NAIST-IS-DT9661205

PhD Thesis

On supervised learning from sequential data with applications for speech recognition

Michael Schuster

February 15th, 1999

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

PhD Thesis
submitted to Graduate School of Information Science
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Michael Schuster

Preface

"Assumption is the mother of all screw-ups."
`fortune`, slakware-3.6.0

Congratulations. You are one of the few who started reading this thesis. If you have the patience, read it all. But who has. So if you just want to skip through it, read at least this preface to know in what order to skip.

This thesis deals with supervised learning from sequential data, always having the quote above in mind. Read the introduction (chapter 1), if you don't know what I mean by *supervised learning* or *sequential data*. Chapter 2 summarizes the necessary basics to understand the underlying problem and possible approaches to solve it. Don't be afraid, the ideas (chapter 3 and 4) presented in this thesis are, compared to what you can find elsewhere, relatively simple. In chapter 3 a recurrent neural network structure is extended to a bidirectional structure to model probabilistic expressions occuring when you treat 'learning from sequences' as a pattern recognition problem. The probably most interesting section is the one about the recurrent mixture density networks. Read chapter 4 if you want to know how I implemented a stack decoder for speech recognition, a challenging sequential-data problem. If you wonder why two so very different topics are addressed in one thesis, read chapter 2 again, because: To predict (recognize) a sequence you need always two parts, a generative part (chapter 3) and a search part (chapter 4), given the current state of research.

This thesis is available in postscript from the WWW server of the Nara Institute of Science and Technology:

`http://isw3.aist-nara.ac.jp/IS/Shikano-lab/database/library/paper/DT9661205.ps.gz`

`http://isw3.aist-nara.ac.jp/IS/Shikano-lab/database/library/paper/DT9661205withJapanese.ps.gz`

There are two version, one completely in English and one that contains a Japanese abstract and some Japanese references. If you don't have Japanese postscript fonts then the Japanese version will not display correctly in your postscript viewer. Also, you need a printer with Japanese fonts to print it out.

On supervised learning from sequential data with applications for speech recognition [*]

Michael Schuster

Abstract

Many problems of engineering interest, for example speech recognition, can be formulated in an abstract sense as *supervised learning from sequential data*, where an input sequence $\mathbf{x}_1^T = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{T-1}, \mathbf{x}_T\}$ has to be mapped to an output sequence $\mathbf{y}_1^T = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots, \mathbf{y}_{T-1}, \mathbf{y}_T\}$. This thesis gives a unified view of the abstract problem and presents some models and algorithms for improved sequence recognition and modeling performance, measured on synthetic data and on real speech data.

A powerful neural network structure to deal with sequential data is the recurrent neural network (RNN), which allows one to estimate $P(\mathbf{y}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t)$, the output probability distribution at time t given all previous input. The first part of this thesis presents various extensions to the basic RNN structure, which are

a) a *bidirectional recurrent neural network* (BRNN), which allows the estimation of expressions of the form $P(\mathbf{y}_t|\mathbf{x}_1^T)$, the output at t given *all* sequential input, for uni-modal regression and classification problems,

b) an extended BRNN to directly estimate the posterior probability of a symbol sequence, $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$, by modeling $P(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_1, \mathbf{x}_1^T)$ without explicit assumptions about the shape of the distribution $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$,

c) a BRNN to model multi-modal input data that can be described by Gaussian mixture distributions conditioned on an output vector sequence, $P(\mathbf{x}_t|\mathbf{y}_1^T)$, assuming that neighboring $\mathbf{x}_t, \mathbf{x}_{t+1}$ are conditionally independent, and

d) an extension to c) which removes the independence assumption by modeling $P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_1, \mathbf{y}_1^T)$ to estimate the likelihood $P(\mathbf{x}_1^T|\mathbf{y}_1^T)$ of a given output sequence without any explicit approximations about the use of context.

The second part of this thesis describes the details of a fast and memory-efficient one-pass stack decoder for speech recognition to perform the *search* for the most probable word sequence. The use of this decoder, which can handle arbitrary order N-gram language models and arbitrary order context-dependent acoustic models with full cross-word expansion, led to the best reported recognition results on the standard test set of a widely used Japanese newspaper dictation task.

Keywords: speech recognition, recurrent neural networks, stack decoder

[*]PhD Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9661205, February 15th, 1999.

情報系列からの教師付き学習とその音声認識への応用*

シュスター・マイク

内容梗概

音声認識に代表される情報系列の認識問題は工学的に非常に興味深い分野であり、その多くの場合において、入力系列 $\mathbf{x}_1^T = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{T-1}, \mathbf{x}_T\}$ から出力系列 $\mathbf{y}_1^T = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots, \mathbf{y}_{T-1}, \mathbf{y}_T\}$ へのマッピングを、情報系列データから教師付きで学習する問題としての定式化が可能である。本論文では、種々の問題を一般の情報系列認識問題として統一的に扱う視点に立ち、モデリング性能、及び認識性能を向上させるための枠組みとアルゴリズムについて述べるとともに、これらを擬似的に合成したデータと実際の音声データを用いて評価した結果を示す。

リカレント・ニューラル・ネットワーク（RNN）は、従来のニューラル・ネットワークに対して情報系列を取り扱うのに適した構造を持たせた強力なモデリング手法であり、これにより時刻 t までに与えられた全ての入力情報に対する出力情報出現確率 $P(\mathbf{y}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t)$ が推定可能である。本論文の前半部では、基本的な RNN を以下に示す通り様々に拡張したものについて述べる。

a) 双方向リカレント・ニューラル・ネットワーク（BRNN）：最終時刻までの全ての入力情報に対する出力情報出現確率 $P(\mathbf{y}_t|\mathbf{x}_1^T)$ を推定可能とするものであり、単峰形関数に基づく回帰、及び識別問題に適用可能

b) 拡張 BRNN： $P(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_1, \mathbf{x}_1^T)$ をモデリングすることによって、出力情報系列の出現事後確率 $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$ を直接推定可能とするもの

c) 多峰形の分布に従う入力情報に対して、ある出力情報系列の条件下での出現確率 $P(\mathbf{x}_t|\mathbf{y}_1^T)$ を、隣接する入力情報 $\mathbf{x}_t, \mathbf{x}_{t+1}$ が互いに独立と仮定して推定する BRNN

d) c) の拡張として、隣接入力情報間に独立性が仮定できない場合に、 $P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_1, \mathbf{y}_1^T)$ をモデリングすることによって、 $P(\mathbf{x}_1^T|\mathbf{y}_1^T)$ を近似なしに推定可能な BRNN

また、本論文の後半部では、最も確からしい単語系列を高速に、かつ少ないメモリ量で探索可能な、音声認識用 one-pass スタックデコーダについて述べる。本デコーダは、任意次数の N-gram 言語モデル、及び単語境界音素環境も考慮した任意次数の音素環境依存音響モデルが使用可能で、現在広く用いられている日本語新聞ディクテーションタスクでの評価により最高レベルの性能が示された。

キーワード: 音声認識, リカレントニューラルネットワーク, スタック デコダ

*奈良先端科学技術大学院大学 情報科学研究科 情報処理学専攻 博士論文, NAIST-IS-DT9661205, １９９９年２月１５日.

Acknowledgments

First and foremost I would like to thank my supervisor Prof. Shikano and my co-supervisor Prof. Tohkura from the Nara Institute of Science and Technology. I deeply respect the always warm and helpful atmosphere typical for Japan as much as the supervision and freedom they gave me in deciding what research to pursue, which probably helped me most in developing my view of scientific research. The required monthly update talks about the progress of my work and the critical discussions afterwards helped me to a large extent to stay focussed and realistic. I also would like to thank Prof. Matsumoto, Prof. Campbell and Associate Prof. Nakamura from the Nara Institute of Science and Technology for taking their time to review this thesis and for giving valuable comments. I am especially grateful to Associate Prof. Nakamura, who spent a fair amount of time with me discussing my research and who has helped me on a number of other occasions as well.

It was only possible to enroll in the PhD program because I was supported by ATR Interpreting Telecommunications Laboratories, where I have worked as a researcher since October 1995. I would like to thank my department head, Dr. Sagisaka, as well as the current head of ATR-ITL, Dr. Yamamoto, and the previous head, Dr. Yamazaki, for their continuous support of giving me the freedom for research and for the chance of attending many conferences and visiting other research labs all over the world. ATR-ITL helped me to understand more of the Japanese research atmosphere and gave me the chance of meeting many interesting people inside and outside of ATR, who in some way had an influence on this thesis. There are too many to name individually, but I would like to thank in particular some members and visitors of ATR-ITL and ATR-HIP for scientific discussions, for the never-ending need for help with the Japanese language, for the fun at basketball and for being good friends: Kuldip Paliwal, Li Deng, Toshiaki Fukada, Harald Singer, Qiang Huo, Michiel Bacchiani, Dimitri Rtischev, Gregor Möhler, Detlef Koll, Hirofumi Yamamoto, Atsushi Nakamura, Hajime Tsukada, Tomoyo and Tetsuya Takiguchi, Masaki Naito, Hirokazu Masataki, Hideyuki Watanabe, Erik McDermott and Eric Woudenberg. I want to thank especially Erik for giving me a critical review of this thesis, and Atsushi Nakamura for translating the abstract into Japanese.

I am especially grateful to Li Deng, who invited me to the 1998 Switchboard Speech Recognition Workshop at Johns Hopkins University, which turned out to be a great experience.

Some people not affiliated with ATR who helped me by answering E-mails about their research or by reviewing paper drafts are Tony Robinson, Steve Renals, Jürgen Fritsch and John Bridle.

I had my first contact with neural networks, speech recognition and learning from data at the Gerhard-Mercator-University in my home town Duisburg, Germany. Several members of Prof. Gerhard Rigoll's department I joined for a few months before coming to ATR, Christoph Neukirchen, Jörg Rottland and Daniel Willett, became good friends and we have had many fruitful scientific and non-scientific discussions since. I want to thank Christoph and Daniel especially for giving me a very detailed review of this thesis.

I am grateful to the "Studienstiftung des deutschen Volkes", a German scholarship association, which funded me in 1993/94 when I came after my master thesis the first time to Kyoto and Tokyo, Japan, to learn about language and people. I would like to thank Prof. Kikuchi from Tokyo University, who gave me the chance to work with him and his group for six months in 1994.

Finally, I want to thank my parents for always supporting my interests, and I have to hug my wife Keiko and my two kids Julia and Yuuji for distracting me from research whenever it was necessary.

Contents

List of Figures **xi**

List of Tables **xiii**

List of Symbols **xvi**

1 Introduction **1**

1.1 MOTIVATION AND BACKGROUND 1

1.1.1 Learning from examples 1

1.1.2 Does the order of the data samples matter? 3

1.1.3 Example applications 3

1.1.4 Related scientific areas 4

1.2 THESIS STRUCTURE 5

2 Supervised learning from sequential data **7**

2.1 DEFINITION OF THE PROBLEM 7

2.2 Decomposition into a generative and a prior model part 10

2.2.1 Context-independent model 10

2.2.2 Context-dependent model 12

2.3 Direct decomposition 13

2.4 HIDDEN MARKOV MODELS 14

2.4.1 Basic HMM formulation 15

2.4.2 Calculation of state occupation probabilities 17

2.4.3 Parameter estimation for output probability distributions 20

2.4.4 Parameter estimation for transition probabilities 23

2.5 SUMMARY 23

3 Neural networks for supervised learning from sequences **25**

3.1 BASICS OF NEURAL NETWORKS 26

3.1.1 Parameter estimation by maximum likelihood 26

3.1.2 Problem classification 27

3.1.3 Neural network training 29

3.1.4 Neural network architectures 32

3.2 BIDIRECTIONAL RECURRENT NEURAL NETWORKS 37
 3.2.1 Prediction assuming independent outputs 37
 3.2.2 Experiments and results 40
 3.2.3 Prediction assuming dependent outputs 49
 3.2.4 Experiments and results 52
3.3 MIXTURE DENSITY RECURRENT NEURAL NETWORKS 54
 3.3.1 Basics of mixture density networks 56
 3.3.2 Mixture density extension for BRNNs 60
 3.3.3 Experiments and results 61
 3.3.4 Discussion 65
3.4 SUMMARY 67

4 Memory-efficient LVCSR search using a one-pass stack decoder 71
4.1 INTRODUCTION 72
 4.1.1 Organization of this chapter 73
 4.1.2 General 73
 4.1.3 Technical 75
 4.1.4 Decoder types 80
4.2 A MEMORY-EFFICIENT ONE-PASS STACK DECODER 82
 4.2.1 Basic algorithm 83
 4.2.2 Pruning techniques 87
 4.2.3 Stack module 89
 4.2.4 Hypotheses module 91
 4.2.5 N-gram module 92
 4.2.6 LM lookahead 94
 4.2.7 Cross-word models 96
 4.2.8 Fast-match with delay 98
 4.2.9 Using word-graphs as language model constraints 98
 4.2.10 Lattice rescoring 99
 4.2.11 Generating phone-/state-alignments 99
4.3 EXPERIMENTS 99
 4.3.1 Recognition of Japanese 100
 4.3.2 Recognition results for high accuracy 101
 4.3.3 Recognition results for high speed and low memory 103
 4.3.4 Time and memory requirements for modules 104
 4.3.5 Usage of cross-word models 105
 4.3.6 Usage of fast-match models 105
 4.3.7 Effect of on-demand N-gram smearing 105
 4.3.8 Lattice/N-best list generation and lattice rescoring 106
4.4 CONCLUSIONS 107
4.5 ACKNOWLEDGMENTS 107

5 Conclusions **109**
5.1 SUMMARY 109
5.2 CONTRIBUTIONS FROM THIS THESIS 111
5.3 SUGGESTIONS FOR FUTURE WORK 112

A Improved initialization of Multi-Layer-Perceptrons **115**

B List of Publications **117**

List of Figures

1.1 Learning from sequential data 2

2.1 Supervised learning from sequences 9
2.2 Markov Model history length expansion 12
2.3 Example Hidden Markov Model structure 16

3.1 Elements of general feed-forward neural networks 33
3.2 Recurrent neural network 35
3.3 How much input information is used? 36
3.4 Bidirectional recurrent neural network 38
3.5 Results for artificial regression problem 42
3.6 Results for artificial classification problem 43
3.7 Forward probability estimation in extended BRNN 51
3.8 Extended bidirectional recurrent neural network 52
3.9 An artificial multi-modal regression problem 55
3.10 Conventional Gaussian mixtures for multi-modal regression 56
3.11 Polynomial segment models for multi-modal regression 57
3.12 BRNNs for multi-modal regression 58
3.13 Mixture density network 59
3.14 BRNN mixture density extension (Type II) 61
3.15 Mixture density BRNNs for multi-modal regression: Results 69

4.1 Example for a tree-lexicon 84
4.2 Hypotheses storage format 91
4.3 N-gram storage format 93
4.4 Uni-gram smearing 95
4.5 On-demand N-gram smearing 96
4.6 Cross-word model incorporation 97

List of Tables

3.1 Common setups for neural networks. 34
3.2 Types of experiment for prediction assuming independent outputs. . . . 41
3.3 Results for prediction assuming independent outputs. 46
3.4 Results for phoneme recognition on the TIMIT database using BRNNs as observation probability estimators. 49
3.5 Results for prediction assuming dependent outputs. 53
3.6 Number of parameters for different types of models 64

4.1 Used phone set for Japanese recognition 100
4.2 Recognition results for high accuracy 101
4.3 Parameter settings and search statistics for results from Table 4.2 102
4.4 Results for high speed and low memory 103
4.5 Parameter settings and search statistics for results from Table 4.4 103
4.6 Relative time and memory requirements for modules 104
4.7 Effect of cross-word effects 105
4.8 Effect of fast-match models 105
4.9 Effect of on-demand N-gram smearing 106
4.10 Relative time and memory for different search modes 106

A.1 Inversion of output activation functions. 116

List of Symbols

N, n	number of data samples, index
$\mathcal{X}$	input space
$\mathcal{Y}$	output space
D, d	dimensionality of input vector, component index
K, k	dimensionality of output (target) vector, component index
$\mathbf{x}_n, x^d$	nth input vector, dth component
$\mathbf{y}_n, y^k$	nth output vector, kth component
$\mathbf{t}_n, t^k$	nth target vector, kth component
s	categorical variable, symbol
$'$	indicates transposed vector or matrix
M, m	number of mixtures, index
$\mathbf{M}$	model
$\mathbf{A}, a_{ij}$	transition matrix, its entries
$b(\cdot)$	(observation) probability distribution, discrete or continuous
$\mathbf{B}$	collection of several $b(\cdot)$
t, T	time, total time of a sequence
$\mathbf{X} = \mathbf{x}_1^T = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{T-1}, \mathbf{x}_T\}$	input vector sequence
$\mathbf{Y} = \mathbf{y}_1^T = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots, \mathbf{y}_{T-1}, \mathbf{y}_T\}$	output vector sequence
$\mathbf{T} = \mathbf{t}_1^T = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \ldots, \mathbf{t}_{T-1}, \mathbf{t}_T\}$	target vector sequence
$S = s_1^T = \{s_1, s_2, s_3, \ldots, s_{T-1}, s_T\}$	categorical symbol sequence
$Q = q_1^T = \{q_1, q_2, q_3, \ldots, q_{T-1}, q_T\}$	Markov state sequence
J, j	number of Markov states, index
q_t^j	jth Markov state at time t
q^I, q^E	initial state, final exit state
$q(t)$	state index of state q
$N(q^j), N(q^i q^j)$	number of times state q^j (state pair $q^i q^j$) appears
$P(\cdot)$	probability *or* probability density
$\alpha_j(t)$	forward likelihood for jth state at time t
$\beta_j(t)$	backward likelihood for jth state at time t
$\gamma_j(t)$	state occupation probability for jth state at time t
$\gamma_{ij}(t)$	state pair occupation probability at time t
$\phi_j(t)$	Viterbi likelihood for jth state at time t
$\chi_j(t)$	Viterbi backtrace state index for jth state at time t

$\mathcal{L}$	likelihood
c_m	mixture weight for mth mixture component
$\boldsymbol{\mu}_m, \mu_m^d$	mean vector for mth mixture component, dth component
$\mathbf{\Sigma}_m$	covariance matrix for mth mixture component
σ_m	root of radial covariance for mth mixture component
σ_m^d	root of dth diagonal covariance component for mth mixture component
$\mathcal{D}$	data
W, w	number of weights (*or* word sequence), weight index
$\mathbf{W}$	all possible word sequences
$\mathcal{W}, \mathbf{w}, w_w$	parameter weight vector space, weight vector, wth weight
$E(\cdot)$	error function, objective function
δ_w	stepsize for wth weight
η	multiplication factor for gradient
ρ	multiplication factor for momentum
τ_+, τ_-	stepsize multiplication factors for RPROP
$a, f_{act}(a)$	activation, activation function
$\langle \cdot \rangle$	expectation operator
$\propto$	proportional to
$\approx$	approximately equal to
$:=$	is assigned to

Chapter 1

Introduction

Abstract

In this chapter the motivation and background for the research conducted for this thesis is discussed. A few examples to classify basic learning problems and links to related scientific areas are shown to define the scope of the thesis. Finally, the general structure of the thesis is briefly explained.

1.1 MOTIVATION AND BACKGROUND

This thesis was started out of an interest for the general problem of *learning from examples*, also called *learning from data*. The simple concept of having only a limited amount of data and a procedure to learn from it to explain and imitate human learning behavior and reasoning is very attractive, not only because of its simplicity, but also for the purpose of making practical use of it to enhance our living standard.

1.1.1 Learning from examples

Learning from data can be divided into the two parts

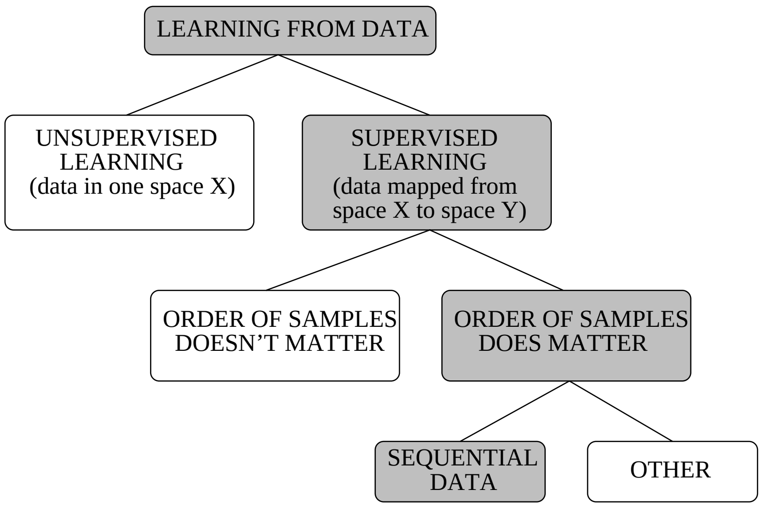
- *unsupervised learning* and
- *supervised learning*,

as illustrated in Figure 1.1.

1.1.1.a Unsupervised learning

Unsupervised learning refers to model data from one space $\mathcal{X}$, which is usually a vector space such that data samples from that space are vectors notated as $\mathbf{x}$. Learning in that case means to discover structure in the unknown space $\mathcal{X}$ by looking at examples from it. Once the structure is discovered, or approximated by a *model*, it can be used to predict certain areas in the space $\mathcal{X}$ which haven't been observed. An example for unsupervised learning is the task of understanding the range of the outside *temperature*

Figure 1.1: Learning from sequential data


LEARNING FROM DATA
UNSUPERVISED LEARNING (data in one space X)
SUPERVISED LEARNING (data mapped from space X to space Y)
ORDER OF SAMPLES DOESN'T MATTER
ORDER OF SAMPLES DOES MATTER
SEQUENTIAL DATA
OTHER

Classification of the problem of supervised learning from sequential data as used in this thesis.

x, defining the space $\mathcal{X}$, which has in this case as elements scalars (one-dimensional vectors). Let's assume the only way to find out about the outside temperature is to leave the house and measure it. Before leaving the house the first time we make certain assumptions about the temperature, for example that it is continuous and not discrete – we define a *model* M for it. We also decide not to have a preference for any specific temperature. If we would never leave the house, the outside temperature could be anything, it could be $x = -80°C$ or $x = 80°C$, each possible temperature would have the same probability. Leaving the house a few times (to collect a few samples of data) would tell us for example that the temperature is usually around $x = 20°C$ and rarely below zero or above $x = 30°C$. By leaving the house many, many times to collect a large number of data samples we will understand more about the properties of the temperature data space $\mathcal{X}$ – for example we will assume that x never reaches $80°C$, because we haven't observed a sample near to it, like $x = 70°C$ or $x = 60°C$. Now we have learned something about the structure of the data space that allows us to make predictions about areas of it we haven't observed. We could make for example statements of the form: "The temperature is with 60% chance between $25°C$ and $30°C$", meaning $P(25 < x < 30) = 0.6$", or "The temperature is never above $80°C$", meaning $P(x > 80) = 0$.

1.1.1.b Supervised learning

Supervised learning means to model data that is mapped from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$. Another expression for that is to say that $\mathcal{Y}$ *is conditioned* on $\mathcal{X}$.

An example would be the task of predicting the outside *temperature* (output space $\mathcal{Y}$) by checking the *brightness* of the incoming light through a window (input space $\mathcal{X}$). Measuring brightness and outside temperature or *observing samples of data pairs* will tell us how to make predictions about the outside temperature without leaving the house[1]. We will for example say that if it is very bright, i.e. $x = 100$, it is with 80% chance between 25°C and 30°C, or $P(25 < y < 30 \;\; given \;\; x = 100) = 0.8$.

1.1.2 Does the order of the data samples matter?

A further subdivision of *learning from data* can be made by assuming that the

- *order* of observing the data samples *doesn't matter* or that the
- *order* of observing the data samples *matters.*

1.1.2.a Order of data samples doesn't matter

An example for the case where the order doesn't matter would be a measurement of the outside temperature by many different people at the same time, each contributing one sample of the data. Since it is reasonable to assume that a measurement by Mrs. A doesn't influence the measurement by Mrs. B, the order of the samples does not matter in this case – the samples are assumed to be *statistically independent.*

1.1.2.b Order of data samples matters

An example for the case where the order matters would be when measurements for the temperature are made on consecutive days. Although the range of the temperature over the whole year might be between $-20°C$ and $+40°C$, the temperature does never change by 60°C over two consecutive days. Therefore the *order* of the data samples contains information that allows us to build a better model for the data space to make predictions. In this thesis this kind of data will be referred to as *sequential data*, when the structure of the order can be represented in a one-dimensional space.

1.1.3 Example applications

Supervised learning from sequential data has many possible applications. In practical systems it is in general required to predict a sequence in the output space given a sequence from the input space. Some of the applications in this context are for example:

Speech recognition: Speech recognition is often defined as the automatic transcription of human utterances as sequences of words. In the training phase, given a large number of utterances with their correct transcriptions, the task is to learn the mapping from the acoustic signal to the word sequence, so as to later be able to recognize new, previously unknown utterances. Considering the number of

[1] a definite enhancement of our living standard!

examples and the complexity of the models, speech recognition is currently one of the most advanced areas that uses the principle of learning from examples.

Automatic translation: The task is to learn the mapping from a sequence of words in one language to a sequence of words in another language, which can be viewed as a sequence prediction problem with categorical variables in the input and output space. Automatic translation with useful results between two arbitrary languages is an unsolved problem. However, for some pairs of languages with similar roots and grammar structures, like French and English, statistical methods similar to the ones addressed in this thesis have been applied with reasonable success.

Hand-writing recognition: Hand-writing recognition can today be done to some extent by many hand-held computers and PDAs. Most of the used approaches to the problem of online hand-writing recognition again use the principle of 'learning from examples'. A large number of words are written by an appropriate number of people with different writing styles – the movement of the pen is recorded using a digitizer board. Based on the sequential data describing the relationship between pen-movement and the corresponding words or letters, models are built that are used to recognize new words.

Stock market prediction: The problem of predicting the value of stocks or currencies can be formulated as a sequence prediction problem. History provides the example sequences. For example, the change of the ruling party of a country usually has some influence on the stock market. One of the input variables recorded over time could be a categorical variable which would indicate either 'party A' or 'party B'. The output variable could be the country's currency value, that could be predicted by learning from examples. The training data in this case would be the currency value and its history conditioned on the status of the 'party' variable.

1.1.4 Related scientific areas

All of the sequence prediction problems discussed above seem to be of very different nature. Any scientific approach to solving them involves first a unification and abstraction of all the specific problems to one scientific core problem, which allows us to use established scientific methods and knowledge from other scientific areas, as in this case:

Pattern recognition: *Pattern recognition* (Berger, 1985; Bishop, 1995; Duda and Hart, 1974; Ripley, 1996; Sivia, 1996) is the basis for many problems discussed in this thesis, and is often understood as *learning from examples* like discussed above. Pattern recognition involves the definition of stochastic models like neural networks (Bishop, 1995; Hertz et al., 1991) or Hidden Markov Models (Huang et al., 1990; Rabiner and Juang, 1993), which are trained on given training data and tested on unseen test data. It relies on the practical aspects of probability theory (MacKay, 1999), and is strongly related to coding and compression (Bell, 1990).

Information theory: *Information theory* (Hamming, 1986; MacKay, 1999; Mildenberger, 1992) defines consistent ways of measuring the amount of information in data, called the *entropy*. Many problems of pattern recognition, which are formulated to maximize a probability, can often also be formulated to maximize the amount of information flow through some channel.

Computer science: Pattern recognition and information theory live from their usage in implementations to solve real world problems. Computer science gives the algorithms and tools (Cormen et al., 1990; Press et al., 1992) to make pattern recognition work on computers. It is amazing how much time in research is spent on issues that are more related to the application of computer science than on issues related to the original problem.

1.2 THESIS STRUCTURE

The introduction defined the problem of supervised learning from sequential data in a loose way. Some examples for different learning problems were given and some real world applications were briefly discussed. The remainder of this thesis is organized as follows:

Chapter 2: The second chapter reviews basic notation and algorithms which are used throughout the thesis. A classification of the type of problems occuring for sequence modeling is given. Two frequently used approaches to decompose the posterior probability of a complete sequence conditioned on an input sequence are discussed. The basics of Hidden Markov Models, an important type of model for sequence modeling and sequence prediction, are reviewed.

Chapter 3: In the third chapter, various approaches to supervised learning from sequences using artificial neural networks are discussed. First the necessary basics of neural networks, commonly used architectures and their problems with respect to sequence processing are reviewed. Then a new architecture, based on a recurrent neural network, called a *bidirectional recurrent neural network*, that is especially useful for supervised learning from sequences, is introduced and evaluated for uni-modal regression and classification problems. Another extension is presented which allows the estimation of the posterior probability of a vector sequence conditioned on an input vector sequence. Finally the architecture is extended to model target sequences which can be described by multi-modal continuous distributions.

Chapter 4: In the fourth chapter, the concept of *search* for the prediction of symbol sequences conditioned on vector input sequences is addressed. The search procedure is an important part in the general class of sequence processing problems, and has generated a lot of research interest in the area of speech recognition. In this chapter a time- and memory-efficient implementation of a search algorithm for large vocabulary continuous speech recognition is presented. Advantages and problems of different search algorithms are discussed. Requirements for a modern decoder from an expert user's point of view are defined. It is shown how most of these can be integrated into a single decoder. Finally results from experiments on a Japanese newspaper dictation task are presented.

Chapter 5: Chapter five summarizes the main issues of this thesis, gives directions for future work and discusses extensions and improvements of the models and algorithms presented.

The bibliography concludes the thesis. All used figures, tables and the most common mathematical symbols are listed before the first chapter.

Chapter 2

Supervised learning from sequential data

Abstract

This chapter discusses some of the basic techniques which are necessary for supervised learning from sequential data and introduces basic concepts, algorithms and notation, which are used in or strongly related to later chapters of this thesis.

As discussed in the introduction chapter, the prediction or recognition of sequences conditioned on another sequence has a number of potential applications. A unification of the core problems of these applications, which can be tackled with similar concepts, and creating a framework that is open to measurable improvements of these, requires an abstract definition of the problem. This includes a consistent notation and a metric that can be used to measure and compare the performance of sequence prediction algorithms, which can then be improved by applying new ideas and concepts.

This chapter first introduces the concept and a notation for the sequence prediction problem which is used throughout the thesis. Then a suitable measure based on a sound statistical technique is discussed. Two approaches to decompose the original problem into smaller independent problems are shown, which make it possible to use established methods from other areas. It is also shown how sequence prediction necessarily always decomposes into a modeling problem for hypothesized sequences and a search part for selecting between them. Challenges and specific problems regarding the influence of context in sequences are discussed. Finally, (Hidden) Markov Models, an important group of models specifically designed to be used for sequence modeling and prediction, are reviewed.

2.1 DEFINITION OF THE PROBLEM

Consider a (time) sequence of *input data vectors* of dimensionality D

$$\mathbf{X} = \mathbf{x}_1^T = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{T-1}, \mathbf{x}_T\} \tag{2.1}$$

from the input sequence data space $\mathcal{X}$ and a sequence of corresponding *output target data vectors* of dimensionality K

$$\mathbf{T} = \mathbf{t}_1^T = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \ldots, \mathbf{t}_{T-1}, \mathbf{t}_T\}, \tag{2.2}$$

in an output sequence data space[1] $\mathcal{Y}$, with neighboring data-pairs (in time) are assumed to be statistically dependent on each other. Given a finite amount of paired sequences $\mathbf{x}_1^T$ and $\mathbf{t}_1^T$ as training data, the aim is to learn the rules to predict the output data given the input data, which is equivalent of finding the mapping $\mathbf{M}$ from the input space to the output space, $\mathcal{X} \xrightarrow{\mathbf{M}} \mathcal{Y}$. The performance of the model that has been trained can be checked on a set of unseen sequences to measure an *error rate* that is yet to be defined. Then the question is:

- How is it possible to build and improve models for sequence prediction in a consistent way?

Treating sequences as *patterns* allows to apply the theory of pattern recognition, which is well described in several books (Duda and Hart, 1974; Fukunaga, 1990; Bishop, 1995). If a sequence is treated as a block (pattern), then the aim becomes to predict correct *and* complete sequences, which is a great simplification of the original problem. A sequence that has one predicted component wrong is as wrong as a sequence that has all components wrong, which seems to be a crude assumption since a human would definitely grade this performance, but models built using this assumption work reasonably well in practice.

The aim of pattern recognition is to minimize the *expected* number of wrong predictions when learning the mapping function $\mathcal{X} \xrightarrow{\mathbf{M}} \mathcal{Y}$. Since in general this mapping is not deterministic (there is *not* exactly one $\mathbf{Y}$ for each $\mathbf{X}$), which could be easily realized by a implementing a look-up table $\mathbf{Y} = f(\mathbf{X})$, and the training data samples in general don't cover the complete space $\mathcal{X}$ and $\mathcal{Y}$, it is appropriate to formulate the problem in an *approximate* manner by introducing *probability distributions* [2] over $\mathcal{X}$ and $\mathcal{Y}$. The problem then becomes to maximize the *posterior probability* of the output given the input. The sequence prediction problem can then be written as:

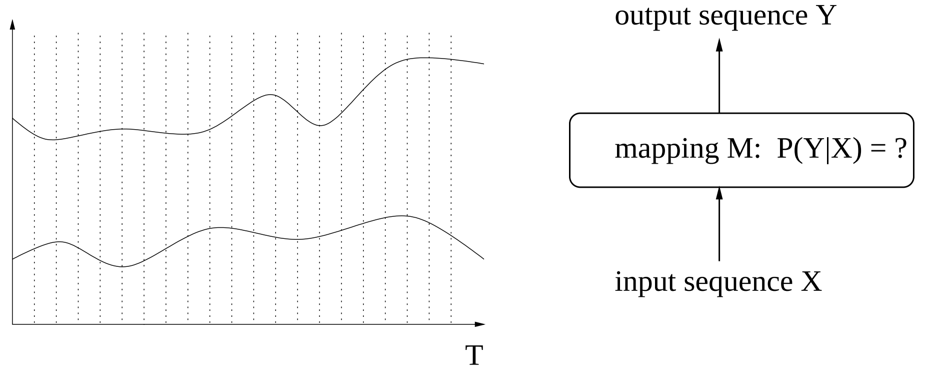
$$\boxed{\mathbf{Y}^\star = \arg\max_{\mathcal{Y}} P(\mathbf{Y}|\mathbf{X})} \tag{2.3}$$

with $\mathbf{X} = \mathbf{x}_1^T$ being the input sequence, $\mathbf{Y} = \mathbf{y}_1^T$ being any valid output sequence and $\mathbf{Y}^\star$ being the predicted sequence with with the highest probability among all sequences. The general problem is visualized in Figure 2.1.

