed data-set and the language model training corpus.

An interesting group of words are those that are present in the untranscribed set, and not in the transcribed one. The paper reports accuracy improvement 19%→32% for such words caused by the self-training.

The paper proposes confidence-based sentence selection, while the sentence confidence is computed in two stages. First, the per-word confidence is trained as logistic regression on a development set with binary targets corresponding to word correctness. Second, the sentence-level confidence is trained as a logistic regression with binary targets, where the two classes group the sentences where WER is below or above the average in the development set. Such confidence model is used to choose sentences with predicted better than average error rate.

Our comment: It will be interesting to compare the WER recoveries from the GMM-HMM models with the recoveries from the DNN-HMM models. In our work, we do not focus on the role of language model. We use a language model trained on the transcripts for building the seed model (Babel Vietnamese) or the Fisher training transcripts (Switchboard setup).

We would also like to briefly mention other works on semi-supervised training of GMM-HMM systems. In [Gollan et al., 2007] the author performs word selection according to 'word-posterior confidence' or the 'allophone state posterior confidence'. In the thesis [Gollan, 2014] the author further extends the topic by discussion of Automatic learning (self-learning) in Chapter 8. Yet another work [Fraga-Silva et al., 2011] reports improvements from self-training with soft-probabilistic targets, i.e. lattice-based training compared to 1-best training without data filtering (we tried same idea for DNN training, but in our setup the data selection was better than the training with the 'soft targets').

5.3.2 Confidence methods based on single ASR system

In this section we summarize confidence methods based on a single ASR system. The C_{max} [Wessel et al., 2001] confidence was combined with 'phone occurrence confidence' in [Thomas et al., 2013]. Later [Zhang et al., 2014a] shown that the neural network posterior confidence C_{pos} is better than the 'phone occurrence confidence' alone, while we will show that the sentence confidence from Minimum Bayes Risk decoding statistics is better than C_{pos} (section 7.2.4).

Phone occurrence confidence

[Thomas et al., 2013] has shown, how much can be achieved with 1h of the transcribed training data when training a Tandem system with bottleneck features and GMM-HMM acoustic model, using the Call Home English data. The bottleneck feature extractor is pre-trained both with multi-lingual data and selected semi-supervised data. Then it is re-trained with the 1h of correctly transcribed data. The GMM-HMM model is at first trained with the 1h dataset, while the automatically transcribed data are added. The absolute WER improvement is impressive 16% (WER 71.2 $\rightarrow$ 55.2, out of which 75% came from improving the bottleneck features and 25% from the self-training of GMM-HMM model).

The actual confidence is a fusion of C_{max} [Wessel et al., 2001] and 'phone occurrence confidence' C_{occ}. C_{occ} is based on matching the decoded word with the phone posteriors from the output of the NN used to extract bottleneck features. The phone posteriors are selected along the Viterbi path through the word, while calculating the ratio of frames in which the selected posteriors were above a threshold.

However, we should be cautious about interpreting the article. For example, it does not compare the following three confidences: a) C_{max}, b) C_{occ}, c) $f(C_{max}, C_{occ})$. So, we do not know how important it was to include the confidence C_{occ}. Then, the article reports a setup with speakers overlapping in the correctly transcribed and the untranscribed data, which increases the likelihood, that the untranscribed data will be recognized correctly.

Neural network posterior confidence

[Zhang et al., 2014a] studied semi-supervised training for meeting recognition (English). The seed-system is trained either on matched data (scenario 'S1') or on mismatched data (scenario 'S2'). In the mismatched scenario, the seed system is trained on ICSI meeting recordings (U.S. English), while both the untranscribed and the test data are from AMI meeting corpus with 'European', mostly non-native, English.

The structure of the ASR system is a Tandem consisting of NN-based features and GMM-HMM acoustic model. The article studies the effect of replacing the NN-output classes by: tied-states, mono-phone states, mono-phones and grouped mono-phones (8 classes). In the

matched scenario 'S1', it was good to use the tied-states for the self-training of the NN-based feature extractor, while for mismatched scenario 'S2', it was better to use the mono-phones.

The proposed confidence C_{pos} is based on averaging the log-posteriors from the front-end neural network along the Viterbi path of the decoded word. The sentence confidence is the arithmetic mean of word confidences; sentence confidence is used for selecting the sentences. The paper claims that the posterior confidence for sentences is better than the 'phone occurrence confidence' C_{occ} from [Thomas et al., 2013]. However, the C_{occ} was originally used in combination with C_{max}, so it is not clear which article proposes better confidence. Nevertheless, we will compare C_{pos} with the Minimum Bayes Risk decoding statistics in section 7.2.4 to show that the MBR statistics are better than C_{pos}.

Statistics from Minimum Bayes Risk decoding

We already introduced this method in section 'Per-word confidence' on page 51, where we used it to assign confidences to words from best path in lattice.

However, the method can also 'replace' words in the hypothesis: The MBR decoding method from [Xu et al., 2011] is used to decode the word (or state) sequence, which minimizes expected word (or state) error rate, where the expectation is taken w.r.t. the decoded lattice. As shown in [Xu et al., 2011, table 1], the decoding method leads to a better WER than both the Maximum-a-posteriori decoding and Consensus-network decoding. It can also be used as a powerful method for system combination. In [Xu et al., 2011, table 4], we see that it outperforms both ROVER and the Confusion-network combination (CNC). The system combination by MBR decoding was also studied in [Swietojanski et al., 2013].

We use MBR decoding for combining two ASR systems in chapter 9. The decoded word-strings are used as automatic transcripts for semi-supervised training.

Calibrating confidences by logistic regression

The calibration of per-word confidences is well described in [Yu et al., 2011]. It assumes a situation of a developer who has no access to the ASR engine internals. The Max-entropy model from the article is equivalent to training a logistic regression on the proposed input features: polynomial expansion of continuous features (per-frame acoustic model score, background model score, noise score, ...), word-identity as 1-of-M encoding and the context expansion represented by confidences of the adjacent words. The aim of the paper is to improve the confidences to make them closer to the 'ideal' confidence. The paper does not consider semi-supervised training. We will try to use this to enhance the word confidences in our experiments.

Out Of Vocabulary word detection

The use of per-frame confidences is reported rarely. An exception is the detector of Out-of-vocabulary words (OOVs), developed for the JHU workshop in 2007 [Burget et al., 2008]. Here, the core idea was to detect the OOVs from the discrepancy of per-frame phone posteriors obtained from the lattices generated by a 'strongly constrained' word-based LVCSR system and a 'weakly constrained' phone recognizer, while both systems use the same acoustic model. The 'raw' posteriors are combined by a small neural network, where adding the temporal context to the input is helpful. The paper suggests to use the central frame and 2 side-frames with the offsets +6 and -6 frames. The network has 3 classes: silence, non-OOV and OOV. It is shown that the OOV detector can be used as a generic detector of

recognition errors, not limited only to OOVs.

Our comment: In our case, we are not interested in OOV detection, but still the confidences from the 'weakly constrained' system can be helpful. To simplify the setup, we can first work with the per-frame confidences from the 'strongly constrained' LVCSR system, which we will do in section 6.1.

5.3.3 Other methods

Entropy minimization

Another family of the semi-supervised methods incorporates the uncertainty about the unlabeled data into the objective function. The system is trained to reduce the entropy of the probabilistic assignment which the decoder gives to untranscribed data. [Grandvalet and Bengio, 2004, Huang and Hasegawa-Johnson, 2010, Yu et al., 2010b, Manohar et al., 2015]. The entropy is computed from posteriors of lattice-paths. It becomes minimal, if all probability mass is assigned to the strongest path from lattice.

Feature-space manifold

Yet another family of semi-supervised methods is based on the feature-space manifold assumption using a graph-based framework [Malkin et al., 2009], where the nearest supervised data-point suggests the label. These methods rely on building a large index with the features. The index is used for searching the labeled data that are similar with the unlabeled data.

Two-softmax semi-supervised training

An interesting idea for DNN self-training is to use *two softmax layers* [Su and Xu, 2015] on the output. The first softmax output is trained by the transcribed data, the second softmax is trained with the automatically transcribed data, while the hidden layers are shared. The self-training is done without data selection or frame-weighting. The paper proposes to discard the output layer and train a new softmax output with the transcribed data.

The paper also experiments with data selection, in which the sentences with the top 20% or bottom 20% confidences are used. However according to importance sampling experiment in [Huang et al., 2013], the largest improvements are achieved by self-training with sentences having 'middle' confidence.

A similar setup with two softmaxes is described in [Manohar et al., 2015]. In the self-training without the data-filtering, it is beneficial to use the second softmax layer for the training with the automatic labels, while lowering the weight of the unlabeled data. The semi-supervised scenario is further extended to the sequence-discriminative training in which sMBR objective is used for the transcribed data. The untranscribed sentences are trained to maximize the Negative Conditional Entropy on the second softmax. The approach is very interesting. However, the absolute WER improvements from the semi-supervised training were about 3x lower than what we achieved in [Veselý et al., 2013b].

Multi-system confidence re-calibration, importance sampling

The scenario in [Huang et al., 2013] uses thousands of hours of untranscribed data for semi-supervised training. The importance sampling is implemented by splitting the untranscribed data into 10 bins of 1M sentences according to the re-calibrated confidence. Each bin is used for the semi-supervised training. The best result is achieved with the middle-range of confidences (5th bin), which leads to better result than than selecting the bin with highest confidences.

The confidence re-calibration is based on ROVER system combination that is followed by a round-robin of exponential re-calibration based on number of supplementary systems that agree with the main system.

Our comment: The importance sampling experiment showing that the most helpful data are not those with the best confidence, but the middle ones, is interesting. The paper suggests that re-calibrated confidence helps to select data with lower PER, but it does not explicitly compare the original and re-calibrated confidences in the self-training experiments.

Multi-system automatic transcripts, Condition random fields (CRF)

The idea of multi-system semi-supervised training was recently re-explored in [Li et al., 2016], in which the automatic transcripts are combined from two systems by a hierarchy of CRF models.

5.3.4 What we believe to be interesting

To summarize, there are many interesting ideas in the literature. Here is a list of important topics:

- different methods to obtain per-word confidences
- per-frame confidences
- calibration of confidences
- re-training with transcribed data
- importance sampling
- iterative semi-supervised training
- realigning automatic transcripts
- two-softmax semi-supervised training
- entropy minimization semi-supervised training
- multi-system automatic transcripts and confidences

Clearly, we cannot re-explore all these ideas. Instead, in the spirit of the Occam's razor, we will try to build the simplest possible system that will work well in practice. With this literature survey, we broadened our know-how of semi-supervised training and we also created the context to situate our experiments into.

Chapter 6

Initial experiments with semi-supervised training

6.1 Frame selection by confidence (ASRU2013)

This chapter is based on our first article about the semi-supervised training [Veselý et al., 2013b]. The key idea was to use two types of confidences (per-sentence, per-frame), to be able to perform 'smart' data-selection in SGD training. At this point we are using the uncalibrated 'raw' confidences, which are easy to obtain and good enough for the data selection. In this section we use Babel Vietnamese setup described in section 4.1.

In our scenario, we use 10h of transcribed data and 74h of untranscribed data. For seeding, we use a state-of-the-art DNN system built by RBM pre-training, frame-classification training and sMBR training as we previously introduced in section 3. In this section we use the Babel Vietnamese database, from which we take the LimitedLP condition.

6.1.1 Confidence measures

In general, it is beneficial to incorporate some form of confidence measure into the semi-supervised training. In self-training, we decode using the seed system and treat the hypothesis as a reference. If we had a number that tells us how certain the decoder was about the decoded hypothesis, we could use it to pre-select the data and remove hypotheses which are more likely to contain errors. However, it is not clear on which level the confidence scores should be extracted.

A popular approach [Huang et al., 2013, Thomas et al., 2013] is to compute word-level confidences, which are then used to derive sentence-level confidences. However, in our initial experiments, we use the frame-level confidence.

As we already introduced on page 51, the per-frame confidence $c_{\bar{s}_t}$ is taken from the lattice posterior $\gamma(t, s)$, which is obtained by the forward-backward algorithm (see section 2.3.8). The posterior $\gamma(t, s)$ corresponds to the probability of being at time t in the tied-state s. Supposing that we have a sequence of tied-states for the best-path $\bar{S} = (\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_N)$, the confidence value for frame t is extracted with its associated state $\bar{s}_t$ as:

$$c_{\bar{s}_t} = \gamma(t, \bar{s}_t) \tag{6.1}$$

The values of this confidence measure reside in interval $(0, 1)$, they can be used either for threshold-driven *data selection* or training with *weighted data*, as introduced on page 53.

6.1.2 Seed system

Feature extraction, auxiliary GMM-HMM system

For the paper [Veselý et al., 2013b] we used the PLP-fMLLR features, which were described in section 3. At that time, the Kaldi pitch extractor did not exist, instead we used a pitch algorithm described in [Talkin, 1995], while the 'raw' pitch was mean-normalized per speaker and the non-voiced parts were bridged-over by linear interpolation. All the input features were then processed by speaker-based mean/variance normalization. The baseline GMM-HMM system with 2300 cross-word triphone tied states and 10 Gaussians per state is used to prepare LDA+STC+fMLLR features.

For the supervised data, we compute fMLLR transforms from forced-alignments. For the unsupervised data we compute fMLLR from lattices by using two passes of decoding. The GMM-HMM system is also used to produce DNN training targets by forced-alignment to the transcription, these triphone-state targets are used for the frame-classification training. The DNN triphone tree is inherited from the baseline GMM-HMM system.

DNN-HMM seed system

The DNN training procedure and topology were described in section 3.1. The DNN has 6 hidden layers of 2048 sigmoidal neurons, there is 440 dimensional input and 2.3k dimensional softmax output. The 40 dimensional fMLLR features are spliced by +/- 5 frames and re-normalized for the DNN input.

We use generative RBM pre-training [Hinton et al., 2006] to initialize the 6 hidden layers. The next step is frame-classification training till the convergence, re-alignment of triphone tied-state target labels, and second run of the training. Finally, the network is fine-tuned by sequence-discriminative training with sMBR objective. As proposed in [Veselý et al., 2013a], we re-generate the lattices and reference alignment after the first epoch, running 1 + 4 epochs while using a fixed small learning rate of 1e-5.

6.1.3 Supervised experiments

As described in previous section, the seed model is trained in several stages. The auxiliary GMM-HMM model is trained by mixing-up maximum likelihood training, where the last stage produces LDA+STC+fMLLR features. Then a DNN is built on top of fMLLR features by using layer-wise pre-training, two runs of frame-classification training and 1+4 iterations of sequence-discriminative training.

We applied the above described training procedure to both the LimitedLP and FullLP conditions. By looking at table 6.1, we see system performance at individual stages of the training. In the last stage, the LimitedLP system (that is later used as seed model) has 13.6% worse WER than the FullLP system. This large difference is not solely due to lower amount of acoustic model training data, but also due to smaller lexicon, language model trained on smaller set of transcripts, different segmentation and smaller number of triphone states. In FullLP condition the optimum was 4800 states, while for LimitedLP it was 2300.

In order to get an idea how much a DNN can improve by unsupervised self-training from the seed model, knowing the correct transcripts, we performed an 'SST w. oracle ref'. The rest of the LimitedLP system was the same as before (LM, lexicon, GMMs, fMLLR features). The WER we obtained is 57.0%, we will consider this to be the upper bound performance for the semi-supervised training and the calculation of WER recovery.

Table 6.1: *Baseline performance of LimitedLP system trained on 10h data (2% of low-confident segments removed); upper bound FullLP performance*

Dataset [WER] (transcribed hrs.)	LimitedLP (10.8 hours)	FullLP (84.8 hours)
GMM (fMLLR)	69.0	58.6
DNN	63.1	50.4
(+ SST w. oracle ref.)	(57.0)	-
DNN-sMBR	60.6	47.0

6.1.4 Semi-supervised experiments

Our objective is to make such use of unannotated in-domain speech data, that the WER performance of DNN ASR system improves. In this section, we will search for an optimal strategy to achieve this goal.