[1] a sample sequence of the *training target data* is denoted as $\mathbf{T}$, while an output sequence in general is denoted as $\mathbf{Y}$, both live in the output space $\mathcal{Y}$

[2] to simplify notation, throughout this thesis random variables and their values, are often *not* denoted as different symbols, when their identity is obvious from the context. This means, $P(\mathbf{x}) = P(X = \mathbf{x})$.

Figure 2.1: Supervised learning from sequences



Visualization of the general problem of supervised learning from sequences, here shown for a mapping with one-dimensional continuous input and one-dimensional continuous output. Each dotted line represents one data pair input/output vector. Inputs and/or outputs can in general be categorical also.

Training of a sequence prediction system corresponds to estimating the probability distribution [3] $P(\mathbf{Y}|\mathbf{X})$ from a number of samples which includes (a) defining an appropriate model $\mathbf{M}$ and (b) estimating its parameters $\mathbf{w}$ by maximizing some predefined optimality criterion. In practice the model $\mathbf{M}$ consists of several modules with each of them being responsible for a different part of $P(\mathbf{Y}|\mathbf{X})$.

Testing (usage) of the trained system or *recognition* for a given input sequence $\mathbf{X}$ corresponds principally to the evaluation of $P(\mathbf{Y}|\mathbf{X})$ for all possible output sequences to find the best one $\mathbf{Y}^\star$. This procedure is called the *search* and its efficient implementation is important for many applications. The search problem is discussed in more detail in chapter 4 of this thesis.

In order to build a model to predict sequences it is necessary to decompose the sequences such that modules responsible for smaller parts can be build. There are two basic possibilities of decomposing the sequence posterior probability $P(\mathbf{Y}|\mathbf{X})$ into smaller parts, that is

- Decomposition into a generative model part and a prior model part, and
- Direct decomposition,

which are discussed in the next two sections.

[3]throughout this chapter there is no distinction made between probability mass and density, usually denoted as P and p, respectively. If the variable $\mathbf{Y}$ to model is categorical, a probability mass is assumed, if it is continuous, a probability density is assumed.

2.2 Decomposition into a generative and a prior model part

Using Bayes' rule $P(B|A) = P(A|B)P(B)/P(A)$ decomposition into two parts is possible as:

$$\begin{aligned}
\mathbf{Y}^\star &= \arg\max_{\mathcal{Y}} P(\mathbf{Y}|\mathbf{X}) \\
&= \arg\max_{\mathcal{Y}} \frac{P(\mathbf{X}|\mathbf{Y})P(\mathbf{Y})}{P(\mathbf{X})} \\
&= \arg\max_{\mathcal{Y}} P(\mathbf{X}|\mathbf{Y})P(\mathbf{Y}) \qquad (2.4)
\end{aligned}$$

For a certain observed input sequence $\mathbf{X}$ the denominator $P(\mathbf{X})$ is constant for any choice of $\mathbf{Y}$ and can therefore be omitted because only the $\mathbf{Y}$ with maximum $P(\mathbf{Y}|\mathbf{X})$ is needed, not the maximum $P(\mathbf{Y}|\mathbf{X})$ itself. The first term in (2.4), the conditional probability of the input $\mathbf{X}$ given the output $\mathbf{Y}$ is called the *likelihood* of an input sequence $\mathbf{X}$ assuming it was generated by a hypothesized output sequence $\mathbf{Y}$. The second term is the *prior* probability of the hypothesized output sequence totally independent of the input.

2.2.1 Context-independent model

Using Bayes' rule and the product rule of probability theory $P(A,B) = P(A)P(B|A)$ the conditional sequence probability $P(\mathbf{Y}|\mathbf{X})$ can for a simple example be broken down to three terms as:

$$\begin{aligned}
\mathbf{Y}^\star &= \arg\max_{\mathcal{Y}} P(\mathbf{X}|\mathbf{Y})P(\mathbf{Y}) \\
&= \arg\max_{\mathcal{Y}} P(\mathbf{x}_1^T|\mathbf{y}_1^T)P(\mathbf{y}_1^T) \\
&= \arg\max_{\mathcal{Y}} \Big[\prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{t-1},\mathbf{y}_1^T)\Big]\Big[\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_{t-1})\Big] \qquad (2.5) \\
&\approx \arg\max_{\mathcal{Y}} \Big[\prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{y}_t)\Big]\Big[\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t-1})\Big] \qquad (2.6) \\
&= \arg\max_{\mathcal{Y}} \Big[\prod_{t=1}^{T} \frac{P(\mathbf{y}_t|\mathbf{x}_t)}{P(\mathbf{y}_t)} P(\mathbf{y}_t|\mathbf{y}_{t-1})\Big] \qquad (2.7)
\end{aligned}$$

making some simplifying approximations. These are for this example:

(1) Every output $\mathbf{y}_t$ depends only on the previous output $\mathbf{y}_{t-1}$ and not on all previous outputs, making it a *first order Markov model*:

$$P(\mathbf{y}_t|\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_{t-1}) \Rightarrow P(\mathbf{y}_t|\mathbf{y}_{t-1}) \qquad (2.8)$$

(2a) The inputs are assumed to be statistically independent in time:

$$P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_1^T) \Rightarrow P(\mathbf{x}_t|\mathbf{y}_1^T) \tag{2.9}$$

(2b) The likelihood of an input vector x_t given the complete output sequence $\mathbf{y}_1^T$ is assumed to depend only on the output found at t and not on any other ones, making it in this simple case a *context-independent* model:

$$P(\mathbf{x}_t|\mathbf{y}_1^T) \Rightarrow P(\mathbf{x}_t|\mathbf{y}_t) \tag{2.10}$$

(2.6) and (2.7) or very similar expressions are the basis for many frequently used approaches to predict sequences. In these equations there are two kinds of probabilistic terms,

- $P(A|B)$: a conditional probability expression, and
- $P(A)$: an unconditional probability expression,

for which efficient types of models, depending on whether the variables are continuous or categorical, are known. Several of these model types will be discussed throughout this thesis. The remaining probability expressions in (2.6) and (2.7) to model are:

(1) $P(\mathbf{x}_t|\mathbf{y}_t)$ – the likelihood of input vector $\mathbf{x}$ given output vector $\mathbf{y}$ at time t. For this case $\mathbf{x}$ is often continuous and $\mathbf{y}$ categorical, which is usually coded such that exactly one component of $\mathbf{y}$ is one and all others zero. Then it is possible to build an *unconditional* model for each category, e.g. $P_k(\mathbf{x})$ for $1 \leq k \leq K$. A popular model for unconditional density estimation is a parametric representation as *Gaussian mixtures*, which is discussed further in section 2.4.3.

(2) $P(\mathbf{y}_t|\mathbf{x}_t)$ – the posterior probability of vector $\mathbf{y}$ given input vector $\mathbf{x}$ at time t. For this type of problem often *neural networks* as discussed in chapter 3 are used as models. Depending on the type of the variable $\mathbf{y}$ (categorical or continuous) certain assumptions are made about the distribution of $\mathbf{y}$, which leads to different types of neural networks.

(3) $P(\mathbf{y}_t)$ – the unconditional prior probability, which in the case of a categorical $\mathbf{y}$ is approximated by a discrete distribution, for example by the relative frequency of observing the possible categories of $\mathbf{y}$ in the training data, and in the case of a continuous $\mathbf{y}$ by a parametric distribution.

(4) $P(\mathbf{y}_t|\mathbf{y}_{t-1})$ – the conditional probability of observing category $\mathbf{y}_t$ given the neighboring category $\mathbf{y}_{t-1}$, also called *transition probability*, which can also be estimated from the training data using the techniques from section 2.4.1 in the case of a categorical variable $\mathbf{y}$.

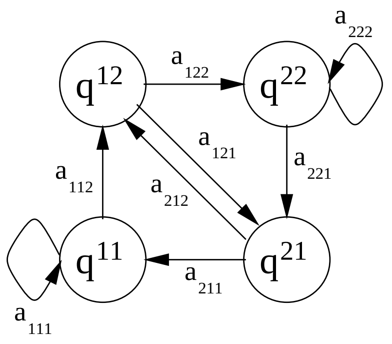
Often the expression $\tilde{P}(\mathbf{x}_t|\mathbf{y}_t) = P(\mathbf{y}_t|\mathbf{x}_t)/P(\mathbf{y}_t)$ is referred to as the *scaled likelihood*, because it is proportional to the real likelihood $P(\mathbf{x}_t|\mathbf{y}_t)$.

2.2.2 Context-dependent model

The approximations used to derive (2.6) ignored all context effects, on the input side by ignoring neighboring vectors of $\mathbf{x}_t$ and on the output side by ignoring the effects of hypothesized neighboring output vectors of $\mathbf{y}_t$. For the listed approximations (1), (2a) and (2b) several improvements are possible and used frequently in real-world applications.

(1) Although the basic model is often a first order Markov model, with the transition probabilities depending only on the previous state, long-span dependencies can be used by artificially increasing the state space. Assume the categorical output variable $\mathbf{y}$ has dimensionality K, representing K categories with K states and K^2 transition probabilities, if first order Markov Models using a history of length one are used and every transition is possible. A model that can differentiate a history of length two can be created by increasing the number of states to K^2 representing all combinations of any two original states, simulating a second order Markov Model in the original state space. The number of possible transitions increases then from K^2 to K^3. Note that the number of possible observation probability distributions associated to states, as introduced in section 2.4.1, does *not* increase. An example is shown in Figure 2.2.

Figure 2.2: Markov Model history length expansion



Expansion of a first order Markov Model (left) to a second order Markov Model (right) to differentiate a history of length two instead one.

(2a) Neighboring inputs $\mathbf{x}_t$,$\mathbf{x}_{t+1}$ are assumed to be statistically independent, that is $P(\mathbf{x}_t,\mathbf{x}_{t+1}) = P(\mathbf{x}_t)P(\mathbf{x}_{t+1})$. This means that by knowledge of any $\mathbf{x}_t$ nothing is known about $\mathbf{x}_{t+1}$. Since this is in general a poor assumption for sequential data, efforts are made to make neighboring input vectors less dependent on each other. One method successfully applied in speech recognition and hand-writing recognition is the extension of the input vectors $\mathbf{x}_t$ at each time t by estimates of the first and second order derivatives of an assumed smooth function that the original sequence $\mathbf{x}_1^T$ describes.

(2b) The likelihood of an input vector is, after applying assumption (2a), conditioned on the complete hypothesized output sequence, which has been approximated by only the output vector at time t for the context-independent model described in section 2.2.1. The likelihood can be made dependent on more context information like neighboring output vectors or groups of output vectors as used in speech recognition to construct for example *triphones* (a phone in the context of two other phones, each modeled by a group of output symbols or states). Assume that for the context-independent model it is required to model the likelihood of a vector given a certain class s as $P(\mathbf{x}|s)$, then for the context-dependent model one would have to model $P(\mathbf{x}|s,\phi)$ if ϕ describes the context of s.

The use of context-dependent models raises a number of issues which are beyond the scope of this introduction. Including additional context usually increases the number of potential output classes which in general has to be reduced by clustering and parameter sharing to have enough training data to robustly estimate the parameters of the associated distributions. A number of such clustering techniques and generation of context-dependent models successfully used for speech recognition are discussed in for example (Bahl et al., 1991) and (Odell, 1995).

In case of an implementation using (2.7), which requires the posterior and prior probability of a class s in its context ϕ, the resulting expression for the scaled likelihood $\tilde{P}(\mathbf{x}|s,\phi)$ is often decomposed as

$$\begin{aligned}\tilde{P}(\mathbf{x}|s,\phi) &= \frac{P(s,\phi|\mathbf{x})}{P(s,\phi)} \\ &= \frac{P(\phi|s,\mathbf{x})}{P(\phi|s)} \cdot \frac{P(s|\mathbf{x})}{P(s)}, \qquad (2.11)\end{aligned}$$

which results in four independent probability expressions that can be estimated individually. A more detailed decomposition consisting of more terms is a straightforward extension of (2.11). A very appealing method to discriminate thousands of classes by automatically growing a tree structure, where decomposition at each node in the tree is based on the basic idea of decomposing posterior probabilities, has been applied successfully to a speech recognition problem and is described in (Fritsch, 1998a) and (Fritsch, 1998b).

A new modeling approach which removes the independence assumption (2a) and the context-dependency assumption (2b) completely, using neural networks as models, is shown in section 3.3.

2.3 Direct decomposition

Another possibility to split up the posterior sequence probability $P(\mathbf{Y}|\mathbf{X})$ is to use the rule $P(A,B) = P(A)P(B|A)$ for decomposing joint probability expressions directly T

times to yield:

$$
\begin{aligned}
\mathbf{Y}^\star &= \arg\max_{\mathcal{Y}} P(\mathbf{Y}|\mathbf{X}) \\
&= \arg\max_{\mathcal{Y}} P(\mathbf{y}_1^T|\mathbf{x}_1^T) \\
&= \arg\max_{\mathcal{Y}} \underbrace{\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t+1},\mathbf{y}_{t+2},\ldots,\mathbf{y}_T,\mathbf{x}_1^T)}_{\text{backward posterior probability}} \qquad (2.12) \\
&= \arg\max_{\mathcal{Y}} \underbrace{\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_{t-1},\mathbf{x}_1^T)}_{\text{forward posterior probability}} \qquad (2.13)
\end{aligned}
$$

Now the posterior sequence probability $P(\mathbf{Y}|\mathbf{X})$ has been broken down into a product of conditional probabilities of output vectors $\mathbf{y}_t$ given the complete input sequence $\mathbf{x}_1^T$ plus the output vectors on either the left hand side of t, that is $\{\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_{t-1}\}$, or on the right hand side of t, that is $\{\mathbf{y}_{t+1},\mathbf{y}_{t+2},\ldots,\mathbf{y}_T\}$. These decompositions have the disadvantage that no parts of $P(\mathbf{Y}|\mathbf{X})$ have completely been separated as done in section 2.2, which limits the flexibility of this approach. The advantage is though, that, at least up to this point, no approximations have been made to decompose $P(\mathbf{Y}|\mathbf{X})$ – the decomposition results from a strict application of probability rules. For a practical implementation it will of course be necessary to make approximations by cutting input and/or output sequences appropriately. Note that the two decompositions shown here are not the only ones possible, because the sequence can be broken down in any order and not necessarily sequentially from left to right or right to left.

This decomposition will be explored in chapter 3, where a neural network structure is developed, that exactly models expression 2.12 or 2.13 to allow the estimation of conditional sequence probabilities by a single neural network.

2.4 HIDDEN MARKOV MODELS

Hidden Markov Models (HMMs) are simple, but very powerful and successful models for the prediction of categorical symbol sequences. They are the basis for many applications like for example speech recognition and on-line hand-writing recognition. The main reason for their success is the fact, that they can not only model *given* sequences reasonably well, but their use also greatly simplifies the *search* to *predict* sequences (Viterbi-Search).

This section introduces the basic terminology and algorithms which are related to this thesis. There exist a number of well written tutorials on HMMs (Huang et al., 1990; Rabiner and Juang, 1993), so here only a short introduction is given. The notation for HMMs used in this section conforms mainly with that found in (Young et al., 1997) and (Odell, 1995).

2.4.1 Basic HMM formulation

An HMM is a model for a random process generating a sequence of vectors that can be described by a progression through a *hidden state sequence* to model exactly probabilistic expressions of the form (2.6), and are therefore well suited for the prediction of sequences. Elements of HMMs are a number of J *states* $Q = \{q^1, q^2, \ldots, q^J\}$ which are connected by directed arcs, called *transitions*. They are called *first order* HMMs, when each transition depends only on the previous state and not on more states. Each state allows the calculation of a likelihood for a certain *observation vector* (input vector) $\mathbf{x}$, which is called the *observation likelihood*, notated as $b_i(\mathbf{x}) = P(\mathbf{x}|q^i)$ and often jointly notated as $\mathbf{B}$. The probability distributions $b(\cdot)$ can be discrete or continuous and must obey the rules of probability, i.e. for discrete distributions

$$\sum_{\mathcal{X}} b_i(\mathbf{x}) = 1 \tag{2.14}$$

and

$$\int_{\mathcal{X}} b_i(\mathbf{x})\ d\mathbf{x} = 1 \tag{2.15}$$

for continuous distributions. For simpler notation it is sometimes convenient to think of an *initial state* q^I and a final *exit state* q^E which do not have an observation probability distribution $b(\cdot)$ associated with them and are therefore *non-emitting states*. The connecting arcs have probabilities associated that are called *transition probabilities*, which are notated as a_{ij} for the transition probability from state i to state j and collected in a transition matrix $\mathbf{A}$. Transitions can also be *self-transitions* describing the probability of staying in a state for one time-step. All transitions from a certain state i have to sum up to one:

$$\sum_{j=1}^{J} a_{ij} = 1 \tag{2.16}$$

A typical single HMM representing a certain symbol, notated here as $\mathbf{M} = \{\mathbf{A}, \mathbf{B}\}$, as often used in speech recognition is shown in Figure 2.3.

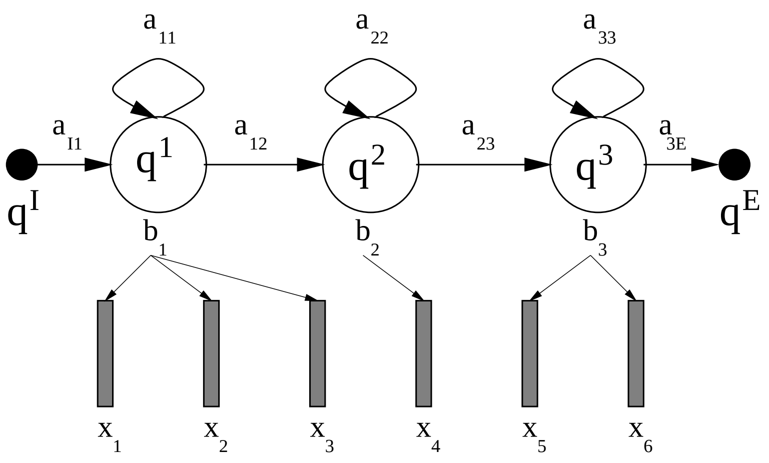
An observed input sequence $\mathbf{x}_1^6 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$ could have been generated by the HMM in the example by first staying three time steps in state q^1 for the generation of input vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$, then one time step in state q^2 to generate $\mathbf{x}_4$, and finally two time steps in state q^3 to generate $\mathbf{x}_5$ and $\mathbf{x}_6$. The likelihood that this particular state sequence was generated by the HMM model $\mathbf{M}$ can be calculated as:

$$P(\mathbf{x}_1^6|q^1q^1q^1q^2q^3q^3, \mathbf{M}) = a_{I1}b_1(\mathbf{x}_1)a_{11}b_1(\mathbf{x}_2)a_{11}b_1(\mathbf{x}_3)a_{12}b_2(\mathbf{x}_4)a_{23}b_3(\mathbf{x}_5)a_{33}b_3(\mathbf{x}_6)a_{3E}$$

This is only one possibility for an *alignment* of input vectors to states. Many more possible different progressions generating the same sequence of vectors $\mathbf{x}_1^6$ are possible, which can all be calculated in a similar manner and are in general:

$$P(\mathbf{X}|Q, \mathbf{M}) = a_{I1} \prod_{t=1}^{T} b_{q(t)}(\mathbf{x}_t)\ a_{q(t)q(t+1)} \tag{2.17}$$

Figure 2.3: Example Hidden Markov Model structure



Example for a typical Hidden Markov Model structure as often used in speech recognition.

for a single state sequence. The sum over all possible state sequences through the model (which are not observed or *hidden*) for a certain input sequence $\mathbf{X}$ is the *full likelihood* for the generation of the sequence by the HMM model $\mathbf{M}$ and can be written as:

$$P(\mathbf{X}|\mathbf{M}) = \sum_Q P(\mathbf{X}|Q,\mathbf{M}) \tag{2.18}$$

$$= \sum_Q a_{I1} \prod_{t=1}^{T} b_{q(t)}(\mathbf{x}_t)\ a_{q(t)q(t+1)} \tag{2.19}$$

An alternative to calculating the full likelihood is the likelihood of only the single best state sequence, that is:

$$\widehat{P}(\mathbf{X}|\mathbf{M}) = \underset{Q}{\mathrm{MAX}} \left\{ a_{I1} \prod_{t=1}^{T} b_{q(t)}(\mathbf{x}_t)\ a_{q(t)q(t+1)} \right\} \tag{2.20}$$

Direct evaluation of these expressions is infeasible, but an efficient recursive algorithm to calculate them is known and described in section 2.4.2.a.

There are two important problems to be solved to make use of HMMs as models for the prediction of sequences.

Problem 1: How do we train the model? Given the observation input sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ and the HMM model $\mathbf{M} = \{\mathbf{A}, \mathbf{B}\}$, how do we adjust the model parameters $\mathbf{A}, \mathbf{B}$ to maximize $P(\mathbf{X}|\mathbf{M})$?

Problem 2: How do we use the model? Given the observation input sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ and the HMM model $\mathbf{M} = \{\mathbf{A}, \mathbf{B}\}$ how do we efficiently find the single state sequence that is *most* probable?

The idea for training HMMs runs roughly as follows: Assume a *best* state sequence for every training pattern is known, that is, every observation vector $\mathbf{x}$ belongs to a certain state q that is was generated by. Then all observation vectors for a particular state q^i could be collected to update the parameters of its observation probability distribution $b_i(\cdot)$, such that the emission of the observation vectors by this distribution becomes most likely for that state. Knowledge of this alignment from observation vectors to states would also imply how to set the transition probabilities $\mathbf{A}$ as relative frequencies $a_{ij} = N(q^i q^j)/N(q^i)$ by counting from the aligned training data.

The problem is that a *best* state sequence is not known, because a particular observation vector sequence could have been generated by many state sequences as discussed above. In that case the assignment to states becomes fuzzy and a smooth value for the counts would replace the original alignments, which then can be used to update the observation distributions and transition probabilities. An algorithm called the *Forward-Backward algorithm* to calculate these statistics efficiently is explained in section 2.4.2.a.

Usage of HMMs for *recognition* (prediction) or a simplified training procedure requires a decision for a certain set of models to progress through, which generated the observed vector sequence. In the simplest case the progression through the set of models is approximated by the best single state sequence that has the highest probability, which can be understood as a simplification of the Forward-Backward algorithm.

2.4.2 Calculation of state occupation probabilities

Updating the parameters of the model requires the calculation of a *state occupation probability* $\gamma_i(t)$, the probability of the transition from the initial state q^I to the final exit state q^E and being in state q^i at time t given the model $\mathbf{M}$ and the observed vector sequence $\mathbf{x}_1^T$, i.e.

$$\gamma_i(t) = P(q^I \rightarrow (q_t = q^i) \rightarrow q^E | \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, \mathbf{M}), \qquad (2.21)$$

which is necessary to update the observation distribution parameters $\mathbf{B}$. A *state pair occupation probability* $\gamma_{ij}(t)$, the probability of the transition from the initial state q^I to the final exit state q^E and being in state q^i at time t and and state q^j at time $t+1$, i.e.

$$\gamma_{ij}(t) = P(q^I \rightarrow (q_t = q^i, q_{t+1} = q^j) \rightarrow q^E | \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, \mathbf{M}), \qquad (2.22)$$

is necessary to update the transition probabilities $\mathbf{A}$. These probabilities can exactly be calculated efficiently using the *Forward-Backward algorithm*, which is explained in the next section. An often used approximation is the *Viterbi algorithm* (section 2.4.2.b) that is also used for recognition.

2.4.2.a Forward-backward algorithm

Consider the two intermediate probability expressions:

- the *forward likelihood* of the joint event of starting in state q^I and being in state q^i at time t after observing the partial sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$, given the model $\mathbf{M}$:
$$\alpha_i(t) = P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, q^I \to (q_t = q^i)|\mathbf{M}), \tag{2.23}$$
- and the *backward likelihood* of observing the partial sequence $\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_T$, given the model $\mathbf{M}$ and that the state sequence starts in state q^i at time t and ends in state q^E:
$$\beta_i(t) = P(\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_T|(q_t = q^i) \to q^E, \mathbf{M}), \tag{2.24}$$
which is not symmetrical to the forward likelihood.

These expressions can efficiently be calculated recursively for all J emitting states with the *Forward algorithm* as

$$\begin{aligned} \alpha_j(1) &= a_{Ij} b_j(\mathbf{x}_1) \\ \alpha_j(1 < t \leq T) &= \sum_{i=1}^{J} \alpha_i(t-1) a_{ij} b_j(\mathbf{x}_t) \\ \alpha_J(T^+) &= \sum_{i=1}^{J} \alpha_i(T) a_{iE}, \end{aligned} \tag{2.25}$$

and in a similar way with the *Backward algorithm* as:

$$\begin{aligned} \beta_i(T) &= a_{iE} \\ \beta_i(1 \leq t < T) &= \sum_{j=1}^{J} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_j(t+1) \\ \beta_1(1^-) &= \sum_{j=1}^{J} a_{Ij} b_j(\mathbf{x}_1) \beta_j(1) \end{aligned} \tag{2.26}$$

Here is enforced that any state progression starts in the initial state q^I and ends in the final exit state q^E. The time indices $t = 1^-$ and $t = T^+$ are a convenient notation for the boundary indices, with $t = 1^-$ being the time index before $t = 1$ and $t = T^+$ being the time index after $t = T$. If (2.25) and (2.26) are implemented exactly as notated, the recursively used multiplications will lead to a numerical underflow. Therefore either an implementation based on logarithms or some normalization of the $\alpha_j(\cdot)$ and $\beta_i(\cdot)$ after each time step must be used.

The full likelihood $\mathcal{L} = P(\mathbf{X}|\mathbf{M})$ based on the definitions of the forward and backward likelihoods $\alpha_i(t)$ (2.23) and $\beta_i(t)$ (2.24) is then

$$\begin{aligned}
\mathcal{L} = P(\mathbf{X}|\mathbf{M}) &= P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, q^I \rightarrow (q_{T^+} = q^E)|\mathbf{M}) \\
&= \alpha_J(T^+) \qquad (2.27) \\
&= P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, (q_{1^-} = q^I) \rightarrow q^E|\mathbf{M}) \\
&= \beta_1(1^-) \qquad (2.28)
\end{aligned}$$

One possible expression for the state occupation probability $\gamma_j(t)$ is therefore using (2.21):

$$\begin{aligned}
\gamma_j(t) &= P(q^I \rightarrow (q_t = q^j) \rightarrow q^E|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, \mathbf{M}) \\
&= \frac{P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, q^I \rightarrow (q_t = q^j) \rightarrow q^E|\mathbf{M})}{P(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, q^I \rightarrow q^E|\mathbf{M})} \\
&= \frac{\alpha_j(t)\beta_j(t)}{\mathcal{L}} \qquad (2.29)
\end{aligned}$$

The state pair occupation probability $\gamma_{ij}(t)$ is then using (2.22):

$$\gamma_{ij}(t) = \frac{\alpha_i(t) a_{ij} b_j(\mathbf{x}_{t+1}) \beta_j(t+1)}{\mathcal{L}} \qquad (2.30)$$

2.4.2.b Viterbi algorithm

The *Viterbi algorithm* is a simplification of the Forward-Backward algorithm, that leads to the likelihood $\widehat{P}(\mathbf{X}|\mathbf{M})$ of the single best state path through the model. The path itself, like necessary for recognition, can easily be reconstructed with little computational overhead.

Instead of summing over all possible states at each time t, a decision is made and only the best local path is considered. If $\phi_j(t)$ is the likelihood of the single most likely state sequence ending in state q^j at time t, then:

$$\begin{aligned}
\phi_j(1) &= a_{Ij} b_j(\mathbf{x}_1) \\
\phi_j(1 < t \leq T) &= \operatorname*{MAX}_{i=1}^{J} \{\phi_i(t-1) a_{ij}\} b_j(\mathbf{x}_t) \qquad (2.31) \\
\phi_J(T^+) &= \operatorname*{MAX}_{i=1}^{J} \{\phi_i(T) a_{iE}\}
\end{aligned}$$

According to the definition of $\phi_j(t)$ and (2.20) the Viterbi likelihood is then:

$$\widehat{P}(\mathbf{X}|\mathbf{M}) = \phi_J(T^+) \qquad (2.32)$$

The best state sequence can be reconstructed by a *trace-back* giving the best state number for each time t. This requires that the maximum decision made at each time during the forward progression is recorded as:

$$\chi_j(1 < t \leq T) = \arg\max_{i=1}^{J}\{\phi_i(t-1)a_{ij}\} \tag{2.33}$$

$$\chi_J(T^+) = \arg\max_{i=1}^{J}\{\phi_i(T)a_{iE}\}, \tag{2.34}$$

which can then be back-traced to recover the most likely state-sequence as:

$$q(T) = \chi_J(T^+) \tag{2.35}$$

$$q(1 \leq t < T) = \chi_{q(t+1)}(t+1) \tag{2.36}$$

The use of the Viterbi algorithm for training leads to state occupation probabilities $\gamma_j(t)$ of either one or zero for any state q^j at time t, because only a single path and no weighted average of paths is considered. The same is true for the state pair occupation probabilities.

Note that the computational complexity of finding the best state sequence *guaranteed* is only of order $O(J^2T)$ (by examining (2.31), for each t and for each state q^j the maximum over J states has to be found) and not of order $O(J^TT)$, which would define an exhaustive search over all possible state sequences for a fully connected HMM (for each of the J^T state combinations T operations would be necessary to calculate the likelihood, the backtrace operations are ignored). This improvement in efficiency for the search is one of the main reasons for the success of Hidden Markov Models in many applications.

2.4.3 Parameter estimation for output probability distributions

For the output probability distributions $b(\cdot)$ to generate the likelihoods many different parametric and non-parametric approaches are possible. The most common are

- likelihoods from continuous density distributions with Gaussian kernels,
- likelihoods from discrete distributions and
- scaled likelihoods by posterior conversion,

which are discussed in the next sections.

2.4.3.a Continuous density likelihoods

Continuous density likelihoods come often from a mixture of M basic functions

$$p(\mathbf{x}|q = q^j) = b_j(\mathbf{x}) = \sum_{m=1}^{M} c_{jm} b_{jm}(\mathbf{x}) \tag{2.37}$$

with Gaussian kernels

$$\begin{aligned} b_{jm}(\mathbf{x}) &= \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}) \\ &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_{jm}|^{1/2}} exp\Big\{ -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_{jm})'\boldsymbol{\Sigma}_{jm}^{-1}(\mathbf{x}-\boldsymbol{\mu}_{jm})\Big\} \qquad (2.38) \\ & \qquad (2.39) \end{aligned}$$

where the dimensionality of the data is D and c_{jm}, $\boldsymbol{\mu}_{jm}$ and $\boldsymbol{\Sigma}_{jm}$ being the *mixture weights*, *means* and *covariance matrices* for the m-th component of the Gaussian mixture for state q^j. The parameters of the model can be estimated efficiently using the expectation-maximization (EM) algorithm (McLachlan and Krishnan, 1997) to maximize the likelihood of the data. The maximum likelihood estimates for observation distributions of Markov chains using the state occupation probabilities are weighted averages as

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} \gamma_{jm}(t)}{\sum_{t=1}^{T} \gamma_j(t)} \qquad (2.40)$$

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{t=1}^{T} \gamma_{jm}(t) \cdot \mathbf{x}_t}{\sum_{t=1}^{T} \gamma_{jm}(t)} \qquad (2.41)$$

$$\begin{aligned} \hat{\boldsymbol{\Sigma}}_{jm} &= \frac{\sum_{t=1}^{T} \gamma_{jm}(t) \cdot (\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jm})(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jm})'}{\sum_{t=1}^{T} \gamma_{jm}(t)} \\ &= \frac{\sum_{t=1}^{T} \gamma_{jm}(t) \cdot \mathbf{x}_t\mathbf{x}_t'}{\sum_{t=1}^{T} \gamma_{jm}(t)} - \hat{\boldsymbol{\mu}}_{jm}\hat{\boldsymbol{\mu}}_{jm}' \qquad (2.42) \end{aligned}$$

where $\mathbf{x}_t$ is the tth vector of the training data sequence and $\gamma_{jm}(t)$ is the probability that observation vector $\mathbf{x}_t$ was produced by mixture m of state q^j given by

$$\gamma_{jm}(t) = \gamma_j(t)\frac{c_{jm}b_{jm}(\mathbf{x}_t)}{b_j(\mathbf{x}_t)}. \qquad (2.43)$$

This set of equations, which is easily generalized to deal with multiple training sequences, is called in combination with the Forward-Backward algorithm to calculate the state occupation probabilities *Baum-Welch re-estimation* (Baum et al., 1970; Baum, 1972), which is a form of the EM algorithm. In general the data used for the M-step in (2.40), (2.41) and (2.42) is the same as for the E-step in (2.43), but it is sometimes useful to estimate parameters of new distributions in the M-step based on state occupation probabilities from well trained models from the E-step. The expressions (2.40) – (2.43) can be found in similar form in (Young et al., 1997) (pages 138-143), (Huang et al., 1990) or in (Rabiner and Juang, 1993).

2.4.3.b Discrete likelihoods

For discrete likelihoods all state distributions are based on a number of M *codebook vectors* usually found by *vector-quantizing* (Gray, 1984) the observations, such that the sequence of continuous vectors becomes a sequence of categorical symbols. Observation vectors are assigned to the closest codebook vector usually based on an Euclidean distance measure.