RBM pre-training

As the first experiment, we tried to add data to the RBM pre-training; this is trivial since the Contrastive Divergence algorithm does not need any labels. As can be seen in table 6.2,

Table 6.2: *Adding more data to unsupervised RBM pre-training*

Pre-training data [h]	10.8	84.8
Pre-training iterations	10	3
Fine-tuning data [h]	10.8	10.8
WER	63.8	63.8

we tried pre-training with more iterations on smaller set and less iterations on larger set, which contains both the annotated and unannotated data. In both cases, the fine-tuning (frame-classification training) was performed with the same annotated dataset. However, there is no WER difference between the two systems, hence adding the untranscribed data to RBM pre-training is not helpful. This is consistent with the observations previously published in [Yu et al., 2010a, Swietojanski et al., 2012].

Frame-classification training (cross-entropy)

As the pre-training is not promising, we focus on frame-classification training. In the first experiment, we add all the sentences of unannotated data. Due to mini-batch SGD training, the annotated and unannotated data are mixed together. As can be seen in table 6.3, we obtain significantly better results by simply adding all the automatically transcribed segments[1].

Due to the large disproportion between the amount of annotated and unannotated data (≈1:7), we tried to include the annotated data several times to the training set. This leads

[1] In [Veselý et al., 2013b], we worked with a premise that the per-sentence confidences are not helpful for the semi-supervised training. This was suggested by [Veselý et al., 2013b, table 4]. After repeating the same experiment later, we realized that a small improvement can be achieved by removing 30-50% of the sentences (table 7.4). However, the frame-selection still leads to better results than sentence selection.

Table 6.3: *Adding unannotated segments (based on table 4 in [Veselý et al., 2013b])*

Added segments	0%	100%
WER	63.1	**62.0**

to stronger focus on transcribed data during the SGD training. In table 6.4, we see that a slight WER improvement can be achieved by including the annotated data 3x[2].

Table 6.4: *Including several copies of annotated data, while using 100% unannotated segments*

No. copies	1x	2x	3x	4x	5x
WER	62.0	62.0	**61.7**	61.8	61.9

As we believed that per-sentence confidences are not helpful, we experimented with the data selection based on the per-frame confidences. By doing the data-selection on the frame-level, we are able to discard the frames (i.e. data-points), where the decoder was uncertain. The results in table 6.5 indicate that we can achieve significant WER improvement by using the frame-selection. Again, the WER is relatively insensitive within the interval of high thresholds.

Table 6.5: *Dropping frames from unannotated part according to threshold on per-frame confidence, while using 100% unannotated segments and including annotated part 3x*

Threshold	0.0	0.5	0.7	0.8	0.9	0.95
Removed frames	0%	11%	18%	23%	28%	32%
WER	61.7	61.2	**60.9**	60.9	61.0	61.0

In the last experiment, we performed frame-weighted training. We used the per-frame confidences, i.e. the posteriors of being in the correct state, to re-scale the vectors with error derivatives that are used for backpropagation. As the gradient depends linearly on error derivative through Jacobians, scaling the derivatives is equivalent to scaling of gradients. We combine the frame-weighting and frame-selection. The results in table 6.6 show, that with threshold 0.5, there is a small WER improvement compared to the same threshold in table 6.5. However, with threshold 0.7, which corresponds to the best system, there was no WER difference, therefore frame-weighting and frame-selection do not seem to be complementary[3].

The overall absolute WER improvement coming from semi-supervised frame-classification training is 2.2%, from which 1.1% is caused by adding all the unannotated segments and 1.1% comes from thresholded frame-selection. This corresponds to WER recovery [Novotney and Schwartz, 2009] of 36%.

[2]Later, we found that adding the copies of transcribed data is not always helpful. Particularly, when the we tune the exponential scale of confidences, the WER improvement from several copies disappears (the confidences scale SGD gradients).

[3]As the confidence was not calibrated, we should be cautious and perform further investigation.

Table 6.6: *Frame-weighting by a confidence, while using 100% unannotated segments, including annotated part 3x and using thresholded frame-selection*

Frame-selection threshold	WER
0.0	61.3
0.5	60.9
0.7	60.9

Sequence-discriminative training (sMBR)

So far, we have observed WER improvements from the semi-supervised frame cross-entropy training, on the other hand our seed system was trained using sequence-discriminative criterion (sMBR). To outperform the seed system, we need to apply sequence-discriminative training as well.

It is not clear whether the strategy based on thresholded frame-selection will be efficient also for the sMBR training. At first, we try to outperform the seed system by applying the supervised sequence-discriminative training to the best self-trained DNN, using the 10.8 hours of the correctly transcribed data. As can be seen in table 6.7, the WER improvement slightly lowered, it was 2.2% after the frame cross-entropy training, and it became 1.8% after the sMBR training.

Table 6.7: *Supervised sequence-discriminative training of the best self-trained DNN*

	baseline	semi-supervised	
cross-entropy data	10.8h	84.8h	
sMBR data	10.8h	10.8h	
	WER	WER	abs. improvement
cross-entropy training	63.1	60.9	2.2
sMBR training	60.6	**58.8**	1.8

Our intuition is that further improvements are possible by using the semi-supervised sequence-discriminative training. However, we did not succeed to create a plausible sequence-discriminative recipe that would use the untranscribed data and that would also lead to a better performance. Hence, we keep on using sMBR with the correctly transcribed data after the semi-supervised frame-classification training is finished. We can see it as a 'fine-tuning' with the data that have the correct labels.

Summary

In this section, based on our conference article [Veselý et al., 2013b], we experimented with the semi-supervised frame-classification training. We found out that frame selection according to the per-frame confidence leads to reasonable results. However, we did not yet consider the possibility of post-processing the confidences. For sMBR training, we use the correctly transcribed data.

6.2 Re-scaling the frame posteriors

By looking again at table 6.5, we see that the confidences are not calibrated. For example, with the threshold 0.5 we discard only 11% of frames, while with the calibrated confidence we should discard 50% frames. In this section we will present a simple technique which approximately calibrates the confidences, making it more suitable for the frame-weighted training. Here, we use Babel Bengali and other languages described in section 4.2.

6.2.1 Analyzing the per-frame confidences

In section 6.1, we suggested to use the per-frame confidence to *select* reliable frames for the mini-batch SGD training. The per-frame confidence is the lattice posterior $c_{\bar{s}_t} = \gamma(t, \bar{s}_t)$, of tied states $\bar{S} = (\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_T)$ obtained from best path in lattice. The state sequence $\bar{S}$ is used as target labels for NN training. The per-frame confidences $c_{\bar{s}_t}$ are in the interval $(0, 1)$, and we will use them both for the *frame selection* and the *confidence-weighted* training.

Intuitively, an *ideal confidence* can be seen as the probability that the automatically generated label is correct. To see how well the frame-confidence matches the label accuracy, we created figure 6.1. Here, the 'state-accuracy' is the percentage of matching elements in tied-state sequences, comparing the *best-path* in lattice and *forced-alignment*.

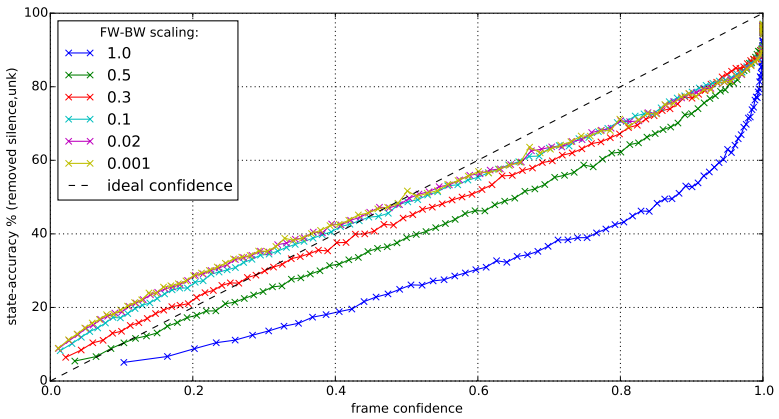


Figure 6.1: *Tied-state accuracy as function of confidence. By lowering the lattice-scale λ the curve becomes closer to the 'ideal confidence'. — We split the data into 100 bins, each bin has frames of similar confidence. Then, each point on the curve is the frame accuracy of one bin. The data contains 48% of 'correct' frames (after removing silence frames, 35%). Created from Babel Bengali data.*

In figure 6.1 we see that the original frame-confidence (the dark blue curve) is far beneath the dashed line which represents the 'ideal confidence'. This means that the confidence is too 'optimistic': for many frames, the confidence is much higher than the actual frame accuracy. To reduce the mismatch, we apply the lattice-scale λ both to the acoustic and graph scores:

$$(P_{AM}(X|W)^{\kappa} P_G(W)^{\varrho})^{\lambda} . \tag{6.2}$$

This controls the 'sharpness' of lattice-posteriors from forward-backward algorithm. The lattice-scale λ was previously discussed on page 20.

The curves for different lattice-scales λ in figure 6.1 show that as the scaling factor gets lower, the curves get closer to the dashed line representing the ideal confidence, while the curves do not change much for scales 0.1 and lower. The $\lambda = 1.0$ corresponds to the original scaling (the acoustic scale of $\kappa = 0.1$ and graph scale $\varrho = 1.0$).

As changing the lattice-scale λ affects the distribution of confidences, we plot it in figure 6.2, and we realize that with the lower scales the confidences become more uniform.

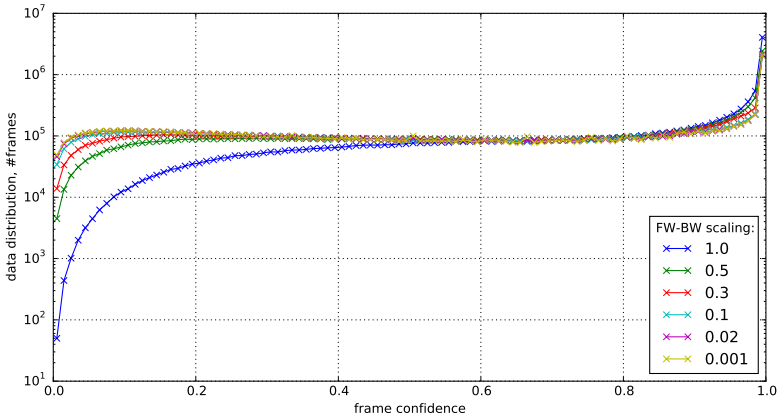


Figure 6.2: *Distribution of the per-frame confidences, when the lattice-scale λ is applied. Low scales lead to a more uniform distribution.*

Lastly, it is interesting to realize how the lattice-scale λ affects the posteriors of paths through the lattice. With low scales, the posteriors of all the paths get closer. Let's illustrate the effect on the following example: suppose we have a lattice with 3 paths having posteriors $p_1 = 0.9,\ p_2 = 0.09,\ p_3 = 0.01$. We apply scaling factor 0.1 and re-normalize to sum up to one. What we get is $p'_1 = (p_1)^{0.1}/Z \doteq 0.35,\ p'_2 = (p_2)^{0.1}/Z \doteq 0.33,\ p'_3 = (p_3)^{0.1}/Z \doteq 0.32$. It is very likely that the actual scores in the lattice become less important than the 'depth' of the lattice, which explains the non-changing curves for $\lambda = 0.1$ and lower in figure 6.1.

6.2.2 The seed system

In this section, we use Babel Bengali and the 'Other Babel languages': Assamese, Haiti, Zulu and Lao. The experimental setup for both the 'seed systems' and 'semi-supervised systems' was described in section 4.2.

Feature extraction, auxiliary GMM-HMM system The setup is similar to the description in section 6.1.2. The differences are that here, the PLP features are extended by the 3-dimensional Kaldi pitch features [Ghahremani et al., 2014], which replace the former pitch algorithm. We also increased the number of tied-states in the GMM-HMM from 2000 to 4000, which leads to roughly 7 Gaussians per state. Also, the fMLLR transforms are obtained by three passes of lattice generation and fMLLR estimation, previously there were two passes.

With the final GMM-HMM system, we produce the initial labels for the DNN training and the binary decision tree for the tied-state clustering.

DNN-HMM system The DNN training procedure is the same as we previously described in section 6.1.2. The only difference is that here, we removed the <unk> words from the frame-CE and sMBR training. The <unk> words are: OOVs, words from other languages and incomprehensible words.

6.2.3 Experiments with lattice-scale λ, frame-weighted training

Motivated by promising analysis, we tested the approximate confidence calibration method based on tuning the lattice-scale λ. In table 6.8, we see the frame-CE training results. For scale $\lambda = 0.02$, we obtained 0.3% lower WER, compared to the original λ 1.0, while the other λ scales 0.5, 0.1 and 0.001 lead to the results that are almost as good as with the scale 0.02.

Table 6.8: *Tuning the lattice-scale λ in weighted frame-CE training. We use 100% of automatically transcribed segments and the manually transcribed data are included 1x.*

lattice-scale λ:	1.0	0.5	0.3	0.1	0.02	0.001
WER (Bengali)	61.3	61.1	61.2	61.1	**61.0**	61.1

In next set of experiments, we fixed the scale λ to 0.02, and we combined the weighted training with the data selection according to confidence threshold. In table 6.9, we see that the WER performance is relatively insensitive to the confidence-threshold. In case of Bengali, we get 0.1% better result with threshold 0.5, however, this improvement did not repeat for Zulu. If we train on all the frames (i.e. without frame selection), there seems to be no significant hit in performance and the recipe becomes simpler.

Table 6.9: *Tuning frame rejection threshold, while using lattice-scale $\lambda = 0.02$, and including manually transcribed part 1x. We train only with frames whose confidence is above threshold.*

Rejection threshold:	0.0	0.25	0.5	0.75	1.0
WER (Bengali)	61.0	61.1	**60.9**	61.7	63.6
WER (Zulu)	**67.6**	67.8	67.8	68.0	69.1

In section 6.1, we saw that repeating the manually transcribed data 3x helps. However, here, the same repeating of data (3x) caused a little degradation 61.0 → 61.2 (Babel Bengali). Hence, it is sufficient to include the manually transcribed data 1x, which again simplifies the setup.

With the best setup for Bengali, we achieved a WER recovery of 49%. The related performances are in table 6.10, in which the last line is the replica of the best recipe from [Veselý et al., 2013b] applied to Bengali (frame selection with threshold 0.7, repeating transcribed data 3x). We see that frame-weighting is on-par with the frame-selection, that we used previously *(but we'll show later in section 7.4 that the weighted-training becomes a little better, after we introduce the exponential scale α).*

Finally, table 6.11 shows the WER improvements for the LimitedLP condition of the five development languages from the second year of Babel program. We obtain consistent WER improvements for all the languages ranging 2.0-3.0% absolute, when comparing the 'seed model' and the '+ sMBR' system. The improvement is coming from the semi-supervised frame Cross-Entropy training, using our new strategy based of frame-weighted SGD training, which uses re-scaled lattice-posteriors as the weights (lattice-scale $\lambda = 0.02$). After the semi-

Table 6.10: *Performance of semi-supervised training for Bengali, frame Cross-Entropy training. The WER recovery is 49%.*

system	WER	abs. improvement
No data added	63.6	-
Semi-supervised	60.9	2.7
Oracle transcripts	58.1	5.5
ASRU13 Recipe	60.9	2.7

supervised training, we continue with the sMBR training, for which we use the smaller 10h set of manually transcribed data.

Table 6.11: *Semi-supervised training for five languages from the second year of Babel program.*

WER	Assamese	Bengali	Haiti	Zulu	Lao
seed model (sMBR)	60.1	62.0	57.4	67.4	53.2
per-frame baseline	62.0	63.6	58.6	68.6	54.5
+ semi-supervised	59.6	61.0	55.5	66.5	52.0
+ sMBR	57.9	59.8	54.4	65.4	50.8
overall abs. improvement	2.2	2.2	3.0	2.0	2.4

6.2.4 Summary

We have proposed a simple heuristic to approximately calibrate the frame confidences, based on re-scaling the lattice-posteriors with lattice-scale λ. This simple trick removes a big portion of the distance to the 'ideal confidence'.

The refined confidences are then used in semi-supervised mini-batch SGD training, where the frames are weighted with the confidence. The experiments have shown that we no longer need to tune the frame-selection threshold. Also, the duplication of transcribed data is no longer needed. This simplifies the semi-supervised recipe we have previously proposed in section 6.1, while the WER performance is on-par for both approaches.