Because discrete distributions are implemented as look-up tables, they can be evaluated quickly, which is their most important advantage. Their disadvantage is, that the vector quantizer introduces additional noise into the observations, which usually leads to suboptimal results. Further discussion of models using discrete likelihoods can be found in (Huang et al., 1990).

For discrete distributions the relative codebook vector frequencies for each model must be estimated based on the state occupation probabilities. Suitable expressions can be derived by thinking of a discrete distribution as a mixture distribution over *all* available mixture components in the model, with the number of mixtures being the number of codebook vectors. This leads then to expressions similar to (2.40) and (2.43).

2.4.3.c Scaled likelihoods

(2.7) suggests an alternative way of obtaining likelihoods of the form $P(\mathbf{x}|q = q^j)$. If the posterior probability of the state given the observation vector $P(q = q^j|\mathbf{x})$ is known, then a *scaled likelihood* can be calculated using Bayes' rule by division through the prior probability $P(q = q^j)$ of that state (and ignoring the prior probability $P(\mathbf{x})$, because it is constant for a given observation sequence $\mathbf{x}_1^T$), i.e.

$$P(\mathbf{x}|q = q^j) \propto \frac{P(q = q^j|\mathbf{x})}{P(q = q^j)}, \tag{2.44}$$

which is an equivalent expression for the determination of the most likely output sequence. This approach can be used when suitable models for the estimation of the posterior probability are available, e.g. *neural networks* as sometimes used for speech recognition, or other non-parametric models whose outputs can be interpreted as posterior probabilities.

For distributions found by likelihood conversion to scaled likelihoods usually only the Viterbi algorithm is used for training (Robinson et al., 1996), giving a distinct best state sequence for every training sequence, although training based on a soft alignment is also possible (Senior and Robinson, 1996). Parameter estimation for these distributions means to train a separate model to estimate the posterior probability $P(q = q^j|\mathbf{x})$, for which often neural networks are used. This is discussed in more detail in chapter 3. The maximum likelihood estimates for the prior probabilities $P(q = q^j)$ are in this case the relative frequencies of the states in the training data.

2.4.4 Parameter estimation for transition probabilities

A formula similar to (2.40) is used to update the transition probabilities using the state pair occupation probabilities $\gamma_{ij}(t)$ as

$$a_{ij} = P(q_t = q^j | q_{t-1} = q^i) = \frac{\sum_{t=1}^{T} \gamma_{ij}(t)}{\sum_{t=1}^{T} \gamma_i(t)}, \tag{2.45}$$

which is easily generalized for multiple utterances.

2.5 SUMMARY

This chapter has outlined some basic techniques and models for supervised learning from sequences. Two different approaches to decompose the posterior probability of a complete sequence conditioned on an input sequence have been shown and discussed. Basics of Hidden Markov Models, an important type of model for sequence prediction, have been reviewed. Further chapters will develop some of the shown approaches – an improved model to learn from sequences (chapter 3) to model expressions of the form $P(\mathbf{y}_t|\mathbf{x}_1^T)$, a model to directly estimate the posterior probability of sequences (chapter 3) by modeling expressions of the form $P(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_1, \mathbf{x}_1^T)$, a model to estimate expressions of the form $P(\mathbf{x}_t|\mathbf{y}_1^T)$ and $P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_1, \mathbf{y}_1^T)$ (chapter 3) and an extension of the basic Viterbi algorithm such that it is suitable for large vocabulary speech recognition (chapter 4).

Chapter 3

Neural networks for supervised learning from sequences

Abstract

In this chapter, various approaches of supervised learning from sequences using artificial neural networks are discussed. First, the basics of neural networks, commonly used architectures and their problems with respect to sequence processing are reviewed. It is shown why recurrent neural networks are particularly useful for supervised learning from sequential data. Then, a new architecture based on a recurrent neural network, is introduced and evaluated for uni-modal regression and classification problems assuming that the output data within the sequence is statistically independent. To deal with statistically dependent output data, a variation of this architecture is developed. Finally, the architecture is extended to model target sequences which can be described by multi-modal continuous distributions.

Many classification and regression problems of engineering interest are currently solved with statistical approaches using the principle of "learning from examples". For a certain *model* with a given *structure*, inferred from the prior knowledge about the problem and characterized by a number of *parameters*, the aim is to estimate these parameters accurately and reliably using a finite amount of training data to estimate the target data's underlying distribution, which is conditioned on the input data. In general, the parameters of the model are determined by a supervised training process, while the structure of the model is defined in advance. Choosing a proper structure for the model is often the only way for the designer of the system to put in prior knowledge about the solution of the problem.

Artificial neural networks (NNs) are one group of models, which take the principle "infer the knowledge from the data" to an extreme, since the structure is in general less specified than for other types of models. This chapter deals with NN structures for one particular class of problems which are represented by temporal sequences of input-output data pairs. For these types of problems, which occur for example in speech recognition, time series prediction, dynamic control systems, etc., one of the challenges is to choose an appropriate network structure which at least theoretically is able to use all available input information to predict a point in the output space.

Many NN structures have been proposed in the literature to deal with time varying patterns. Multi-layer perceptrons (MLPs) have the limitation that they can only deal with static data patterns (i.e., input patterns of a predefined dimensionality), which requires to define the size of the input window in advance. (Waibel, 1989) have pursued time delay neural networks (TDNNs), which have proven to be a useful improvement over regular MLPs in many applications. The basic idea of a TDNN is to tie certain parameters in a regular MLP structure without restricting the learning capability of the NN too much. Recurrent neural networks (RNNs) (Rumelhart et al., 1986; Giles et al., 1994; Pearlmutter, 1989; Robinson, 1994; Robinson et al., 1996) provide another alternative for incorporating temporal dynamics and are discussed in more detail in a later section.

In this chapter, several different NN structures for incorporating temporal dynamics are investigated. A number of experiments using both artificial and real-world data are conducted. The superiority of RNNs over the other structures is demonstrated. After pointing out some of the limitations of RNNs, a modified version of an RNN (called a bidirectional recurrent neural network) is proposed, which overcomes these limitations. Various extensions of the bidirectional structure and their potential applications are discussed.

3.1 BASICS OF NEURAL NETWORKS

Artificial neural networks (see (Bishop, 1995) for an excellent introduction) can be used for many supervised learning tasks. Given as training data N input/target data vector pairs $\mathbf{D} = \{\mathbf{x}_n, \mathbf{t}_n\}$ (a mapping from input to a target data), with dimensions D and K, respectively, the aim of a supervised learning process is to learn how to predict output data given new input data, which is written as a K-dimensional function $y^k(\mathbf{x}_n; \mathbf{w})$ depending on the current input vector $\mathbf{x}$ and the NN parameter value vector $\mathbf{w}$ with W weights. The weights are combined in *structures*, whose different types are discussed in more detail in a later section.

3.1.1 Parameter estimation by maximum likelihood

The estimation of the parameters of the model is often guided by the *maximum likelihood* principle, which can be stated as follows:

There is given training data $\mathcal{D}$ and a model structure $\mathbf{M}$, which is characterized by the parameter vector $\mathbf{w}$ out of all possible parameter vectors in the space $\mathcal{W}$, and all parameter value sets are assumed to be equally probable

$$P(\mathbf{w}) = const. \tag{3.1}$$

Then the goal of the parameter estimation process is to find a single set of parameter values $\mathbf{w}^\star$ that maximize the probability of the model parameters given the data, which is called MAP (maximum a posteriori) estimation. Given assumption (3.1) and the fact that the unconditional probability of the data $\mathcal{D}$ is independent of the parameters $\mathbf{w}$ this can be simplified to

$$\begin{aligned}
\mathbf{w}^\star &= \arg\max_{\mathbf{w}}\{P(\mathbf{w}|\mathcal{D})\} && (3.2)\\
&= \arg\max_{\mathbf{w}}\left\{\frac{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})}{P(\mathcal{D})}\right\} && (3.3)\\
&= \arg\max_{\mathbf{w}}\{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})\} && (3.4)\\
&= \arg\max_{\mathbf{w}}\{P(\mathcal{D}|\mathbf{w})\} && (3.5)
\end{aligned}$$

such that the problem becomes now to maximize the *likelihood* of the data given the model with its parameters $\mathbf{w}$ without having a preference for certain parameter sets that seem to be more plausible than others ($\Rightarrow p(\mathbf{w}) = const.$).

In the case of supervised learning with training data $\mathbf{D} = \{\mathbf{x}_n, \mathbf{t}_n\}$, whose data pairs are assumed to be conditionally independent given $\mathbf{w}$, the likelihood $\mathcal{L}$ becomes

$$\begin{aligned}
\mathcal{L} &= P(\mathcal{D}|\mathbf{w}) && (3.6)\\
&= \prod_N P(\mathbf{t}_n, \mathbf{x}_n|\mathbf{w}) && (3.7)\\
&= \prod_N P(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w})P(\mathbf{x}_n|\mathbf{w}) && (3.8)\\
&\propto \prod_N P(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w}), && (3.9)
\end{aligned}$$

because for neural networks as considered in this thesis the inputs are not modeled and do not depend on $\mathbf{w}$, the term $\prod_N P(\mathbf{x}_n|\mathbf{w})$ is constant for all $\mathbf{w}$ and therefore vanishes during maximization of $\mathcal{L}$. With the negative logarithm taken, the function becomes

$$\begin{aligned}
E(\mathcal{D}, \mathbf{w}) &= -ln \prod_N P(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w}) && (3.10)\\
&= -\sum_N ln\ P(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w}) && (3.11)
\end{aligned}$$

which is called an *objective function* or *error function* that has to be minimized during training. Making appropriate assumptions for the conditional output distribution $P(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w})$ makes it possible to classify the types of problems to be solved by neural networks, which is discussed in the next section.

3.1.2 Problem classification

Problems suitable to be solved with neural networks can be divided into three groups depending on the type of input and target variables. Inputs and targets can, in general, be continuous and/or categorical variables, which defines the two categories of supervised learning problems. When targets are continuous, the problem is known as a *regression problem*, when they are categorical (class labels), the problem is known as a *classification problem*. Regression can again be subdivided into *uni-modal regression* and *multi-modal regression*, depending on whether the output can be characterized by a distribution with only one mode (usually a Gaussian) or requires one with many modes (mixture of Gaussians), respectively. Within this chapter, the term *prediction* is used as a general term which includes classification and both types of regression.

3.1.2.a Uni-modal regression

For uni-modal regression or *function approximation*, the components of the output vectors are continuous variables. The NN parameters are estimated to minimize some predefined error criterion, e.g. maximize the likelihood of the target data given the model and the input data

$$P(\mathcal{D}|\mathbf{w}) \propto \prod_{n=1}^{N} P(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w}) \tag{3.12}$$

as discussed above. When it is assumed that

a) the distribution of the errors between the desired target and the estimated output vectors is a single Gaussian with zero mean and a global data-dependent variance and

b) all K outputs are conditionally independent,

then the likelihood criterion reduces to the convenient Euclidean distance measure between the desired and the estimated output vectors or the *squared-error function*,

$$E = \sum_{N} \sum_{K} (y^k(\mathbf{x}_n; \mathbf{w}) - t_n^k)^2 \tag{3.13}$$

which has to be minimized during training. It has been shown several times (Bishop, 1995) that neural networks can estimate the conditional average of the desired target vectors at their network outputs; i.e., $y^k(\mathbf{x}; \mathbf{w}^\star) = \langle t^k|\mathbf{x}\rangle$, where $\langle\cdot\rangle$ is an expectation operator and $\mathbf{w}^\star$ is the parameter (weight) vector at the minimum of the error function. For uni-modal regression the network outputs can be interpreted as the mean of a Gaussian distribution, which varies depending on the current input.

3.1.2.b Multi-modal regression

For multi-modal regression, the components of the output vectors are, as in the uni-modal case, continuous variables, and the parameters are estimated to maximize the likelihood of the target data. The difference to uni-modal regression is that the distribution of errors between the desired target and the estimated output vectors is not assumed to be a single Gaussian, but a weighted sum of Gaussians with a given number of mixture components and a given covariance type as discussed for a simple case in (Bishop, 1995). The objective function to be maximized is here the full likelihood of the target data given the input data and the model.

$$E = \sum_N ln\ P(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w}) \tag{3.14}$$

For multi-modal regression the network outputs represent all parameters of the assumed distribution, i.e. mixture weights, mean vectors and covariance matrices. This case is discussed in more detail in section 3.3.

3.1.2.c Classification

In the case of a classification problem, one seeks the most probable class out of a given pool of K classes for each input vector $\mathbf{x}_n$. To make this kind of problem suitable to be solved by a NN, the categorical target variables are usually coded as vectors as follows. Consider that k is the desired class label for an input vector $\mathbf{x}$. Then construct a K-dimensional target vector $\mathbf{t}$ such that its kth component is one and all other components are zero. The kth component can be interpreted as the probability of $\mathbf{x}$ belonging to class k. The target vectors $\mathbf{t}_n$ constructed in this manner along with the input vectors $\mathbf{x}_n$ can be used to train the NN under some optimality criterion, usually the *cross-entropy function*,

$$E = -\sum_N \sum_K t_n^k ln(y^k(\mathbf{x}_n; \mathbf{w})) \tag{3.15}$$

which results from a maximum likelihood estimation assuming a multinomial target distribution (Bishop, 1995). It has been shown (Richard and Lippman, 1991; Bishop, 1995) that the kth network output can be interpreted as an estimate of the conditional posterior probability of class membership ($y^k(\mathbf{x}; \mathbf{w}^\star) = P(C = k|\mathbf{x})$), with the quality of the estimate depending on the amount of training data and the complexity of the network.

3.1.3 Neural network training

Training of neural networks as discussed here is equivalent to adjusting the weights $\mathbf{w}$ iteratively such that an error function is minimized, which is depending on the type of problem either the squared error, the cross-entropy or the general likelihood of the target data. Function minimization is a problem occuring in many disciplines of science and standard procedures are well documented (Bishop, 1995; Press et al., 1992; Battiti, 1992). Usual approaches for neural networks are

(1) first order methods, which use the first derivative of the error function ($\frac{\partial}{\partial \mathbf{w}} E$) to be minimized, for example gradient descent, gradient descent with momentum, RPROP, and

(2) second order methods (Shepherd, 1997), which also use the second derivative (Hessian) or approximations to it, for example quasi-Newton, conjugate gradient, Levenberg-Marquardt, BPQ (Saito and Nakano, 1997) or Quickprop (Fahlmann, 1988).

The first (and also second) derivative of the error function in feed-forward neural networks can be calculated efficiently with a procedure called *back-propagation* (Rumelhart et al., 1986; Bishop, 1995), which requires a *forward pass* (calculate $y^k(\mathbf{x}_n;\mathbf{w})\ \forall\ k$) and a *backward pass* (calculate $\frac{\partial}{\partial \mathbf{w}} E$ vector) through the network for each of the N training vector pairs. All training procedures can be

(1) *off-line* or *batch* methods, for which the weights $\mathbf{w}$ are updated after all N training samples have been used to calculate the first derivative, or

(2) *on-line* methods, for which only a part of the training samples is used to get an estimate of the first derivative which is then used to update the weights.

The use of neural networks for large scale real-world problems like for example speech recognition adds two practical problems to training: (1) The number of parameters W (weights) is considerably high, often in the range of 10000 - 2000000. (2) The number of training data vectors N is often in the range of one million to 100 million, being also much higher than for the average NN application (medical applications etc.). These two problems rule out many of the theoretically superior and more sophisticated second order training algorithms because of insufficient memory resources and/or a too complicated implementation. Algorithms used in practice for large scale problems are currently mostly first order methods, i.e. (1) online gradient descent and (2) online RPROP procedures.

3.1.3.a Gradient descent training

Gradient descent training refers to adjusting the weight vector $\mathbf{w}$ after each iteration i by a small vector $\Delta\mathbf{w}$ proportional to the negative gradient $-\frac{\partial}{\partial \mathbf{w}} E^{(i)} = -\frac{\partial}{\partial \mathbf{w}} E(\mathbf{D}, \mathbf{w}^{(i)})$:

$$\Delta\mathbf{w}^{(i)} = -\eta \frac{\partial}{\partial \mathbf{w}} E^{(i)} \tag{3.16}$$

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \Delta\mathbf{w}^{(i)} \tag{3.17}$$

This procedure can be refined by making the weight change $\Delta\mathbf{w}$ linearly dependent on the previous change

$$\Delta\mathbf{w}^{(i)} = -\eta \frac{\partial}{\partial \mathbf{w}} E^{(i)} + \rho \cdot \Delta\mathbf{w}^{(i-1)}, \tag{3.18}$$

which often leads to a considerable speed-up. Good values for η and ρ heavily depend on the used NN structure, the training data and the initialization of $\mathbf{w}$ (which is often random using small values) and have to be found by experiments. An improved initialization procedure based on the training data and the structure of the network, that uses a combination of data driven methods for the input layer weights and supervised procedures for the output layer weights, is discussed in appendix A.

If an online procedure is used, then the estimated gradient to be used for one update depends only on a small part of the available training data, which might lead to a large fluctuation of the gradient and therefore to slower training. In this case the local gradient estimate may be smoothed and improved by

$$\frac{\partial}{\partial \mathbf{w}} E^{(i)} := (1-\alpha) \cdot \frac{\partial}{\partial \mathbf{w}} E^{(i-1)} + \alpha \cdot \frac{\partial}{\partial \mathbf{w}} E^{(i)} \tag{3.19}$$

with $0 \leq \alpha \leq 1$ controlling the amount of smoothing. As an additional improvement α can be made variable slowly increasing towards one during training.

3.1.3.b RPROP training

A procedure, that has been named RPROP in (Riedmiller and Braun, 1993), is a simple, heuristic first order procedure, that has been proposed in many variations by different researchers (see (Bishop, 1995)), and works reasonably well also for large scale problems. The idea is to keep a *step-size* δ_w for each weight individually and make the update dependent only on the sign of the wth component of the gradient $\frac{\partial}{\partial w} E^{(i)}$ as:

$$\begin{array}{rll} \text{IF} & \frac{\partial}{\partial w} E^{(i)} > 0 & \text{THEN} \quad w_w^{(i+1)} := w_w^{(i)} - \delta_w^{(i)} \\ \text{ELSE IF} & \frac{\partial}{\partial w} E^{(i)} < 0 & \text{THEN} \quad w_w^{(i+1)} := w_w^{(i)} + \delta_w^{(i)} \end{array}$$

The step-size itself is updated depending on the gradient component change as

$$\begin{array}{llll} \text{IF} & \frac{\partial}{\partial w} E^{(i)} \cdot \frac{\partial}{\partial w} E^{(i-1)} > 0 & \text{THEN} & \delta_w^{(i+1)} = \delta_w^{(i)} \cdot \tau_+ \\ \text{ELSE} & & & \delta_w^{(i+1)} = \delta_w^{(i)} \cdot \tau_- \end{array}$$

with good values being $\tau_+ = 1.2$ and $\tau_- = 0.5$ for many problems. It is useful to limit δ_w to not exceed a certain range, which is not very critical and often set to $0.000001 < \delta_w < 50$. A good initial start value for δ_w is often $\delta_w = J/10$, with J being the number of input weights to a certain neuron. An online version of RPROP using gradient smoothing like shown above was used for most experiments described in this thesis.

3.1.3.c ARPROP training

A simple adaptive refinement of the RPROP procedure, that was found to be very robust against variations of the data block sizes used for an update of the weights and therefore well suited for online training, was used for some experiments of this thesis. The problem of RPROP for online training, which is necessary to train large networks with lots of data, is that in general the step sizes approach zero too quickly. This can be avoided by heuristically increasing the step-sizes after a certain number of iterations, or by an automatic procedure, here called *Automatic* RPROP or ARPROP. An efficient method was found to be the following:

- An *average* step-size $\delta_{average}$ is calculated after each update of the weights as

$$\delta_{average} = \frac{1}{W} \sum_{w=1}^{W} \delta_w .$$

- If the average step-size of the current update $\delta^{(i)}_{average}$ is below the average step-size from the previous update $\delta^{(i-1)}_{average}$, the parameter τ_+ controlling the step-size increase, is decreased by a small value C, otherwise it is increased, that is

$$\begin{array}{llll} \text{IF} & \delta^{(i)}_{average} > \delta^{(i-1)}_{average} \;\; \text{AND} \;\; \tau^{(i)}_+ > 1.0 & \text{THEN} & \tau^{(i+1)}_+ = \tau^{(i)}_+ - C \\ \text{ELSE} & & & \tau^{(i+1)}_+ = \tau^{(i)}_+ + C, \end{array}$$

with $C = 0.01$, $\tau^{(0)}_+ = [1.1; 1.2]$ and $\tau_- = [0.5; 0.9]$ being useful values for all experiments conducted for this thesis. In this way the average step-size decreases very slowly and a complete training until convergence could be run without any human-induced restarts.

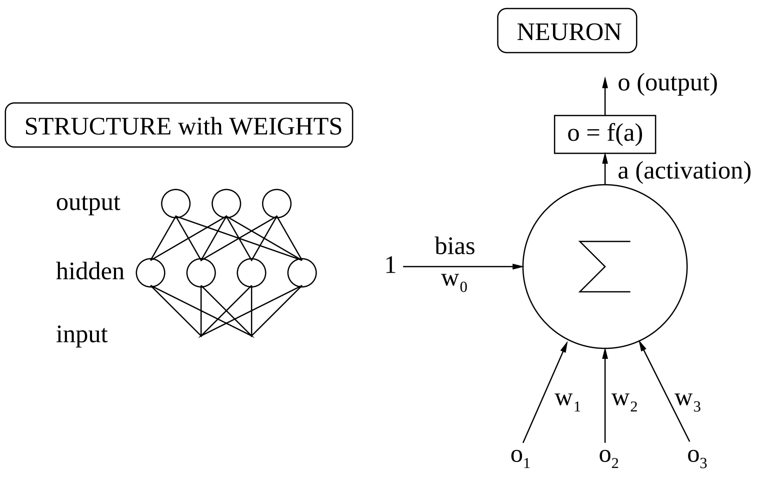
3.1.4 Neural network architectures

For supervised learning (from sequences) several different neural network architectures are in use. The most common are Multi-Layer-Perceptrons (MLPs), Time Delay Neural Networks (TDNNs) and Recurrent Neural Networks (RNNs) and are briefly discussed below. Other common architectures like Radial Basis Functions (RBFs) and Hierarchical Mixtures of Experts (HMEs) (Jordan and Jacobs, 1994) have interesting properties but haven't been used for this thesis and their explanation is therefore omitted here.

The type of neural networks discussed here have as elements *neurons* connected by directed *connection weights* representing scalar parameters w, which are combined in a *structure* to provide an D- (input) to K-dimensional (output) mapping. Each neuron has one output o and many (for example J) inputs connected to outputs of other neurons or the input vector itself (Figure 3.1). The output o of each neuron is a function of its *activation* a, so $o = f_{act}(a)$, with the activation calculated as a sum of all inputs to the neuron multiplied by its corresponding weight, $a = \sum_J o_j w_j$. Usually there is also a *bias* weight which acts as an additional input constantly set to one and in general treated like one of the J inputs.

The neurons are often organized in *layers* as groups of neurons, with consecutive layers being usually *fully connected*, meaning that each neuron of a layer is connected to all neurons of the next layer. When neurons' outputs are used as one of the K neural network outputs, they belong to the *output layer*, otherwise they belong to one of the *hidden layers*. As activation functions for hidden layer neurons commonly the *sigmoid* function $f_{act}(a) = 1/(1 + e^{-a})$ or its equivalent by a linear transformation, the *tanh*-function $f_{act}(a) = (e^a - e^{-a})/(e^a + e^{-a})$ is used (note that $2f^{sigmoid}_{act}(2a) - 1 = f^{tanh}_{act}(a)$), with the latter one leading often to slightly faster convergence using commonly used training procedures. The choice of the sigmoid activation function is motivated by

Figure 3.1: Elements of general feed-forward neural networks



Elements of feed-forward neural networks as used in this thesis are *neurons* and *connection weights* in a *structure*.

its distinct property of being the *discriminant function* for a two-class classification problem that makes the output the posterior probability of class membership, if the input distributions are Gaussian with equal covariance matrices (Bishop, 1995). The choice of activation functions for the output layer depends on the problem to be solved. If it is a uni-modal regression problem, the output of the network represents the mean of a Gaussian distribution, which shouldn't be bounded. Therefore usually the *linear* activation function $f_{act}(a) = a$ is used. If it is a classification problem, the *softmax* (Bridle, 1989) function $f_{act}(a) = e^a / \sum_K e^{a_k}$ is used, which can be interpreted as the generalized sigmoid for the K-class classification problem (Bishop, 1995). For multi-modal regression problems the outputs of the network represent various parameters of a distribution, i.e. means, covariances and mixture weights. Their range requires different types of activation functions, e.g. the linear function to model the unbounded means, the partially bounded exponential activation function $f_{act}(a) = e^a$ to model the variances (actually standard variations) and the softmax activation function to model the mixture weights to account for the fact that the sum of the mixture weights has to sum up to one (Bishop, 1995). Multi-modal regression is discussed in more detail in section 3.3.1. The combinations of objective function and output activation functions depending on the problem to be solved are summarized in Table 3.1.

3.1.4.a Multi-Layer-Perceptrons

Multi-Layer-Perceptrons (MLPs) are the most common type of architecture, in many practical applications only with two layers of weights, one hidden and the output layer

Table 3.1: Common setups for neural networks.

Problem	Objective Function	Output Activation Function
uni-modal regression	squared error	linear
multi-modal regression	log-likelihood	linear, exponential, softmax
classification	cross-entropy	softmax

Typical combinations of objective functions and output activation functions depending on the problem to solve.

like shown in the left part of Figure 3.1. More layers are possible but not necessary, since there are proofs that any mapping can be approximated with arbitrary accuracy with only two layers ((Bishop, 1995) and references therein), although using more layers *can* be a more efficient realization of a certain mapping. In practice, although, more than two layers are rarely used because of little expected performance gain and practical problems during training.

For sequence processing with neighboring vector pairs being correlated, it is common to use besides the current input vector $\mathbf{x}_t$ also information from its $2L$ neighboring vectors $\mathbf{x}_{t-L}, \mathbf{x}_{t-L+1}, \ldots, \mathbf{x}_{t-1}$ and $\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_{t+L}$ from a *window* as input to the MLP to improve prediction. The optimal width of the window to reach a good performance on unseen test data drawn from the same distribution depends obviously heavily on the type of data and the predefined structure. If the window is too small, not enough information will be present to provide an optimal prediction. If the window is very large, the performance on the training data will be almost perfect, but the performance on test data will be poor. Hence, an optimal window size has to be found by trial and error on development data.

3.1.4.b Time-Delay Neural Networks

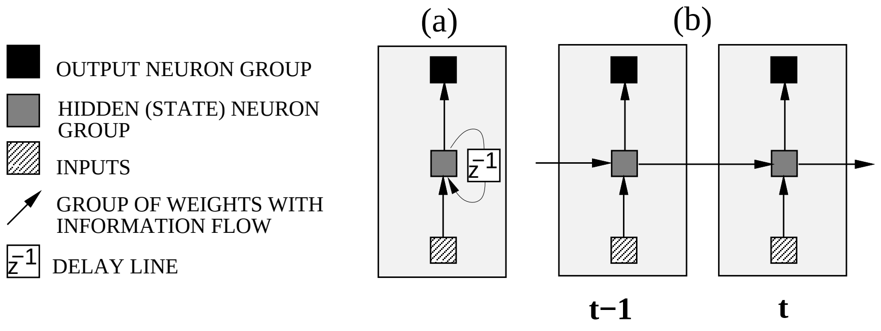
Time-Delay-Neural-Networks (TDNNs) (Waibel, 1989) have the same structure as a regular MLP, but a reduced number of total weight parameters, and have proven to be a useful improvement over regular MLPs in many applications. This is achieved by a user-defined *hard-tying* of parameters, meaning forcing certain parameters to have the same values. Which parameters are useful to tie depends heavily on the used data and can only be found by experiments.

3.1.4.c Recurrent Neural Networks

For many applications the data $\mathcal{D}$ is not a collection of vector pairs in arbitrary order, but the data comes in sequences of vector pairs, with the order being not arbitrary. Speech recognition is a typical example for this case – every preprocessed waveform is an array of vectors $\mathbf{x}_1^T$, that is to be mapped to an array of target classes s_1^T in the form of K-dimensional vectors $\mathbf{t}_1^T$.

One type of Recurrent Neural Networks (RNNs) provides an elegant way of dealing with (time) sequential data that embodies correlations between data points that are close in the sequence. Figure 3.2 shows a basic RNN architecture with a delay line and unfolded in time for two time steps. In this structure, the input vectors $\mathbf{x}_t$ are fed one at a time into the RNN. Instead of using a fixed number of input vectors as done in the MLP and TDNN structures, this architecture can make use of all the available input information up to the *current* time frame t_c (i.e., $\{\mathbf{x}_t, t = 1, 2, \ldots, t_c\}$) to predict $\mathbf{y}_{t_c}$. How much of this information is captured by a particular RNN depends on its structure and the training algorithm. An illustration of the amount of input information used for prediction with different kinds of NNs is given in Figure 3.3.

Figure 3.2: Recurrent neural network

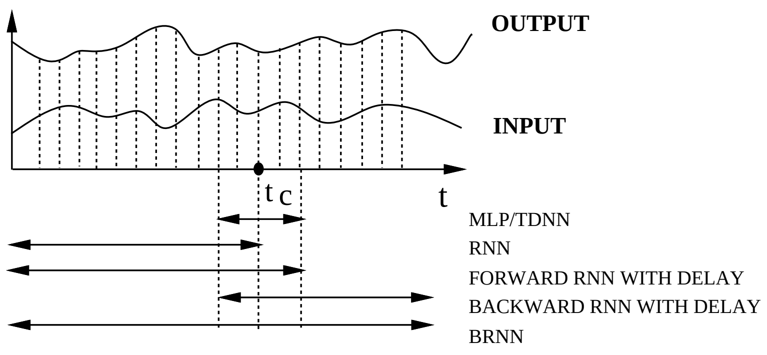

OUTPUT NEURON GROUP
HIDDEN (STATE) NEURON GROUP
INPUTS
GROUP OF WEIGHTS WITH INFORMATION FLOW
DELAY LINE
(a)
(b)
z^{-1}
t−1
t

General structure of a regular unidirectional RNN shown (a) with a delay line, and (b) unfolded in time for two time steps.

Future input information coming up later than t_c is usually also useful for prediction. With an RNN, this can be partially achieved by delaying the output by a certain number of S time frames to include future information up to $\mathbf{x}_{t_c+S}$ to predict $\mathbf{y}_{t_c}$ (Figure 3.3). Theoretically S could be made very large to capture all the available future information, but in practice it is found that prediction results drop if S is too large. A rough explanation for this could be that with rising S the modeling power of the RNN is more and more concentrated on remembering the input information up to $\mathbf{x}_{t_c+S}$ for the prediction of $\mathbf{y}_{t_c}$, leaving less modeling power for combining the prediction knowledge from different input vectors.

While delaying the output by some frames has been used successfully to improve results in a practical speech recognition system (Robinson, 1994) (this was also confirmed by the experiments described in this thesis) the optimal delay is task dependent and has to be found by the "trial and error" error method on a validation test set. Certainly, a more elegant approach would be desirable. One possibility to get around this user-defined delay is to use *bidirectional recurrent neural networks* (BRNNs), which are discussed in the next chapter and in (Schuster and Paliwal, 1997).

Figure 3.3: How much input information is used?



Visualization of the amount of input information used for prediction by different network structures.

To use all available input information, it is possible to use two separate networks – one for each time direction, and then somehow merge the results. Both networks can then be called experts for the specific problem the networks are trained on. One way of *merging the opinions of different experts* is to assume that the opinions are statistically independent and distributed itself by some hyper-distribution. Making certain simplifying assumptions about these distributions leads to the often used arithmetic averaging for uni-modal regression and to geometric averaging (what corresponds to an arithmetic averaging in the log-domain) for classification. These merging procedures are referred to as *linear opinion pooling* and *logarithmic opinion pooling*, respectively (Berger, 1985; Jacobs, 1995). Although simple merging of network outputs has been applied successfully in practice (Robinson et al., 1994), it is generally not clear how to merge network outputs in an optimal way, since different networks trained on the same data cannot be regarded as independent anymore.

Because of their recurrent connections, training of RNNs is slightly more complicated than for feed-forward neural networks like MLPs. An often used training procedure is *back-propagation through time* (BPTT). For BPTT first the RNN structure is unfolded up to the length of the training sequence as shown for two time steps in Figure 3.2, which transforms the RNN in a large feed-forward neural network. Now regular back-propagation can be applied, only at the beginning and the end of the training data sequence some special treatment is necessary. The state inputs at $t = 1$ are not known and can be set to an arbitrary, but fixed, value in practice. Also, the local state derivatives at $t = T$ are not known and can be set to zero, assuming that input information beyond that point is not important for the current update, which is for the boundaries certainly the case.

3.2 BIDIRECTIONAL RECURRENT NEURAL NETWORKS

In this section a regular recurrent neural network is extended to a bidirectional recurrent neural network (BRNN). Given a series of paired input/target vectors $\{(\mathbf{x}_t, \mathbf{t}_t), t = 1, 2, \ldots, T\}$, the bidirectional recurrent neural networks is to be trained to perform the following two tasks:

Prediction assuming independent outputs: Unimodal regression (i.e., compute $\mathbf{y}_t = \hat{\mathbf{t}}_t = \langle \mathbf{t}_t | \mathbf{x}_1^T \rangle)$ or classification (i.e., compute $\hat{y}_t^k = P(C_t = k | \mathbf{x}_1^T)$ for each output class k and decide the class using the maximum a posteriori decision rule). In this case, the outputs are treated to be statistically independent. Experiments for this part are conducted for artificial toy data as well as for real data.

Prediction assuming dependent outputs: Estimation of the conditional probability of a complete sequence of classes of length T using all available input information (i.e., compute $P(s_1, s_2, ..., s_T | \mathbf{x}_1^T)$). In this case, the outputs are treated to be statistically dependent, which makes the estimation more difficult and requires a slightly different network structure than the one used in the first part. For this part, results of experiments using real data are reported.