However, later we found that the 'ideally calibrated confidence' is not guaranteed to be the best possible confidence for semi-supervised training with weighted data-points. Later, in table 7.16 on page 82 will be shown that the results are better with the 'default' $\lambda = 1.0$ and the tuned $\alpha = 5.0$. Also, the lattice scale $\lambda = 0.02$ did not generalize to Switchboard recipe (table 8.1 on page 89). Therefore, the technique is not suitable for practical use, but it was an important 'stop' on the way to more systematic study, which is in next chapter.

Chapter 7

What is the best granularity of confidences?

Previously, in section 5.2, we 'layed-out' the area for our semi-supervised training experiments. The key questions were: A) "Should we work with per-sentence, per-word or per-frame confidences?" B) "Should we use data selection or data weighting?" C) "Should we calibrate the confidence?" In this chapter we explore these questions systematically. We begin with 'oracle' experiments, followed by three sections with confidence-types from question A). We also introduce re-training with manually transcribed data, which is necessary to better decide which semi-supervised setup is preferable. Here, we returned to our Babel Vietnamese setup (LimitedLP) from section 4.1

Seed system The seed system is built with full recipe as described in chapter 3. It includes the training of initial GMM-HMM system to produce PLP-pitch-fMLLR features. Then the recipe continues with RBM pre-training, frame cross-entropy training and sMBR training. The seed system is built with 10 hours of transcribed Babel Vietnamese LimitedLP data, the performance of the seed system is 59.6% (in section 6.1 we had 60.6, the difference is caused by improved pitch features [Ghahremani et al., 2014]. We also increased the number of tied-states in the fMLLR-GMM from 2000 to 4000, which improves the oracle results, but does not change seed system performance dramatically.) The performance of intermediate stages is in table 7.1.

Table 7.1: *Performance of seed system we use in this chapter for semi-supervised experiments. The initial fMLLR-GMM system was used to produce input features, the CE-DNN was trained with frame cross-entropy loss, then it was re-trained with sMBR objective.*

	fMLLR-GMM	CE-DNN	**sMBR-DNN**
WER	66.1	61.7	**59.6**

Semi-supervised training Again, we do frame-CE semi-supervised training with intermixed automatically and manually transcribed data. The automatic labels for NN training are the tied-states $\bar{S} = (\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_T)$ from the best-path in lattice $\mathcal{L}$, which was generated with sMBR-DNN seed system. The specific confidences and the way they are used is described later in the chapter.

7.1 Oracle experiments

To better understand the effect of calculating confidences for smaller or larger units, we designed the oracle experiments which use the 'ground truth' transcripts of the untranscribed data. The transcripts are used to *mark the correct words* in the recognized text by using the word-alignment from scoring, or to *mark the correct frames* in the best state-sequence $\bar{S}$ by comparing it with the forced-alignment to transcripts. We create the 'oracle' confidences in the following way:

Per-sentence confidence is related to the estimate of word accuracy in the sentence, rather than the correctness of the whole sentence. The oracle value is the word-accuracy in the automatically transcribed sentence. The DNN self-training is done with the sentences scaled by their word-accuracies, which are post-processed by tuning the exponential scale α (see table 7.2).

Per-word confidence is related to the probability that the word is correct. Here, according to the word-alignment from scoring, the correct words get oracle confidence 1 and incorrect 0, this confidence is then assigned to the frames of the word. The confidences for silence frames on sentence ends or between words are filled with linear interpolation (35% of all frames).

Per-phone confidence would be the probability of labeling a frame with the correct phone. For the oracle confidence, we compare the phones on the best-path from lattice with the phones in forced-alignment. The frames where the phones match get confidence 1, otherwise 0. The self-training is done with these per-frame weights. For the sake of comparison, we simplified the Vietnamese phone-set not to consider the tones or the position in the word.

Tied-state confidence is similar to the per-phone confidence, with the difference that we are comparing the sequence of acoustic model classes (i.e. pdfs, tied-states). Again, we compare the lattice best-path with the forced-alignment. The frames with the same tied-states have confidence 1, otherwise 0. We train with weighting the frames by these confidences, which in fact selects the 'correct' frames from the forced alignment of the untranscribed data-set. This oracle confidence is thus very optimistic.

Table 7.2: *Per-sentence oracle confidences, the sentences are weighted by word accuracy (scaled exponentially by α: $c' = c^{\alpha}$). Although the WER of seed systems with sMBR training was 59.6%, we should also compare to CE-DNN baseline with WER 61.7%. The results in table are from frame-CE training.*

Scale α	0.5	1.0	1.5	2.0	**2.5**	3.0
WER	59.4	59.0	58.9	58.8	**58.7**	59.2

The results

As can be seen in table 7.3, the oracle confidences bring nice improvements. The first two lines in the table show the performance of the baseline CE-DNN system and the sMBR-DNN, which is our seed system. The next two lines compare the systems with no untranscribed data added, or the case when we simply add all the untranscribed data without using any

confidences. The system 'No untranscribed' is better than 'Baseline system', because we replaced the training labels from GMM-alignment with a 'Seed system' alignment. The group of four lines with the proposed *oracle confidences* of different granularity is in the middle. The last line in the table is the best possible oracle, where we trained with the correct transcripts of the 'untranscribed' data.

Table 7.3: *DNN self-training with oracle confidences (optimizing frame Cross-Entropy)*

	WER	WER recovery
Basline system (CE-DNN)	61.7	-
Seed system (with sMBR)	59.6	-
No untranscribed	60.8	0
All untranscribed, no confidence	60.1	8
Oracle confidences:		
Per-sentence confidence	58.7	25
Per-word confidence	57.7	36
Per-phone confidence	56.1	55
Tied-state confidence	55.3	64
With correct transcripts	52.3	100

From the results in table 7.3 we see that:

1. all the oracle confidences are better than using no confidence
2. the results improve as the confidence unit becomes smaller (sentence → word → frame)

This oracle experiment suggests that the tied-state confidences are the most promising. However, the method did not consider the actual confidence values that will be available. In the real world scenario we do not know which frames are correct. Inevitably, the training set will contain some wrongly labeled frames, while we will miss some of the correct frames. The actual results will be certainly worse than in this oracle experiment.

Also, we have to perform the real experiments with the sentence-level and word-level confidences. Only a careful comparison across techniques can verify if the frame confidences are the best.

7.1.1 Analyzing tied-state frequencies in the best path of lattice

After realizing that frame confidences are promising, we compare the label frequencies in the data that are a) automatically transcribed (best-path of lattices) and b) manually transcribed (forced-alignment to transcripts).

Figure 7.1 shows the log-scale frame counts of tied-states. The thick black curve corresponds to tied-state counts in the alignment. The green points around are the counts of the matching tied-states in the best-path from lattice.

We see that the frequencies are well matched for the more represented tied-states ($> 10^3$ frames), while the point cloud deviates from the black curve for the low-represented tied-states ($< 10^3$ frames). The extra frames, attributed to the low-represented states seem to be

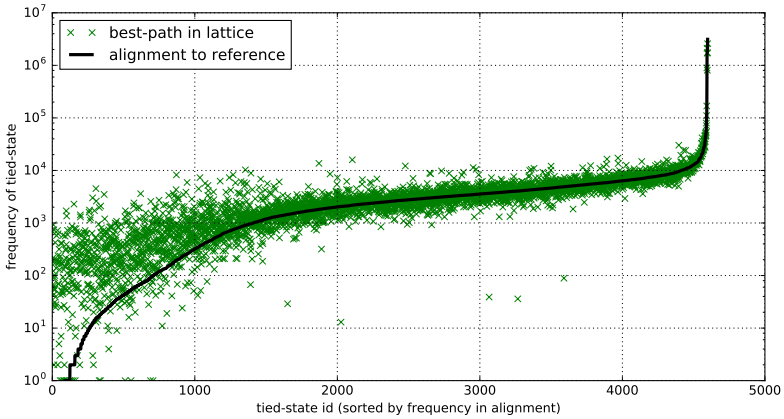


Figure 7.1: *Tied-state frequencies in Babel Vietnamese, comparison of the alignment and the best-path in lattice. (The WER of the seed system was 59.6%.)*

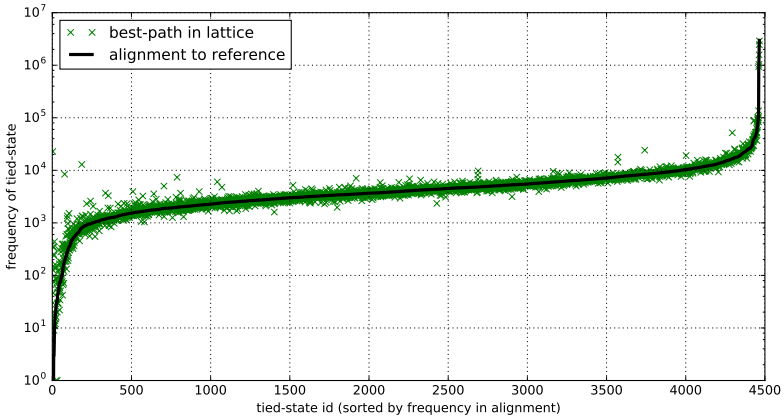


Figure 7.2: *Tied-state frequencies in Switchboard, comparison of alignment and best-path in lattice. (The WER of the seed system was 26.9%.)*

taken from the silence states as there are not many other states with a significant decrease of the frame count.

We also generated the same plot for the Switchboard setup (figure 7.2). Here, the cloud of green points closely follows the black curve, regardless of the tied state frequency. There certainly is a difference between the two databases. We will continue working with the Vietnamese setup, which is more challenging (the WERs are higher). There is more space for improvement and it is more likely that the good techniques will generalize to easier databases than the other way round.

7.2 Per-sentence confidences

In this section we will experiment with the per-sentence confidences based on the MBR statistics or based on the NN-posteriors. We will address data selection, weighted training and confidence calibration. Finally we show that it is beneficial to further re-train the acoustic model with the correctly transcribed data.

7.2.1 Minimum-Bayes risk confidence

As mentioned earlier in section 5.2.1, the sentence-level confidence c_{sent} is calculated as the average word confidence within a sentence: $c_{sent} = \frac{1}{M}\sum_{i=1}^{M} c_{\bar{w}_q}$. The per-word probabilities $c_{\bar{w}_q}$ are the posteriors $\gamma(q, \bar{w}_q)$ from the Minimum Bayes Risk decoding [Xu et al., 2011, section 7.1], where we fixed the word sequence $\bar{W} = (\bar{w}_1, \bar{w}_2, \ldots, \bar{w}_T)$ to the best path in lattice $\bar{W} = \text{wrds}(\bar{S})$. The confidences $c_{\bar{w}_q}$ were extracted with the scales that were used to generate the lattices (i.e. the acoustic scale $\kappa = 0.1$, graph scale $\varrho = 1.0$).

Data-selection, no weighting

The simplest experiment, which can be done with the per-sentence confidences, is the data-selection, in which we gradually add sentences ranked by the confidence.

Table 7.4: *Data selection by per-sentence confidence.*

Added sentences	0%	30%	**50%**	**70%**	90%	100%	Oracle
WER	60.9	60.1	**59.8**	**59.8**	60.0	60.1	58.7

From table 7.4, we see that it is good to leave out 30-50% of sentences, which brings a 0.3% WER improvement compared to adding 100% sentences. The oracle WER 58.7 achieved by weighting the sentences with their re-scaled true word accuracy indicates that there is a space for further improvement.

Weighting the sentences by confidence

To make our setup more similar to the oracle experiment, we should weight the sentences in the SGD training. Before doing it, it is good to show how well the confidence matches the word accuracy.

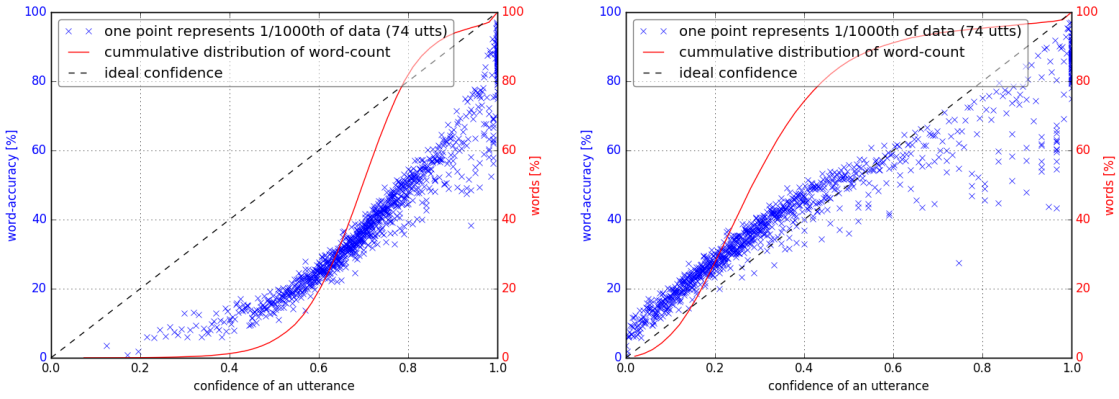
The scatter plot of confidence and accuracy in figure 7.3a revealed that the confidence is more optimistic than the actual word-accuracy, as the blue cloud is far beneath the dashed line representing the ideal match. The steep section of red curve indicates that majority of words comes from sentences with confidence ranging from 0.6 to 0.8.

If the confidence gets closer to the word accuracy, we should also get closer to the oracle performance. For a better match, we can warp the confidence by an exponential scale α as follows: $c'_{sent} = c^{\alpha}_{sent}$. By a grid search over α in table 7.5, we found the best $\alpha = 3.5$. From the results, we see that the distance to Oracle shrank from 1.1% to 0.6% WER.

Using the tuned $\alpha = 3.5$, we regenerated the confidence-accuracy scatter plot. In figure 7.3b, we clearly see that large part of mismatch is removed, as the blue cloud nearly overlaps with the dashed line.

Also the red curve indicates that a majority of words now comes from sentences with confidence below 0.4, the sentence weights became smaller. In other words, the scaling by α can have a similar effect as the repetition of the transcribed data in table 6.4: in the first case we decreased the weight of the untranscribed data, while in the second case the weight of transcribed data is increased by repeating them. As the repetition was helpful in section 6.1 and unhelpful in section 6.2, we no longer consider using it, which simplifies the recipe.

We also combined the weighted-training with data-selection, the grid search of scale α was repeated while we added the top 90% 70% and 50% of the automatically transcribed



(a) Scale $\alpha = 1.0$ (b) Scale $\alpha = 3.5$

Figure 7.3: *Sentence-confidence from MBR statistics for weighted training.*

Table 7.5: *Weighted training with sentence-confidence from MBR statistics (scaled exponentially by α, keeping all the sentences)*

Scale α	1.0	2.0	2.5	3.0	**3.5**	4.0	Oracle
WER	59.8	59.6	59.6	59.5	**59.3**	59.5	58.7

sentences. In table 7.6, we found a further drop by 0.1% WER, in case of adding 90% with confidence scale $\alpha = 4.0$.

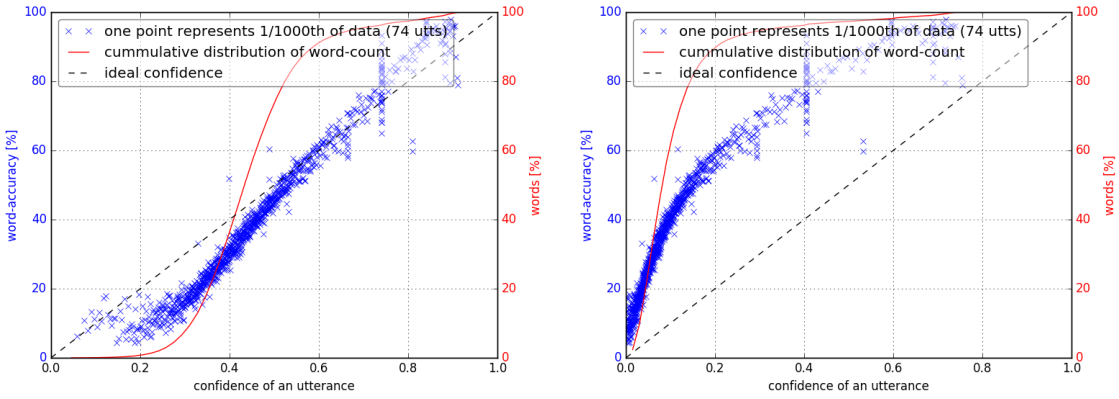
7.2.2 Calibration of confidences

A common approach to calibrate confidences is to train a logistic regression on the annotated development data, which are disjoint both from the training set of ASR seed-model and the untranscribed data [Yu et al., 2011]. In this section we verify if such calibration of the per-word MBR statistics can improve the efficiency of semi-supervised training with weighted sentences.