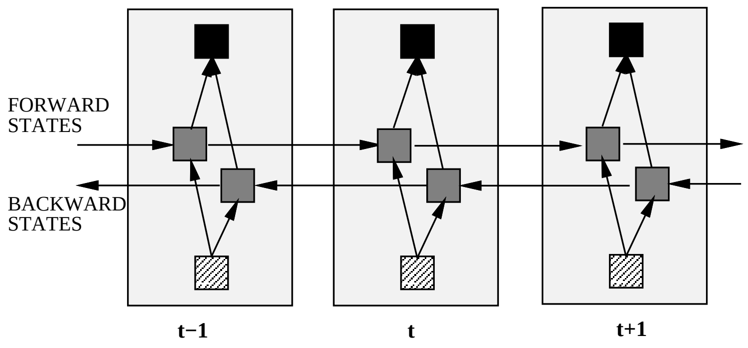
3.2.1 Prediction assuming independent outputs

To overcome the limitations of a regular RNN outlined in the previous section, here a bidirectional recurrent neural network (BRNN) which can be trained using all available input information in the past and future of a specific time frame, is proposed.

3.2.1.a BRNN architecture

The idea is to split the state neurons of a regular RNN in one part which is responsible for the positive time direction (forward states) and a second part for the negative time direction (backward states). Outputs from forward states are not connected to inputs of backward states and vice versa. This leads to the general structure which can be seen in Figure 3.4, where it is unfolded over three time steps. It is not possible to display the BRNN structure in a figure similar to Figure 3.2 with the delay line, since the delay would have to be positive and negative in time. Note that without the backward states this structure simplifies to a regular unidirectional forward RNN, like shown in Figure 3.2. If the forward states are taken out, it results in a regular RNN with a reversed time axis. With both time directions taken care of in the same network, input information in the past and the future of the currently evaluated time frame can directly be used to minimize the objective function, without the need of delays to include future information as for the regular unidirectional RNN discussed above.

Figure 3.4: Bidirectional recurrent neural network



General structure of the bidirectional recurrent neural network (BRNN) with hidden states in forward and backward time direction, shown unfolded in time for three time steps.

3.2.1.b BRNN training

The BRNN can in principle be trained with the same algorithms as a regular unidirectional RNN because there are no interactions between the two types of state neurons, and therefore can be unfolded into a general feed-forward network. However, if for example any form of back-propagation through time (BPTT) is used, the forward and backward pass procedure is slightly more complicated, because the update of state and output neurons can not be done one at a time anymore. If BPTT is used, the forward and backward pass over the unfolded BRNN over time are done almost in the same way as for a regular MLP - only at the beginning and the end of the training data some special treatment is necessary. The forward state inputs at $t = 1$ and the backward state inputs at $t = T$ are not known. Setting these could be made part of the learning process, but here they are set arbitrarily to a fixed value (0.5). Also, the local state derivatives at $t = T$ for the forward states and at $t = 1$ for the backward states are not known, and are set here to zero. The training procedure for the unfolded bidirectional network over time can be summarized as follows:

1. *FORWARD PASS*

   Feed all input data for one time slice $1 \leq t \leq T$ into the BRNN and determine all predicted outputs.

   (a) Do forward pass just for forward states (from $t = 1$ to $t = T$) and backward states (from $t = T$ to $t = 1$).

   (b) Do forward pass for output neurons.

2. *BACKWARD PASS*

   Calculate the part of the objective function derivative for the time slice $1 \leq t \leq T$ used in the forward pass.

   (a) Do backward pass for output neurons.
   (b) Do backward pass just for forward states (from $t = T$ to $t = 1$) and backward states (from $t = 1$ to $t = T$).

3. *UPDATE WEIGHTS*

One obvious disadvantage of the bidirectional structure is the fact, that for its usage theoretically the *complete* input sequence must be known, which prohibits any online processing for time-sequential data. This disadvantage can be partially removed by cutting the sequences in shorter chunks, which are used for training and testing. If these chunks are long enough to include all context effects that can be used by the BRNN, then this representation with many short sequences is equivalent to the original one with one long sequence. In this case only a time-lag equal to the length of the chunks would occur, if an online procedure is used.

3.2.1.c Extensions for the BRNN architecture

For BRNNs there are several extensions possible to improve convergence speed, to take advantage of symmetry in the data or to remove some minor disadvantages of the bidirectional structure, which are briefly discussed below.

Short cuts: It is possible to add an additional layer of weights from the inputs directly to the output layer, which can be thought of a one-layer non-recurrent NN working in parallel to the bidirectional structure. In informal experiments training with such *short cuts* converged much faster than without for many problems.

Additional output layer: There are proofs (Bishop, 1995) that an MLP with one sufficiently large hidden layer can approximate *any* mapping to arbitrary accuracy provided the NN has enough degrees of freedom. A similar argument applied to recurrent neural networks would imply that any mapping of input *sequences* to targets could be approximated with arbitrary accuracy. This argument *cannot* be applied to the structure shown in Figure 3.4 for the following reason. The hidden neurons (forward and backward states) cannot represent arbitrary properties of the input data, because the forward neurons cannot make use of the data after t while backward neurons cannot make use of the data before t. The features of the data that depend on information before *and* after t are not fed through another layer of a sufficiently large number of neurons, which would be required to approximate any input sequence to target mapping. In principle an additional layer between the forward/backward state neurons and the output layer can resolve that problem. In practice, however, an additional output layer causes additional difficulty in converging during training of the network and its usefulness depends a lot on the data that is used for training. Informal experiments showed that for the data as used for this thesis no performance gain could be achieved by adding this layer.

Weight sharing for symmetrical data: Assume that the given sequential input data is known to be symmetrical around every t in the absence of input noise. In practice, though, input noise is present that causes distortions in the symmetry.

One way to include the prior knowledge that the input data is symmetrical is to *hard-tie* the weights that connect the inputs to the forward and backward states, meaning to force them to have the same values. Another way is to use a regular uni-directional RNN and train it with every training sequence twice, once in positive time direction and once in negative time direction.

Long term delays: In (Bengio et al., 1994) it is shown that long-term delays are difficult to learn for NNs trained with gradient descent procedures. One way of favoring the learning of long-term dependencies is to change the structure of the network by allowing not only connections between neighboring forward (or backward) neurons in time, but to connect hidden neurons that are further apart. In informal experiments with small networks this improved convergence speed and results in some cases, depending on the used data.

3.2.2 Experiments and results

In this section a number of experiments with the goal to compare the performance of the BRNN structure with that of other structures, is described. In order to provide a fair comparison, different structures with a comparable number of parameters as a rough complexity measure were used. Experiments were run for artificial data for both uni-modal regression and classification tasks with small networks to allow extensive experiments, and for real data for a phoneme classification task with larger networks.

3.2.2.a Experiments with artificial data

Description of data: In these experiments, an artificial data set was used to conduct a set of uni-modal regression and classification experiments. The artificial data was generated as follows. First a stream of 10000 random numbers between zero and one was created as the one dimensional input data to the NN. The one-dimensional target data (the desired output) was obtained as the weighted sum of the inputs within a window of 10 frames to the left and 20 frames to the right with respect to the current frame. The weighting falls of linearly on both sides as follows:

$$y(t) = \frac{1}{10}\sum_{\Delta t=-10}^{-1} x(t+\Delta t)\cdot(1-\frac{|\Delta t|}{10}) + \frac{1}{20}\sum_{\Delta t=0}^{19} x(t+\Delta t)\cdot(1-\frac{|\Delta t|}{20}). \tag{3.20}$$

The weighting procedure introduces correlations between neighboring input/target data pairs which become less for data pairs further apart. Note that the correlations are not symmetrical, being on the right side of each frame twice as "broad" as on the left side. For the classification experiments, the output data was mapped to two classes, with class 0 for all output values below (or equal to) 0.5 and class 1 for all output values above 0.5, giving approximately 59% of the data to class 0 and 41% to class 1.

Experiments: Separate experiments were conducted for uni-modal regression and classification tasks. For each task, four different architectures were tested (Table 3.2). Type "MERGE" refers to the merged results of type RNN-FOR and RNN-BACK, being regular unidirectional recurrent neural networks trained in forward and backward time direction, respectively. The first three architecture types were also evaluated over different shifts of the output data in positive time direction, allowing the RNN to use future information as discussed above.

Table 3.2: Types of experiment for prediction assuming independent outputs.

Structure	Neurons (forward/backward)	Shift Range
RNN-FOR	2/0	-5 to +20
RNN-BACK	0/2	+5 to -20
MERGE	2/2	-2/+2 to +10/-10
BRNN	2/2	none

Architectures evaluated for uni-modal regression and classification. The shift range is the number of frames that the target data has been moved artificially in positive time direction seen from the time axis of the network to include future context effects.

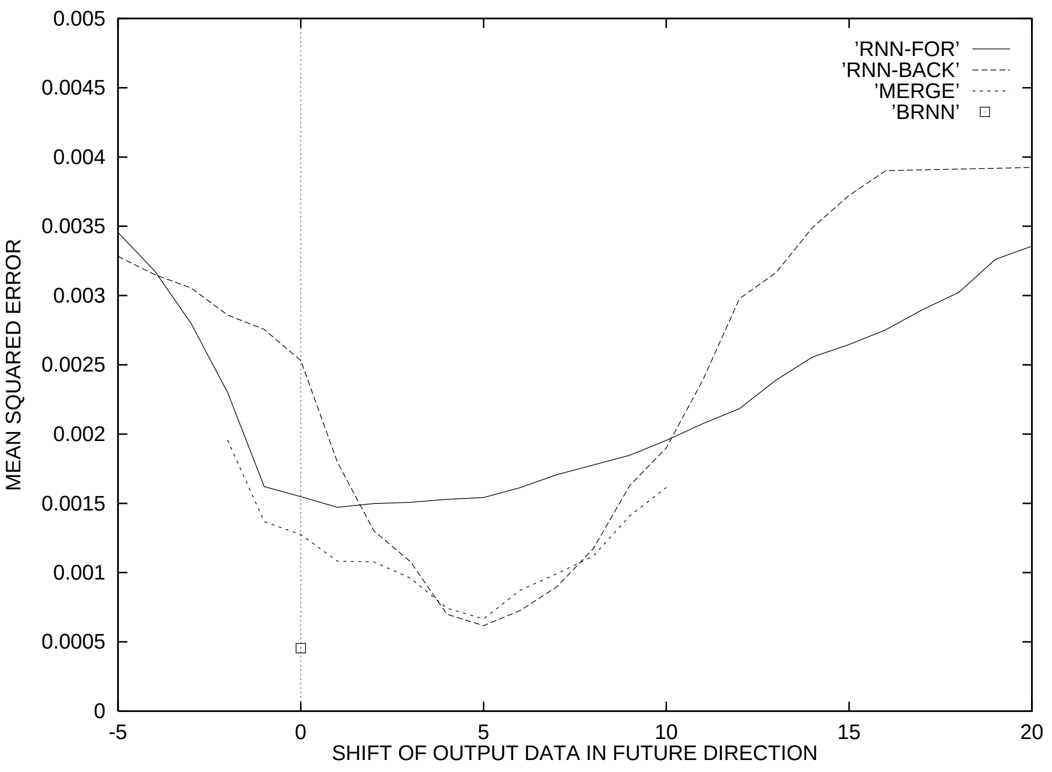
Every test (NN training/evaluation) was run 100 times with different initializations of the NN to get at least partially rid of random fluctuations of the results due to convergence to local minima of the objective function. All networks are trained with 200 cycles of a modified version of the resilient propagation (RPROP) technique, extended to a RPROP through time variant. All weights in the structure were initialized in the range of $[-1,1]$ drawn from the uniform distribution, except the output biases, which were set so the corresponding output gives the prior average of the output data in case of zero input activation.

For the uni-modal regression experiments, the networks contain the $tanh()$ activation function for the hidden layers and linear activation function for the output layer, and were trained to minimize the squared-error objective function. For type "MERGE", the arithmetic mean of the network outputs of "RNN-FOR" and "RNN-BACK" was taken (*linear opinion pool*).

For the classification experiments, the output layer uses the "softmax" output function, so that outputs add up to one and can be interpreted as probabilities. As commonly used for NNs to be trained as classifiers, the cross-entropy objective function is used as the optimization criterion. Because the outputs are probabilities assumed to be generated by independent events, for type "MERGE" the normalized geometric mean (*logarithmic opinion pool*) of the network outputs of "RNN-FOR" and "RNN-BACK" is taken.

Results: The results for the regression and the classification experiments averaged over 100 training/evaluation runs can be seen in Figure 3.5 and Figure 3.6, respectively. For the regression task, the mean squared error depending on the shift of the target data in positive time direction seen from the time axis of the network is shown. For the classification task, the recognition rate instead of the mean value of the objective function (which would be the mean cross-entropy) is shown, because it is a more familiar measure to characterize results of classification experiments.

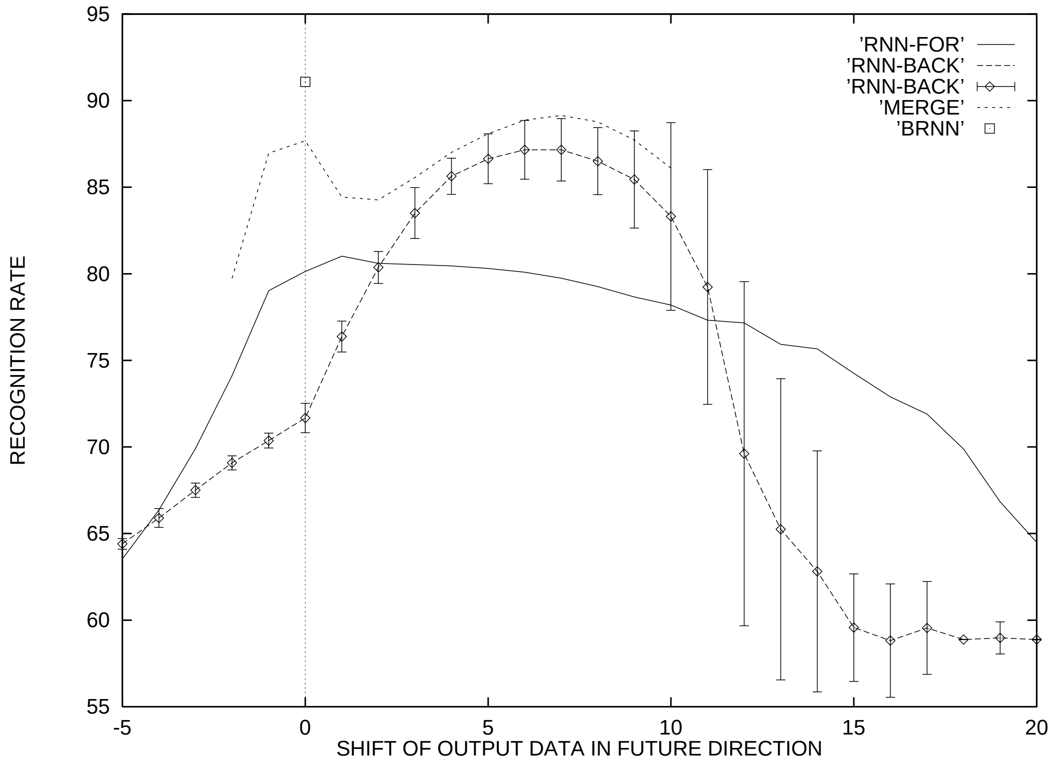
Figure 3.5: Results for artificial regression problem



Averaged results (100 runs) for the regression experiment on artificial data over different shifts of the output data with respect to the input data in future direction (viewed from the time axis of the corresponding network) for several structures.

Several interesting properties of RNNs in general can be directly seen from these figures. The minimum (maximum) for the regression (classification) task should be at 20 frames delay for the forward RNN and at 10 frames delay for the backward RNN because at those points all information for a perfect regression (classification) has been fed into the network. Neither is the case because the modeling power of the

Figure 3.6: Results for artificial classification problem



Averaged results (100 runs) for the classification experiment on artificial data over different shifts of the output data with respect to the input data in future direction (viewed from the time axis of the corresponding network) for several structures.

networks given by the structure and the number of free parameters is not sufficient for the optimal solution. Instead, the single time direction networks try to make a trade-off between "remembering" the past input information which is useful for regression (classification), and "knowledge combining" of currently available input information. This results in an optimal delay of one (one) frame for the forward RNN and five (six) frames for the backward RNN. The optimum delay is larger for the backward RNN because the artificially created correlations in the training data are not symmetrical, with the important information for regression (classification) being twice as dense on the left side as on the right side of each frame. In the case of the backward RNN, the time series is evaluated from right to left with the denser information coming up later. Because the denser information can be evaluated easier (less parameters are necessary for a contribution to the objective function minimization), the optimal delay is larger for the backward RNN. If the delay is so large that almost no important information can be saved over time, the network converges to the best possible solution based only on prior information. This can nicely be seen for the classification task with the backward RNN which converges to 59 % (prior of class 0) for more than 15 frames delay.

Another sign for the trade-off between "remembering" and "knowledge combining" is the variation in the standard deviation of the results which is only shown for the backward RNN in the classification task. In areas where both mechanisms could be useful (3 to 17 frames shift), different local minima of the objective function correspond to a certain amount to either one of these mechanisms which results in larger fluctuations of the results than in areas where "remembering" is not very useful (-5 to 3 frames shift) or not possible (17 to 20 frames shift).

If the outputs of forward and backward RNNs are merged, so that all available past and future information for regression (classification) is present, the results for the delays tested here (-2 to 10) are in almost all cases better than with only one network. This is no surprise because besides the use of more useful input information the number of free parameters for the model doubled.

For the BRNN, it does not make sense to delay the output data because the structure is already designed to cope with all available input information on both sides of the currently evaluated time point. Therefore, the experiments for the BRNN are only run for SHIFT = 0. For the regression and classification tasks tested here, the BRNN clearly performs better than the network "MERGE" built out of the single time-direction networks "RNN-FOR" and "RNN-BACK", with a comparable number of total free parameters.

3.2.2.b Experiments with real data I

The goal of the experiments with real data is to compare different NN structures for the classification of phonemes from the TIMIT speech database. Several regular MLPs and recurrent neural network architectures, which make use of different amounts of acoustic context, are tested here.

Description of data: The TIMIT phoneme database is a well established database consisting of 6300 sentences spoken by 630 speakers (10 sentences per speaker). Following official TIMIT recommendations, two of the sentences (which are the same for every speaker) are not included in our experiments, and the remaining data set is divided into two sets, the

1) training data set consisting of 3696 sentences from 462 speakers, and the

2) test data set consisting of 1344 sentences from 168 speakers.

The TIMIT database provides hand-segmentation of each sentence in terms of phonemes and a phonemic label for every segment out of a pool of 61 phonemes. This results in 142910 phoneme segments for training and 51681 for testing.

In our experiments, every sentence was transformed into a vector sequence using three levels of feature extraction. First, features were extracted every frame to represent the raw waveform in a compressed form. Then, with the knowledge of the boundary locations from the corresponding label files, segment features were extracted to map the information from an arbitrary length segment to a fixed dimensional vector. A third transformation was applied to the segment feature vectors to make them suitable as inputs to a neural net. These three steps are briefly described below.

1. **Frame Feature Extraction:** As frame features, 12 regular MFCCs (from 24 mel-space frequency bands) plus the log-energy are extracted every 10 ms with a 25.6 ms Hamming window and a preemphasis of 0.97. This is a commonly used feature extraction procedure for speech signals at the frame level (Young, 1996).

2. **Segment Feature Extraction:** From the frame features, the segment features are extracted by dividing the segment in time into five equally spaced regions and computing the area under the curve in each region, with the function values between the data points linearly interpolated. This is done separately for each of the 13 frame features. The duration of the segment is used as an additional segment feature. This results in a 66-dimensional segment feature vector.

3. **Neural Network Preprocessing:** Although NNs can principally handle any form of input distributions, it was found in the experiments conducted here that the best results are achieved with Gaussian input distributions which matches the experiences from (Robinson, 1994). To generate an "almost-Gaussian distribution", the inputs were first normalized to zero mean and unit variance on a sentence basis, and then every feature of a given channel[1] was quantized using a scalar quantizer having 256 reconstruction levels (1 byte). The scalar quantizer is designed to maximize the entropy of the channel for the whole training data. The maximum entropy scalar quantizer can be easily designed for each channel by arranging the channel points in ascending order according to their feature values and putting (almost) an equal number of channel points in each quantization cell. For presentation to the network, the byte-coded value is remapped with $value = \mathrm{erf}^{-1}(2 \cdot (\mathrm{byte} + 1/2)/256 - 1)$, where ($\mathrm{erf}^{-1}$ is the inverse error function, erf() is part of math.h library in C). This mapping produces on average a distribution that is similar to a Gaussian distribution.

The feature extraction procedure described above transforms every sentence into a sequence of fixed dimensional vectors representing acoustic phoneme segments. The sequence of these segment vectors (along with their phoneme class labels) were used to train and test different NN structures for classification experiments as described below.

Experiments: Experiments were performed here with different NN structures (e.g., MLP, RNN and BRNN), which allow the use of different amounts of acoustic context. The MLP structure is evaluated for three different amounts of acoustic context as input:

[1]Here each vector has a dimensionality of 66. Temporal sequence of each component (or, feature) of this vector defines one channel. Thus, there are 66 channels.

(1) one segment,

(2) three segments (middle, left and right), and

(3) five segments (middle, two left and two right).

The evaluated RNN structures are unidirectional forward and backward RNNs which use all acoustic context on one side, two forward RNNs with one and two segment delays to incorporate right hand information, the merged network built out of the unidirectional forward and backward RNNs, and the BRNN. The structures of all networks were adjusted so each of them has about the same number of free parameters (approximately 13000 here). The networks were trained with RPROP.

Results: Table 3.3 shows the phoneme classification results for the full training and test set. Although the database is labeled to 61 symbols, a number of researchers have chosen to map them to a subset of 39 symbols. Here results are given for both versions, with the results for 39 symbols being simply a mapping from the results obtained for 61 symbols. Details of this standard mapping can be found in (Robinson, 1991).

Table 3.3: Results for prediction assuming independent outputs.

Structure	Classification Rate %	Classification Rate %
	TRAIN 61 (39)	TEST 61 (39)
MLP-1 (1 segment)	61.32 (70.20)	59.67 (68.95)
MLP-3 (3 segments)	68.37 (75.74)	65.69 (73.48)
MLP-5 (5 segments)	66.97 (74.60)	64.32 (72.35)
FOR-RNN	65.42 (74.27)	63.20 (72.51)
BACK-RNN	64.57 (72.83)	61.91 (70.94)
FOR-RNN (1 delay)	68.45 (75.37)	65.83 (73.00)
FOR-RNN (2 delay)	65.97 (73.03)	63.27 (70.77)
MERGE (FOR+BACK)	66.94 (75.01)	65.28 (73.73)
BRNN	**70.73 (77.33)**	**68.53 (75.48)**

TIMIT phoneme classification results for full training and test data sets with $\approx$ 13000 parameters. The BRNN performs best among all evaluated architectures.

The baseline performance assuming neighboring segments to be independent gives a recognition rate of 59.67% (MLP-1) on the test data. If three consecutive segments are taken as the inputs (MLP-3), loosening the independence assumption to three segments, the recognition rate goes up to 65.69%. Using five segments (MLP-5), the structure is not flexible enough to make use of the additional input information, and as a result the recognition rate drops to 64.32%. The forward and backward RNNs (FOR-RNN, BACK-RNN), making use of input information only on one side of the current segment, give lower recognition rates (63.2% and 61.91%) than the forward RNN with one segment delay (65.83%). With a two segment delay, too much information has to be saved over time and the result drops to 63.27% (FOR-RNN, 2 delay), although theoretically more input information than for the previous network is present. The merging of the outputs of two separate networks (MERGE) trained in each time direction results in a recognition rate of 65.28%, and is worse than the forward RNN structure using one segment delay. The bidirectional recurrent neural network (BRNN) structure results in the best performance (68.53%).

3.2.2.c Experiments with real data II

In a second set of experiments on the TIMIT database not classification of phoneme segments, but *recognition* of phoneme sequences (utterances) was evaluated. This comes closer to actual speech recognition, because part of the problem becomes now not only the assignment of acoustic segments to phonemes, but also the segmentation of the feature vector sequence into acoustic segments.

Description of data: Each sentence of the TIMIT phoneme database, as described in section 3.2.2.b, was first transformed into a feature vector sequence representing the waveform in a compressed form in a similar way as shown in the last section.

1. **Frame Feature Extraction:** As frame features, 14 regular MFCCs (from 24 mel-space frequency bands) plus the log-energy are extracted every 16 ms with a 25.6 ms Hamming window and a preemphasis of 0.97. First order derivatives of the underlying smooth progression of feature vectors were appended to the original sequence to decorrelate neighboring vectors in the sequence, although if used as inputs to BRNNs this might not really be necessary since the network itself can approximate the generation of delta-features. This leads to 30-dimensional feature vectors, altogether to 702438 from the training data and 256617 from the test data.

2. **Neural Network Preprocessing:** In the same way as described in 3.2.2.b, the resulting vector streams were transformed to "almost-Gaussian distributions" for each vector component, which also had the effect of adjusting the range of the data, such that it is suitable for input to a neural network with initial weights drawn from a uniform distribution in the range $]-1, 1[$.

Experiments: The goal of the experiments was to find the best output frame sequence by using the concept of (2.7), which requires to get estimates for three probability expressions. The posterior probability of a context-independent phone class c_i (out of 61 possible) given the acoustic features, i.e. $P(c_i|\mathbf{x})$, was approximated by several BRNNs with a different number of hidden forward and backward neurons. The prior class probability $P(c_i)$ was estimated by the relative frequency of the frames of that class in the training data. The transition probabilities $P(c_j|c_i)$ were also estimated by relative frequencies as $P(c_j|c_i) = N(i,j)/N(i)$ like explained in 2.4.1. To account for unseen transitions ($N(i,j) = 0$), which would result in a transition probability of zero, $N(i,j)$ was set to 0.5 in those cases to *floor* the transition probabilities. To find the single best sequence using (2.7) a simple Viterbi search (section 2.4.2.b) was implemented. Note that the models used here learn to use an arbitrary amount of acoustic context because of their bidirectional structure, but they are still context-independent models in the sense of section 2.2.2, because no use of the context on the output side is made.

Training of the BRNNs was the most difficult part of the experiments because of the relatively large number of training vectors (large compared to other tasks where neural networks are used) and the large number of parameters in the networks. In the first stages of the experiments high-end workstations were not available. Therefore the first version of the training was done on a multi-processor machine (Kendall Square Research I with 96 processors). The training data was divided into blocks of approximately 30 sentences each. A weight update was done after the gradient for each block was calculated, smoothed like shown in section 3.1.3.a. The basic algorithm was RPROP, extended to a RPROP through-time variant. The problem of this training procedure was that up to a 128 passes through the complete training data were necessary to achieve sufficient convergence.

Once faster workstations became available, training was moved to single processor machines (like the DEC ALPHA 500 MHz) using the better automatic RPROP procedure shown in section 3.1.3.c. One pass through the training data then took approximately 20 min (35000 vectors/min), and because of the better ARPROP algorithm with smaller block sizes less passes were necessary (< 32) to reach sufficient convergence. In addition, training was sped up by parallelizing it to run on workstation clusters using the publically available P4 library.

Results: For these experiments two measures of performance are of interest:

- the *frame classification rate* is what is optimized by training the NN using the cross-entropy measure like discussed in section 3.1.2.c, and
- the *phoneme recognition rate*, which is found by application of (2.7) using a Viterbi search. The phoneme recognition rate has to be calculated by aligning the output phoneme sequence to the correct phoneme sequence with a dynamic programming technique (Rabiner and Juang, 1993). The recognition rate is defined as

$$r = \frac{N - I - D - S}{N} \tag{3.21}$$

  with N being the number of symbols in the correct sequence, I being the wrongly inserted, D the deleted and S the number of substituted symbols in the recognized sequence.

Results from experiments with the complete TIMIT database showing both performance measures for various sizes of BRNNs are summarized in Table 3.4.

Table 3.4: Results for phoneme recognition on the TIMIT database using BRNNs as observation probability estimators.

Structure forward/backward states	Rec.-Rate (frame-rate) % TRAIN	Rec.-Rate (frame-rate) % TEST
32/32	70.6 (68.31)	64.4 (62.74)
64/64	71.5 (70.21)	66.2 (63.94)
128/128	73.6 (72.12)	68.0 (65.11)
176/0 (+4 delay)	75.4 (70.57)	69.3 (65.31)

Phoneme recognition (and frame classification) results for the full TIMIT training and test data with 61 symbols. The last column states the results from (Robinson, 1994), which only contains phoneme recognition results for the test data. The missing numbers were obtained from the software that was used to reproduce these results, which is publically available on the FTP server mentioned in (Robinson, 1994).

Recognition results using a BRNN for a different number of hidden forward/backward states are compared to the results published in (Robinson, 1994), where a regular uni-directional RNN with a shift of four frames was used to include right-hand context. The BRNN was trained without any delay of the data, because the bidirectional structure can already make use of all available input without any delay. The first notable and unexpected result is that none of the systems with the BRNNs achieved a better phoneme recognition rate than the system with the regular RNN described in (Robinson, 1994), although the frame recognition rate for the training data is almost 1.5% higher in the best case. This is partially due to an over-training effect since the difference for the frame classification results for the BRNN systems are larger than for the uni-directional RNN system. But even on the training data for the best BRNN the phoneme recognition results are almost two absolute percent worse. The reason for this might be, that either the BRNN parameters are stuck in a worse local minimum than the RNN, or that the modeling assumptions used to derive (2.7) have a more severe effect when it comes to evaluate the overall phoneme recognition rate.

These results show that choosing an appropriate NN architecture for the problem is essential for obtaining good results. In the experiments here neither the complexity of the network nor the number of training passes were particularly optimized for the specific problem of maximizing the phoneme recognition rate on the test data.

3.2.3 Prediction assuming dependent outputs

In the preceding section, an efficient architecture for the estimation of the conditional posterior probability $P(C_t = k|\mathbf{x}_1^T)$ of a *single* class k at a certain time point t given the sequence of input vectors $\mathbf{x}_1^T$ was presented. For some applications, it is necessary to estimate instead of $P(C_t = k|\mathbf{x}_1^T)$ the conditional posterior probability $P(s_1, s_2, \ldots, s_T|\mathbf{x}_1^T)$ of a *sequence* of all classes from $t = 1$ to $t = T$ given the sequence of input vectors. This is a difficult problem and no general practical solution is known, although this type of estimation is essential for many pattern recognition applications where sequences are involved.

3.2.3.a Approach

Bidirectional recurrent neural networks can provide an approach to estimate terms of the form $P(s_1, s_2, \ldots, s_T|\mathbf{x}_1^T)$. Using the rule $P(A, B) = P(A|B)P(B)$, the sequence posterior probability can be decomposed as follows (section 2.3):

$$P(s_1, s_2, \ldots, s_T|\mathbf{x}_1^T) = \underbrace{\prod_{t=1}^{T} P(s_t|s_{t+1}, s_{t+2}, \ldots, s_T, \mathbf{x}_1^T)}_{\text{backward posterior probability}} \tag{3.22}$$

$$= \underbrace{\prod_{t=1}^{T} P(s_t|s_1, s_2, \ldots, s_{t-1}, \mathbf{x}_1^T)}_{\text{forward posterior probability}} \tag{3.23}$$

The probability term within the product is the conditional probability of an output class given all the input to the right and left hand side plus the class sequence on one side of the currently evaluated input vector. The two ways of decomposing $P(s_1, s_2, \ldots, s_T|\mathbf{x}_1^T)$ are here referred to as the *forward* and the *backward* posterior probabilities. Note that these decompositions are only a simple application of probability rules; i.e., no assumptions concerning the shape of the distributions are made yet. Also note, that many other decompositions are possible. The two chosen here are just the ones which are most convenient and most popular.

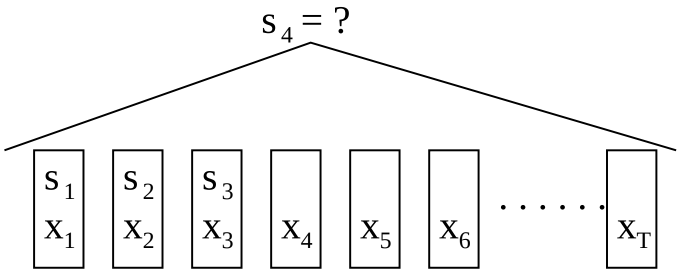
In the present approach, the goal is to train a network to estimate conditional probabilities of the kind $P(s_t|s_1, s_2, \ldots, s_{t-1}, \mathbf{x}_1^T)$, which are of the form of the probability terms in the products in (3.22) and (3.23).

The estimates for these probabilities can then be combined by using the formulae above to estimate the full conditional probability of the sequence. It should be noted that the forward and the backward posterior probabilities are exactly equal, provided that the probability estimator is perfect. However, if neural networks are used as probability estimators, this will rarely be the case because different architectures or different local minima of the objective function to be minimized correspond to estimators of different performance. It might therefore be useful to combine several estimators to get a better estimate of the quantity of interest using the methods of the previous section. Two candidates that could be merged here are $P(s_t|s_1, s_2, \ldots, s_{t-1}, \mathbf{x}_1^T)$ and $P(s_t|s_{t+1}, s_{t+2}, \ldots, s_T, \mathbf{x}_1^T)$ at each time point t.

3.2.3.b An architecture to estimate the posterior probability of a symbol sequence

A slightly modified BRNN structure can be used efficiently to estimate the conditional probabilities of the kind $P(s_t|s_1, s_2, \ldots, s_{t-1}, \mathbf{x}_1^T)$, which is conditioned on continuous ($\mathbf{x}_1^T$) and discrete sequential inputs $(s_1, s_2, \ldots, s_{t-1})$. Figure 3.7 shows a visualization of the problem.