Training the logistic regression The training targets are obtained from the Levenshtein alignment that is done as part of the scoring. The logistic regression is trained for a two-class problem with label 1 for correct words and 0 for incorrect words. The model is trained globally on all the words in the hypothesis for the development data (9.8 hours for Babel Vietnamese). The model contains only a small number of trainable parameters (934 parameters), it was trained by L-BFGS algorithm.

Table 7.6: *Weighted training with sentence-confidence from MBR statistics (scaled exponentially by α, adding portion of top N% of untranscribed sentences)*

Scale α	1.0	2.0	2.5	3.0	3.5	**4.0**	4.5	Added sentences
	59.8	59.6	59.6	59.4	59.3	**59.2**	59.3	**90%**
WER	59.6	59.5	59.5	59.4	59.5	**59.2**	59.3	**70%**
	59.6	59.6	59.5	59.6	59.5	59.5	59.5	50%



(a) Scale $\alpha = 1.0$ (b) Scale $\alpha = 3.0$

Figure 7.4: *Sentence-confidence from calibrated MBR statistics (per-word).*

Input features of the logistic regression As the input features, we used:

1. logit of the MBR posterior $c_{\bar{w}_q}$ of the word $\bar{w}_q$ (the 'raw' MBR statistics, which reside in the interval $(0, 1)$),
2. logarithm of the number of phonemes in the word $\bar{w}_q$,
3. unigram log-probability of the word $\bar{w}_q$ from the language model,
4. indicator of word identity with 1-of-K encoding; words with >20 examples have own classes, the less frequent words are pooled. This allows the model to have a word-specific bias, a majority of trainable parameters is here.

From the re-calibrated MBR statistics, the per-sentence confidence is again the arithmetic mean of per-word confidences: $c_{sent} = \frac{1}{M}\sum_{i=1}^{M} c_{\bar{w}_q,cal}$. Now, our confidence corresponds to the word-accuracy in the sentence, see verification in figure 7.4a. Still, we did a grid search over the exponential scales α applied as $c'_{sent} = c^{\alpha}_{sent}$, and surprisingly the results in table 7.7 are favoring by 0.2% WER the scales around $\alpha = 3.0$, for which the confidence scatter is shown in figure 7.4b. Perhaps, the cause is the imperfect calibration in figure 7.4a, where the lower left tail of the cloud is deviating from the dashed line representing the ideally calibrated confidence. An alternative hypothesis is that $\alpha = 3.0$ is preferred because it makes the confidences generally smaller as indicated by the red curve in figure 7.4b.

As we achieved the same best performance in tables 7.5 and 7.7, we can conclude that the *calibration did not lead to a performance improvement* in the semi-supervised training with the weighted sentences.

Table 7.7: *Weighted training with per-sentence confidence from the calibrated word-confidences (derived from 'raw' MBR statistics, scaled exponentially by α, keeping all the sentences)*

Scale α	1.0	1.5	2.0	2.5	**3.0**	3.5	4.0	4.5	Oracle
WER	59.5	59.5	59.4	59.3	**59.3**	59.3	59.4	59.6	58.7

7.2.3 NN posterior-based confidence

For comparison, we experimented also with the best confidence from [Zhang et al., 2014a]: $C_{pos}(u)$ based on neural network posteriors. The confidence is extracted in two stages, at first the word-confidences are calculated by accumulating the NN log-posteriors from the frames of the hypothesized word w_i, while the state-sequence of the word is used to select the NN-outputs $\bar{s}_t$:

$$C_{pos}(w_i, t_s, t_e) = \frac{1}{t_e - t_s + 1} \sum_{t=t_s}^{t_e} \log p(t, \bar{s}_t) \ , \tag{7.1}$$

second, the per-sentence confidence is obtained as the arithmetic mean of the word confidences:

$$C_{pos}(u) = \frac{1}{K} \sum_{i=1}^{K} C_{pos}(w_i, t_{s,i}, t_{e,i}) \tag{7.2}$$

For our experiments, we exponentiated the value as $\exp(C_{pos}(u))$, which is not in the original formulation. This converts the confidence from log-domain back to the interval $(0, 1)$. The order of sentences sorted by confidence remains the same.

In [Zhang et al., 2014a], the confidence was used to select the top 70% of the sentences for training the HMM-GMM system and its bottleneck feature extractor. In our scenario, we weight the sentences for training a DNN-HMM system, which worked for us better than the simple sentence selection.

Again, we searched for the optimal scaling of the sentence confidence by tuning the parameter α, applied as $C_{pos}(u)' = \exp(C_{pos}(u))^{\alpha}$. This time the optimal scale is $\alpha = 0.25$ as shown in table 7.8. The confidence scatter plots for the default scale 1.0 and for the tuned scale 0.25 are in figure 7.5. By comparing the best results from tables 7.8 and 7.5 we see that the weighted semi-supervised training with NN-posterior based confidence (59.4%) is a little worse than the sentence confidences from MBR-statistics (59.3%).

Table 7.8: *Training with NN-posterior based confidence.*

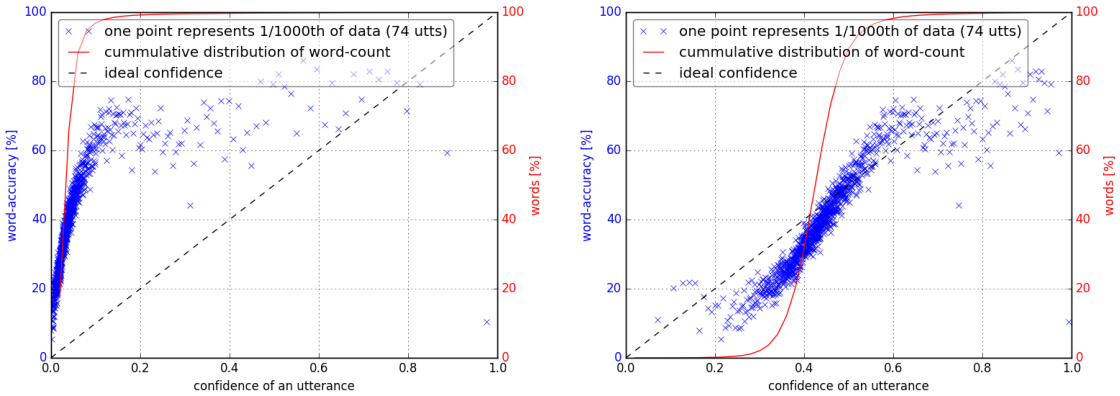
Scale α	1.0	0.5	0.33	**0.25**	0.20	Oracle
WER	59.7	59.8	59.7	**59.4**	59.7	58.7

7.2.4 Summary

In this section, we compared three types of per-sentence confidence based on 1) MBR-decoding statistics, 2) calibrated MBR decoding-statistics, 3) neural network posteriors. First, we found that weighted training leads to better results than simple data selection (WER% 59.8 → 59.3). However, for weighted training, we need a 'probabilistic' confidence, which we further tune by an exponential scale α, and this tuning is time consuming.

The best results for the three different confidence methods are summarized in table 7.9, where we see that the calibration of MBR statistics by logistic regression did not improve the system and the confidence from NN-posteriors [Zhang et al., 2014a] was a little worse than the MBR based confidences.

Apart from building the ASR models, we can compare the confidences by how well they select the sentences that contain less errors. The 3 methods are compared in figure 7.6, where we are adding sentences (horizontal axis) and observe the WER in the selection (vertical



(a) Scale $\alpha = 1.0$ (b) Scale $\alpha = 0.25$

Figure 7.5: *Sentence-confidence from NN-posteriors for weighted training.*

Table 7.9: *Weighted SGD training with various sentence confidences.*

Confidence method	WER
MBR statistics	59.3
Calibrated MBR statistics	59.3
NN-posteriors	59.4
Oracle	58.7

axis). The best selection is done with the blue curve, which shows that the calibration was done properly. The second best is the red curve, which represents the confidence from uncalibrated MBR statistics. The third is the green curve corresponding to the NN-posterior confidence. Note that the curves are invariant to the scale α, the shape depends only on the order of sentences.

It is slightly disturbing that although there clearly is a difference in the quality of the confidences (seen in figure 7.6), the results in table 7.9 revealed that all three methods lead to similar performance when applied as sentence weights in the semi-supervised training.

Finally, we achieved additional 0.1% WER improvement by a combination of data-selection (top 90% sentences) and data-weighting (by average MBR statistics, scaled by $\alpha = 4.0$), see table 7.6. We got close to the oracle performance (59.2 vs. 58.7 for oracle), while, in the oracle experiment, we weighted the sentences by their word-accuracy exponentiated by $\alpha = 2.5$. Still, the oracles with confidences for smaller units (words, frames) indicated even more promising results. We will explore them in the following sections.

7.2.5 Re-training with transcribed data

In literature, we can find that it is beneficial to post-process the model trained with the inter-mixed transcribed and untranscribed data. In [Thomas et al., 2013], the DNN output layer was discarded and trained again from random initialization, by using only the correctly transcribed data.

We also tried an alternative scenario in which we keep the output layer 'as-is' and continue training with the 10 hours of correctly transcribed data, while using a smaller

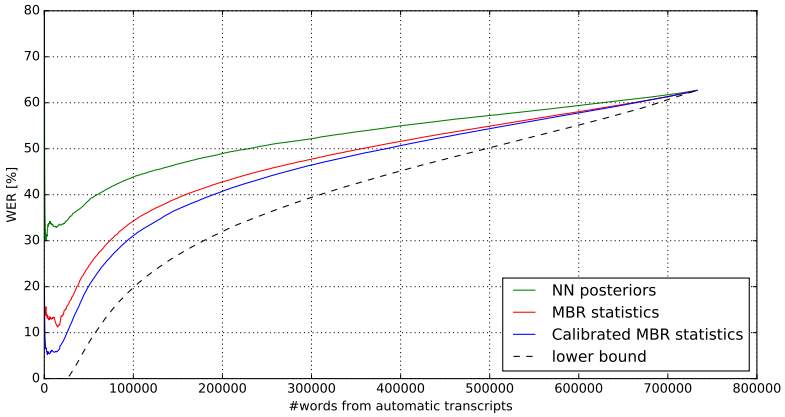


Figure 7.6: *WER as function of word-count of the selected data. The 'lower bound' curve is obtained by selecting sentences sorted by WER.*

learning rate (initial learning rate 0.001 instead of the original value 0.008). The model post-processing in this spirit was described in [Grézl and Karafiát, 2014] as 'fine-tuning'.

To verify that the per-sentence confidences are helpful, we post-process these two 'initial models':

1. no confidence: the initial model was built by adding all the automatically labeled sentences, no confidences are used

2. best per-sentence confidence: the initial model is built by training with weighted sentences ($\alpha = 4.0$), while 90% sentences are added (the best result from table 7.6).

The difference of these two scenarios is the improvement from using the per-sentence confidences. The results in table 7.10 show the post-processing by both the 'frame CE' training and the subsequent 'sMBR' training. Both is done with the 10 hours of correctly transcribed data.

If we focus on 'frame CE' numbers, we see that re-training with the correctly transcribed data improves the results in all the cases, while we were picking the best initial learning rate from 0.008, 0.001 and 0.0001.

Table 7.10: *Re-training the 'initial model' with 10 hours of correctly transcribed data.*

WER	no confidence (added 100% sentences)	best per-sentence confidence (added 90% sentences, $\alpha = 4.0$)	
initial model (semi-supervised)	**60.1**	**59.2**	frame CE
	58.3	57.6	sMBR
(A) discard last layer	58.7 (lrate 0.008)	58.7 (lrate 0.008)	frame CE
	57.5	57.6	sMBR
(B) re-train the initial model	58.7 (lrate 0.008)	**58.3 (lrate 0.001)**	frame CE
	57.6	**57.2**	sMBR

Then, if we (A) discard the output layer, there is no difference whether the initial model was built with confidences or not, see the 'frame CE' line.

If we (B) re-train the initial model, it is better to start from the model trained with the best per-sentence confidences, see the 'frame CE' line.

This improvement in (B) persists also after the sMBR training (last row in table 7.10, 3rd column). Based on this sMBR result, we can conclude that it is good to introduce the 'frame CE' re-training before the sMBR training. By skipping it, the performance would degrade by 0.4%, from 57.2 to 57.6.

Also, we can conclude that the use of per-sentence confidences is beneficial. The best result without confidences is 57.5, while with the confidences we obtained 57.2.

7.3 Per-word confidences

In this section, we will use the Minimum Bayes Risk statistics $c_{\bar{w}_q} = \gamma(q, \bar{w}_q)$ directly as the word-confidences. Primarily, we are interested to decide if the confidences are more efficient when used per-sentence or per-word. The literature presents both approaches (per-word [Wessel and Ney, 2005], per-sentence [Novotney et al., 2009]), but we have not seen a direct comparison. Per-sentence confidences are used more frequently, probably because of their simpler use. The oracle experiment in chapter 7.1 was favoring the per-word confidences over the per-sentence confidences.

The experiments will be conducted following a similar pattern as in the previous section 7.2. The behavior can be different if the processing decisions are done on the word-level basis.

7.3.1 Minimum Bayes Risk confidence

Again, we start from the non-calibrated MBR statistics; we'll use them for word-selection or word-weighting. Previously, with the per-sentence confidences, 'weighting' was better than 'selecting'.

Word-selection, no weighting

In this experiment, we are adding words into the *frame CE* training, starting from the highest confidence. The frames at the selected words have training weight 1, while the frames of rejected words have weight 0. The weights in eventual silence gaps between words are filled by linear interpolation.

In table 7.11, we tune the fraction of added words. We directly got to WER 59.1, which is a little better than the best performance achieved with per-sentence confidences (see table 7.6). This indicates that the per-word confidences are at least as good as the per-sentence ones.

Table 7.11: *Data selection by per-word confidence, Babel Vietnamese.*

	Added words [%]								Word	Seed
	0	20	30	**40**	50	60	70	100	oracle	system
WER	60.9	59.5	59.2	**59.1**	59.2	59.3	59.6	60.1	57.7	59.6

Previously, in table 7.6, we used training with weighted sentences, while table 7.11 was prepared with simple hard-selection of words. It is also remarkable that the optimal amount of added words seem to coincide with the word-accuracy of the seed system, which is $(100 - 59.6 = 40.4)$. We will re-visit this later in chapter 8 with a different experimental setup.

Weighting words by confidence

Next, we replace the data selection with weighted SGD training. The confidence of a word is used to scale the gradients over its time-span. Again, the silences are bridged by linear interpolation of the confidences from the adjacent words. We extend the *frame CE* training set with all the untranscribed data, while we tune the exponential scale α applied to word-confidences: $c'_{\bar{w}_q} = (c_{\bar{w}_q})^{\alpha}$. The 'raw' word confidences were generated with the lattice-scale $\lambda = 1.0$ (i.e. with default scaling from lattice generation by $\kappa = 0.1$, $\varrho = 1.0$).

Table 7.12: *Weighted training with MBR statistics (uncalibrated per-word confidence, scaled exponentially by α, all words were added)*

Scale α	1.0	2.0	3.0	4.0	5.0	6.0	7.0	Oracle
WER	59.5	59.2	59.2	59.1	59.0	59.0	59.1	57.7

Scale α	8.0	9.0	10.0	11.0	**12.0**	13.0	14.0	Oracle
WER	58.9	59.0	59.0	58.9	**58.8**	58.9	59.0	57.7

In table 7.12, we see that the best alpha 12.0 leads to WER 58.8%, which is by 0.3% better than we had with the word-selection in table 7.11. The WERs seem to fluctuate between 59.1 and 58.8. The best exponent value 12.0 is relatively high, we can see it as a soft version of data selection. For example, the words with confidence 0.68 get a training weight $c_{\bar{w}_q} = 0.68^{12.0} \doteq 0.01$, which in our case almost removed 43% words from the training. Another evidence which supports this view is the experiment in which we combined the word-selection and word-weighting. In table, 7.13 we see that the $\alpha = 12.0$ is no longer the best value, after we remove some of the words. We also see that there is no gain from introducing the word-selection in combination with the word-weighting, which would happen if the weighting already 'contained' the selection.