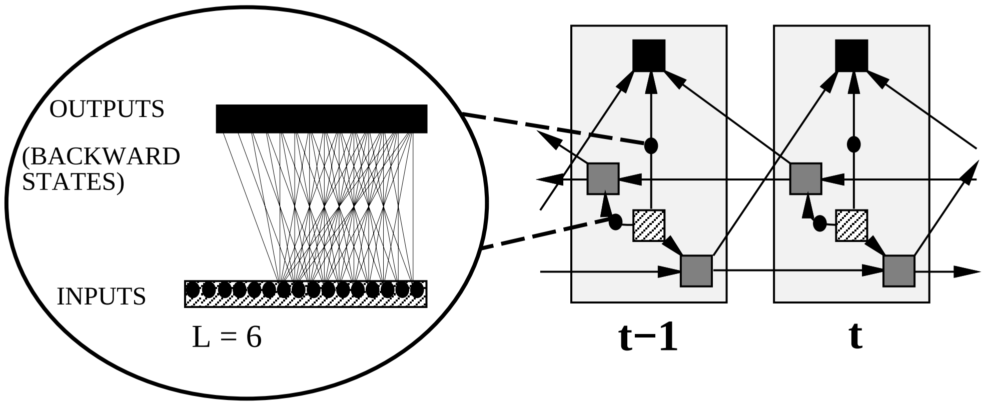
Figure 3.7: Forward probability estimation in extended BRNN



Visualization of the subproblem occuring during the estimation of the posterior probability of vector sequences conditioned on another sequence, here for the *forward probability* part $P(s_4|s_1, s_2, s_3, \mathbf{x}_1^T)$.

Assume that the input for a specific time t_c is coded as one long vector containing the target output class s_t and the original input vector $\mathbf{x}_t$, with for example the discrete input s_t coded in the first L dimensions of the input vector. One way of making the BRNN suitable to estimate $P(s_t|s_1, s_2, \ldots, s_{t-1}, \mathbf{x}_1^T)$, two changes of the original architecture from Figure 3.4 are necessary. First, instead of connecting the forward and backward state neurons to the current output neurons, they are connected to the next and previous output neurons, respectively, and the inputs are directly connected to the outputs. Second, if in the resulting structure the first L weight connections from the inputs to the backward states and the inputs to the outputs are cut, then only discrete input information from $t < t_c$ can be used to make predictions. This is exactly what is required to estimate the forward posterior probability $P(s_t|s_1, s_2, \ldots, s_{t-1}, \mathbf{x}_1^T)$. Figure 3.8 illustrates this change of the original BRNN architecture. Cutting the input connections to the forward states instead of the backward states provided the architecture for estimating the backward posterior probability. Theoretically all discrete and continuous inputs $s_1, s_2, \ldots, s_{t-1}, \mathbf{x}_1^T$ which are necessary to estimate the probability are still accessible for a contribution to the prediction. During training the bidirectional structure can adapt to the best possible use of the input information, as opposed to structures which do not provide part of the input information because of the limited size of the input windows (e.g., in MLP and TDNN) or one-sided windows (unidirectional RNN).

Figure 3.8: Extended bidirectional recurrent neural network



Modified bidirectional recurrent neural network structure to estimate the posterior probability of a hypothesized sequence conditioned on an input sequence, shown here with extensions for the forward posterior probability estimation. Note that the extensions apply to both marked layers.

3.2.4 Experiments and results

Experiments using the modified BRNN structure to estimate the full conditional posterior probability of symbol sequences were run on the TIMIT database, which was also used for some experiments in the previous section of this chapter. The goal was here to estimate the probability of a phone sequence conditioned on a waveform, that was preprocessed to a feature vector sequence. Each vector of the input sequence represented an arbitrary long part of the original waveform.

3.2.4.a Experiments

Experiments were performed using the full TIMIT data set. To include the output (target) class information, the original 66-dimensional feature vectors, that were also used in section 3.2.2.b, were extended to 72 dimensions. In the first six dimensions, the corresponding output class is coded in a binary format (binary [0,1] $\rightarrow$ network input [-1,1]). Two different structures of the modified BRNN, one for the forward, and the other one for the backward posterior probability, were trained separately as classifiers using the cross-entropy objective function. The output neurons had the softmax activation function, and the remaining ones the $tanh()$ activation function. The forward (backward) modified BRNN has 64 (32) forward and 32 (64) backward states. Additionally, 64 hidden neurons were implemented before the output layer. This resulted in a forward (backward) modified BRNN structure with 26333 weights. These two structures by themselves as well as their combination, merged as a linear and a logarithmic opinion pool, were evaluated for phoneme classification on the test data.

3.2.4.b Results

The results for the phoneme classification task are shown in Table 3.5.

Table 3.5: Results for prediction assuming dependent outputs.

Structure	Classification Rate % TRAIN 61 (39)	Classification Rate % TEST 61 (39)
forward modified BRNN	79.11 (84.42)	72.70 (79.08)
backward modified BRNN	79.38 (83.27)	72.74 (77.44)
both merged, linear	83.57 (87.17)	77.53 (82.11)
both merged, logarithmic	83.89 (87.45)	77.75 (82.38)

Classification results for full TIMIT training and test data with 61 (39) symbols. Hypothesized input sequences are all test sentences.

It can be seen that the combination of the forward and backward modified BRNN structures results in much better performance than the individual structures. This shows that the two structures, though trained on the same training data set to compute the same probability $P(s_1, s_2, \ldots, s_T|\mathbf{x}_1^T)$, are providing different estimates of this probability, and as a result the combination of the two networks is giving better results. The slightly better results for the logarithmic opinion pool in contrast to the linear opinion pool show that it is reasonable to assume the two estimates for the probability $P(s_1, s_2, \ldots, s_T|\mathbf{x}_1^T)$ as independent, although the two structures are trained on the same data set.

It should be noted that the modified BRNN structure is only a tool to estimate the conditional probability of a *given* class sequence, it does not *provide* a class sequence with the highest probability, which is the ultimate goal. For this purpose, all possible class sequences have to be searched to get the most probable class sequence. The search itself involves many more challenging problems. A discussion and an example for a real application of a search for symbol sequences is given in chapter 4. In the experiments reported in this section, the class sequence provided by the TIMIT data base was used as the hypothesized class sequence that is fed into the network. Therefore, the context on the (right or left) output side is known and *correct.*

3.3 MIXTURE DENSITY RECURRENT NEURAL NETWORKS

One way of obtaining the optimal output vector sequence is to use the approach suggested by (2.7), which involves to build models that estimate an output vector given the input. This approach has been focussed on in section 3.2. A second approach, which is addressed in this section, is to use (2.6) and (2.5) directly. This involves, in contrast to the approach discussed up to now, building models of the input sequence given a hypothesized output sequence, which is equivalent to estimating the conditional sequence likelihood as

$$P(\mathbf{X}|\mathbf{Y}) = \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_1^T). \tag{3.24}$$

or

$$P(\mathbf{X}|\mathbf{Y}) = \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_{t+2}, \ldots, \mathbf{x}_T, \mathbf{y}_1^T) \tag{3.25}$$

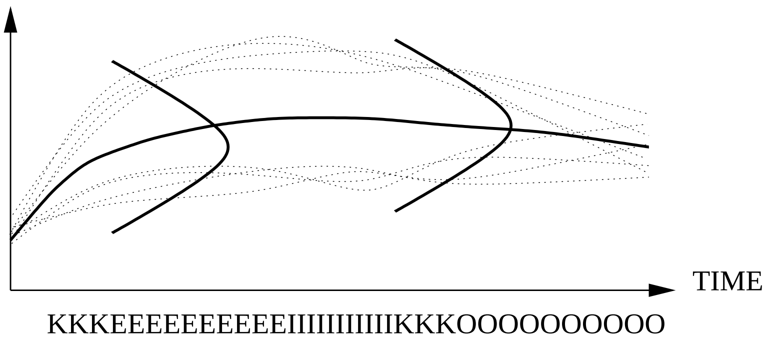
in its symmetrical version. These models are often called *generative* models, because they can be used to generate data with statistical properties that are similar to those of the data used to train the model.

Assume we want to model a continuous vector sequence, conditioned on a sequence of categorical variables (symbols) as shown in Figure 3.9. One approach, as discussed in section 3.2.2.a, is to assume that the vector sequence can be modeled by a uni-modal Gaussian distribution with a constant variance, making it a uni-modal regression problem. There are many practical examples where this assumption doesn't hold, requiring a more complex output distribution to model multi-modal data as briefly discussed in section 3.1.2.b. One example is the attempt to model the sounds of phonemes based on data from multiple speakers. A certain phoneme will sound completely different depending on its phonetic environment or on the speaker, and using a single Gaussian with a constant variance would lead to a crude averaging of all examples.

One traditional approach is to build models for each symbol (or group of symbols) separately as described in section 2.2.1 and 2.2.2. If conventional Gaussian mixtures are used to model the observed input vectors, then the parameters of the distribution (means, covariances, mixture weights) usually do not change with the temporal position of the vector to model within a given state segment of that symbol. This has many practical advantages regarding the parameter estimation process, but is probably suboptimal since the models' parameters are constant within the state and change in discrete steps at state boundaries as shown in Figure 3.10.

When used to model speech, a procedure often used to cope with this problem is to increase the number of symbols by grouping often appearing symbol sub-strings into a new symbol as described in section 2.2.2 and by subdividing each original symbol into a number of states as discussed in section 2.4.1. Another approach is to model the parameters of the distribution by time-varying functions, usually polynomials, as done

Figure 3.9: An artificial multi-modal regression problem

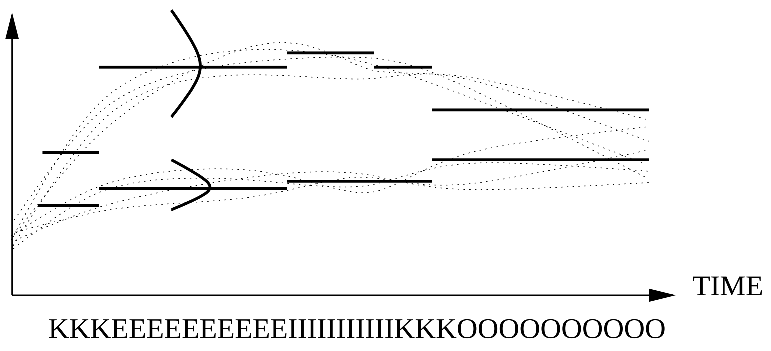


Abstract visualization of the problem to model human speech. A large number of example sequences of observation vectors (shown connected as continuous trajectories) depending on a given sequence of class labels, with each class representing for example a phoneme (here the name *Keiko* with given durations). In this synthetic example, the one-dimensional target data would be represented poorly by a uni-modal Gaussian distribution with a constant variance (which corresponds to using the squared-error objective function), which would average the two separate branches, indicated by the fat lines as the mean and constant variance of the single Gaussian. Compare this figure with Figure 3.10, Figure 3.11 and Figure 3.12 to see a subsequent improvement of the model.

for the Gaussian means in (Gish and Ng, 1996). These *polynomial segment models* can be extended to other distribution parameters at the cost of a more complex formalism (Fukada et al., 1998). A summary of various kinds of segment models can be found in (Ostendorf, 1996). An example of this approach is visualized in Figure 3.11.

Yet another alternative is explored in this section, where all parameters of the distribution modeling the continuous multi-modal targets are predicted by one bidirectional recurrent neural network, extended to model mixture densities conditioned on a vector sequence. The basic underlying problem has been discussed in section 3.1.2.b. Basics of non-recurrent mixture density networks, as introduced in (Bishop, 1994) and (Bishop, 1995), are reviewed in section 3.3.1. The bidirectional mixture density model structure proposed here removes some disadvantages of traditional approaches to model speech data. In particular it allows the modeling of smoothly time-varying means, variances (actually standard variations) and mixture weights of a Gaussian mixture distribution conditioned on a hypothesized output class sequence, as shown in Figure 3.12. An extension to the architecture similar to the one discussed in section 3.2.3 allows the estimation of time varying mixture densities conditioned on a hypothesized output sequence *and* a continuous vector sequence to model exactly the probabilistic terms occuring in (3.24) without *any* explicit approximations about the use of context. Experiments (section 3.3.3) show that both proposed approaches give a higher likelihood on test data when compared to the traditional approach using models of similar complexity.

Figure 3.10: Conventional Gaussian mixtures for multi-modal regression



One traditional approach to model multi-modal observation sequences conditioned on symbol states (here letters). Every state has an observation probability distribution, whose parameters are estimated from the observed data. Shown here are the means (and two selected variances for the state 'E') of a Gaussian distribution with two mixture components (fat lines). In general the parameters don't change within a state, which can be a poor approximation to represent the data in some areas, as indicated by the shown variances.

3.3.1 Basics of mixture density networks

In this section non-recurrent mixture density networks, as described for a simple case in (Bishop, 1994; Bishop, 1995) to model multi-modal target data distributions, are reviewed. The multi-modal target data distribution for the observation data $\mathbf{x}$ is here chosen as a mixture of kernel functions

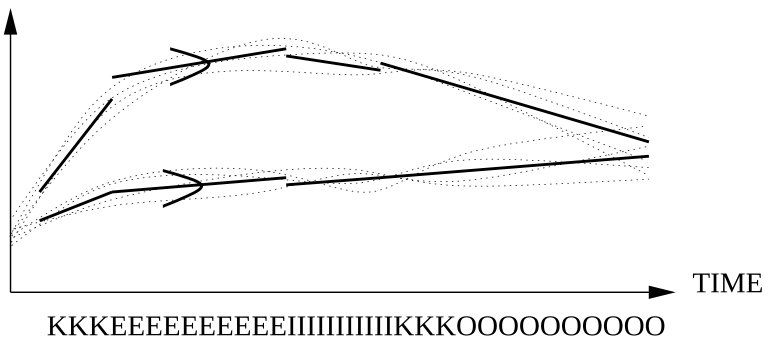
$$b(\mathbf{x}|\mathbf{y}) = \sum_{m=1}^{M} c_m(\mathbf{y}) b_m(\mathbf{x}|\mathbf{y}), \tag{3.26}$$

with M being the number of mixture components and $c_m(\mathbf{y})$ being the parameters for the mixing coefficients conditioned on the input data $\mathbf{y}$ to the network. The function $b_m(\mathbf{x}|\mathbf{y})$ represents the conditional density of $\mathbf{x}$ for the mth kernel. Note that the *target* vectors, that have to be modeled, live here in contrast to the rest of this thesis in the D-dimensional space $\mathcal{X}$ and are not denoted by a special symbol to simplify notation. The simplest choice for the kernel functions $b(\mathbf{x}|\mathbf{y})$ are Gaussians, here with *radial* covariances (diagonal covariance matrix with one common variance per mixture component)

$$b_m(\mathbf{x}|\mathbf{y}) = \frac{1}{(2\pi)^{D/2}\sigma_m^D(\mathbf{y})} exp\left\{ -\frac{1}{2}\sum_{d=1}^{D} \left(\frac{x^d - \mu_m^d(\mathbf{y})}{\sigma_m(\mathbf{y})} \right)^2 \right\} \tag{3.27}$$

with $\sigma_m^2(\mathbf{y})$ denoting the single variance per mixture component and $\boldsymbol{\mu}_m(\mathbf{y})$ being the mean vector of the mth mixture component, both conditioned on the K-dimensional

Figure 3.11: Polynomial segment models for multi-modal regression



An extension of the approach shown in Figure 3.10 allows the means in each state to vary with time constrained to a polynomial, here of first order. Compared to the model used in Figure 3.10 this model can be expected to give a better representation of the data, also indicated by smaller variances. Note that there are still discontinuities between states.

data $\mathbf{y}$. An example structure for a mixture density network with radial covariances is shown in Figure 3.13.

Another useful choice for the kernel functions are Gaussians with diagonal covariances

$$b_m(\mathbf{x}|\mathbf{y}) = \frac{1}{(2\pi)^{D/2}\prod_{d=1}^{D}\sigma_m^d(\mathbf{y})} exp\Big\{ -\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x^d - \mu_m^d(\mathbf{y})}{\sigma_m^d(\mathbf{y})}\right)^2\Big\} \qquad (3.28)$$

with $\sigma_m^d(\mathbf{y})$ representing the root of the variance for the dth data vector component in the mth mixture component.

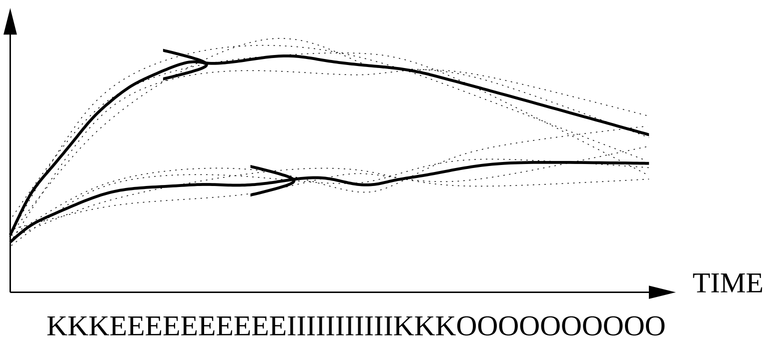
The goal of training is to maximize the likelihood of the training data (section 3.1.1), which leads for mixture density networks to the *objective function*

$$E = \sum_N ln\left\{\sum_{m=1}^{M} c_m(\mathbf{y}_n) b_m(\mathbf{x}_n|\mathbf{y}_n)\right\}. \qquad (3.29)$$

Compared to the uni-modal regression case discussed in section 3.1.2.a, the use of this objective function doesn't require the assumption that the output distribution can be described by a single Gaussian nor that the outputs have to be conditionally independent. For the case discussed here, though, the covariance matrices of the kernel functions have no off-diagonal elements and therefore cannot model correlations between any components of the observation data.

A neural network modeling the parameters of the distribution (3.27) will have D outputs for every mean vector from each of the M mixture components, denoted as activation $a[\mu_m^d]$ before being fed through the output layer, M outputs for the root of the variance per mixture component $\sigma_m(\mathbf{y})$, denoted as $a[\sigma_m]$ before being fed through the output layer, and M outputs for mixing coefficients $c_m(\mathbf{y})$, denoted as $a[c_m]$ before

Figure 3.12: BRNNs for multi-modal regression



Bidirectional mixture density networks allow to model smoothly time-varying means, variances and mixture weights of a Gaussian distribution (means shown as fat lines with variances at two selected points) conditioned on a state sequence.

being fed through the output layer, giving in this case $M(D+2)$ outputs altogether. For diagonal covariances using (3.28) the total number of outputs is $2M(D+1)$.

The predicted parameters have to fulfill certain constraints to insure that the resulting distribution is normalized, that is $\int_{\mathcal{X}} b(\mathbf{x})d\mathbf{x} = 1$. This can be achieved by choosing appropriate activation functions (see section 3.1.4) for the network outputs modeling the means, variances and mixture weights. In particular this means:

- **mean vectors $\boldsymbol{\mu}_m(\mathbf{y})$**

  There are no constraints on the range of the mean vectors, making the *linear* activation function $f_{act}(a) = a$ the most appropriate, such that

  $$\mu_m^d(\mathbf{y}) = a[\mu_m^d]. \tag{3.30}$$

- **roots of variances $\sigma_m(\mathbf{y}), \sigma_m^d(\mathbf{y})$**

  Variances and corresponding standard deviations cannot be negative. An appropriate activation function is the always non-negative *exponential* function $f_{act}(a) = e^a$, modeling the root of the variances as

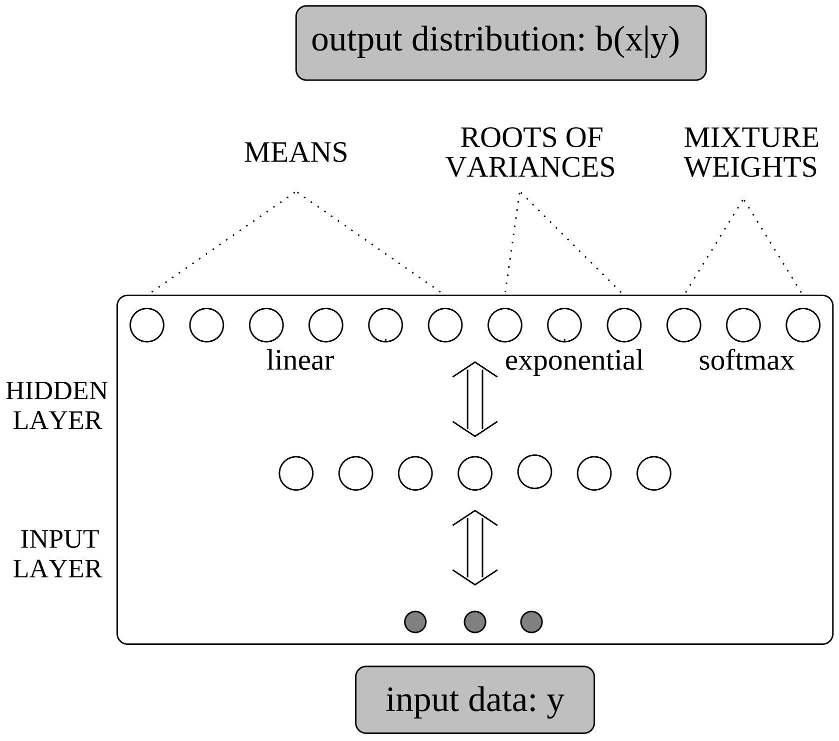
  $$\sigma_m(\mathbf{y}) = e^{a[\sigma_m]} \tag{3.31}$$
  $$\sigma_m^d(\mathbf{y}) = e^{a[\sigma_m^d]}. \tag{3.32}$$

- **mixture weights $c_m(\mathbf{y})$**

  The mixture weights have to be greater zero and have to sum up to one, making the *softmax* function the appropriate activation function, such that

  $$c_m(\mathbf{y}) = \frac{e^{a[c_m]}}{\sum_{i=1}^{M} e^{a[c_i]}}. \tag{3.33}$$

Figure 3.13: Mixture density network



Basic non-recurrent mixture density network structure, here shown for 3-dimensional input data and 2-dimensional output data for 3 mixture components and a radial covariance matrix.

The choice of the output activation functions together with the objective function (3.29) to minimize leads to the equations to calculate the gradient of (3.29) with respect to the network weights, which can then be used to update the network parameters iteratively by gradient descent procedures. The gradient for all weights can be calculated using back-propagation when suitable expressions of the "error" at the outputs of the network are available. For radial covariances this leads for the contribution of the nth data pattern using to

$$\frac{\partial E_n}{\partial a[\mu_m^d]} = \pi_m(\mathbf{x}_n,\mathbf{y}_n)\left\{\frac{\mu_m^d(\mathbf{y}_n)-x_n^d}{\sigma_m^2(\mathbf{y}_n)}\right\} \quad (3.34)$$

$$\frac{\partial E_n}{\partial a[\sigma_m]} = -\pi_m(\mathbf{x}_n,\mathbf{y}_n)\left\{\sum_{d=1}^{D}\left[\left(\frac{x_n^d-\mu_m^d(\mathbf{y}_n)}{\sigma_m(\mathbf{y}_n)}\right)^2-1\right]\right\} \quad (3.35)$$

$$\frac{\partial E_n}{\partial a[c_m]} = c_m(\mathbf{y}_n)-\pi_m(\mathbf{x}_n,\mathbf{y}_n) \quad (3.36)$$

and for diagonal covariances to

$$\frac{\partial E_n}{\partial a[\mu_m^d]} = \pi_m(\mathbf{x}_n, \mathbf{y}_n)\left\{\frac{\mu_m^d(\mathbf{y}_n) - x_n^d}{(\sigma_m^d(\mathbf{y}_n))^2}\right\} \tag{3.37}$$

$$\frac{\partial E_n}{\partial a[\sigma_m^d]} = -\pi_m(\mathbf{x}_n, \mathbf{y}_n)\left\{\left(\frac{x_n^d - \mu_m^d(\mathbf{y}_n)}{\sigma_m^d(\mathbf{y}_n)}\right)^2 - 1\right\} \tag{3.38}$$

$$\frac{\partial E_n}{\partial a[c_m]} = c_m(\mathbf{y}_n) - \pi_m(\mathbf{x}_n, \mathbf{y}_n) \tag{3.39}$$

with

$$\pi_m(\mathbf{x}_n, \mathbf{y}_n) = \frac{c_m(\mathbf{y}_n) b_m(\mathbf{x}_n|\mathbf{y}_n)}{\sum_{i=1}^M c_i(\mathbf{y}_n) b_i(\mathbf{x}_n|\mathbf{y}_n)}, \tag{3.40}$$

which can be back-propagated to calculate $\partial E/\partial \mathbf{w}$. Detailed derivations of the expressions (3.34), (3.35) and (3.36) in similar form can be found in (Bishop, 1995). In an actual implementation care must be taken to insure that

$$\sum_{i=1}^{M} c_i(\mathbf{y}_n) b_i(\mathbf{x}_n|\mathbf{y}_n) \neq 0 \tag{3.41}$$

for any of the N data patterns, which might be violated if D is large and the network is initialized to give mean and variance values far away from the actual solution. A simple solution in these cases is to set $\pi_m = 1/M$, corresponding to an unconditional uniform prior for the probability that vector $\mathbf{x}_n$ was generated by any mixture component.

3.3.2 Mixture density extension for BRNNs

Here two types of extensions of recurrent neural networks to mixture density networks are considered:

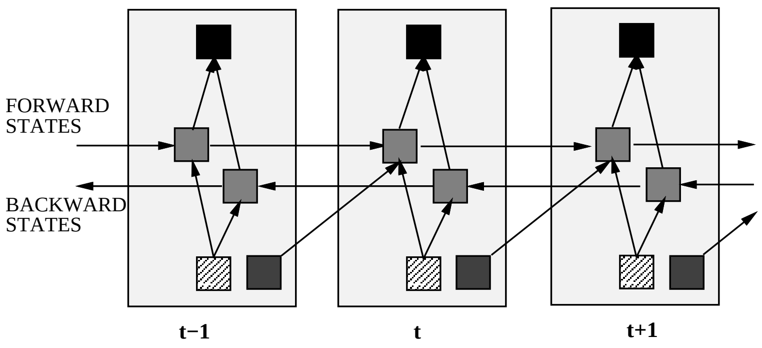
I) An extension to model expressions of the type $P(\mathbf{x}_t|\mathbf{y}_1^T)$, a probability distribution of a continuous vector conditioned on a vector sequence $\mathbf{y}_1^T$, here labeled as mixture density BRNN of *Type I.*

II) An extension to model expressions of the type $P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_1^T)$, a probability distribution of a continuous vector conditioned on a vector sequence $\mathbf{y}_1^T$ *and* on its previous context in time $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}$. This architecture is labeled here as mixture density BRNN of *Type II.*

The first extension of recurrent neural networks, as shown in Figure 3.4, to mixture density networks is not particularly difficult compared to the non-recurrent implementation, because for recurrent neural networks the outputs are treated the same way as for non-recurrent networks (that is, independently, which is necessary to use expressions of the form (3.11)). It is important to notice that this case is still an approximation to (3.24), because the generated distribution is not conditioned on the previous observations as required.

The second extension is very similar to the idea that was used in section 3.2.3, but here a slightly different approach was chosen. The basic architecture is shown in Figure 3.14. Note that this architecture allows to estimate the terms in (3.24) or (3.25) in its symmetrical version without making any explicit assumptions (see section 2.2.1), since all the information $\mathbf{x}_t$ is conditioned on is theoretically available.

Figure 3.14: BRNN mixture density extension (Type II)



A BRNN is extended to model expressions of the form $P(\mathbf{x}_t|\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{t-1},\mathbf{y}_1^T)$, with $\mathbf{x}$ being continuous and $\mathbf{y}$ being categorical in the case discussed here. This structure is similar to the one shown in Figure 3.8. For any t the neighboring $\mathbf{x}_{t-1},\mathbf{x}_{t-2},\ldots$ are incorporated by adding an additional set of weights to feed the hidden forward states with the extended inputs (the targets for the outputs) from the time step before. This includes $\mathbf{x}_{t-1}$ directly and $\mathbf{x}_{t-2},\mathbf{x}_{t-3},\ldots\mathbf{x}_1$ indirectly through the hidden forward neurons.

Different from non-recurrent mixture density networks, the extended BRNNs can predict the parameters of a Gaussian mixture distribution conditioned on a vector *sequence* rather than a single vector, that is, at each (time) position t one parameter set (means, variances, mixture weights) conditioned on $\mathbf{y}_1^T$ for the BRNN of type I and $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_{t-1},\mathbf{y}_1^T$ for the BRNN of type II.

3.3.3 Experiments and results

The goal of the experiments is to show that mixture density BRNNs are more suitable to model speech data than traditional approaches, because they rely on fewer assumptions. The speech data used here has observation vector sequences representing the original waveform in a compressed form, where each vector is mapped to exactly one out of K phonemes. Here several approaches are compared, which allow the estimation of the likelihood $P(\mathbf{X}|\mathbf{Y})$ with various degrees of approximations:

- Conventional Gaussian mixture model (baseline)
- Bidirectional recurrent mixture density network (Type I)
- Bidirectional recurrent mixture density network (Type II)

Conventional Gaussian mixture model (baseline):

$$P(\mathbf{X}|\mathbf{Y}) \approx \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{y}_t) \tag{3.42}$$

According to (2.6) and section 2.4.3 the likelihood of a phoneme class vector is approximated by a conventional Gaussian mixture distribution, that is, a separate mixture model is built to estimate $P(\mathbf{x}|\mathbf{y}) = P_k(\mathbf{x})$ for each of the possible K categorical states in $\mathcal{Y}$. In this case two assumptions are made, as already discussed in section 2.2.1:

I) The local likelihood is assumed not to depend on context around $\mathbf{y}_t$, which can be relaxed by introducing more states per phoneme and/or more states depending on neighboring phonemes (section 2.2.2).

II) The local likelihood is assumed not to depend on the neighboring vectors around $\mathbf{x}_t$, which is equivalent of assuming that neighboring $\mathbf{x}$ are independent of each other. As introduced in section 2.2.1, this assumption is called the *independence* assumption (2.9) and is for most sequential data considered to be a strong assumption, since neighboring vectors are often in a similar range and therefore not independent.

For the variance a radial covariance matrix (single diagonal variance for all vector components) is chosen to match it to the conditions for the BRNN cases below. The data is preprocessed such that all components of the input data vectors have a unit variance. For each of the K phonemes a separate model with a given number of mixture components is built based on the data associated to that phoneme class. The number of parameters for the complete model is $KM(D+2)$. Several models of different complexity were trained (Table 3.6).

Bidirectional recurrent mixture density network (Type I):

$$P(\mathbf{X}|\mathbf{Y}) \approx \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{y}_1^T) \tag{3.43}$$

One mixture density BRNN of type I, with the same number of mixture components and a radial covariance matrix for its output distribution as in the approach above, is trained by presenting complete sample sequences to it. Note that for type I all possible context-dependencies (assumption I) are automatically taken care of, because the probability is conditioned on complete sequences $\mathbf{y}_1^T$. The complete sequence $\mathbf{y}_1^T$ contains for any t not only the information about neighboring phonemes, but also the position of a frame within a phoneme. In conventional systems this can only be crudely modeled by introducing a certain number of states per phoneme. The number of outputs for each network depends on the number of mixture components and is $M(D+2)$. The total number of parameters can be adjusted by changing the number of hidden forward and backward state neurons, and was set here to 64 each.

Bidirectional recurrent mixture density network (Type II):

$$P(\mathbf{X}|\mathbf{Y}) = \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_1^T), \tag{3.44}$$

One mixture density BRNN of type II, again with the same number of mixture components and a radial covariance matrix for its output distribution as in the approach above, is trained by presenting complete sample sequences to it. Note that in this case assumption I *and* assumption II are taken care of, because exactly expressions of the required form (3.24) can be modeled by a mixture density BRNN of type II.

3.3.3.a Description of data

The recommended training and test data of the TIMIT database was used for the experiments. The number of possible categorical classes is the number of phonemes, $K = 61$. The TIMIT database comes with hand-aligned phonetic transcriptions for all utterances, which were transformed to sequences of categorical class numbers. The categorical data (input data for the BRNNs) is represented as K-dimensional vectors with the kth component being one and all others zero. The feature extraction for the waveforms, which resulted in the vector sequences $\mathbf{x}_1^T$ to model, was done in the same way as for the experiments described in section 3.2.2.c. Note that because of the normalization of the variances a single variance for each mixture component is a reasonable choice.

3.3.3.b Experiments

All three model types were trained with $M = 1, 2, 3, 4$, the conventional Gaussian mixture model also with $M = 8, 16$ mixture components. The number of resulting parameters, used as a rough complexity measure for the models, is shown in Table 3.6. The states of the triphone models were not clustered.

Training for the conventional approach using mixtures of Gaussians with radial covariances for each model was done using the EM algorithm, which converged after a few iterations (10 were used). The means of the Gaussians were initialized with a *k-means* clustering algorithm, with the number of clusters increased one at a time by splitting the cluster with the highest absolute likelihood, until the desired number of mixture components was reached. The variances were initialized with the cluster variances, the mixture weights were set to the relative frequency of the data vectors belonging to a certain cluster.

Training of the BRNNs of both types must be done using a gradient descent algorithm. Here ARPROP (section 3.1.3.c) was used ($\tau_+ = 1.1, \tau_- = 0.5$), with a weight

Table 3.6: Number of parameters for different types of models

mixture components	mono61 1-state	mono61 3-state	tri571 3-state	BRNN I	BRNN II
1	1952	5856	54816	20256	22176
2	3904	11712	109632	24384	26304
3	5856	17568	164448	28512	30432
4	7808	23424	219264	32640	34560
8	15616	46848	438528	–	–
16	31232	93696	877056	–	–

Note that the number of parameters of the BRNN models (64 forward-/backward neurons each) is not as sharply increasing as the number of parameters of the conventional Gaussian mixture models, because the number of hidden neurons was not altered.

update after each presentation of 1/100 of the training data set. All weights were initialized randomly in the range $w = [-0.01; 0.01]$, the initial step-size was chosen as $\delta_{start} = 0.001$. The training of the networks, using a cluster of eight workstations, converged to useful solutions after a few (8-16) passes through the training data. Total training time was between two and five hours per network.

The measure used in comparing these approaches is the log-likelihood of training and test data given the models built on the training data. In absence of a search algorithm to perform recognition this is a valid measure to evaluate the models since maximizing log-likelihood is the training criterion. It has to be noted though, that an increase in log-likelihood is not necessarily a sign of better performance, because it is possible that it is caused by pathological solutions caused by data sparseness (variances approaching zero), as discussed in (Bishop, 1995)(page 63). For the training of the conventional Gaussian mixture models, this case was treated by decreasing the number of mixture components for the problematic model until a stationary point of the likelihood was reached. During training of the BRNN models pathological solutions were not found in the experiments conducted for this thesis. Note that the given alignment of vectors to phoneme classes for the test data is used in calculating the log-likelihood on the test data – there is no search for the best alignment. A search procedure using the proposed mixture density BRNN models, necessary for a problem like speech recognition, is a separate issue and is addressed in the discussion of the results.

3.3.3.c Results

Figure 3.15 shows the average log-likelihoods depending on the number of mixture components for all tested approaches on training (upper line) and test data (lower line). The baseline 1-state monophones give the lowest likelihood. The 3-state monophones are slightly better, but have a larger gap between training and test data likelihood. For comparison on the training data a system with 571 distinct triphones with 3 states each was trained also. Note that this system has a lot more parameters than the BRNN systems (see Table 3.6) it was compared to. The results for the traditional Gaussian mixture systems show how the models become better by building more detailed models for different (phonetic) context, i.e., by using more states and more context classes.

The mixture density BRNN of type I gives a higher likelihood than the traditional Gaussian mixture models. This was expected because the BRNN type I models are, in contrast to the traditional Gaussian mixture models, able to include all possible phonetic context effects – i.e. a frame of a certain phoneme surrounded by frames of any other phonemes with theoretically no restriction about the range of the contextual influence.

The mixture density BRNN of type II, which in addition removes the independence assumption (2.9), gives a significant higher likelihood than all other models. Note that the difference in likelihood on training and test data for this model is very small, indicating a useful model for the underlying distribution of the data. An interesting point is, compared to the BRNN of type I, the approaching saturation in likelihood for the BRNN II model for four mixture components. It is reasonable to assume that because the predicted distribution at each time is also conditioned on the previous input observations, the resulting distribution is not as spread out as it would be without being conditioned on the previous input observations, such that a model with a low number of mixture components can approximate the real distribution reasonably well.

It is important to notice that the number of parameters for both BRNN models, used as a rough complexity measure, doesn't increase as quickly as the number of parameters for the traditional Gaussian mixture models. This explains the slower increase in likelihood for the BRNN models.

3.3.4 Discussion

The BRNN models of both types are significantly different from conventional Gaussian mixture models, which leads to a number of advantages and disadvantages. These and possible improvements are discussed in this section.

Modeling context-dependencies: In the conventional Gaussian mixture approach context-dependency is modeled by increasing the state space – within a phone by usually three states per phone, and across phones by building triphones or even higher order context-dependent models. Also, context-dependency is taken care of by allowing a large number of mixture components per state.

The BRNN model of type I automatically takes care of all possible context-dependencies without increasing the number of output classes, but *speaker dependencies* are not modeled at all, such that a rising number of mixture components for the BRNN I model can be expected to have the effect of modeling speaker differences.

Modeling speaker dependencies: In the conventional Gaussian mixture approach speaker dependencies are taken care of by allowing a large number of mixture components per state. This is also true for the BRNN model of type I. For the BRNN II model speaker dependencies are better incorporated, because the likelihood is also conditioned on the previous observations. Knowledge of the previous observation vector limits the range of the distribution to predict significantly, leading to considerably higher likelihoods as shown in Figure 3.15.

Automatic parameter sharing: One significant difference of the BRNN models compared to the Gaussian mixture models is, that all available parameters are automatically shared in the BRNN model, which has to be taken care of explicitly for the Gaussian mixture models. The use of all model parameters is optimized during training to maximize the likelihood on the data.

Using mixture density BRNNs for search: The ultimate goal is to perform *prediction* or recognition of sequences, as done in speech recognition. This is difficult using the BRNN models discussed here, because the estimation of the output distributions is conditioned on the *complete* hypothesized output sequence $\mathbf{y}_1^T$, which corresponds to an infinite number of states, if it would be used with a Viterbi search. To make these models suitable for recognition using a Viterbi search approximations will have to be made.

It is reasonable to assume that the local likelihood does depend significantly on only the hypothesized symbols near $\mathbf{y}_t$ and not on symbols far away. Therefore it is possible to use only context within a window of a few frames, for example

$$P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_1^T) \approx P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{t-1}, \mathbf{y}_t, \mathbf{y}_{t+1}), \tag{3.45}$$

which limits in this case the number of Markov states to a maximum of K^3, if K is the number of possible states in space $\mathcal{Y}$. This expression can be estimated by a mixture density BRNN, if all training and test sequence samples are cut to length three, ignoring any further context. If the window width is very small, an alternative to the recurrent version of the mixture density BRNN is to use a non-recurrent mixture density network.

Introducing additional prior knowledge: In the experiments conducted here there was one input per phoneme, which was either set to 1 or 0. A better representation might be the description of phonemes by an *attribute vector*, that describes the phonemes in a lower-dimensional space. In this way the dimensionality of the input vectors could be reduced and this method would allow to present additional prior knowledge about phonemes to the network (for example, vowel or consonant, plosive or non-plosive etc.), that is usually included by clustering states using a decision tree.

Computational efficiency: One disadvantage of the mixture density BRNN models is, that the likelihood calculation time compared to conventional Gaussian mixture models will increase if the same number of mixture components is used, because besides evaluation of the likelihood using a given distribution this distribution has to be generated first using the mixture density BRNN. This doesn't necessarily mean that applications using mixture density networks will be slower, because the run-time of practical applications depends heavily on the quality of the models.

In summary, the mixture density BRNN seems to be a very suitable model for multi-modal regression problems involving sequences. In particular, one simple extension to the basic model makes it possible to remove the *independence assumption* (2.9) completely.

3.4 SUMMARY

In this chapter to some extent the use of neural networks for supervised learning from sequences was discussed.

Basics of neural networks: In the first section, necessary basics of neural networks were reviewed and common architectures for supervised learning from sequences were shown.

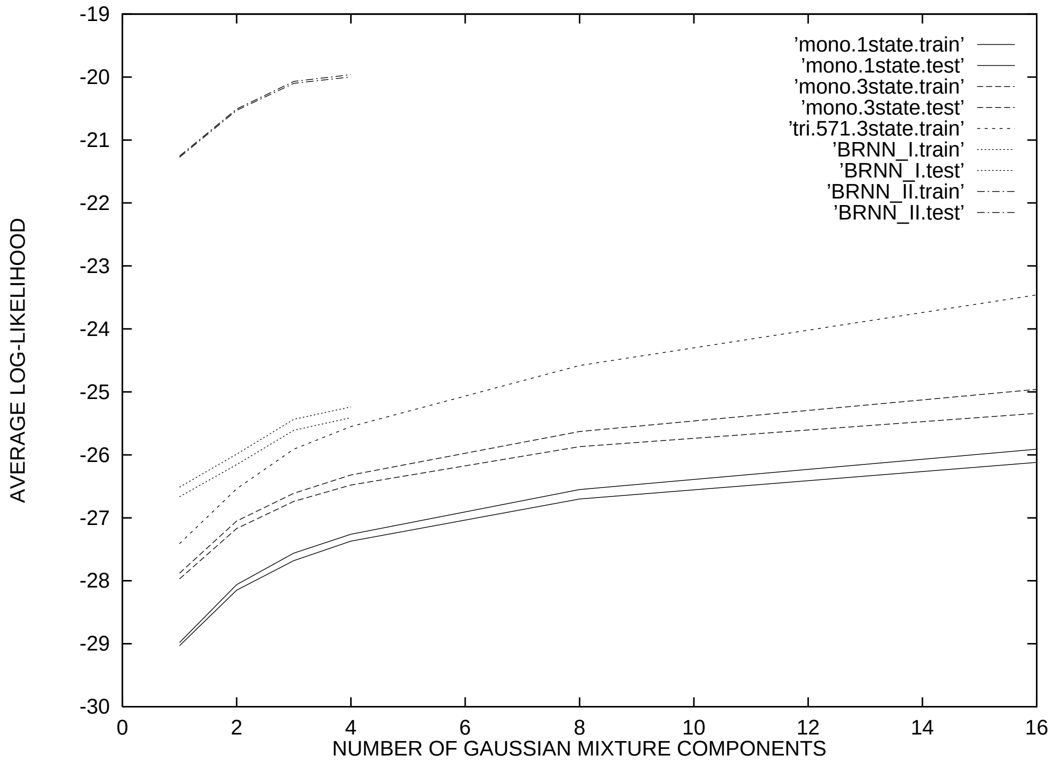
Bidirectional recurrent neural networks: In the second section, a simple extension to a regular recurrent neural network structure has been presented which makes it possible to train the network in both time directions simultaneously to model probabilistic expressions of the type $P(\mathbf{y}_t|\mathbf{x}_1^T)$. Because the network concentrates on minimizing the objective function for both time directions simultaneously, there is no need to merge outputs from two separate networks, each being responsible for one time direction. There is also no need to search for an "optimal delay" (an additional search parameter during development) to minimize the objective function in a given data/network structure combination, because all future and past information around the currently evaluated time point is theoretically available and does not depend on a predefined delay parameter. Through a number of experiments, it has been shown that the bidirectional recurrent neural network structure leads to better results than other NN structures for many problems. In all these comparisons the number of free parameters, used as an approximate measure for the complexity of the models, has been kept approximately the same. The training time for the BRNN is therefore about the same as for the other RNNs.

In the second section it has been shown how to use a slightly modified bidirectional recurrent neural network structure to directly estimate the conditional probability of hypothesized symbol sequences by modeling expressions of the form $P(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \dots, \mathbf{y}_1, \mathbf{x}_1^T)$ without making any explicit assumption about the shape of the output probability distribution. It should be noted that this modified BRNN structure is only a tool to estimate the conditional probability of a *given* class sequence, it does not provide the class sequence with the highest probability. For this, all possible class sequences have to be searched to get the most probable class sequence.

Mixture density recurrent neural networks: In the third section the proposed bidirectional structure was extended to model sequences which can be described by multi-modal target distributions. It was shown that recurrent mixture density recurrent neural networks can accurately model the parameters of a Gaussian distribution with a variable number of mixture components conditioned on a label input sequence. With the experiments it was shown that the predicted progression of speech conditioned on a label input sequence with the proposed models was better than with conventional methods based on Gaussian mixture distributions by comparing likelihoods measured on test data.

The mixture density BRNN model of type I, which takes care of all possible context dependency on the output side by modeling expressions of the form $P(\mathbf{x}_t|\mathbf{y}_1^T)$, improved the likelihood on the test data over the traditional approach with conventional Gaussian mixture models. The mixture density BRNN model of type II, which in addition removes the independence assumption for the observation vectors by modeling expressions of the form $P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_1^T)$, gave the best results. The BRNN model of type II can be regarded as the optimal model architecture in the sense that it makes it possible to model expressions which are necessary for sequence recognition problems without *any* explicit assumptions about the use of context. The complexity of both types of BRNN models can be controlled easily by changing the number of hidden neurons.

Figure 3.15: Mixture density BRNNs for multi-modal regression: Results



Average log-likelihoods for training (upper line) and test data (lower line) from the TIMIT database for all three tested approaches depending on the number of mixture components, using a radial covariance. The baseline 1-state monophones give the lowest likelihood. The 3-state monophones are slightly better, but have a larger gap between training and test data likelihood. For comparison on the training data a system with 571 distinct triphones with 3 states each was trained also. Note that this system has a lot more parameters than the BRNN systems (see Table 3.6) it was compared to. The results for the traditional Gaussian mixture systems show how the models become better by building more detailed models for different (phonetic) context, i.e., by using more states and more context classes. The mixture density BRNN of type I gives a higher likelihood than the traditional Gaussian mixture models. This was expected because the BRNN type I models, in contrast to the traditional Gaussian mixture models, are able to include all possible phonetic context effects – i.e. a frame of a certain phoneme surrounded by frames of any other phonemes with theoretically no restriction about the range of the contextual influence. The mixture density BRNN of type II, which in addition removes the independence assumption, gives a significant higher likelihood than all other models. Note that the difference in likelihood on training and test data for this model is very small, indicating a useful model for the underlying distribution of the data. An interesting point is, compared to the BRNN of type I, the approaching saturation in likelihood for the BRNN II model for four mixture components. It is reasonable to assume that because the predicted distribution at each time is also conditioned on the previous input observations, the resulting distribution is not as spread out as it would be without being conditioned on the previous input observations, such that a model with a low number of mixture components can approximate the real distribution reasonably well. It is important to notice that the number of parameters for both BRNN models, used as a rough complexity measure, doesn't increase as quickly as the number of parameters for the traditional Gaussian mixture models. This explains the slower increase in likelihood for the BRNN models.

Chapter 4

Memory-efficient LVCSR search using a one-pass stack decoder

Abstract

In this chapter a time- and memory-efficient implementation of a search algorithm for large vocabulary continuous speech recognition is presented. Advantages and problems of different search algorithms are discussed. Requirements for a modern decoder from an expert user's point of view are defined. It is shown how most of these can be integrated into a single decoder. Finally results from experiments on a Japanese newspaper dictation task are presented.

Speech recognition is one of the most attractive sequence processing problems. Finding the word sequence with the highest posterior probability given a stream of preprocessed acoustic input using all available constraints from the acoustic model, the language model and the pronunciation dictionary is a non-trivial task. There are in general thousands of possible words in the dictionary, which theoretically can be recognized in any combination to form an immense number of possible sentences. A challenging problem is how to do the search for the *best* sentence efficiently in time- and memory requirements.

This chapter describes the details of a fast, memory-efficient one-pass stack decoder for efficient evaluation of the search space for large vocabulary continuous speech recognition. A modern, efficient search engine is not based on a single idea, but is a rather complex collection of separate algorithms and practical implementation details, which only in combination make the search efficient in time and memory requirements. Being the core of a speech recognition system, the software design phase for a new decoder is often crucial for its later performance and flexibility. This chapter tries to emphasize this point – after defining the requirements for a modern decoder, it describes the details of an implementation that is based on a stack decoder framework. It is shown how it is possible to handle arbitrary order N-grams, how to generate N-best lists or lattices next to the first-best hypothesis at little computational overhead, how to efficiently handle cross-word acoustic models of any context order, how to efficiently constrain the search with word-graphs or word-pair grammars, and how to use a fast-match with delay to speed up the search, all in a single left-to-right search pass. The details of a disk-based representation of an N-gram language model are given, which make it possible to use LMs of arbitrary (file) size in only a few hundred kB of memory. *On-demand N-gram smearing*, an efficient improvement over the regular unigram smearing used as an approximation to the LM scores in a tree lexicon, is introduced. It is also shown how lattice rescoring, the generation of forced alignments and detailed phone-/state-alignments can efficiently be integrated into a single stack decoder.

The decoder named "Nozomi" [1] was tested on a Japanese newspaper dictation task using a 5000 word vocabulary. Using computationally cheap models it is possible to achieve realtime performance with 89% word recognition accuracy at about 1% search error using only 4 MB of total memory on a 300 MHz Pentium II. With computationally more expensive acoustic models, which also cover cross-word effects, that are essential for the Japanese language, more than 95% recognition accuracy [2] is reached.

4.1 INTRODUCTION

Large vocabulary continuous speech recognition (LVCSR), here defined as the recognition of arbitrary, continuously spoken sentences using a vocabulary of 5000 words or more, is currently limited to workstations and fast high-end laptops with a lot of memory. To make LVCSR work on PDAs, cellular phones, user-interfaces, wrist watches etc., it is necessary to find time- and memory-efficient algorithms. The efficiency of the *search engine* of a speech recognition system, that takes as input an utterance and generates in its simplest form the most probable word string, is unfortunately not based on a single algorithm, but on a complex collection of ideas and implementation details which only in combination make the search efficient. While the basic ideas can often be stated in a few words, their details and the implementation, which is crucial for good performance, is often not obvious and should be explained to the necessary detail in those cases.

Because the search engine combines all parts (pronunciation dictionary, feature vectors, acoustic models, language models) of a speech recognition system, it often defines the formats for module communication and is to a great extent responsible for the overall complexity of the whole system. The author's observation is, that the problem of too marginal improvements of state-of-the-art LVCSR systems has its origin not necessarily in a lack of innovative ideas, but often is due to a lack of possibilities for a scientific procedure to test them. The reason is in general an overwhelming complexity of the complete system, and research has to be aimed at reducing it.

Therefore, the goal for implementation of any search engine must be to

- minimize **time** and **memory requirements** and to

[1] "Nozomi" is the name of the fastest, most comfortable and most expensive bullet train in Japan, and also means "hope" in Japanese

[2] these are currently the best reported results on this task

- minimize overall **complexity** of the system while
- maximizing its **flexibility**

using all available knowledge sources to search for the desired output.

4.1.1 Organization of this chapter

In the first (general) part of the introduction (section 4.1.2) the term “search” for speech recognition is used in a loose way and necessary requirements for a modern search engine are defined. In the second (technical) part (section 4.1.3) definitions for the used terms are given and explained using the necessary mathematical equations. In the third part (section 4.1.4) known decoder types are classified and briefly explained. Section 4.2 explains the details of a memory-efficient one-pass stack decoder. Section 4.3 shows experiments and results for a 5000 word Japanese newspaper dictation task using this decoder. Specific problems regarding decoding for the Japanese language are discussed. The chapter concludes with section 4.4.

4.1.2 General

The essential content of any search algorithm for the best hypothesis in a LVCSR system can be summarized in simple words as:

1. Consider all possible hypotheses (different word sequences, pronunciations, alignments) using the dictionary
2. Assign a score to each hypothesis using the language model and the acoustic model
3. Put out the hypothesis with the highest score.

If this method would be applied in this form in practice, it would be impossible to find the best hypothesis because of the very large number of possible combinations of words, pronunciations and alignments for any reasonable sized dictionary in combination with the commonly used trigram language model.

As discussed above, the primary goal of any search algorithm must be to minimize the *time* and *memory* requirements for finding the best hypothesis while maintaining a minimal search error. Any practical search implementation (Alleva, 1997; Gopalakrishnan, 1995; Ney and Aubert, 1996; Odell, 1995; Paul, 1992; Ravishankar, 1996; Renals and Hochberg, 1996; Schwartz et al., 1996; Soong and Huang, 1991; Robinson and Christie, 1998) based on 1st order Hidden Markov Models (HMMs) uses various methods to achieve that. Some of them are: the *Viterbi search* to linearize the search with respect to time, the *beam search* to heuristically reduce the number of hypothesis at any time point, the use of a *tree lexicon* for the pronunciation dictionary to share computations for beginnings of words, the *language model lookahead* (Steinbiss et al., 1994), to approximate LM scores within words, the *fast-match* (Bahl et al., 1992; Gopalakrishnan and Bahl, 1996) to generate quickly acoustically likely word hypotheses.

A second goal for a search engine that is used in a research environment, or in cases where the output of the search engine is used as input to post-processing modules like translation engines, is its *flexibility*. It is often not enough to allow as input only a sequence of feature vectors to produce a word sequence with the highest score. In many cases more detailed outputs like lattices, N-best lists or detailed word, phone, or state-alignments are required. As language model search constraints one might want to use arbitrary order N-gram language models, word-pair grammars, word-graphs to simulate finite state automatons, or transcriptions to produce forced alignments. These and other requirements for a modern search engine, from an expert user's point of view, can be listed as:

- **possible inputs:**
  - utterance feature vectors (for on-demand likelihood calculation) or precalculated likelihoods (as often produced by neural network based systems)
  - lattice in standard lattice format (SLF)
- **possible outputs:**
  - first-best hypothesis (text or SLF)
  - N-best (text or SLF))
  - lattice in SLF
  - phone-/state-alignments
- **tree lexicon** (possibly > 65536 words) with multiple pronunciations and optional pronunciation scores
- **possible LM search constraints:**
  - arbitrary order N-gram language models
  - word-pair grammar (with scores)
  - word-graph in SLF
  - word transcription (for forced alignment)
- support for **word-within/cross-word context-dependent acoustic models** of any context order without needing to change the monophone dictionary
- optional **disk-based LM** to save memory
- **efficient LM lookahead** (unigram-smearing or on-demand N-gram smearing) to incorporate LM scores in tree lexicon as early as possible
- optional use of **fast-match models** to speed up search

A third goal is the realization of the search in a single left-to-right pass, using all available search constraints as early as possible. This reduces overall complexity of the search process, is conceptually attractive and is essential for on-line systems. Being able to run the search in one pass of course doesn't imply that it *has* to be run in one pass. In many cases, especially in a research environment, it often turns out that multi-pass strategies are more time-efficient for finding optimal solutions.

4.1.3 Technical

Speech recognition relies on the framework of statistical pattern recognition (Bishop, 1995; Duda and Hart, 1974; Huang et al., 1990), which has been shown to work well in practice. The goal for the search engine is to find the word sequence $\hat{W} = w_1, w_2, \ldots, w_M$ with the highest probability among all possible word sequences $\mathbf{W}$, which is conditioned on a feature vector sequence $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_T$. Every word of the *dictionary* (see 4.1.3.a for definition of terms), is usually mapped to a sequence of Hidden Markov Models (HMMs) (Huang et al., 1990), which themselves consist of *states* q, such that every word is equivalent to a Markov state sequence $Q = q_1, q_2, \ldots, q_{t-1}, q_T$. Using Bayes' rule $P(B|A) = P(A|B)P(B)/P(A)$ and the product rule of probability $P(A,B) = P(A)P(B|A)$ the conditional sequence probability $P(W|\mathbf{X})$ can be broken down to three terms and simplified as:

$$\begin{aligned}
\hat{W} &= \arg\max_{\mathbf{W}} P(W|\mathbf{X}) \\
&= \arg\max_{\mathbf{W}} P(\mathbf{X}|W) \cdot P(W) \\
&= \arg\max_{\mathbf{W}} \sum_{Q} P(\mathbf{X}|W,Q) \cdot P(W,Q) \\
&\approx \arg\max_{\mathbf{W}} \sum_{Q} P(\mathbf{X}|Q) \cdot P(W,Q) \\
&\approx \arg\max_{\mathbf{W}} \operatorname*{MAX}_{Q} P(\mathbf{X}|Q) \cdot P(W,Q) \\
&= \arg\max_{\mathbf{W}} \operatorname*{MAX}_{Q} P(\mathbf{X}|Q) \cdot P(W) \cdot P(Q|W) \\
&= \arg\max_{\mathbf{W}} \operatorname*{MAX}_{Q \epsilon Q_W} P(\mathbf{X}|Q) \cdot P(W) \cdot P(Q) \qquad (4.1)
\end{aligned}$$

Several assumptions have been made in this derivation:

a) The likelihood of the feature vector sequence given the state *and* the word sequence is equal to the likelihood of the feature vector sequence given *only* the state sequence, $P(\mathbf{X}|W,Q) = P(\mathbf{X}|Q)$. This implies that all acoustic information is captured by the state sequence and is independent of the actually uttered words.

b) The sum over all possible state sequences for a particular word sequence is approximated by the single best state sequence, which is termed the *Viterbi approximation*. This assumption in general doesn't effect the result but greatly simplifies the actual search and makes it possible to speak of state alignments and actual word boundaries (which would be fuzzy, if this assumption wouldn't be used).

The remaining three expressions stand for:

a) The *observation likelihood* [3]

$$P(\mathbf{X}|Q) = \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, q_1^T) \approx \prod_{t=1}^{T} P(\mathbf{x}_t|q_t), \tag{4.2}$$

which is generally modeled by a continuous density Gaussian mixture model or by a neural network. The evaluation of $P(\mathbf{x}_t|q_t)$ during the search usually takes a great percentage (typically 40-80 %) (Beyerlein and Ullrich, 1995) of the actual search time, so effort has to be made to reduce the number of likelihood calculations as much as possible.

b) The *transition probability* of the state sequence within words

$$P(Q) = \prod_{t=1}^{T} P(q_t|q_1, q_2, \ldots, q_{t-1}) \approx \prod_{t=1}^{T} P(q_t|q_{t-1}), \tag{4.3}$$

which is usually approximated by a first order Markov model.

c) The unconditional probability of the word sequence (*language model probability*)

$$\begin{aligned} P(W) &= \prod_{m=1}^{M} P(w_m|w_1, w_2, \ldots, w_{M-1}) \\ &\approx \prod_{m=1}^{M} P(w_m|w_{m-1}, w_{m-2}, \ldots, w_{m-(N-1)}), \end{aligned} \tag{4.4}$$

which is often approximated by an *N-gram*; the probability of a word given its $N-1$ predecessors.

In practical systems the search is never based on the raw probability estimates, but on their logarithms to stay in the given floating point range of current computers. This also converts the multiplications in (4.2), (4.3) and (4.4) to simpler additions. It is then usual to speak of a *score* rather than of a probability.

In practice it is found, that an exact implementation of (4.1) is often not optimal to achieve the best word recognition results. In general acoustic and language models are estimated on completely different corpora and many assumptions have to be made to make a practical implementation of a speech recognition system possible. To cope with these assumptions it is usually useful to weight the LM score against the acoustic score,

[3] throughout this chapter there is no distinction made between probability mass and density, usually denoted as P and p, respectively, because it is not necessary for discussion of the search

which is often realized by a multiplication of the *language model score* (*log* $P(W)$) by a *language model scale factor* λ. Also, there is often a *word deletion penalty* WDP, which is added to the LM score at every word end. A high WDP encourages word insertions, therefore penalizes word deletions. For M words in the hypothesis the use of these two heuristic parameters can be summarized as:

$$LMscore = \lambda \cdot log\ P(W) + M \cdot WDP \qquad (4.5)$$

4.1.3.a Definitions

Here definitions of terms are collected, which are frequently used in the context of search for speech recognition, and also in this thesis.

word: the ASCII sequence defining a word in the conventional sense, for example "car"

word-ID: a unique identification number or ASCII sequence for any logical word in the dictionary (note that homonyms like "arm" (part of body) and "arm" (weapon) would have a different word-ID)

word-ID list: a list of all word-IDs that are used during the search, contains information about whether a word triggers the language model or not (like for example silence, laughter, cough etc.)

(physical) state: smallest units of the acoustic model, which are each characterized by a method (function) to calculate its observation likelihood $P(\mathbf{x}_t|q^{(i)})$ at any time; in typical systems there are between 500 and 30000 different physical states

(logical) state: smallest unit of an HMM model, is characterized by its observation number (from the physical state) and its directed connections to other logical states (transitions)

HMM model: a collection of logical states, typically three to model a phone plus a non-emitting init and exit state; in a tied-state system different HMM models can share several physical states

phone: smallest modeling unit for a word, represented by a single HMM model; there are context-independent phones (monophones) or context-dependent phones (triphones, quintphones etc.) – context-dependent phones that depend on information beyond word boundaries are called *cross-word models*

pronunciation: a sequence of phones which specify the pronunciation of a word; can have a pronunciation weight associated

recognition unit: a word-ID plus its pronunciation; equal word-IDs with different pronunciations (and vice versa) are different recognition units

dictionary: a list of recognition units (word-IDs plus pronunciation), optional outputs and optional pronunciation weights; three example lines:

```
arm_1 [arm] 0.234 aa r mh
arm_2 [arm] 0.456 aa r mh
armageddon 0.55 aa r mh ae g ae dd n
```

a word-ID can occur several times to account for alternative pronunciations of a word

tree lexicon: internal representation of the pronunciation dictionary; a tree-based collection of all pronunciations in the dictionary as *lexical nodes* each representing a phone (HMM model), such that equivalent beginnings of pronunciations are shared

lexical node: smallest unit of the tree lexicon, representing an HMM model; a node is an *end-node* if a pronunciation ends at it – note that end-nodes are not necessarily leaf-nodes of the tree ("arm" and "armageddon" share the first three phones and "arm" ends within the pronunciation of "armageddon"); equal pronunciations will have the same lexical end-node

acoustic model: collection of HMM models, which allow the computation of $P(\mathbf{X}|Q)$ and $P(Q)$ for any valid state sequence; is typically based either on continuous density Gaussian mixtures, discrete distributions or on neural networks

language model: the module which allows the computation of

$$P(W) = \prod_{m=1}^{M} P(w_m|w_1, w_2, \ldots, w_{M-1})$$

N-gram: language model which makes the approximation

$$P(W) = \prod_{m=1}^{M} P(w_m|w_{m-1}, w_{m-2}, \ldots, w_{m-(N-1)}),$$

with N being typically three (trigram) or two (bigram); usually allows the computation of $P(W)$ for any W using a backoff procedure

word-pair grammar: language model which makes the approximation

$$P(W) = \prod_{m=1}^{M} P(w_m|w_{m-1})$$

for a limited set of word-pairs; $P(w_m|w_{m-1})$ for word-pairs not in the set are zero

hypothesis: a word sequence including its pronunciation and word start/stop times, which is hypothesized by the decoder

language model state: two hypotheses are in the same LM state, if their tail cannot be distinguished by the currently used language model (example: the LM histories "I love you" and "I don't love you" are in the same LM state using a trigram LM, because the last two words are the same)

first-best hypothesis: the hypothesis with the highest total score

lattice: a graph made out of *arcs* and *nodes*, containing all hypotheses considered during the search including all different alignments and pronunciation variants

standard lattice format (SLF): a lattice format that can be passed around easily between modules (usually an ASCII string); a useful format is suggested in (Young et al., 1997)

node: part of a lattice, that joins partial hypotheses which end at the same time and are in the same LM state

arc: part of a lattice joining two nodes; an arc represents a recognition unit associated with at least its acoustic score

N-best list: the N best hypotheses, which differ by at least one word-ID (different alignments or pronunciations of the same word-ID sequence belong to the same hypothesis for this purpose)

state/phone alignment: every frame of an utterance labeled with a state number and a phone number

pass: one *pass* means to search once from left to right through the utterance (or from right to left) incorporating more of the available knowledge than in the last pass

full search: exhaustive search over all possibilities given the dictionary and the LM constraints $\Rightarrow$ in general not feasible

beam search: at any time point t only partial hypotheses of a score within a *beam* around some best score at that point are kept ($L_t >= L_{BEST,t} - beam$); heuristic use of beams makes any search non-admissible

admissibility: a search is called *admissible* if the algorithm guarantees to find the best hypothesis

LM lookahead: heuristic approximation of the LM scores within words, usually used with a tree lexicon

fast-match: method to quickly find acoustically likely matches for words

stack: collection of partial word hypotheses

search error: error that is caused by the search algorithm (usually by too heavy pruning) and not by a badly estimated acoustic model or language model

4.1.4 Decoder types

Every decoder implementation is different and a clear distinction between different decoder types can often not be made. In this thesis, it has been tried to distinguish them by their basic search strategy, namely the time-synchronous *transition network decoders* and the usually time-asynchronous *stack decoders*.

4.1.4.a Transition network decoders

The majority of the decoders currently in use are transition network decoders (Alleva, 1997; Murveit et al., 1993; Gauvain et al., 1994; Ney and Aubert, 1996; Odell, 1995; Ravishankar, 1996; Schwartz et al., 1996; Shimizu et al., 1997; Soong and Huang, 1991) which are based on a *transition network* of words (as HMM state sequences) that incorporates the used language model in its word transitions. In its simple static form all word-ends are connected to all word-beginnings via transitions that contain word bigram probabilities, such that the whole network can be viewed as a large first-order HMM containing thousands of logical states. This makes it possible to use the efficient and admissible *Viterbi algorithm* as well explained in (Rabiner and Juang, 1993) (pages 339–340) and (Young et al., 1997) (pages 11–13) to search for the optimal state sequence time-synchronously. Discarding states with a relatively low score at each time t has proven to efficiently reduce the amount of needed computation time to find the first-best hypothesis at no or little search errors. Pruning of states is often based on a heuristic beam around the best state or/and on a predefined number of states with a high score that remain active.

It is easy and efficient to use word unigrams and bigrams in such a network, because their scores can be incorporated into the transition network before the actual search starts, but it doesn't extend automatically to long-span language models (3-grams, 4-grams), which are necessary to reduce modeling assumptions and to achieve good performance in LVCSR. Long-span LMs are either incorporated through dynamic building of the network during the search or through multi-pass rescoring strategies (Schwartz et al., 1996), which are often also necessary to construct lattices or true N-best lists. These implementations require then a dynamic LM score lookup which is not needed if only unigrams (in case of a tree lexicon) or bigrams (in case of a linear lexicon) are used.

Since transition network decoders are run time-synchronously, meaning the state-space evaluation over for $t + 1$ is done after it was done for t, it is possible to run real on-line recognition without any additional delay imposed by the decoding algorithm.

4.1.4.b Stack decoders

Stack decoders can be defined as decoders that during decoding use some kind of a *stack* of partial sentence hypotheses each consisting of a certain number of words. In general the partial hypotheses on a stack are expanded by complete words time-synchronously using the dictionary to create new partial hypotheses which are inserted into other stacks. When all stacks except the last (result stack) are empty, the result stack will contain the first-best hypothesis, the N-best hypotheses or the respective lattices depending on the search mode.

Although in the context of decoders the storage container for partial hypotheses is historically called *stack*, which should be a Last-In-First-Out buffer (LIFO) given its name, it is in practice rather often a simple list or a tree of hypotheses ordered by some kind of total score. The total score the hypotheses on the stack(s) are ordered by can be a) the partial hypothesis' log-likelihood, b) an estimate of the log-likelihood of the complete utterance ($A^{\star}$ criterion) (Soong and Huang, 1991), or c) some other score that expresses the belief in the partial hypothesis' correctness (Gopalakrishnan, 1995), (Renals and Hochberg, 1996).

There are at least two different types of implementations for stack decoders: a) with only one stack that contains all partial hypotheses which might have different end-times (Paul, 1991; Paul, 1992) or b) with one stack for each time point, where each stack contains only hypotheses ending at that time (Renals and Hochberg, 1996). If there are many stacks, the *stack expansion* can either be time-synchronous (expand stack t before expanding stack $t+1$, which has been termed start-synchronous in (Renals and Hochberg, 1996) or time-asynchronous (any stack can be expanded next, completely or partially, depending on some algorithm to pick a stack that will probably lead to the first-best hypothesis (Gopalakrishnan, 1995)). Even when the stack expansion is time-synchronous, stack decoders are often said to search time-asynchronously, because the global state progression through the utterance is in general not time-synchronous like for transition network decoders.