Table 7.13: *Weighted training with MBR statistics (uncalibrated per-word confidence, scaled exponentially by α, adding portion of top N% words)*

Scale α	1.0	2.0	3.0	3.5	4.0	4.5	**6.0**	8.0	**12.0**	Added words
WER	59.5	59.2	59.2	59.1	59.1	59.1	59.0	58.9	**58.8**	**100%**
	59.3	59.2	59.1	59.0	59.1	59.0	**58.8**	59.0	59.0	**70%**
	59.1	59.0	59.0	59.1	58.9	59.0	59.0	58.9	59.1	50%
	59.1	59.0	59.1	59.0	59.1	59.1	59.1	59.0	59.1	40%

7.3.2 Weighting words by calibrated confidence

In section 7.2.2, we calibrated the MBR statistics by logistic regression. We re-used the same calibrated per-word confidences for experiment in table 7.14. The best alphas 2.0, 3.0

lead to WER 58.8. Here also, the calibration of confidence did not bring a performance improvement (same best result in tables 7.12 and 7.14).

Table 7.14: *Weighted training with calibrated word-confidence, (calibration by logistic regression, exponential scaling by α, keeping all the words)*

Scale α	0.5	1.0	1.6	**2.0**	2.5	**3.0**	4.0	5.0	6.0	Oracle
WER	59.6	59.4	59.1	**58.8**	59.0	**58.8**	58.9	59.1	59.3	57.7

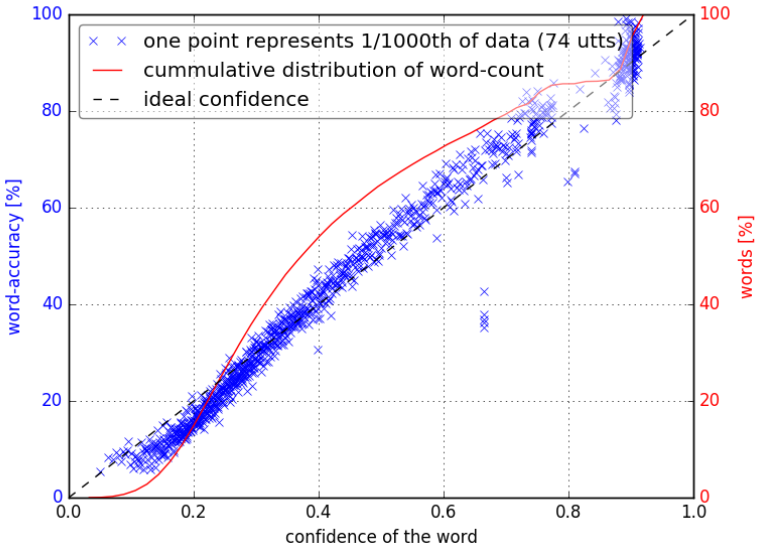


Figure 7.7: *Calibrated word-confidences, each blue point represents 750 words of similar confidence (in previous plots from section 7.2 the points were groups of sentences). The point cloud reasonably copies the ideal confidence (dashed black line). The cumulative distribution (red curve) shows that the distribution of confidences is not too far from uniform.*

However, in figure 7.7, we see that the calibration is done properly, as the confidences nicely match with the word accuracy.

7.3.3 What happens in data selection?

To demonstrate that the data selection done per-word is better than per-sentence, we created figure 7.8, where we compare the WER as function of number of selected words. We see that by selecting the individual words, we can create a subset with lower WER (blue curve) than if we select whole sentences (red curve). With the 'un-weighted' data selection, the WER was the following: 59.8 for per-sentence confidences (table 7.4), 59.1 for per-word confidence (table 7.11). With the weighted training, the difference shrank from 0.7 to 0.4, as we had 59.2 with per-sentence confidences and 58.8 with per-word confidences.

7.3.4 Re-training with transcribed data

Similarly to the previous section, we take the best *'initial model'* and post-process it by re-training with the 10 hour set of the correctly transcribed data. As we found earlier, we

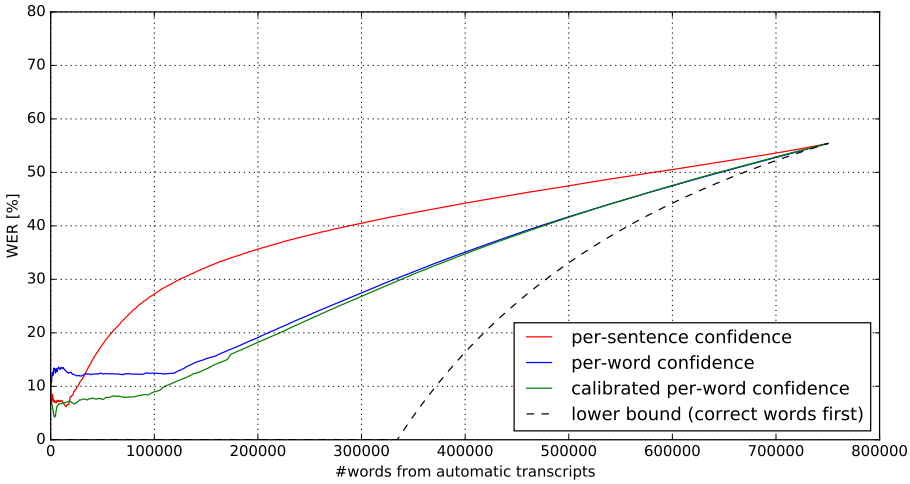


Figure 7.8: *WER in subset of data selected by the confidence, starting from the highest. The per-word confidences (blue curve) allow us to select data with lower WER, than the per-sentence confidences (red curve). The calibration by logistic regression further improves the data selection (green curve). However, the better calibration did not translate into improvement of the semi-supervised training. All the confidences are based on Minimum Bayes Risk decoding statistics. The WER on this plot is without deletions and we normalized by the length of the hypothesis, as we cannot select a 'deleted' word from the automatic transcripts.*

first re-train by 'frame CE' training and continue with 'sMBR' training. As a sanity check we also compare with the sMBR done directly from the initial model.

The results in table 7.15 show that the final sMBR WER 56.9 is by 0.3% better than we previously obtained with the per-sentence confidences. This makes the per-word confidences better than the per-sentence confidences. We believe that this explicit comparison has not been shown in any study yet.

Table 7.15: *Re-training the best 'initial models' with 10 hours of correctly transcribed data. The 'initial models' are obtained by three variants of semi-supervised training: a) no confidences at all, b) best per-sentence confidences and c) best setup with per-word confidences (weighted training with uncalibrated per-word confidences, exponential scale $\alpha = 12.0$, added all the words.*

WER	a) no confidence	b) best sentence confidence	c) best word confidence	
initial model	**60.1**	**59.2**	**58.8**	frame CE
(semi-supervised)	58.3	57.6	57.4	sMBR
re-train the initial CE model	58.7 (lrate 0.008)	58.3 (lrate 0.001)	**58.2** (lrate 0.001)	frame CE
	57.6	57.2	**56.9**	sMBR

7.4 Tied-state confidences

In this section, we return to the frame confidences $c_{\bar{s}_t} = \gamma(t, \bar{s}_t)$, which we previously introduced on page 51 and explored in sections 6.1 and 6.2. In section 6.2, we have shown that the lattice posteriors for states from best path can be used as frame confidence in weighted mini-batch SGD training. We observed that it is beneficial to tune the 'lattice-scale' λ by rescaling both the acoustic and graph scores in the lattice, which shifts the confidences closer to the ideal confidence (probability that the label is correct). In this section, we will extend this approach in two directions:

- we further introduce the exponential scale α that is applied to the extracted frame confidences (i.e. the frame posterior from lattice at states on the best path). The ideal confidence may not be the best weight for the SGD training, so we 'bend' the distribution of the confidences by exponential scaling $c'_{\bar{s}_t} = (c_{\bar{s}_t})^{\alpha}$.
- we introduce the per-pdf calibration, assuming that the confidences behave differently across tied-states. The calibration helps to select data-points with less annotation errors than if we used the global calibration.

In this section we want to see if the tied-state confidences lead to better results than we obtained with the per-word confidences.

7.4.1 Tuning confidence scale α in frame-weighted SGD training

Table 7.16: *Weighted training with per-frame confidences (the tied-state posteriors from lattice for the state on the best path). The confidences were extracted with the lattice scales* $\lambda = \{1.0, 0.3, 0.02\}$ *(applied both to acoustic and graph scores in the lattice). After the extraction, the frame confidences were re-scaled by exponential scale* α*.*

Scale α	1.0	2.0	3.0	4.0	5.0	6.0	Lattice scale λ
	59.4	59.1	59.0	59.1	**58.9**	59.0	1.0
WER	59.2	**59.0**	**59.0**	59.2	59.3	59.3	0.3
	59.1	59.1	59.2	59.4	59.4	59.5	0.02

In table 7.16, we see that for any lattice scale, we can always tune such alpha for which the WER becomes similar regardless of the initial lattice scale. We decided to fix the lattice-scale to $\lambda = 0.3$ for further experiments, as it reasonably pre-calibrates the confidences and the scale is not too far from $\lambda = 1.0$ (see red curve in figure 6.1 on page 64).

7.4.2 Per-phoneme analysis of the frame confidences

So far, we have been analyzing the confidences for the whole untranscribed data-set. To get an idea how the confidences differ for individual phonemes, we show them separately in figure 7.9 (frame-accuracy as function of confidence).

In the graphs, we see that the curves of phonemes with similar frame counts look similar. The curves of highly represented phonemes are smooth, while the curves for low-represented phonemes are noisy. Then we see that the curves of the 5 most represented phonemes (vowels *a, e, i, o* and *n*) begin with higher frame-accuracies than the other phonemes. We also see that silence, which is the most represented label, has both high confidence and high accuracy.

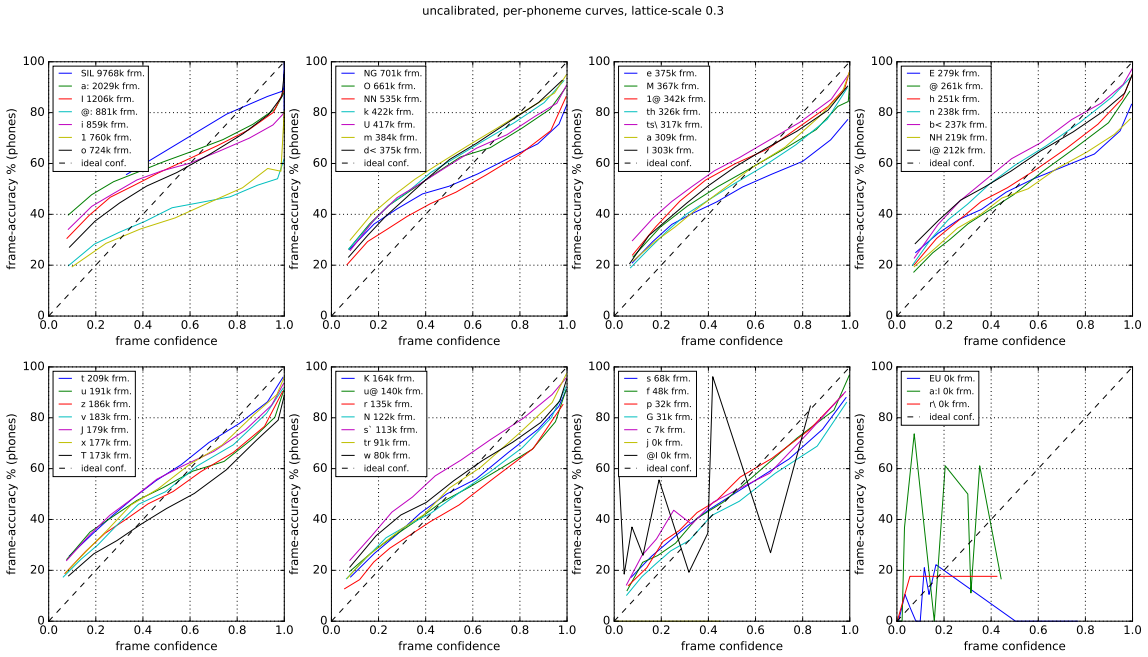


Figure 7.9: *Per-phoneme frame-accuracy as function of confidence (uncalibrated), from lattice-posteriors with scaling $\lambda = 0.3$. Here, a frame is considered correct when the phoneme label matches both in the alignment and the best-path from lattice. The curves are sorted by frame-count of phonemes, which are marked in the legend.*

This is understandable, it is harder to confuse the silence with a phone than to confuse two phones.

It would be good to use the plots for suggestions to the experiments, however it is not clear how exactly. We can confirm that there definitely are differences among the phoneme curves, these will be reduced by our calibration.

7.4.3 Calibration by logistic regression

Because we saw in figure 7.9 that there are differences among the accuracy curves of the individual phonemes, we can apply a pool of calibration models to reduce the differences. Each model is a simple logistic regression with 2-parameters (scale, bias) for each class of acoustic model (pdf, tied-state), the set of models is trained on the transcribed development set. Yes, we do not build a per-phoneme model, but a per-tied-state one (i.e. we have a model per neural network class). Note that the oracle results in table 7.3 were the best for the per-pdf confidences.

As we saw in figure 7.1, the pdf's are represented by various amounts of data. For some pdf's, there is too little data for reliable estimation of calibration model parameters. To solve this, we adopted a three-level hierarchy of models: a) per-pdf models, b) per-phoneme models and c) a global model. In case, there is not enough data for a given unit, we switch to higher-level model a) $\rightarrow$ b) $\rightarrow$ c). By observing their loss function values, we found that for a minimum of 100 data-points, the logistic regressions are already trained reasonably. The per-pdf models 'a)' were used for 91.5% frames, the per-phoneme models 'b)' for 8% frames, the the global model 'c)' for 0.5% frames. We started from the lattice posterior frame confidences with the lattice-scale $\lambda = 0.3$.

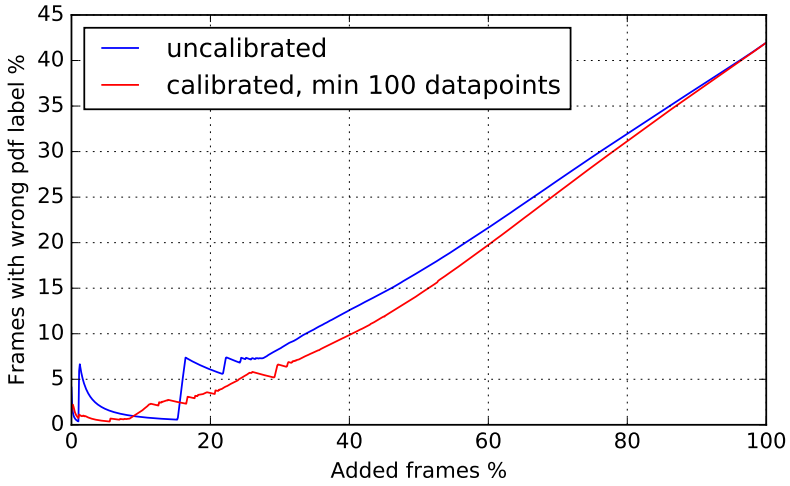


Figure 7.10: *Frame selection according to frame-confidence with or without the calibration (hierarchical three-level calibration). The minimum number of data-points for training a calibration model (logistic regression) was 100. We see that the calibrated confidences allow us to select frames with less errors (red curve) than without the calibration (blue curve). The benefit comes from training the calibration models per-pdf (tied-state), which changes the overall ranking of the frames.*

In figure 7.10, we see how our three-level calibration changes the gradual selection of frames, starting from the highest confidence. The gap between the curves is promising as it means that with the calibration, we can select data-points containing less wrong labels.

Frame selection experiments

In the first set of experiments, we perform the 'data-selection', starting from the highest confidence: the weights are either 0 (dropped frame) or 1 (selected frame).