All stack decoders operate at least on two levels of search: a) the outer level, which loops over the stacks (*word-level search*), and b) the inner level, which loops over time and states (or states and time (Robinson and Christie, 1998)) to search for complete words, starting from the end-time of the hypothesis to expand, which is called *state-level search* or word-within search. Every time a potential word-end is found during the time-synchronous word-within search, its language model score is looked up using the found word plus its history using the hypotheses which are to be expanded. Because the dynamic LM score lookup can take any word history into account, stack decoders can easily make use of any kind of N-th order Markov language model and also of non-Markov language models like link grammars etc. Especially N-gram models of any order are simple to implement (section 4.2.5), which is one of the major advantages of stack decoders over the transition network decoders.

The decoupling of the language model from the Viterbi search in the state space has several other advantages. Because the hypotheses generation is completely independent of the word-within search, the word-within search can be realized memory-efficiently without the need for token passing or backtrace pointer storage (section 4.2.1.b). Word lattices can be created easily in the first pass at little computational overhead (section 4.2.3.a). Using a similar procedure N-best lists can be created, optionally with all different alignments and pronunciation variants in a lattice within each N-best hypothesis, again in the first pass (section 4.2.3.b). LM lookahead procedures depending on the scores of the word history to expand are easily integrated as a separate module (section 4.2.6).

In stack decoders there are several ways to implement cross-word context-dependent acoustic models, which are necessary for good recognition results. A procedure shown to be computationally efficient for cross-word models of *any* context order is discussed in section 4.2.7. This procedure leads naturally to a possible use of fast-match models to generate acoustically likely word candidates quickly. In this chapter a novel version of using a fast-match in a stack decoder is discussed (section 4.2.8), which avoids some disadvantages of earlier implementations.

Historically stack decoders have often been used for lattice rescoring to integrate higher order LMs and to optimize search parameters, often in combination with $A^\star$ procedures (Soong and Huang, 1991). This type and other types of often needed lattice rescoring procedures are discussed in section 4.2.10, which all can be implemented as additions to the regular decoder.

The usage of word-graphs constraining the search using stack decoders is closely related to the usage of word-pair grammars and the generation of forced word-alignments (section 4.2.9). Detailed phone- and state-alignments, which are not available when, as mentioned above, no state-based backtrace pointers are stored, will have to be created on demand. This turned out to be particularly easy for the implementation described in this chapter (section 4.2.11).

One disadvantage of stack decoders is the fact that they usually evaluate the state space time-asynchronously within a certain range, which makes real online decoding impossible – there will be a time lag being equal to the range of the state evaluation. Although in practice this time lag is short (less than a second) compared to other time limiting factors during a real search and can also be avoided during silences, it might pose a problem in systems that must have a human-like response time.

A second principal disadvantage is that it is not possible to merge logical state theories within words, because the word-level search is separate from the word-within search, which is discussed in more detail in section 4.2.1.b.

4.2 A MEMORY-EFFICIENT ONE-PASS STACK DECODER

This section describes the details of a memory-efficient one-pass stack decoder, that is based on a multi-stack implementation with one stack per time frame, which is equivalent to a one-stack implementation with the stack entries ordered primarily by time and then by score.

4.2.1 Basic algorithm

As discussed above, a stack decoder works on two levels of search, the word-level search looping over stacks and the state-level search looping over time and logical states.

4.2.1.a Word-level search

Looping over stacks for the word-level search can be done time-synchronously (start-synchronously) or time-asynchronously depending on the stack expansion mode. Independent of this mode, which is a function of the *stacklist* (collection of all stacks), the basic word-level search, as shown at the end of this section, works as follows: First an initial temporary stack *stack* containing only an initial empty root hypothesis is generated. Then all partial hypotheses on the temporary stack *stack* are extended by one word using the state-level search that knows about the *stacklist*, such that the new partial hypotheses can be inserted into the correct stacks. When the current temporary stack is finished, a new temporary stack is popped from the *stacklist*. This can be any of the currently held stacks in *stacklist*, which will be the earliest one in time in case of a synchronous stack expansion, and any one of the available ones in case of a asynchronous stack expansion depending on the selection criterion. The temporary stack doesn't necessarily have to contain *all* partial hypotheses of the stack in *stacklist* it was generated from. Again, depending on the selection criterion, these could be only a subset of that stack. When there are no more stacks to be popped, the method finishes with returning the result (first-best, N-best, lattice, etc.). An example implementation using pseudo C++ code would be:

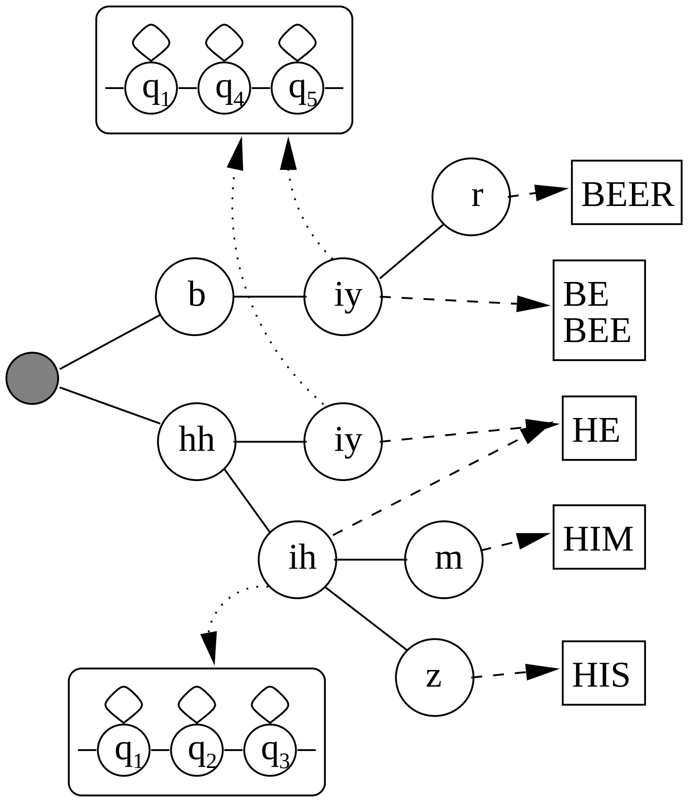
Word-level search:

```
{
  stack = stacklist.GET_INITIAL( hyp.ROOT() );

  do
  {
    statelevel_search.EXTEND( stack );

    stack.FORGET();
  }
  while( (stack = stacklist.POP()) );

  return( stacklist.RESULT() );
}
```

4.2.1.b State-level (word-within) search

The search on the state level extends all hypotheses of the passed temporary stack by one word using the pronunciation dictionary and inserts all generated new hypotheses in the corresponding stacks provided they are within the beam. The search is based on the pronunciation dictionary *dict* which is organized in a tree structure such that equivalent beginnings of pronunciations are shared to save redundant computations (Figure 4.1). This tree lexicon consists of *lexical nodes*, with each node pointing to its

Figure 4.1: Example for a tree-lexicon



Example for a tree-lexicon, as used for the search explained here, made out of only six words. *Lexical nodes* point on associated HMM models, here monophones, but possibly higher order models. Note that words don't have to end at leaf-nodes of the tree and there can be homonyms (words with the same pronunciation but a different meaning, here 'be' and 'bee') as well as multiple pronunciations for words, here 'he' as '`hh iy`' or '`hh ih`'.

associated HMM and all possible recognition units ending at it. The lexicon has a single root-node that does not have an HMM associated with it and defines the beginnings of all words. A node is called *active* if any of its logical HMM states is within the current beam. If a node is active, it carries its current time t and the log-likelihoods of all its states in a dynamically allocated chunk of memory. This memory is released to be used by other nodes if a node is deactivated.

During the state-level search, as shown at the end of this section, it is necessary to keep a list of all active nodes for the current and the next time slice (*alist, alist_next*), which are accessed by PUSH and POP operations. These lists contain only pointers to the corresponding lexical nodes and have to be ordered by the levels of the tree lexicon, such that the nodes closest to the root-node are popped first. This is necessary to insure that during actual propagation all states within a node are in the same time slice.

The state-level search then works as follows: After both active node lists are cleared, the non-emitting root-node of the tree lexicon is activated with the score of the best hypothesis of the stack to expand. It is then pushed on the current active node list (*alist*). The start time for the word-within search is the end-time of the stack to expand plus one.

The active nodes are propagated time-synchronously through the tree lexicon until the end of the utterance T (or some maximum word length) is reached or all nodes fell out of the beam and have been deactivated. The active nodes are popped from the list and are forward-propagated one time step assuming they have been in $t-1$ (`FORWARD()`). Forward propagation involves one Viterbi step within the currently worked on node. Since the node cannot be left during that step, it is sufficient to calculate only the new scores for every node-internal state without using any back-pointers. Although not containing much source code, method (`FORWARD()`) will take the largest part of the actual search time because the time consuming observation likelihood calculation functions for the physical states are called from it. Care should be taken in the loop ordering within `FORWARD()`, such that the expensive likelihood calculation functions are only called when actually needed. Also, already calculated likelihoods should be cached because in time-asynchronous stack decoders they will be used several times even when there are no shared physical states.

After the forward propagation the upper bound of the score at the current time is updated using `UPDATE_UPPERBOUND()`, if the pruning procedure is based on the beam around the best score at any time. It is not necessary when the hypotheses on the stacks are not popped depending on their partial log-likelihood as used in (Gopalakrishnan, 1995) or (Renals and Hochberg, 1996).

If any state of the current node is in the beam, it is a possible candidate for causing a stack expansion, otherwise it is deactivated. If a node in beam has its non-emitting exit-state activated and the node corresponds to a word end, the hypotheses on the temporary stack are expanded by one word (`stack.EXTEND()`), which involves looping over all hypotheses and all recognition units ending at this node, looking up the LM score for $P(rec\_unit|hyp\_history)$, generating the extension if the new partial hypothesis is within the current beam, and pushing it on the corresponding stack. Then, only if the exit-state is active, all successor nodes in the tree lexicon are activated (`ACTIVATE_SUCCESSORS()`), which involves copying the exit-state score of the current node into the init-state of the successor node, and pushing the node on the active node list for the next time slice (*alist_next*). Also, any nodes that are in the beam regardless their exit-states have to be pushed on this list.

Note that because of the LM lookahead procedure explained in section 4.2.6, which leads to an overestimate of scores within words, it is possible to use a *lower* (tighter) beam at word-ends compared to the beam within words.

Finally, when all nodes of the current time slice are finished, the two active node lists are swapped and the time is incremented to be ready for the next time slice.

A possible state-level search implementation, that was found to be efficient, is:

State-level search:

```
{
  alist.CLEAR();
  alist_next.CLEAR();

  dict.ROOTNODE.ACTIVATE( stack.TOPHYP.SCORE() );
  alist.PUSH( dict.ROOTNODE() );

  t = stack.TOPHYP.TIME() + 1;

  while( t < T && alist.NOT_EMPTY() )
  {
    while( (node = alist.POP() )
    {
      FORWARD( node, t );
      UPDATE_UPPERBOUND( node, t );

      if( node.IN_BEAM(t) )
      {
        if( node.EXIT_STATE.ACTIVE() )
        {
          if( node.IS_WORD_END() )
            stack.EXTEND( node, t-1 );

          ACTIVATE_SUCCESSORS( alist, node, t );

          node.EXIT_STATE.DEACTIVATE();

          if( node.NO_STATE_ACTIVE() )
            node.DEACTIVATE();
        }
        if( node.ANY_STATE_ACTIVE() )
          alist_next.PUSH( node );
      }
      else
      {
        node.DEACTIVATE();
      }
      alist.SWAP_WITH(alist_next);

      t++;
    }
  }
}
```

As pointed out in section 4.1.4.b, it is not possible to merge logical states theories *within* words from a state-level search that started at a different time carrying a hypothesis in the same LM state, like it is possible for transition network decoders. This will increase the average number of active states at any given time t. One obvious technical reason for this is that the status of intermediate states during any time of the state-level search is not stored, because it would require additional effort, time and memory. Another reason is the explicitly wanted decoupling of the state-level search from the LM, which prohibits any logical state merging, because the same logical state given only the dictionary will be a different one depending on its history, if as LM anything else than a first order Markov model (bigram with a linear lexicon or unigram with a tree lexicon) is used.

4.2.2 Pruning techniques

In the stack decoder described here several efficient pruning techniques to cut logical states, physical states, mixture components, lexical nodes, words or partial hypothesis with a relatively low score can be applied. Here a complete description of all these techniques is given.

Pruning has in general the effect, that the search becomes faster and uses less memory by increasing the chance of making *search errors* (section 4.1.3.a). There are many possible ways to prune and it is usually not hard to come up with new ideas for heuristic pruning. The disadvantage of using many sophisticated heuristic pruning techniques is, that in general they depend of each other and their outcome becomes harder to control. Some of them will inevitably lead to search errors, whose generating source will be hard to localize.

4.2.2.a Word-within pruning

Word-within pruning refers here to pruning of logical states during the forward propagation of a lexical node. Any of the states in the currently looked at HMM model are checked for validity. If a score of a state at time t is more than a *word-within beam* *width* below the best score ever at time t (or short: below the word-within beam),

$$Score(state, t) < BestScore(t) - WordWithinBeam, \tag{4.6}$$

then the state is discarded.

4.2.2.b Word-end pruning

Word-end pruning, as also mentioned in section 4.2.1.b, has two different functions in this stack decoder:

(1) To prune partial hypotheses that have been created during the state-level search, but whose LM state has a too low score.

(2) To prune partial hypotheses from stacks during their expansion, due to a rise of the best score of that stack (at time t), that pushed once valid hypotheses out of the beam.

Both of these mechanisms work analog to the word-within pruning procedure, such that if

$$Score(hyp, t) < BestScore(t) - WordEndBeam, \tag{4.7}$$

the partial hypothesis is discarded. Experiments show that in general a *tighter* word-end beam can be used without causing additonal search errors.

4.2.2.c Lexical node pruning

In this decoder the lexical nodes (HMM models), which are held in the active node list *alist* and used during the state-level search, are in general pruned only by the word-within-beam pruning and can hold as many nodes as necessary. Note that word-within-pruning doesn't require an ordered active node list.

For an additional pruning strategy, it is also possible to prune this list by only keeping the N best active nodes at any time t. This requires an ordering of the list in some form, but can provide an effective additional speed-up. Also, because the average number of active nodes at any frame is approximately proportional to the total search time, it is possible to approximately set the maximum time allowed to search through an utterance, which can be a great advantage during development, when good beam settings are still unknown.

4.2.2.d Mixture component pruning

If state-tied continuous-density Gaussian mixture models with diagonal covariances are used as acoustic models, then it is possible to use an effective additional pruning strategy during the likelihood calculation. In general the likelihood calculation takes a large amount of the search time, and the routine to look up acoustic likelihoods given a state number and an observation vector will be fairly optimized. This makes it difficult to find pruning strategies at that level that really do lead to a speed up, because in general all additional code in this routine makes the likelihood calculation slower.

Here a pruning strategy was used, that did help in many cases. The idea is the following: For diagonal covariances, the likelihood calculation routine contains a routine that calculates the distance of the observation vector to the mixtures, with each component weighted by the inverse variance for diagonal covariances. If this distance becomes larger than a certain heuristic threshold for a mixture,

$$Distance(mixture, observation) > Threshold \tag{4.8}$$

then the likelihood from this mixture component is ignored. This distance threshold check is done here after each component of the evaluated vector is added.

4.2.2.e Posterior pruning

If neural network acoustic models are used, which in general provide the posterior probability of a certain state given the observation vector, then a simple and effective pruning strategy mentioned in (Renals and Hochberg, 1995a; Renals and Hochberg, 1995b) can be used. If a certain state has a posterior probability below an heuristic threshold,

$$PosteriorProbability(state, observation) < Threshold \tag{4.9}$$

then this state is completely ignored.

4.2.2.f Pruning with triangular beams

Time-asynchronous stack decoders have the disadvantage that during the generation of partial hypotheses usually not all information at that time point is available – especially pruning is based on *currently best likelihoods*, which might and usually will improve the closer the stack expansion time point comes. This leads to the generation of many hypotheses which are later not expanded, in the experiments done here up to 95% not expanded partial hypotheses. This waste of resources can be avoided partially, when a more aggressive pruning is used at times when there's not all information available.

The simplest way to achieve more agressive pruning the further away the state-level search is moving from its start time t_{start}, is to use triangular beams. The beam at each time depends on the position of the current state evaluation time relative to the start time of the state-level search. At $t = t_{start}$ the beams (word-within beam, word-end beam) are at their set width (maximum), at $t = t_{start} + wordlength_{max}$ they become zero (minimum). This type of pruning efficiently speeds up the search and led in the experiments done here to very few or no search errors compared to the regular beams.

4.2.3 Stack module

The collection of *stacks* for each time t are accessed by PUSH() and POP() operations taking partial hypotheses as arguments. Because they are used frequently and usually contain a few to several hundred entries in a typical application, the *stacks* (or more precisely lists as discussed above) have to be set up efficiently. The container types used in other decoders are often special tree-structured lists, which are ordered by score and limited in the number of entries (Renals and Hochberg, 1995b). Here a different method is described which was found to be most efficient and simple to implement.

Pushing a hypothesis on a stack involves a check whether a hypothesis in the same LM state is already on that stack. If yes, the scores of the two hypotheses are compared and the better one is inserted into the stack, the other one discarded. In case of an N-gram LM the LM state check means to compare the last $MAX(N-1,1)$ history word-IDs. One word has to be compared as a minimum to not violate the at least first order Markov assumption for the complete speech model. Although checking for LM state equivalence for N-gram LMs can theoretically be done in $O(1)$ using a hash table with the $N-1$ words history as the key, it was found that it is in practice not more efficient than a simple non-ordered unlimited list that is searched through linearly up to an average stack size of a few hundred hypotheses. Pushing a hypothesis on a stack can also improve the upper bound for the score at this time, which has to be checked for. Popping a hypothesis from a stack is an $O(1)$ process, since it doesn't matter in what order the hypotheses in beam are extended for the implementation described here.

The stacks containing mainly pointers to hypotheses can be set up efficiently in a ring-buffer if a maximum word length is defined, which is necessary for on-line operation.

4.2.3.a Lattice generation

Lattices, as defined in section 4.1.3.a, are a convenient form of storage for the hypotheses that are considered during the search, and their generation is often necessary for systems that need to post-process the recognition output such as for translation engines, information retrieval systems, or multi-pass search strategies. Stack decoders can easily generate lattices with little computational overhead in the first pass by slightly modifying the LM state check procedure. Instead of discarding the hypothesis with the worse score in case of LM state equivalence it can be linked into the lattice. A pointer on the best arc back has to be updated to not loose the best hypothesis for the current LM state and future reference. Compared to the generation of the first-best hypothesis there is only little overall increase in memory for the storage of the additional arcs in the lattices (section 4.3).

4.2.3.b N-best list generation

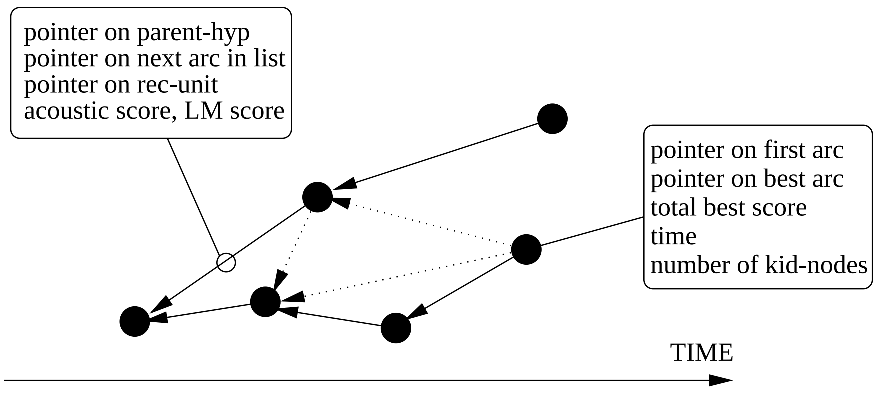
The hypotheses in an N-best list differ by at least one word-ID. This can be directly checked for by extending the LM state check procedure to the complete history instead of just the $MAX(N-1,1)$ history word-IDs like necessary for obtaining the first-best hypothesis. It can be done either exactly by checking each word, or approximately by using a hash function for the history. A lattice within the N-best list, referred to as

N-best lattice, which includes all possible alignments and pronunciation variants for the same word-ID sequence in the possible paths taken backwards from a lattice node, can be produced by merging hypotheses instead of replacing them as discussed above for the first-best lattices. Compared to the lattice generation this procedure uses only little additional memory for the extra nodes of the hypotheses, which are needed because of the increased LM state space, and only little additional time as shown in section 4.3. Since for the generation of N-best lists only the LM state check procedure was modified, they can be generated in the first pass like lattices.

4.2.4 Hypotheses module

A hypothesis in memory is made of objects called *hyp-nodes* and *arcs*, starting at $t = 0$ from a single root node, and ending in either one end-node (first-best or lattice) or in many end-nodes (N-best list, N-best lattice). An example is shown in Figure 4.2. Each node contains its time and best total score of the hypothesis up to this point. The arcs connected to ancestor nodes (parent nodes) are set up as a single linked list starting from the current node, which also contains a pointer to the arc belonging to the best hypothesis going back from this node. Every node contains also a counter on how many *kid-nodes* it is connected to (how many arcs contain a pointer to the current node), which is necessary for efficient memory management of these objects.

Figure 4.2: Hypotheses storage format



Example for the memory-efficient storage format for hypotheses made out of *hyp-nodes* (black dots) and *arcs* (arrows). Shown is a lattice, all arcs but the best are dotted.

An arc defining a recognition unit with scores will contain at least a pointer on the recognition unit in the dictionary, the acoustic score for it, a pointer on its parent hyp-node, and a pointer to the next arc of the linked lists of arcs (see Figure 4.2).

Memory management of hyp-nodes and arcs is best set up using linked lists, such that getting or forgetting them can be done using simple pointer copying. The usage of OS memory management routines can be minimized by allocating blocks of objects if none are left in a buffer made out of linked lists of the needed objects. Forgetting a pruned hyp-node involves also forgetting all linked arcs. Forgetting an arc means also forgetting all hyp-nodes they point to, if these hyp-nodes have no kid-nodes. If implemented in this recursive manner, the total memory for the hypotheses used during the search will be approximately proportional to the active number of hyp-nodes and arcs, which is generally between 10^2 (first-best mode) and 10^5 (lattice mode) for an average LVCSR application. In the implementation described here one hyp-node occupies on average ≈ 30 bytes, one arc ≈ 20 bytes.

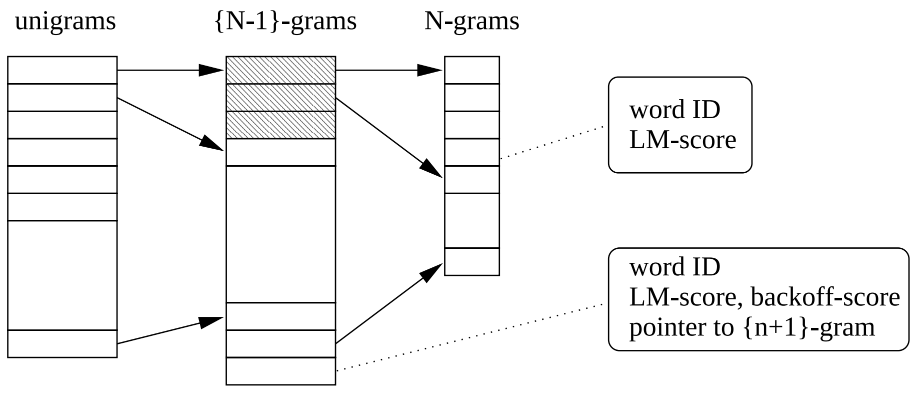
4.2.5 N-gram module

The N-gram module is responsible for generating $P(w_N|w_1, w_2, \ldots, w_{N-1})$, the probability of a word given its $N-1$ words history, which is in general stored in a lookup table and might require backing off to lower order N-grams using the approximation $P(w_N|w_1, w_2, \ldots, w_{N-1}) \approx P_{backoff}(w_1, w_2, \ldots, w_{N-1}) \cdot P(w_N|w_2, w_3, \ldots, w_{N-1})$. The N-gram of an average LVCSR system usually occupies the most memory and is accessed on average a few hundred to a few thousand times per frame, so it has to be stored in a format that is memory-efficient and allows fast access.

A useful format was found to be the following, which is shown in Figure 4.3: For a back-off N-gram LM store all n-grams with $n = 1, 2, \ldots, N$ in a table for each n. Each entry in a table has a word-ID, its LM probability and back-off probability, and a pointer to the beginning of the list of extension word entries in the table holding the $(n+1)$-grams. For the table with the N-grams the pointers are not necessary, since no higher order $(N+1)$-grams are following. Each part of an entry table holding a particular set of extension words is ordered by its word-IDs to allow fast access using a binary search. The number of a set of extension words on any level n doesn't have to be stored because it can be calculated by subtracting the pointer (on level $n-1$) on the current set from the next pointer (also on level $n-1$) on the next set. If the next set on level n doesn't happen to have any extension words, indicated by a NULL pointer on level $n-1$, the next non-NULL pointer on level $n-1$ has to be searched for, which is usually not more than a few entries away. The last entry on any level n has to be treated as a special case – the number of extension words has to be calculated as the pointer difference between the entry and the entry at the beginning of the next level, if levels are stored consecutively in memory.

The memory requirements for this N-gram representation are 8 bytes per entry for all $\{n < N\}$-grams, and 4 bytes for all N-grams, assuming 4-byte pointers, 2-byte word-IDs and 1-byte representations for the LM probability and the back-off probability, uniformly distributed across their log-scores, which was found to be a sufficient accuracy not to cause any errors. Access time for this storage format is of $O(1)$ for the unigrams and of $O((n-1) \cdot log_2(K))$ for the $\{n > 1\}$-grams using a binary search, with K being the average number of words following any n-gram entry. The average access time can be slightly improved by caching LM states and their scores in a hash table for all $\{n > 1\}$-grams that have been accessed before. This improves average access time

Figure 4.3: N-gram storage format



Storage format for a fast accessible and memory-efficient N-gram, which is used in the same form for its disk-based representation. The shadowed region is an example for N-grams that would be loaded into memory for the disk-based LM to search for the correct entry.

to $O(1)$ for already used $\{n > 1\}$-grams, but requires an additional check whether a certain LM state is already in the hash table or not.

4.2.5.a A disk-based N-gram

It has been found that for average LVCSR applications most of the entries in an N-gram are never actually used and a disk-based representation of the N-gram can limit memory requirements to a few hundred kB for N-grams of *any* size (Ravishankar, 1996). The search for the N-gram scores on disk during the search is of course very time-consuming and has to be minimized using an efficient caching scheme. An efficient implementation was found to be the following: Unigrams are stored in memory and all $\{n > 1\}$-grams are stored on disk in exactly the same format that is used for the representation in memory from section 4.2.5, such that looking up an n-gram can be done using the same algorithm. A set of extension words following an n-gram is loaded into temporary memory to run the binary search for the correct word-ID in memory and not on disk. Care must be taken in making this temporary buffer large enough to definitely include all information that is necessary to calculate the number of extension words for any entry within the set of extension words. The LM states that have been used once are cached in a memory-based hash table to minimize disk access. An alternative to caching only the used LM state is to cache all LM states that belong to any set of extension words loaded during the search for the required LM state.

4.2.6 LM lookahead

Most current LVCSR systems use some kind of LM lookahead to approximate the LM scores of the possible current LM states within words and use the exact LM scores only at word-ends. When a tree lexicon is used, the exact LM state often cannot be known until reaching a word-end node. The use of LM lookahead probabilities $p_{lookahead}$, which belong to every node in a tree lexicon, can speed up the search considerably, because nodes with a weak LM score can be pruned early. Suppose the LM lookahead probabilities are already set, they are used during the search through the tree lexicon as:

- When a node is entered, add $p_{lookahead}(node)$ to current total score.
- When a node is left, subtract $p_{lookahead}(node)$ from current total score.

4.2.6.a Unigram smearing

Unigram smearing (Steinbiss et al., 1994; Alleva et al., 1996; Ortmanns et al., 1997) is a commonly used procedure and heuristically sets $p_{lookahead}$ for each node in the lexicon is as follows:

1. Calculate for each word-end node in the lexicon the maximum of all unigram scores of the words that end at this word-end node (a set of words denoted as $W_{word-endnode}$). Note that several words could end at one word-end node because of homonyms and multiple pronunciations.

$$p_{lookahead}(\text{word-end node}) = MAX\{P(w)\} \quad \text{with} \quad w \in W_{word-endnode} \tag{4.10}$$

2. For all non-word-end nodes set $p_{lookahead}$ recursively to the maximum of all child-nodes.

$$p_{lookahead}(\text{non-word-end node}) = MAX\{p_{lookahead}(\text{child-nodes})\} \tag{4.11}$$

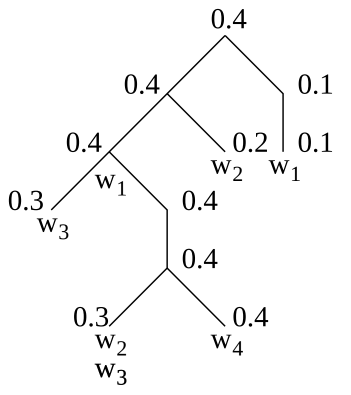
Note that unigram smearing is independent of the currently extended word hypothesis and is therefore a *static* procedure – it has to be calculated only once which can be done in advance. An example is shown in Figure 4.4.

4.2.6.b On-demand N-gram smearing

On-demand N-gram smearing is a LM lookahead procedure that incorporates the LM state constraints of the currently extended hypotheses including their scores (Neukirchen and Willett, 1997). This results in better estimates of the real LM probabilities, compared to the regular unigram smearing procedure, which leads in turn to more accurate pruning and therefore can lead to a faster search. The algorithm works as follows:

1. Initialize all $p_{lookahead}$ with the unigram smearing procedure shown above before the search starts.

Figure 4.4: Uni-gram smearing



trigram language model

1-grams:	w_1	0.1
	w_2	0.2
	w_3	0.3
	w_4	0.4
2-grams:	$w_1 w_3$	0.5
	$w_3 w_4$	0.6
3-grams:	$w_2 w_3 w_1$	0.7

Example for unigram smearing using a tree-lexicon and a trigram LM, which is independent of the hypotheses to extend. Note that there are words with multiple pronunciations (same word ends at several leaf-nodes) as well as pronunciations with multiple words (several words ending at the same leaf-node, called *homonyms*). Also note, that the information from the bi-grams and tri-grams is not used.

2. Calculate, for each set of word hypotheses H_i to expand, the maximum N-gram probability $P(w|H_i)$ of all *existent* N-gram entries (H_i, w) in the language model excluding the unigrams, because they were already set during the unigram initialization (step 1). Identify the corresponding word-end nodes belonging to w (which could be several because of homonyms and multiple pronunciations) and set $p_{lookahead}$ to the maximum of the calculated probability and the unigram $p_{lookahead}$ probability already set.

$$\begin{aligned} p_{lookahead}(\text{word-end node}) &= MAX\{P(w|H_i)\} \;\; \forall\, H_i \\ & \quad \textit{and} \\ & \quad \forall\, w \in \{(H_i, w) \text{ existent in N-gram}\} \end{aligned} \quad (4.12)$$

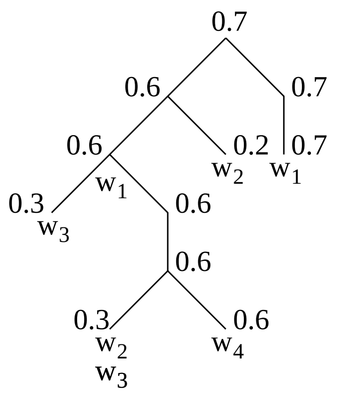
To use not only the LM states but also the relative scores of the current hypothesis to the best hypothesis in the current set to extend use $MAX\{P(w|H_i)\} - \text{score}(H_{top}) + \text{score}(H_i)$ instead of $MAX\{P(w|H_i)\}$.

3. For all non-word-end nodes set $p_{lookahead}$ to the maximum of all child-nodes.

$$p_{lookahead}(\text{non-word-end node}) = MAX\{p_{lookahead}(\text{child-nodes})\} \quad (4.13)$$

Note that this procedure has to be invoked each time a new set of word hypotheses is extended and cannot be done in advance like for unigram smearing. In spite of the additional computation the more accurate LM probabilities lead to more accurate pruning which can lead to a speed-up of the whole search. An example for this procedure is shown in Figure 4.5, which should be compared to Figure 4.4 illustrating the unigram smearing procedure.

Figure 4.5: On-demand N-gram smearing



trigram language model

1-grams:	w_1	0.1
	w_2	0.2
	w_3	0.3
	w_4	0.4
2-grams:	$w_1 w_3$	0.5
	$w_3 w_4$	0.6
3-grams:	$w_2 w_3 w_1$	0.7

hypotheses to extend

1) $w_4 w_1 w_2 w_3$
2) $w_2 w_4$
3) $w_2 w_1 w_1 w_3$

Example of on-demand N-gram smearing using a tree-lexicon and a trigram LM using the constraints from the hypotheses to extend.

Results from experiments showing the impact of on-demand N-gram smearing compared to unigram smearing are shown in section 4.3.

4.2.7 Cross-word models

Cross-word models are context-dependent acoustic models that span over word boundaries. Their use in any decoder is complicated and time consuming. Various implementations have been described (Bahl et al., 1993; Alleva, 1997), which, in the case of transition network decoders, are often limited to cross-word triphones. Whether cross-word modeling is really necessary or not depends heavily on the speaking style and on the definition of a word in the language to be recognized, and is more likely to make a difference when there is no pause at word boundaries. In this chapter it is shown that cross-word modeling is essential for recognition of read newspaper articles in Japanese (section 4.3).