Table 7.17: *Data selection by per-frame confidence, WER.*

Added frames:	0%	20%	*40%*	50%	60%	70%	80%	100%
Uncalibrated conf.	60.9	61.0	*60.1*	59.4	**59.1**	59.3	59.2	60.1
three-level calibration	60.9	60.3	*59.4*	59.1	58.9	**58.8**	59.0	60.1

In table 7.17, we see that the calibration helped a lot when selecting 40% of data, where the gap between curves in figure 7.10 was large. The best results are obtained when selecting 60% or 70% frames, where the confidence calibration brought a WER improvement of 0.3%.

Frame weighted experiments

In the next experiment, we use the calibrated confidences for frame-weighting:

In table 7.18 we obtained the best result so far for semi-supervised training with mixed data (transcribed and untranscribed).

The hierarchical three-level calibration improved the results by 0.3% WER. Previously, the calibration of the per-word confidences, was not bringing improvements (see sections 7.2.2 and 7.3.2). A detailed look at the calibration (frame-accuracy curves per phoneme,

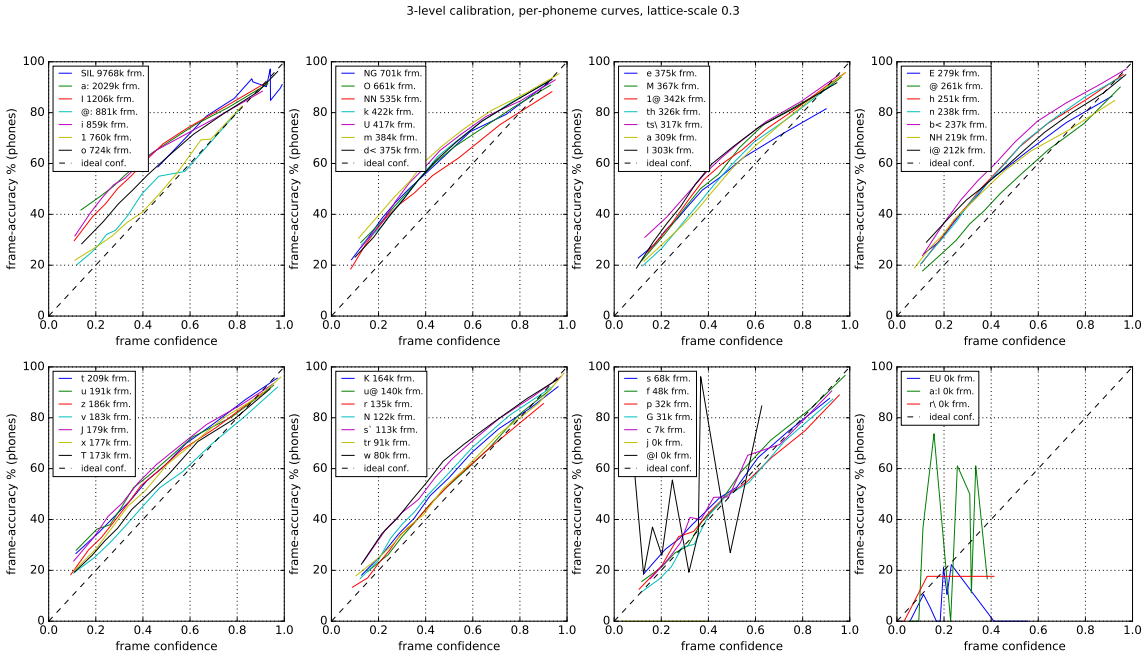


Figure 7.11: *Calibrated version of figure 7.9. Frame-confidences calibrated with three-levels of logistic regression models trained a) per-pdf, b) per-phone, c) global model. We see that the gaps between curves are smaller and many curves look very similar.*

Table 7.18: *Weighted training with calibrated per-frame confidences. The calibration is a three-level hierarchy of calibration models. The calibrated confidences are scaled exponentially by α.*

Scale α	0.5	1.0	1.5	**2.0**	**2.5**	3.0	4.0	5.0	6.0	Oracle
WER	59.6	59.1	59.0	**58.6**	**58.6**	59.0	59.0	59.2	59.4	55.3

figure 7.11) show that the calibration worked well as the curves are very similar (except the few under-represented phonemes at the bottom-right plots).

The performance obtained with the frame-weighted training is by 0.2% WER better than the frame-selection, both for the calibrated and uncalibrated confidences (see the tables 7.16 and 7.18).

Frame selection and weighting experiments

We also tried a combination of frame-selection and frame-weighting, however the results in table 7.19 show no performance improvement from dropping the frames with lowest confidence.

Table 7.19: *Training with both data-selection and data-weighting done per-frame. The calibration is a three-level hierarchy of calibration models. The calibrated confidences are scaled exponentially by α.*

Scale α	0.5	1.0	1.5	**2.0**	**2.5**	3.0	4.0	Selected frames
WER	59.6	59.1	59.0	**58.6**	**58.6**	59.0	59.0	100%
	58.8	58.8	**58.7**	58.8	58.8	58.9	58.8	80%

7.4.4 Re-training with transcribed data

As we did in the previous sections, we further re-train the *'initial model'* that was trained with a mix of transcribed and untranscribed data. The re-training is done with the 10-hour set of correctly transcribed data. We first re-train by 'frame CE' training to continue with 'sMBR' training. As a sanity check we also compare with sMBR done directly from the initial model.

Table 7.20: *Re-training the best 'initial model' with 10 hours of correctly transcribed data, after 'frame CE' training with the mixed data (transcribed and untranscribed). We compare four scenarios of semi-supervised training: a) no confidences at all, b) best per-sentence confidences, c) best setup with per-word confidences and d) best setup with per-frame confidences (frame confidences extracted with lattice-scale $\lambda = 0.3$, re-calibrated by three-levels of logistic regressions and post-processed by exponential scale $\alpha = 2.0$, weighted SGD training). The 'frame-CE' re-training is followed by 'sMBR'*

WER	a) no confidence	b) best sentence confidence	c) best word confidence	d) best frame confidence	
initial model	**60.1**	**59.2**	**58.8**	**58.6**	frame CE
(semi-supervised)	58.3	57.6	57.4	57.1	sMBR
re-train the initial	58.7	58.3	58.2	**58.0**	frame CE
CE model with	(lrate 0.008)	(lrate 0.001)	(lrate 0.001)	(lrate 0.001)	
true transcripts	57.6	57.2	**56.9**	57.0	sMBR

The re-training result in table 7.20 for 'frame CE' objective shows that the calibrated per-frame confidences are the best, with WER of 58.0. From the sMBR results, we see that the best model remained the word-confidence DNN with WER 56.9. The sMBR results from the 'best frame confidence' column reveal that, now, there is almost no difference between the sMBR training from the 'initial model' 57.1 and the re-trained model 57.0, which we did not see in other columns.

7.5 Summary

The observations from this chapter, which was focused on granularity of confidences, can be summarized as follows:

- We found it important to **re-tune the model with a small amount of correctly transcribed data**. The re-tuning is done with smaller learning rate 0.001 (table 7.10, otherwise the initial learning rate is 0.008).

- Also, it is beneficial to **go beyond simple sentence selection**, which can often be seen in the literature (see table 7.21). The very simple, but still powerful approach is to select frames corresponding to the N% words with the best confidence.

- If we compare the results in 'data-selection' table 7.21 with the results in the first column from the 'data-weighting' table 7.22, we see that **'data-weighting' leads to better results than 'data-selection'**. Along the way, we also noticed that data **'weighting' and 'selection' are not complementary**. However, the **data-selection** is more **straightforward to tune-up**, as we can use 'raw' confidences in it.

- For getting better results, we had to **introduce a hyper-parameter that is expensive to tune**: the N% amount of added data for data selection, or the exponential scale α for approximate calibration of 'raw' confidences in role of frame weights.

- For 'proper' confidence calibration with logistic regression, we introduced dependency on the correctly transcribed development set. In some cases, there was no improvement from the confidence calibration (per-sentence, per-word confidences). With the per-frame confidences, we obtained 0.3% WER improvement, this was however absorbed by the re-tuning with frame-CE training and then by sMBR training. Based on this evidence, we can form a conclusion that the **confidence calibration is not necessary for semi-supervised training**.

- Despite the initial optimism from the oracle results (table 7.3), where the tied-state confidence oracle dropped deeply to 55.3% (but this oracle completely ignored the confidence values, as it just selected the frames with correct labels), the final gains from semi-supervised training were more modest, leading to 58.0% WER after re-training with the correctly transcribed data. The final best WER-recovery was 33%.

- In the end, there were **small differences between using the per-sentence, per-word or per-frame confidences**, especially after re-tuning with the 'frame CE' and 'sMBR' objectives using the correctly transcribed data. Still, it is **clearly beneficial to use the confidences in the semi-supervised training** (see the last line in table 7.22).

Table 7.21: *Data selection, summary. Each time we added the optimal amount of data, the amount of data is the only hyper-parameter we needed to tune manually. The two results for the 'frame-selection' refer to 'uncalibrated/calibrated' confidences.*

	WER	taken from
Sentence selection	59.8	table 7.4
Word selection	59.1	table 7.11
Frame selection	59.1/58.8	table 7.17

Table 7.22: *Data weighting, summary. Each time we re-scaled the confidences by tuning the exponential scale α. The frame confidences were calibrated, the sentence and word confidences were better without calibration. The results are from table 7.20.*

	WER	(re-tuned)	(re-tuned + sMBR)
Sentence weighting	59.2	58.3	57.2
Word weighting	58.8	58.2	**56.9**
Frame weighting	**58.6**	**58.0**	57.0
no confidence	60.1	58.7	57.6

In our case, the results for word-confidences and frame-confidences are very similar. Hence, to decide which approach is the best, we will use the Occam's razor. The setup with per-word confidences is simpler as it requires less storage for confidence values. The system without 'proper' calibration of confidences does not depend on development set. The only hyper-parameter we need to tune is the exponential scale α for weighted training or the N% of added words for data-selection.

Chapter 8

Finding generic semi-supervised training approach

Ideally, we are interested in finding such semi-supervised training recipe, that will be efficient for a broad range of scenarios. Until now, we explored the behavior for Babel languages (mainly Vietnamese) and one scenario (10 hours are transcribed, ≈70 hours are untranscribed).

Actually, in chapter 6.2, we evaluated the self-training for 5 languages (table 6.11). But all the 5 languages were from the second year of Babel program, and the word error rates of all the seed systems were around 60% (between 53.2% and 67.4%).

Therefore, we chose Switchboard (described in section 4.3), to test the techniques in a very different setup. As the first step, we applied the procedure from section 6.2 (weighted training with frame-posteriors $c_{\bar{s}_t}$ extracted with lattice-scale $\lambda = 0.02$) and we obtained a disappointing result. In table 8.1 we see that, for Switchboard, we get the same WER 24.8 both with and without using the confidences.

Table 8.1: *Weighting by approximately calibrated frame confidences did not generalize to our custom Switchboard English setup, the result is the same as if we add all the data and use no confidence. The frame confidences $c_{\bar{s}_t}$ were generated with lattice-scale $\lambda = 0.02$.*

WER	Vietnamese	Bengali	Switchboard
The seed system	59.6	62.9	26.9
added 0% data	60.8	64.2	28.0
added 100%, no confidences	60.1	63.2	**24.8**
weighting with frame confidence (scale 0.02)	58.9	62.3	**24.8**
Oracle1 (adding the correctly decoded words)	57.7	-	23.5
Oracle2 (use of true transcripts)	52.3	58.4	22.0

Of course we can re-tune the lattice-scale λ or the exponential scale α. However, if we are searching for a universal recipe without a computationally expensive hyper-parameter tuning, we need to look for a different approach. In section 7.3.1, we saw that the best percentage of added words seems to correspond to the word accuracy of the seed system (in table 7.11 it was good to add 40% words, while the WER of the seed system was 59.6%, i.e. the word accuracy was 40.4%).

A closer look on word-selection is in figure 8.1. We split the automatically transcribed Vietnamese words into 10 bins with same amount of words, while the words with similar

confidence are in the same bin. We see that in the first 4 bins there are more correct words than wrong ones, while by adding the fifth and next bins, we would introduce more incorrectly labeled words than correct ones. In the selection of 40% words, there is 27% WER as can be read from blue curve in figure 7.8 on page 81.

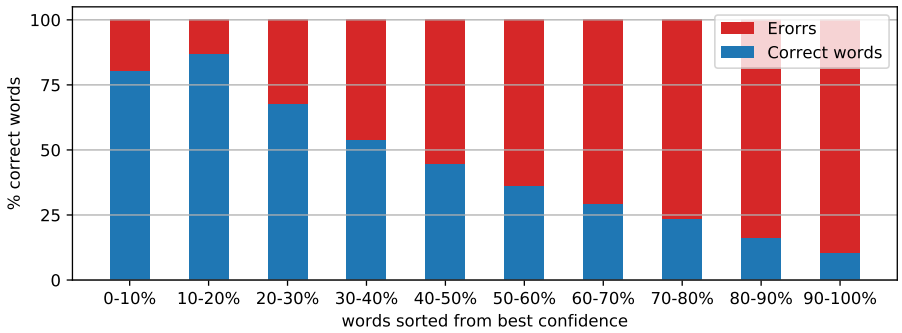


Figure 8.1: *Automatic transcripts of Vietnamese sorted from best word-confidence, split into 10 bins with same word-count. See the correspondence of the bars with results in table 7.11 on page 78 and blue curve in figure 7.8 on page 81.*

Now, let's check if our *word selection rule* based on word accuracy W_{acc} of seed system generalizes to other databases. From results of Babel Bengali in table 8.2 and Switchboard in table 8.3 we see that the same rule holds.

Table 8.2: *Data selection by per-word confidence, babel Bengali.*

Added words	0%	20%	30%	**40%**	**50%**	60%	70%	100%	**Seed $\mathbf{W_{acc}}$**
WER	64.2	62.9	62.5	**62.3**	**62.3**	62.4	62.5	63.2	**37.1**

Table 8.3: *Data selection by per-word confidence. Our Switchboard setup has 14 hours transcribed, 95 hours are untranscribed. The LM is trained on Fisher transcripts. The results are for HUB5-2000 (Switchboard + CallHome), further description of the setup is in section 4.3.*

Added words	0%	50%	60%	**70%**	80%	90%	100%	**Seed $\mathbf{W_{acc}}$**
hub5 WER	28.0	25.3	25.1	**24.4**	24.7	24.5	24.8	**73.1**

For example, for Switchboard in table 8.3, the W_{acc} of the seed system was 73.1%, while the optimal amount of added words was 70%, so the word accuracy of the seed system was the right amount of words to add. Such simple rule works reasonably well for such different setups as the Babel Vietnamese, Babel Bengali or even for Switchboard data.

The bar-graph showing word-selection for Switchboard is in figure 8.2. We see that there is certain amount of incorrect words with high confidence, while the biggest portion of wrong words is located in the last three bins. The selection of top 70% words has a WER of 7.4%.

The only external information we needed for our *word selection rule* is the WER of the seed system calculated on some development set, while we assume that the untranscribed data and development data are similar. We are aware that it is not rigorously 'guaranteed'

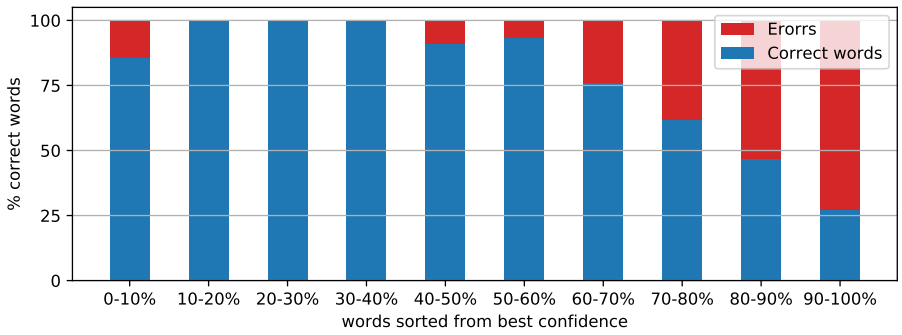


Figure 8.2: *Automatic transcripts of Switchboard sorted from best word-confidence, split into 10 bins with same word-count. See the correspondence of the bars with results in table 8.3.*

that it will always lead to best possible results, in the same time, it will often be a good fit for its simplicity.

8.1 Repeating experiments, Switchboard

In this section, we repeat some of the previously explored methods, this time applied on Switchboard task.

8.1.1 Word-weighted training

At first, we try to replace the word-selection with word-weighting, to see if we can achieve better performance. The word confidences are the uncalibrated MBR statistics.

Table 8.4: *WER from weighted training with MBR statistics (uncalibrated per-word confidence, lattice scales $\lambda = \{1.0, 0.3\}$, re-scaled exponentially by α. We added all untranscribed data.*

lattice-scale λ	Scale α 0.5	0.7	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	Oracle
1.0	-	24.7	24.6	24.6	24.5	24.5	24.5	24.5	**24.4**	24.6	23.5
0.3	24.7	**24.6**	24.7	24.8	24.9	25.1	-	-	-	-	

In table 8.4, we see that here, the *word-weighting* is equally as good as *word-selection*. It also seems to be important to use the lattice-scale $\lambda = 1.0$, as the lattice-scale $\lambda = 0.3$ was responsible for 0.2% worse result. In summary, for Switchboard, the word-weighting was not better than the word-selection, which is preferable as it is simpler.

8.1.2 Calibrated frame-weights

Another method we replicate is the three-level calibration of frame-confidences $c_{\bar{s}_t}$. This time, we used the 4 hour held-out set `train_dev` for training the calibration model.

The results in 8.5 show that it is better to calibrate the posteriors generated with lattice-scale $\lambda = 1.0$. However, the per-pdf calibration did not bring a performance improvement compared to the simple word-selection (see best results in tables 8.3 and 8.5).

With the minimum number of 100 frames for training the per-pdf logistic regression, 50% of frames were calibrated by per-pdf models and the other 50% by the per-phoneme calibration models. By further reducing from 100 to 50 the lowest frame-number for training a per-pdf logistic regression, the results became worse.

Table 8.5: *Weighted training with calibrated per-frame confidences. The calibration is a three-level hierarchy of calibration models. The calibrated confidences are scaled exponentially by α.*

lattice-scale λ	scale α 0.5	1.0	1.5	2.0	2.5	3.0	4.0	min-frame
1.0	24.5	24.6	**24.4**	24.5	24.7	24.7	24.8	100
0.3	24.7	**24.5**	24.7	28.8	24.9	25.0	25.3	100
0.3	24.8	24.8	24.8	**24.7**	24.9	25.0	25.2	50

Again, the *simple word-selection* method is preferable. It leads to the same result, and it does not rely on training a calibration model.

8.2 Re-tuning with correctly transcribed data, Switchboard

As, for Switchboard, we found word-selection to be as good as word-weighting, we take these two systems and proceed with re-tuning. For the final comparison, we re-tune the 'initial models' with the small 14hour set of the correctly transcribed data. We re-tune first by training with 'frame CE' objective and then with 'sMBR'. For comparison, we also run the sMBR training directly from the initial model.

Table 8.6: *Switchboard, re-training the 'initial models' with 14 hours of correctly transcribed data. The 'initial models' are obtained by three variants of semi-supervised training: a) no confidences at all, b) best training with weighted words ($\alpha = 7.0, \lambda = 1.0$) and c) best word selection (selected top 70% of words)*

WER, seed 26.9	a) no confidence	b) best word weighting	c) best word selection	
initial model (semi-supervised)	**24.8**	**24.4**	**24.4**	frame CE
	24.0	**23.6**	23.9	sMBR
re-train the initial CE model	24.3 (lrate 0.001)	**24.1** (lrate 0.001)	24.2 (lrate 0.001)	frame CE
	23.7	**23.5**	23.7	sMBR

For Switchboard (table 8.6), the final sMBR result for word-selection c) was 23.7, which is by 0.2% WER worse than with the word-weighting b). And the sMBR result of c) is the same as if no confidences were used in a). We see that the word-selection did not bring an improvement, while it also was not harmful. The WER recovery of the system c) is 63%, which is much higher than we had for Vietnamese (33%), which can be explained by lower WER in automatic transcripts from Switchboard seed system.

The same set of experiments done for Babel Vietnamese is in table 8.7; here the degradation between word-selection c) and the word-weighting b) is $57.1 - 56.9 = 0.2$, while the word-selection c) is better than the system without confidences a) by 0.5% WER.

Table 8.7: *Babel Vietnamese, re-training the 'initial models' with 10 hours of correctly transcribed data. The 'initial models' are obtained by three variants of semi-supervised training: a) no confidences at all, b) best per-word weighted training ($\alpha = 12.0, \lambda = 1.0$) and c) best word selection (selected 40% words)*

WER, seed 59.6	a) no confidence	b) best word weighting	c) best word selection	
initial model (semi-supervised)	**60.1**	**58.8**	**59.1**	frame CE
	58.3	**57.4**	57.6	sMBR
re-train the initial CE model	58.7 (lrate 0.008)	**58.2** (lrate 0.001)	58.4 (lrate 0.001)	frame CE
	57.6	**56.9**	57.1	sMBR

If the 'simple word-selection' causes either an improvement or no harm, it is still a preferable technique. It it is much faster than the careful tuning of the exponential scale α by a grid search of NN trainings, while such careful tuning brought only a small 0.2% WER improvement.

8.3 Final summary, simple word-selection

The final summary of the WER improvements we obtained with our preferred *simple word-selection* technique is in table 8.8. We see that the overall absolute WER improvement between the seed system and the final sMBR systems is 2.5% for Babel Vietnamese, 2.3% for Babel Bengali and 3.2% for Switchboard. For Bengali and Vietnamese the WER in the automatic transcripts was higher, so the absolute WER improvement from the semi-supervised training is smaller than in the case of Switchboard. At the same time, the use of confidences was more important for Vietnamese (with higher WER), than for Switchboard (with smaller WER).

Table 8.8: *Final WER performance of the semi-supervised training based on 'simple word-selection'. The initial model is trained with the mixed data (transcribed and untranscribed), the re-training is done with the smaller set of correctly transcribed data.*

WER	Vietnamese	Bengali	Switchboard
seed system (sMBR)	59.6	62.9	26.9
initial-model (mixed data)	59.1	62.3	24.4
+ re-trained (frame CE)	58.4	61.6	24.2
+ re-trained (sMBR)	57.1	60.6	23.7
abs. WER improvement	2.5	2.3	3.2

Chapter 9

Two system combination as the seed system

In several articles, good results are reported with a system combination as the seed system [Huang et al., 2013, Li et al., 2016] for the self-training. It is not difficult to try such approach with our 'simple word-selection' method. The recipe consists of the following steps:

1. System combination is done based on weighted union of lattices which is then decoded by Minimum-Bayes risk decoding to get the word-sequence $\bar{W} = (\bar{w}_1, \bar{w}_2, \ldots, \bar{w}_M)$ and per-word confidences $c_{\bar{w}_q} = \gamma(q, \bar{w}_q)$ (this time the word-sequence is updated, it is no longer the best path from the lattice).

2. We align the decoded word-sequence $\bar{W}$ to generate the targets for NN training (we update the word-boundaries for the words and their confidences).

3. We train with the mix of manually and automatically transcribed data, while adding N% of the most reliable words from the automatic transcripts.

We try this recipe with our custom Switchboard system described in section 4.3.

Combining DNN-sMBR and fMLLR-GMM as seed system

In the first experiment, we combine the DNN-sMBR system with the fMLLR-GMM system that we previously used to generate the fMLLR features. The optimal weights for combining lattices are 0.8 for the DNN and 0.2 for the GMM. The fusion improved the results from 26.9 to 26.6 (the WER of the fMLLR-GMM system was 35.3).

Table 9.1: *Fusion of the DNN and the GMM used as the seed system, we do the 'simple word-selection'. The performance is worse than we had with the single seed system.*

Added words:	60%	70%	80%	90%	100%
WER	25.2	25.2	24.7	24.8	25.0

From the results in table 9.1, we see that the self-training is worse than we previously obtained with the single DNN system (24.4%, in table 8.3). The secondary fMLLR-GMM system has however much worse performance than the primary DNN system. We need to try the fusion with other systems with better performance.

Combining DNN-sMBR and Tandem-GMM as seed system In the second experiment, we combine the DMM-sMBR system with the Tandem-GMM system, which is based on Stack-bottleneck features [Grézl and Karafiát, 2010, Vesely et al., 2011] that are post-processed by per-speaker fMLLR. On top of the fMLLR features we train a BMMI-GMM model, which is used to produce the second set of lattices. This system is very different and more complementary, the optimal weights for combining lattices are 0.7 for the DNN and 0.3 for the Tandem-GMM. The fusion improved the results 26.9 → 26.1 (the WER of the Tandem-GMM system was 32.5).

Table 9.2: *Fusion of the DNN-sMBR system and the Tandem-GMM system (Stacked bottleneck features + fMLLR + BMMI) that we used as the seed system. We do the 'simple word-selection'. The performance is the same we had with the single seed system.*

Added words:	60%	70%	80%	90%	100%
WER	25.0	24.6	24.4	24.7	24.9

From the results in table 9.2, we see that the self-training from the fusion leads to the same performance 24.4 as we had with the single DNN system (table 8.3). According to the red curve in figure 9.1, the fusion output leads to word-selection with less errors, however the difference was not sufficient to improve the performance of the semi-supervised training.

Another difficulty is that the word-boundaires in forced-alignment are not always the same as those from MBR decoding, which probably affected the semi-supervised training.

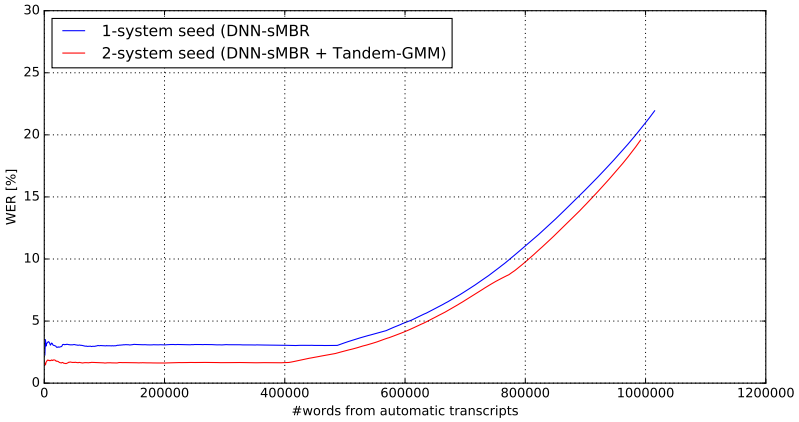


Figure 9.1: *WER in subset of data selected by the confidence, starting from highest. We compare two seed systems: a) single DNN-sMBR system, b) combination of DNN-sMBR and Tandem-GMM. The system combination selects words to a subset which contains less errors.*

Summary Based on our experiments, the system combination did not improve our semi-supervised training recipe. Despite that the system combination outperformed the single system in WER, this did not translate into a better semi-supervised DNN training.

Chapter 10

Final remarks

Initial chapters In this thesis we initially presented a quick introduction to the theory of speech recognition and neural network training, along with our NN-training implementation available as the 'nnet1' training recipe in Kaldi. The recipe is composed of RBM pre-training, mini-batch frame Cross-Entropy training, and sequence-discriminative sMBR training.

Semi-supervised training, what we searched for? Then we switched to the main topic, which is the semi-supervised training of DNN-based ASR systems. Inspired by the literature survey and our initial experiments, we investigated several questions: Firstly, whether the confidences are better to be calculated per-sentence, per-word or per-frame. Then, if the confidences should be used for the data-selection or the data-weighting, which is both compatible with the weighted mini-batch SGD training. It was also not clear whether the confidence calibration can improve the performance of the semi-supervised training. We also investigated how the model should be re-tuned with the correctly transcribed data. And finally we searched for a simple recipe, avoids a grid search over hyper-parameter and that is practical for general use with any dataset.

What we found out The performance differences of the systems with various confidences were relatively small, while it was easier to obtain good results with word-confidences and frame-confidences. The data-weighting (with a tuned scale α as exponent) led to a little better results than the data-selection. The confidence calibration led to minimal or no performance improvements. And the re-training with correctly transcribed data is better to be done first with the 'frame Cross Entropy' objective and then with the 'sMBR' objective.

Finally a **practical recipe** without a computationally expensive hyper-parameter tuning is following: *Use the best-path from lattice as NN training targets. From the best-path, select the words whose confidence is in top N%, where the N% is given by word-accuracy of seed system in the development set. The word confidences are extracted as the 'posteriors' from the MBR decoding, in which the word-sequence is fixed to the words obtained from best-path in lattice.* The frames of selected words have weight 1, the frames of other words get weight 0. Such model is first re-trained by 'frame Cross Entropy' objective (with reduced learning rate 0.001 instead of 0.008) and then with the 'sMBR' objective (standard learning rate). The two-system combination as the seed system did not lead to improvements, when compared to a single seed system.

What we did not cover, what is to be explored later Due to both the space and time limitations, we did not explore some areas which would be interesting. We could for example re-place the feed-forward DNN model with a BLSTM, or replace the PLP-pitch features with bottleneck features. However, intentionally, we kept on using the standard DNN recipe, so that the experiments are comparable.

We also tried the two-softmax 'frame CE' training and the 'entropy-minimization' for untranscribed sentences in sequence-discriminative training. However these did not bring further improvements on top of our existing system, and we did not include the experiments in this thesis.

Final conclusion We found it quite difficult to further improve the performance of the semi-supervised training. Still, we believe, that our findings will be perceived to have practical value. The untranscribed data are abundant and easy to obtain, while our proposed solution brings solid WER improvements (see table 8.8 on page 93) and is not difficult to replicate.

The main results from this thesis were recently published in [Veselý et al., 2017].

Bibliography

[Ackley et al., 1985] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, 9(1):147–169.

[Bahl et al., 1986] Bahl, L. R., Brown, P. F., de Souza, P. V., and Mercer, R. L. (1986). Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *Proc. IEEE ICASSP*, volume 1, pages 49–52.

[Barron, 1993] Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945.

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., Montréal, U. D., and Québec, M. (2007). Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press.

[Bisani and Ney, 2008] Bisani, M. and Ney, H. (2008). Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50:434–451.

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 1st ed. 2006. corr. 2nd printing edition.

[Bourlard and Morgan, 1993] Bourlard, H. A. and Morgan, N. (1993). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA.

[Bridle and Dodd, 1991] Bridle, J. S. and Dodd, L. (1991). An Alphanet approach to optimising input transformations for continuous speech recognition. In *Proc. IEEE ICASSP*, volume 1, pages 277–280.

[Burget et al., 2008] Burget, L., Schwarz, P., Matějka, P., Hannemann, M., Rastrow, A., White, C., Khudanpur, S., Heřmanský, H., and Černocký, J. (2008). Combination of strongly and weakly constrained recognizers for reliable detection of OOVs. In *Proc. of ICASSP*.

[Chen et al., 2015] Chen, G., Xu, H., Wu, M., Povey, D., and Khudanpur, S. (2015). Pronunciation and silence probability modeling for ASR. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 533–537.

[Cohn et al., 1994] Cohn, D. A., Atlas, L. E., and Ladner, R. E. (1994). Improving generalization with active learning. *Machine Learning*, 15(2):201–221.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *MCSS*, 2(4):303–314.

[Dean et al., 2012] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A. W., Tucker, P. A., Yang, K., and Ng, A. Y. (2012). Large scale distributed deep networks. In *Proc. of NIPS*.

[Fraga-Silva et al., 2011] Fraga-Silva, T., Gauvain, J., and Lamel, L. (2011). Lattice-based unsupervised acoustic model training. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*.

[Gers et al., 2000] Gers, F. A., Schmidhuber, J., and Cummins, F. A. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471.

[Ghahremani et al., 2014] Ghahremani, P., BabaAli, B., Povey, D., Riedhammer, K., Trmal, J., and Khudanpur, S. (2014). A pitch extraction algorithm tuned for automatic speech recognition. In *Proceedings of ICASSP*.

[Gibson and Hain, 2006] Gibson, M. and Hain, T. (2006). Hypothesis spaces for minimum Bayes risk training in large vocabulary speech recognition. In *Proc. INTERSPEECH*, pages 2406–2409.

[Glorot et al., 2010] Glorot, X., Bordes, A., and Bengio, Y. (2010). Deep sparse rectifier neural networks. In *Proc. AISTATS'10*, volume 15 (draft) of *W&CP*. JMLR.

[Gollan, 2014] Gollan, C. (2014). *Efficient Setup of Acoustic Models for Large Vocabulary Continuous Speech Recognition*. PhD thesis, RWTH Aachen University, Computer Science Department, RWTH Aachen University, Aachen, Germany.

[Gollan et al., 2007] Gollan, C., Hahn, S., Schlüter, R., and Ney, H. (2007). An improved method for unsupervised training of LVCSR systems. In *INTERSPEECH 2007, 8th Annual Conference of the International Speech Communication Association, Antwerp, Belgium, August 27-31, 2007*.

[Goodfellow et al., 2013] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. C., and Bengio, Y. (2013). Maxout Networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1319–1327.

[Grandvalet and Bengio, 2004] Grandvalet, Y. and Bengio, Y. (2004). Semi-supervised learning by entropy minimization. In *Proc. of NIPS*.

[Graves et al., 2013] Graves, A., Jaitly, N., and Mohamed, A. (2013). Hybrid speech recognition with deep bidirectional LSTM. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, December 8-12, 2013*, pages 273–278.

[Grézl and Karafiát, 2014] Grézl, F. and Karafiát, M. (2014). Combination of multilingual and semi-supervised training for under-resourced languages. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*.

[Grézl and Karafiát, 2010] Grézl, F. and Karafiát, M. (2010). Hierarchical neural net architectures for feature extraction in asr. In *Proc. INTERSPEECH'10*.

[Grézl et al., 2007] Grézl, F., Karafiát, M., Kontár, S., and Černocký, J. (2007). Probabilistic and Bottle-neck Features for LVCSR of Meetings. In *Proc. ICASSP'07*, pages 757–760.

[Hinton et al., 2012] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, pages 14–22.

[Hinton, 2012] Hinton, G. E. (2012). A Practical Guide to Training Restricted Boltzmann Machines. In *Neural Networks: Tricks of the Trade - Second Edition*.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

[Hochreiter, 1991] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universität, München*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Huang and Hasegawa-Johnson, 2010] Huang, J.-T. and Hasegawa-Johnson, M. (2010). Semi-supervised training of Gaussian mixture models by conditional entropy minimization. In *Proc. of INTERSPEECH*, pages 1353–1356.

[Huang et al., 2013] Huang, Y., Yu, D., Gong, Y., and Liu, C. (2013). Semi-supervised GMM and DNN acoustic model training with multi-system combination and confidence re-calibration. In *Proc. of INTERSPEECH*.

[Jaitly et al., 2012] Jaitly, N., Nguyen, P., Senior, A., and Vanhoucke, V. (2012). Application of pretrained deep neural networks to large vocabulary speech recognition. In *Proc. INTERSPEECH*.

[Jelinek, 1976] Jelinek, F. (1976). Continuous Speech Recognition by Statistical Methods. In *Proceedings of the IEEE 64*, pages 532–556.

[Kaiser et al., 2000] Kaiser, J., Horvat, B., and Kačič, Z. (2000). A novel loss function for the overall risk criterion based discriminative training of HMM models. In *Proc. ICSLP*, volume 2, pages 887–890.

[Kingsbury, 2009] Kingsbury, B. (2009). Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *Proc. IEEE ICASSP*, pages 3761–3764.

[Kingsbury et al., 2012] Kingsbury, B., Sainath, T. N., and Soltau, H. (2012). Scalable minimum Bayes risk training of deep neural network acoustic models using distributed Hessian-free optimization. In *Proc. INTERSPEECH*.

[Kneser and Ney, 1995] Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '95, Detroit, Michigan, USA, May 08-12, 1995*, pages 181–184.

[Krogh and Riis, 1999] Krogh, A. and Riis, S. K. (1999). Hidden neural networks. *Neural Computation*, 11(2):541–563.

[Kuwabara, 1996] Kuwabara, H. (1996). Acoustic properties of phonemes in continuous speech for different speaking rate. In *Proceedings of ICSLP, 1996*.

[Li et al., 2016] Li, S., Akita, Y., and Kawahara, T. (2016). Semi-supervised acoustic model training by discriminative data selection from multiple ASR systems' hypotheses. *IEEE/ACM Trans. Audio, Speech & Language Processing*, 24(9):1524–1534.

[Malkin et al., 2009] Malkin, J., Subramanya, A., and Bilmes, J. (2009). On the semi-supervised learning of multi-layered perceptrons. In *Proc. of INTERSPEECH*, pages 660–663.

[Manohar et al., 2015] Manohar, V., Povey, D., and Khudanpur, S. (2015). Semi-supervised maximum mutual information training of deep neural network acoustic models. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 2630–2634.

[Miao et al., 2013] Miao, Y., Metze, F., and Rawat, S. (2013). Deep maxout networks for low-resource speech recognition. In *Proc. ASRU 2013*.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proc. INTERSPEECH 2010*.

[Mohri, 2004] Mohri, M. (2004). *Weighted Finite-State Transducer Algorithms. An Overview*, pages 551–563. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Mohri et al., 2002] Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

[Mohri et al., 1998] Mohri, M., Riley, M., Hindle, D., Ljolje, A., and Pereira, F. C. N. (1998). Full expansion of context-dependent networks in large vocabulary speech recognition. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98, Seattle, Washington, USA, May 12-15, 1998*.

[Ng et al., 2012] Ng, T., Zhang, B., Nguyen, L., Matsoukas, S., Zhou, X., Mesgarani, N., Veselý, K., and Matějka, P. (2012). Developing a Speech Activity Detection System for the DARPA RATS Program. In *Proc. of INTERSPEECH 2012*.

[Novak et al., 2016] Novak, J. R., Minematsu, N., and Hirose, K. (2016). Phonetisaurus: Exploring grapheme-to-phoneme conversion with joint n-gram models in the WFST framework. *Natural Language Engineering*, 22(6):907–938.

[Novotney and Schwartz, 2009] Novotney, S. and Schwartz, R. M. (2009). Analysis of low-resource acoustic model self-training. In *Proc. of INTERSPEECH*, pages 244–247.

[Novotney et al., 2009] Novotney, S., Schwartz, R. M., and Ma, J. Z. (2009). Unsupervised acoustic and language model training with small amounts of labelled data. In *Proc. of IEEE ICASSP*, pages 4297–4300.

[Povey, 2003] Povey, D. (2003). *Discriminative Training for Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, Cambridge, UK.

[Povey et al., 2012] Povey, D., Hannemann, M., Boulianne, G., Burget, L., Ghoshal, A., Janda, M., Karafiát, M., Kombrink, S., Motlícek, P., Qian, Y., Riedhammer, K., Veselý, K., and Vu, N. T. (2012). Generating exact lattices in the WFST framework. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25-30, 2012*.

[Povey et al., 2008] Povey, D., Kanevsky, D., Kingsbury, B., Ramabhadran, B., Saon, G., and Visweswariah, K. (2008). Boosted MMI for model and feature-space discriminative training. In *Proc. IEEE ICASSP*, pages 4057–4060.

[Povey and Kingsbury, 2007] Povey, D. and Kingsbury, B. (2007). Evaluation of proposed modifications to MPE for large scale discriminative training. In *Proc. IEEE ICASSP*, volume 4, pages IV–321–IV–324.

[Povey et al., 2014] Povey, D., Zhang, X., and Khudanpur, S. (2014). Parallel training of deep neural networks with natural gradient and parameter averaging. *CoRR*, abs/1410.7455.

[Recht et al., 2011] Recht, B., Re, C., Wright, S. J., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of NIPS*.

[Rennie et al., 2014] Rennie, S. J., Goel, V., and Thomas, S. (2014). Annealed dropout training of deep networks. In *2014 IEEE Spoken Language Technology Workshop, SLT 2014, South Lake Tahoe, NV, USA, December 7-10, 2014*.

[Robinson, 1992] Robinson, T. (1992). A real-time recurrent error propagation network word recognition system. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 1:617–620.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[Sainath et al., 2014] Sainath, T. N., Chung, I., Ramabhadran, B., Picheny, M., Gunnels, J. A., Kingsbury, B., Saon, G., Austel, V., and Chaudhari, U. V. (2014). Parallel Deep Neural Network training for LVCSR tasks using Blue Gene/Q. In *Proceedings of INTERSPEECH 2014*.

[Sainath et al., 2013] Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. (2013). Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6655–6659.

[Sak et al., 2014] Sak, H., Senior, A. W., and Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128.

[Scanzio et al., 2010] Scanzio, S., Cumani, S., Gemello, R., F., M., and Laface, P. (2010). Parallel implementation of artificial neural network training. In *Proc. ICASSP'10*.

[Seide et al., 2011a] Seide, F., Li, G., Chen, X., and Yu, D. (2011a). Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Proc. IEEE ASRU*, pages 24–29.

[Seide et al., 2011b] Seide, F., Li, G., and Yu, D. (2011b). Conversational speech transcription using context-dependent deep neural networks. In *Proc. of INTERSPEECH*.

[Senior et al., 2015] Senior, A. W., Sak, H., and Shafran, I. (2015). Context dependent phone models for LSTM RNN acoustic modelling. In *Proc. ICASSP 2015*.

[Sharma et al., 2000] Sharma, S., Ellis, D., Kajarekar, S., Jain, P., and Hermansky, H. (2000). Feature extraction using non-linear transformation for robust speech recognition on the Aurora database. In *Proc. ICASSP 2000*, Turkey.

[Smolensky, 1986] Smolensky, P. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Information Processing in Dynamical Systems: Foundations of Harmony Theory, pages 194–281. MIT Press, Cambridge, MA, USA.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

[Su and Chen, 2015] Su, H. and Chen, H. (2015). Experiments on parallel training of deep neural network using model averaging. *CoRR*, abs/1507.01239.

[Su and Xu, 2015] Su, H. and Xu, H. (2015). Multi-softmax deep neural network for semi-supervised training. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 3239–3243.

[Swietojanski et al., 2012] Swietojanski, P., Ghoshal, A., and Renals, S. (2012). Unsupervised cross-lingual knowledge transfer in DNN-based LVCSR. In *Proc. of IEEE SLT*, pages 246–251.

[Swietojanski et al., 2013] Swietojanski, P., Ghoshal, A., and Renals, S. (2013). Revisiting hybrid and GMM-HMM system combination techniques. In *Proc. of IEEE ICASSP*.

[Swietojanski et al., 2014] Swietojanski, P., Li, J., and Huang, J. (2014). Investigation of maxout networks for speech recognition. In *Proc. ICASSP 2014*.

[Talkin, 1995] Talkin, D. (1995). A robust algorithm for pitch tracking (RAPT). In Kleijn, W. B. and Paliwal, K., editors, *Speech Coding and Synthesis*, New York. Elseviever.

[Thomas et al., 2013] Thomas, S., Seltzer, M. L., Church, K., and Hermansky, H. (2013). Deep neural network features and semi-supervised training for low resource speech recognition. In *Proceedings of ICASSP*, pages 6704–6708.

[Trmal et al., 2010] Trmal, J., Pražák, A., Loose, Z., and Psutka, J. (2010). Online TV captioning of Czech Parliamentary Sessions. In *Proceedings of TSD 2010*.

[Tüske et al., 2012] Tüske, Z., Schlüter, R., Ney, H., and Sundermeyer, M. (2012). Context-Dependent MLPs for LVCSR: TANDEM, Hybrid or Both? In *Proceedings of INTERSPEECH'12*.

[Veselý et al., 2017] Veselý, K., Burget, L., and Černocký, J. H. (2017). Semi-supervised DNN training with word-selection for ASR. In *Proceedings of INTERSPEECH 2017*.

[Veselý et al., 2011] Veselý, K., Karafiát, M., and Grézl, F. (2011). Convolutive bottleneck network features for LVCSR. In *Proceedings of ASRU 2011*, pages 42–47.

[Vesely et al., 2011] Vesely, K., Karafiat, M., and Grezl, F. (2011). Convolutive bottleneck network features for LVCSR. *Proc. of ASRU 2011*, pages 42–47.

[Veselý et al., 2016] Veselý, K., Watanabe, S., Žmolíková, K., Karafiát, M., Burget, L., and Černocký, J. H. (2016). Sequence summarizing neural network for speaker adaptation. In *Proceedings of ICASSP'16*.

[Veselý, 2010] Veselý, K. (2010). *Parallel Training of Neural Networks for Speech Recognition*. Brno University of Technology (master thesis).

[Veselý et al., 2010] Veselý, K., Burget, L., and Grézl, F. (2010). Parallel training of neural networks for speech recognition. In *Proc. INTERSPEECH'10*, pages 2934–2937.

[Veselý et al., 2013a] Veselý, K., Ghoshal, A., Burget, L., and Povey, D. (2013a). Sequence-discriminative training of deep neural networks. In *Proc. of INTERSPEECH'13*.

[Veselý et al., 2013b] Veselý, K., Hannemann, M., and Burget, L. (2013b). Semi-supervised training of deep neural networks. In *Proceedings of ASRU*, pages 267–272.

[Wager et al., 2013] Wager, S., Wang, S. I., and Liang, P. (2013). Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 351–359.

[Wang and Sim, 2011] Wang, G. and Sim, K. C. (2011). Sequential classification criteria for NNs in automatic speech recognition. In *Proc. INTERSPEECH*, pages 441–444.

[Wawrzynek et al., 1996] Wawrzynek, J., Asanovic, K., Kingsbury, B., Johnson, D., Beck, J., and Morgan, N. (1996). Spert-II: A Vector Microprocessor System. http://www1.icsi.berkeley.edu/Speech/spert/computer_spert.pdf. Accessed: 2015-11-11.

[Wessel and Ney, 2005] Wessel, F. and Ney, H. (2005). Unsupervised training of acoustic models for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 13(1):23–31.

[Wessel et al., 2001] Wessel, F., Schlüter, R., Macherey, K., and Ney, H. (2001). Confidence measures for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 9(3):288–298.

[Xu et al., 2011] Xu, H., Povey, D., Mangu, L., and Zhu, J. (2011). Minimum Bayes Risk decoding and system combination based on a recursion for edit distance. *Computer Speech & Language*, 25(4):802–828.

[Young et al., 2002] Young, S., Jansen, J., Odell, J., Ollason, D., and Woodland, P. (2002). *The HTK book*. Entropics Cambridge Research Lab., Cambridge, UK.

[Young and Woodland, 1994] Young, S. J. and Woodland, P. C. (1994). State clustering in hidden markov model-based continuous speech recognition. *Computer Speech & Language*, 8(4):369–383.

[Yu et al., 2010a] Yu, D., Deng, L., and Dahl, G. (2010a). Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition. In *Proc. of NIPS*.

[Yu et al., 2011] Yu, D., Li, J., and Deng, L. (2011). Calibration of confidence measures in speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(8):2461 – 2473.

[Yu et al., 2010b] Yu, D., Varadarajan, B., Deng, L., and Acero, A. (2010b). Active learning and semi-supervised learning for speech recognition: A unified framework using the global entropy reduction maximization criterion. *Computer Speech & Language*, 24(3):433–444.

[Yu et al., 2010c] Yu, K., Gales, M. J. F., Wang, L., and Woodland, P. C. (2010c). Unsupervised training and directed manual transcription for LVCSR. *Speech Communication*, 52(7-8):652–663.

[Zeiler et al., 2013] Zeiler, M. D., Ranzato, M., Monga, R., Mao, M. Z., Yang, K., Le, Q. V., Nguyen, P., Senior, A. W., Vanhoucke, V., Dean, J., and Hinton, G. E. (2013). On rectified linear units for speech processing. In *Proc. ICASSP 2013*.

[Zhang et al., 2014a] Zhang, P., Liu, Y., and Hain, T. (2014a). Semi-supervised DNN training in meeting recognition. In *2014 IEEE Spoken Language Technology Workshop, SLT 2014, South Lake Tahoe, NV, USA, December 7-10, 2014*, pages 141–146.

[Zhang et al., 2014b] Zhang, X., Trmal, J., Povey, D., and Khudanpur, S. (2014b). Improving deep neural network acoustic models using generalized maxout networks. In *Proc. ICASSP 2014*.