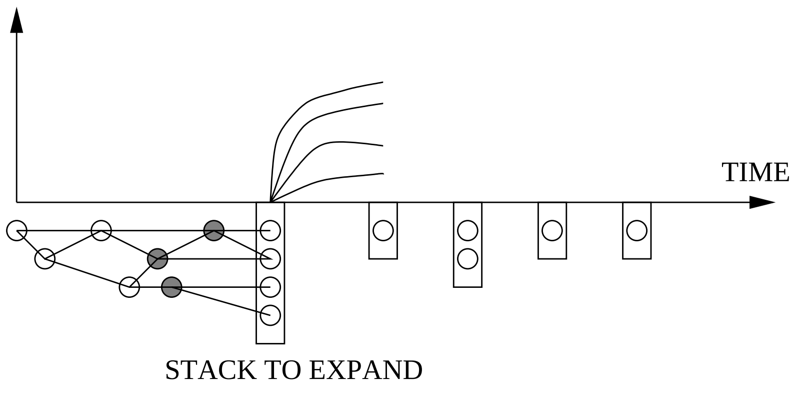
A procedure to deal with cross-word models of *any* order (triphones, quintphones, etc.) incorporating cross-word effects in a delayed manner was found to be very efficient in time and memory requirements, and is especially well suited for a stack decoder:

- Run the state-level search for any set of hypotheses to expand with word-internal context-dependent models only.
- When popping the hypotheses from a stack to expand, realign and rescore the last M words using cross-word models at the word boundaries before entering the state-level search to find the extension words.

- As cross-word effects are incorporated with a one-word delay, it is also necessary to realign the last M words for all hypotheses on the final result stack.

This procedure as illustrated in Figure 4.6 incorporates all cross-word effects within the last M words, and is optimal under the assumption that no search errors are made for cross-word triphones with $M = 2$ for most cases and possibly $M = 3$, if the word before the last word is a one-phone word. To capture all cross-word effects with quintphones theoretically $M = 5$ is necessary, if all words in the dictionary would be one-phone words.

Figure 4.6: Cross-word model incorporation



Visualization of the method to incorporate cross-word models of any context order. Circles denote hyp-nodes, filled circles are the word boundaries that are corrected by the procedure using cross-word models *before* the stack (box) is expanded. In this example only two words are realigned, but there could be more as discussed in the text. The same method is used for the fast-match to rescore acoustically likely word candidates (section 4.2.8).

The realignment for each hypothesis to extend is in detail done as follows: Take the last M words and find the correct (cross-word) HMMs for each phone at the word boundaries which don't already cover the maximum available context given the acoustic model set. Use a local Viterbi search to find M new acoustic scores and possibly $M-1$ new word boundaries. Generate M new arcs and $M-1$ new hyp-nodes and replace the old hypothesis end-hyp-node by the new one.

The *correct* cross-word HMM model is defined as the model which covers the most context around the current center-phone. This definition is also used for finding the correct context-dependent HMM within words during construction of the tree lexicon containing context-dependent models given only a monophone pronunciation dictionary.

Compared to the procedure described in (Bahl et al., 1993), which locally rescores every word that is found during the state-level search, the method described here rescores only words that have been found to be considerably likely being part of stacks to expand. The average number of hypotheses to expand per frame is in general between five and one-hundred and cross-word rescoring is only applied to those few. This requires only very little temporary memory and is fast, because of the low number of hypotheses and because of the fact that most of the states to be evaluated during rescoring for their observation likelihood are already in the cache.

A potential drawback of this method is that because cross-word effects are incorporated delayed, scores might vary more during the lookahead, which might require larger beams than if this delay wouldn't be used.

4.2.8 Fast-match with delay

The method to handle arbitrary cross-word effects from section 4.2.7 is easily extended to allow an efficient acoustic fast-match with a one-word delay, which in a similar form without delay is described in (Bahl et al., 1992; Gopalakrishnan and Bahl, 1996). The basic idea of a fast-match in a stack decoder is to use simple acoustic models to find possible extension words, and rescore them locally with better, but computationally more expensive models. This avoids the use of expensive models for the initial state-level search and can speed up the complete search substantially.

The fast-match procedure described here (see Figure 4.6) keeps the use of the expensive models at a minimum and is almost identical with the method to incorporate cross-word models. Instead of using word-within context-dependent (CD) models for the state-level search, simple monophones with a low number of Gaussians per mixture or small neural-network based models are used in a context-independent tree-lexicon, and the words found are inserted in the corresponding stacks. Rescoring of the last M words including all cross-word effects is done later using the accurate, but expensive CD models, but only when a stack is expanded, such that many of the previously found words will be out of the beam. The difference to the cross-word procedure from section 4.2.7 is, that *all* phones of the last M words have to be mapped to their correct CD HMM model, and not only the ones at the word boundaries. As described above, this can be interpreted as local rescoring with a one-word delay, which limits the number of necessary rescoring turns per frame to less than ten to one-hundred for most applications, and requires very little additional memory.

4.2.9 Using word-graphs as language model constraints

For some applications it is necessary to constrain the search by a finite state grammar, a word-graph or a word-pair grammar, possibly with transition scores. This can be done efficiently in a stack decoder by activating only the pronunciation paths in the tree lexicon that correspond to possible word extensions of the hypotheses to expand. This has to be done on demand before entering the state-level search every time a stack is expanded. The state-level search will only consider the limited number of activated paths which will speed up the search substantially (section 4.3).

The generation of forced word alignments can be interpreted as a search constrained by an extremely simple word-graph consisting of the transcription of word-IDs, which might have several pronunciation variants.

4.2.10 Lattice rescoring

There are two types of often needed lattice rescoring procedures:

I) Use a given word-graph plus the word alignments and the acoustic scores, and change only the LM (often a higher order N-gram) or/and change LM parameters (LM scale and word deletion penalty).

II) Use only a given word-graph ignoring alignments and acoustic scores to constrain the search – use new acoustic models and a new LM. This has been described in section 4.2.9.

Type I is often done using $A^\star$ procedures in a separate search module, because the existence of the complete word-graph with scores allows an efficient estimate of score of the remainder, which is necessary for any $A^\star$ procedure (Nilsson, 1971; Soong and Huang, 1991). In this case there is usually one stack, which is ordered by the $A^\star$ score. For the stack decoder implementation with many stacks like it is described here, a simple replacement of the state-level search makes it possible to integrate lattice rescoring of type I within the stack decoder framework. Instead of the original state-level search through the tree lexicon the possible extension words and their scores for every hypothesis to extend are already calculated in the lattice, so they just have to be located and inserted into the corresponding stacks. Because all other modules remain the same, implementation is simple and all outputs that have been possible before for sequences of feature vectors as inputs (first-best, N-best, first-best lattice, N-best lattice), are then possible for lattices as inputs. Because the time-consuming state-level search doesn't have to be done, this type of lattice rescoring is fast also for large lattices of a few thousand arcs, usually taking between 1/100 and 1/10 realtime. Memory requirements are the same as for the regular search minus the memory that is needed for the state-level search.

4.2.11 Generating phone-/state-alignments

Because state-level backpointers are not stored, phone- or state-alignments have to be created on demand. After a first-best word hypothesis is created, every word is state-aligned using the same routines which are necessary for the cross-word rescoring from section 4.2.7. For a forced state-alignment the word transcription has to be provided as additional input as a word-graph (section 4.2.9).

4.3 EXPERIMENTS

All experiments were conducted using the described one-pass stack decoder for the recognition of read sentences from a Japanese newspaper using a 5000 word pronunciation dictionary with on average 1.5 pronunciations per word. Larger pronunciation dictionaries for Japanese are currently not publically available. Dictionary and acoustic models are based on a set of 43 phones (Table 4.1). The acoustic models are

Table 4.1: Used phone set for Japanese recognition

Phones	Comments
silB, silE	beginning, end silence
sp	short pause (all pauses within utt.)
q	all glottal stops
a, i, u, e, o	short vowels
a:, i:, u:, e:, o:	long vowels
N, w, y, j, z, m, n	voiced consonants
p, t, k, ts, ch, b, d, dy, g, s, sh, h, f, r	unvoiced consonants
my, ky, by, gy, ny, hy, ry, py	combination sounds

The basic phone set on which the dictionary and acoustic models are based on.

gender-dependent decision tree state-clustered Gaussian mixture models trained on 20k sentences and about a 100 speakers per gender from the ASJ and JNAS database of approximately 60 hours of speech. Acoustic preprocessing is standard 12-dimensional MFCCs plus log energy, with applied cepstrum mean subtraction per sentence and first derivatives every 10 ms. A trigram and fourgram language model were trained on around 45 million words from the RWC corpus containing four years of newspaper articles from the Mainichi Shinbun, a regular daily newspaper in Japan. The standard test data are the first ten sentences from the speakers 006, 014, 017, 021, 026, 089, 102, 115, 122 from the JNAS database. The basic phone set, all acoustic models, the initial language models and the initial pronunciation dictionary have kindly been provided by the IPA group (Kawahara et al., 1998), which also defined the test set.

4.3.1 Recognition of Japanese

Speech recognition of Japanese adds a few problems not occurring in Western languages. Japanese has no spaces between words, so the definition of a word for the dictionary is in general not obvious, but the databases used here are already subdivided into words, and word error rate is calculated using these word definitions. Unfortunately the subdivision in words is often ambiguous, which leads to recognition errors (example in English: 'awhile' recognized as 'a' 'while' and vice versa), that shouldn't be counted as errors in Japanese since there are no spaces. Because words are defined by grammatical analysis, there are often no pauses between them, which makes it essential to use cross-word models for Japanese, if this currently common word definition is used (section 4.3.5). These errors are here referred to as *type I* errors.

A second problem is that there are three different alphabets plus the western letters in use, which makes it possible to write the same word with exactly the same meanings and pronunciations using different symbols, a phenomenon that occurs in English only for numbers and for a very limited number of exceptions. It is correct and common to mix alphabets in sentences and use different spellings of the same word (example: there are at least six common ways to spell the Japanese word for 'I', meaning myself, some of them with the exact same pronunciation). This makes the evaluation of Japanese using a word error rate, which is based on word-IDs, more difficult than in Western languages, because different word-IDs shouldn't be counted as errors if their meanings and pronunciations are exactly the same. These errors are here referred to as *type II* errors. For the experiments of this chapter some of the results were cleaned of type I and type II errors to show their relevance.

A third problem specific to decoding is, that because of the many short words and the many homonyms the number of found word-ends, which make stack operations and N-gram accesses necessary, is higher than for example in English. The many short words resulting in on average more word boundaries increase also the need for cross-word modeling.

4.3.2 Recognition results for high accuracy

Table 4.2 shows the results, for which the parameter settings in Table 4.3 were optimized to achieve a low word error rate. The acoustic models are monophones with 129 states and triphones with 2000 and 3000 states with 16 mixture components each. The experiments of this task were run in two modes, a *Katakana* mode, where all word-IDs and all transcriptions are written only in the Katakana alphabet, and in a *Kanji* mode, where all word-IDs and transcriptions are written in a mixture of the three alphabets like they occur in a regular newspaper. Best recognition results in

Table 4.2: Recognition results for high accuracy

states x *mixtures*	cross-word	MALE	FEMALE
	models	Kat/Kan	Kat/Kan
129 x 16 (cleaned)	no	88.7/87.5	91.8/90.8
2000 x 16 (cleaned)	yes	95.2/93.3	**96.9/95.2**
3000 x 16 (cleaned)	yes	**96.4/94.8**	95.9/94.5
129 x 16 (not cleaned)	no	87.9/86.7	91.0/90.0
2000 x 16 (not cleaned)	yes	94.4/92.6	**96.1/94.4**
3000 x 16 (not cleaned)	yes	**95.6/94.0**	95.0/93.6

The upper part shows results which were cleaned of errors that shouldn't be counted in Japanese, the lower results weren't cleaned. All results are given for the Katakana (Kat) and the Kanji (Kan) recognition mode as discussed in the text.

Kanji recognition mode are 5.2% word error rate (WER) for the male speakers using

3000-state models and 4.8% WER for the female speakers using 2000-state models, if the results are cleaned from errors that shouldn't be counted as errors in Japanese as discussed above. The raw outputs from the recognizer are about 15% relative (1% absolute) worse, showing that these errors, which are specific to Japanese, shouldn't be neglected. The Katakana results, which hide misrecognition of homonyms occurring in Japanese more frequently than for example in English, overestimate the score of interest by about 1% absolute on average.

The parameter settings in Table 4.3 show that the word-end-beam can be chosen lower than the word-within-beam as discussed in section 4.2.1.b. The average number of competing HMM model nodes at any time determines to a large extent the overall speed of the search and is a suitable measure to compare different implementations of stack decoders. Note that because active nodes cannot be merged, this number generally will be lower in transition network decoders. The stack statistics show that on average about 75% of the time the language model state of the hypothesis to be inserted is already on the stack, and only 5-10% of the hypotheses remain in the beam to get actually expanded. This implies that at least the number of N-gram accesses could be reduced by a completely time-synchronous scheme, where word- and state-level search both run time-synchronously, which hasn't been tried here.

The average number of N-gram accesses including all back-offs compared to the number of cache accesses within the N-gram module show that many N-grams are used more than once and a cache will be very useful in cases where the N-gram access is slow like for a disk-based LM.

Table 4.3: Parameter settings and search statistics for results from Table 4.2

	129 x 16	2000 x 16	3000 x 16
word-end-beam	30	50	50
word-within-beam	40	80	80
LM-scale	6	11	12
word-deletion-penalty	0	0	0
realtime factor (RTF)	5.5	24	25
active model nodes/frame	1756	10045	8324
pushed hyps/frame	544	1196	1113
inserted/replaced hyps/frame	92/452	246/950	211/902
extended hyps/frame (average stacksize)	36	25	20
on-demand N-gram smearing	no	yes	yes
triangular beam	yes	no	no
N-gram accesses/frame	20070	21029	18749
cache accesses/frame	19837	20834	18600

These results are based on 25-dimensional feature vectors, all log-likelihoods base 10, the realtime factor is for 300 Mhz Pentium II and includes observation likelihood calculation. All results in this table are averaged over genders.

4.3.3 Recognition results for high speed and low memory

Table 4.4 and Table 4.5 show results and parameter settings for experiments that were run to maximize decoding speed at a low (about 1%) search error and to minimize memory requirements, with (a) a regular memory-based trigram LM and (b) a disk-based LM. Almost realtime performance including all observation likelihood calculations is possible with around 90% recognition rate using between 10 and 20 MB of memory. The disk-based LM slows down the search by about a factor of three for the monophone models, for the triphone models only by about a factor of 1.5, because the N-gram access accounts only for a small part of the search time in this case.

Table 4.4: Results for high speed and low memory

states x *mixt.*	disk-LM	cross-word models	MALE Kat/Kan	FEMALE Kat/Kan	MEMORY	RTF
129 x 16	no	no	87.0/86.0	90.2/89.2	10 MB	1.1
129 x 16	yes	no	87.0/86.0	90.2/89.2	4 MB	3.0
2000 x 16	no	yes	93.3/91.5	95.0/93.8	20 MB	9
2000 x 16	yes	yes	93.3/91.5	95.0/93.8	14 MB	14

Results with parameter settings optimized for high speed and low memory, not cleaned of type I/II errors. Memory and realtime factor are for a 300 MHz Pentium II.

Table 4.5: Parameter settings and search statistics for results from Table 4.4

	129 x 16	2000 x 16
word-end-beam	20	40
word-within-beam	30	70
LM-scale	6	11
word-deletion-penalty	0	0
maximum model node pruning	no	150
mixture pruning	no	80
triangular beam pruning	yes	no
active model nodes/frame	685	2993
pushed hyps/frame	149	408
inserted/replaced) hyps/frame	44/105	97/311
extended hyps/frame (average stacksize)	7.9	12.3
on-demand N-gram smearing	no	yes
N-gram accesses/frame	2927	8196
cache accesses/frame	2882	8114

The trigram LM has 5k unigrams, 330k bigrams and 720k trigrams, occupying in total about 6 MB of memory using the techniques of section 4.2.5. An N-gram entry occupies on average 6 bytes, if the complete LM is held in memory, and about 100 kB total for the disk-based LM with bigrams and trigrams on disk which are loaded on demand and cached in a hash table of limited size.

4.3.4 Time and memory requirements for modules

The relative time and memory requirements of the different modules are summarized in Table 4.6. Most of the time is spent on the likelihood calculation and the state-level search, which includes all operations for the active node list. The time for the tree lexicon includes activating and deactivating HMM nodes. The cross-word rescoring procedure includes the on-demand lookup for the correct cross-word HMM model and the local Viterbi search as its most time-consuming parts. The LM state comparison is included in the time listed for the stack operations, which is surprisingly low given the simple linear list implementation shown in section 4.2.3.

Memory requirements are listed for a 5000 word vocabulary with on average 1.5 pronunciations each, giving about 200 bytes/entry. The acoustic model takes most of the memory because of its uncompressed 4-byte mean/variance parameters and the cache for the likelihood calculation. The hypotheses generation itself takes almost no memory but what is needed to represent the currently active hyp-nodes and arcs, which are in the case of first-best recognition not more than a few hundred. Similar, the stack module contains mainly pointers to hyp-nodes, which also don't use more than a few kB.

Table 4.6: Relative time and memory requirements for modules

MODULE	RELATIVE TIME	MEMORY
stack	2%	≈ 0
hyp	1%	≈ 0
state-level search	33%	0.5 MB
word-level search	3%	≈ 0
tree lexicon	5%	1.4 MB
N-gram	12%	5.1 MB
acoustic model	31%	13.0 MB
cross-word rescoring	11%	–
SUM	100%	20 MB

Relative time and memory requirements split up for modules using the 2000 x 16 acoustic model from Table 4.4.

4.3.5 Usage of cross-word models

Given the word definition for Japanese, which was used here, the use of cross-word models is essential for the recognition of read newspaper sentences, as Table 4.7 shows. The additional search time for the local rescoring using cross-word models doesn't effect the overall search time at all for this experiment, possibly because of more accurate partial hypotheses at any time during the search.

Table 4.7: Effect of cross-word effects

CROSS-WORD MODELS	RTF	REC-RATE
yes	24	93.5
no	24	87.0

Recognition of Japanese newspaper articles with and without cross-word 2000x16 models using the same beam settings, but optimized LM-scales and word-deletion penalties. Results are averaged over genders in Kanji recognition mode with search parameters of Table 4.3, not cleaned.

4.3.6 Usage of fast-match models

Table 4.8 shows the effect of using fast-match models to find acoustically likely word hypotheses quickly as described in section 4.2.8. In the case tested here their use required fine tuning of several search parameters to make a difference in recognition time.

Table 4.8: Effect of fast-match models

FAST-MATCH MODELS	RTF	REC-RATE
yes	7	92.5
no	9	92.6

Use of fast-match models to find acoustically likely word hypotheses quickly, averaged over genders in Kanji recognition mode with search parameters of Table 4.5, not cleaned. Fast-match models were 3-state monophones with four mixture components each.

4.3.7 Effect of on-demand N-gram smearing

On-demand N-gram smearing (section 4.2.6.b) can efficiently reduce the number of active model nodes, as Table 4.9 shows. In the cases tested here the reduction of active nodes does not necessarily reduce the search time because of the overhead of the procedure that has to be invoked before each stack is expanded. If the likelihood calculation of the acoustic models would take longer, this method would have a greater effect on the total recognition time.

Table 4.9: Effect of on-demand N-gram smearing

states x *mixt.*	LM lookahead	active models	N-gram accesses	RTF	REC. RATE
129 x 16	unigram	685	2927	1.3	87.6
129 x 16	N-gram	593	2252	1.9	87.5
2000 x 16	unigram	2993	8196	10	92.6
2000 x 16	N-gram	2817	5486	9	92.6

Shows the effect of on-demand N-gram smearing versus unigram smearing. Results are averaged over genders in Kanji recognition mode with search parameters of Table 4.5, not cleaned.

4.3.8 Lattice/N-best list generation and lattice rescoring

The results shown in Table 4.10 compare the time and memory requirements for generating the first-best hypothesis with the time for generating lattices or N-best lists in the first pass. It can be seen that the more complicated LM state check for the N-best lists creates only little overhead, and is almost independent of the length of the N-best lists.

Lattice rescoring as discussed in section 4.2.10 was tested for the generated lattices for both lattice resoring modes. Type I lattice rescoring refers to using only the word-graph as an LM constraint, but all alignments, acoustic scores including cross-word effects and LM scores are recalculated. For type II lattice rescoring only the LM scores are recalculated, which usually includes a new LM scale factor and a new word deletion penalty.

Table 4.10: Relative time and memory for different search modes

SEARCH MODE	RTF	MEMORY
first-best (absolut)	9	20 MB
first-best	100%	100%
lattice	107%	106%
N-best list, N = 10	113%	100.4%
N-best list, N = 50	116%	100.4%
N-best list, N = 100	117%	100.5%
lattice rescoring type I	0.1%	77%
lattice rescoring type II	5.6%	93%

Relative time and memory (as measured by the UNIX `top` command) for several search modes with beams leading to lattices of about 2500 arcs and 500 hyp-nodes, and an average N-best list length of 90 hypotheses, for parameter settings as in Table 4.5. All N-best hypotheses differ by at least one word as defined in section 4.2.3.b.

4.4 CONCLUSIONS

This chapter presented a detailed description of a memory-efficient one-pass stack decoder applied to recognition of read sentences from a Japanese newspaper. The architecture of the time-asynchronous stack decoder made it easily possible to integrate lattice and N-best list construction as well as arbitrary order N-gram LMs and arbitrary order cross-word context-dependent acoustic models in a single decoder. Also, various forms of lattice rescoring and the generation of forced alignments fits well into the framework of the time-asynchronous search technique. Memory requirements at around 1% search error are between 4 and 20 MB using the presented techniques.

In summary, it can be concluded, that a time-asynchronous stack decoder is a conceptually attractive framework for integrating many often needed procedures for speech recognition tasks. Although very efficient in memory requirements and faster than the decoder mentioned in (Kawahara et al., 1998) for the same task, it should be noted that the speed of a time-asynchronous stack decoder like implemented here is probably not optimal for the specific task of generating a first-best hypothesis or a lattice from a feature vector sequence, because the globally time-asynchronous search over the state space results in the generation of many partial hypotheses that are later not expanded. This could be avoided by using a time-synchronous stack decoder with multiple trees, which has not been investigated in this thesis.

4.5 ACKNOWLEDGMENTS

The author would like to thank the Japanese IPA group (Kawahara et al., 1998) for their supply of acoustic models, initial language models and the initial pronunciation dictionary, and especially Prof. Shikano from the Nara Institute of Science and Technology, Japan, for pointing out the importance of cross-word models for the recognition Japanese. Without the many fruitful discussions concerning stack decoders with Christoph Neukirchen and Daniel Willett from the Gerhard-Mercator University, Duisburg, Germany, the implementation of the on-demand N-gram smearing procedure wouldn't have been possible.

Chapter 5

Conclusions

5.1 SUMMARY

Many problems of engineering interest, like for example speech recognition or online hand-writing recognition, can be formulated in an abstract sense as *supervised learning from sequential data*, where an input sequence $\mathbf{X} = \mathbf{x}_1^T = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{T-1}, \mathbf{x}_T\}$ has to be mapped to an output sequence $\mathbf{Y} = \mathbf{y}_1^T = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \ldots, \mathbf{y}_{T-1}, \mathbf{y}_T\}$, that in general embodies correlations between neighboring vectors $\mathbf{x}_t, \mathbf{x}_{t+1}$ and $\mathbf{y}_t, \mathbf{y}_{t+1}$. This thesis tried to give a unified view of the abstract problem of *supervised learning from sequential data* and presented some models and algorithms for improved sequence recognition and modeling performance, measured on synthetic data and on real speech data. In particular, it was possible to remove some assumptions about the data which are necessary using traditional models.

Chapter 2: First the concept of maximizing the *posterior probability* of the output sequence given the input sequence, $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$, to achieve an optimal *sequence recognition rate*, was discussed. Two approaches to decompose the relevant parts of $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$ for recognition of the best sequence $\mathbf{Y}^\star$ into smaller independent expressions were shown, a frequently used approach into a generative and a prior model part,

$$\mathbf{Y}^\star = \arg\max_{\mathcal{Y}} \underbrace{\left[\prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{t-1}, \mathbf{y}_1^T)\right]}_{\text{generative part}} \underbrace{\left[\prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{t-1})\right]}_{\text{prior part}} \quad (5.1)$$

and an approach which decomposes $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$ directly,

$$\mathbf{Y}^\star = \arg\max_{\mathcal{Y}} \prod_{t=1}^{T} P(\mathbf{y}_t|\mathbf{y}_{t+1}, \mathbf{y}_{t+2}, \ldots, \mathbf{y}_T, \mathbf{x}_1^T). \quad (5.2)$$

Necessary approximations to deal with context-dependency on the input and output side in practical applications were discussed, and (Hidden) Markov Models, an important group of models specifically designed to be used for sequence modeling and prediction, were reviewed.

Chapter 3: In chapter 3 various ways of supervised learning from sequences using artificial neural networks were discussed. First the necessary basics of neural networks, commonly used architectures and their problems with respect to sequence processing were reviewed. A powerful neural network structure to deal with sequential data is the recurrent neural network (RNN), which allows to estimate $P(\mathbf{y}_t|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t)$, the output probability distribution at time t given all previous input. This chapter presented various extensions to the basic RNN structure, which are

a) a *bidirectional recurrent neural network* (BRNN), which allows to estimate expressions of the form $P(\mathbf{y}_t|\mathbf{x}_1^T)$, the output at t given *all* sequential input, for uni-modal regression and classification problems,

b) an extended BRNN to directly estimate the posterior probability of a symbol sequence, $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$, by modeling $P(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_1, \mathbf{x}_1^T)$ as found in (5.2) without explicit assumptions about the shape of the distribution $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$,

c) a BRNN to model multi-modal input data that can be described by Gaussian mixture distributions conditioned on an output vector sequence, $P(\mathbf{x}_t|\mathbf{y}_1^T)$, assuming that neighboring $\mathbf{x}_t, \mathbf{x}_{t+1}$ are conditionally independent, and

d) an extension to c) which removes the independence assumption by modeling $P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_1, \mathbf{y}_1^T)$ as found in (5.1) to estimate the likelihood $P(\mathbf{x}_1^T|\mathbf{y}_1^T)$ of a given output sequence without any explicit approximations about the use of context.

Chapter 4: Chapter 4 described the details of a fast, memory-efficient one-pass stack decoder for efficient evaluation of the search space for large vocabulary continuous speech recognition, a challenging sequence modeling problem. After defining the requirements for a modern decoder, it describes the details of an implementation that is based on a stack decoder framework. It is shown how it is possible to handle arbitrary order N-gram language models, how to generate N-best lists or lattices next to the first-best hypothesis at little computational overhead, how to handle efficiently cross-word context-dependent acoustic models of any context order, how to efficiently constrain the search with word-graphs or word-pair grammars, and how to use a fast-match with delay to speed up the search, all in a single left-to-right search pass. The details of a disk-based representation of an N-gram language model are given, which make it possible to use LMs of arbitrary (file) size in only a few hundred kB of memory. On-demand N-gram smearing, an efficient improvement over the regular unigram smearing used as an approximation to the LM scores in a tree lexicon, is introduced. It is also shown how lattice rescoring, the generation of forced alignments and detailed phone-/state-alignments can be integrated efficiently into a single stack decoder. The use of this decoder led to the best reported recognition results of around 3.5–5% word error rate on the standard test set of a widely used Japanese newspaper dictation task. With computationally cheap acoustic models it was possible to achieve around 11% word error rate in nearly real-time on a 300 Mhz Pentium II. Using a disk-based LM the memory usage could be optimized to 4 MB in total.

5.2 CONTRIBUTIONS FROM THIS THESIS

This thesis has addressed the problem of sequence modeling and prediction with applications for speech recognition, and has presented to knowledge of the author several improvements to models and algorithms for sequence modeling for the first time. These are

1) a *bidirectional recurrent neural network* (BRNN) structure, which allows to estimate expressions of the form $P(\mathbf{y}_t|\mathbf{x}_1^T)$, the output at t given *all* sequential input, for uni-modal regression and classification problems,

2) an extended BRNN to directly estimate the posterior probability of a symbol sequence, $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$, by modeling $P(\mathbf{y}_t|\mathbf{y}_{t-1},\mathbf{y}_{t-2},\ldots,\mathbf{y}_1,\mathbf{x}_1^T)$ without explicit assumptions about the shape of the distribution $P(\mathbf{y}_1^T|\mathbf{x}_1^T)$,

3) a BRNN to model multi-modal input data that can be described by Gaussian mixture distributions conditioned on an output vector sequence, $P(\mathbf{x}_t|\mathbf{y}_1^T)$, assuming that neighboring $\mathbf{x}_t,\mathbf{x}_{t+1}$ are conditionally independent, which takes automatically care of *all* possible context effects on the output side,

4) an extension to c) which removes the independence assumption by modeling $P(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_{t-2},\ldots,\mathbf{x}_1,\mathbf{y}_1^T)$ to estimate the likelihood $P(\mathbf{x}_1^T|\mathbf{y}_1^T)$ of a given output sequence without any explicit approximations,

5) a simple extension to a first order gradient descent algorithm to optimize the objective function for neural network training, called ARPROP, which allowed to train the reasonably large networks ($\approx 10^6$ parameters) with reasonably large amounts of data ($\approx 10^7$ vectors) like used for this thesis in a few days on regular workstations without difficult parameter setting,

6) a description of a one-pass stack-decoder for speech recognition, which

   – produces first best, N-best, lattices or state-alignments in a single pass,

   – can handle arbitrary order N-gram language models,

   – can handle arbitrary order context-dependent models with full cross-word expansion, using a time- and memory efficient local rescoring procedure to include all cross-word effects,

– gave the currently best reported results ($\approx 3.5 - 5\%$ word error rate) on the standard test set of a widely used Japanese newspaper dictation task,
– can run in 4 MB total memory with a 5000 word vocabulary and a back-off trigram.

5.3 SUGGESTIONS FOR FUTURE WORK

The models and algorithms presented in this thesis can be extended in various ways and can be expected to be useful in real applications, which are briefly summarized here:

Mixture density BRNNs for speech recognition: In this thesis mixture density BRNNs, which allow to model speech data with less assumptions than previously necessary, were shown to give increased likelihood on test speech data, but they weren't used for actual recognition. Since machines still have a poor performance, compared to a human being, when recognizing *spontaneous speech*[1], which seems to be largely related to the quality of the acoustic models, mixture density BRNNs could be an interesting alternative to conventional unconditional Gaussian mixture models.

Mixture density BRNNs for speech synthesis: A currently popular approach to speech synthesis is a clever assembly of units of pre-recorded speech from a database. While within a unit the speech is in general almost as natural as the recording, the joints between the units usually inhibit discontinuities, which make the synthesized speech unnatural and sometimes incomprehensible.

Mixture density BRNNs could be used for one part of the speech synthesis problem by predicting a smooth feature vector sequence, conditioned on a stream of phonemes with preset durations, to avoid the unnatural discontinuities. When trained for a single speaker, a mixture density BRNN with a single Gaussian might be sufficient to get a reasonably accurate model, which in this case would be very easy to use since the predicted mean of the Gaussian at time t could be used as the predicted feature vector. The quality of the prediction could be controlled by changing the number of free parameters of the model during training.

Extension of BRNNs to higher-dimensional structures: This thesis dealt with *sequential data*, that is data whose order can be represented as a simple (one-dimensional) sequence. Many of the presented extensions to the basic recurrent neural network structure can be generalized to two- or even higher-dimensional structures, which would for example allow to estimate expressions of the form $P(\mathbf{y}_{ij}|\mathbf{x}_{1<=i<=N_i,1<=j<=N_j})$, an output depending on an *area* of $N_i \times N_j$ inputs.

[1]the best state-of-the-art system in the 1998 Switchboard/CallHome evaluations for English, which involves the transcription of telephone conversations over various topics, had 39.5% word error rate

Time-synchronous stack decoder: As briefly mentioned in chapter 4, a probably faster, but less memory-efficient way of finding the best hypothesis for a speech recognition task might be to use a completely time-synchronous stack decoder with multiple trees, which would avoid the need of a heuristic maximum word length and the heuristic triangular beam pruning procedure.

Finally, the ultimate academic goal of conditional sequence prediction must be to perform the *conditional modeling* of the sequence, in categorical as well as in continuous output spaces, as well as the *search* for the best sequence with a *single* model, whose parameters are all optimized during a single training procedure.

Unfortunately all currently used approaches are still based on two separate procedures, the production of hypothesized sequences and the evaluation of their scores to select between them, which limits the possibilities of improving the search procedure automatically during training of a single model using the simple but powerful principle of *learning from data.*

Appendix A

Improved initialization of Multi-Layer-Perceptrons

When training MLPs, an often used initialization procedure is to use small values around zero for all weights without using the training data. An improved procedure that has been used for various experiments in this thesis, which gave in general better results than the random initialization by converging to better local minima, is described here.

The general procedure is to use the properties of the input data to initialize the input layer and supervised methods for the output layer by approximating the original problem (uni-modal regression, multi-modal regression, classification) by a *general least square problem* formulated as the *normal equations*, which can be solved using the methods described in (Press et al., 1992).

1. Initialization of input layer (D inputs, J hidden neurons)
   (a) Set the D weights to the jth neuron randomly excluding the bias weights.
   (b) Forward-propagate all N input vectors through the input layer excluding the bias weights and calculate the mean μ_j of the hidden activations.
   (c) Set the jth bias weight to the negative mean of the jth activation, such that the average activation becomes zero.
   (d) Forward-propagate all input vectors through the input layer using the bias weights and calculate the variance σ_j of the hidden activations.
   (e) Divide all $D + 1$ weights of the jth hidden neuron by $\sqrt{\sigma_j}$ to set a unit variance for all activations.
2. Initialization of output layer (J hidden neurons, K outputs)
   (a) Forward-propagate all input vectors through the input layer using the bias weights and store the results in a $N \times (J + 1)$ *design matrix* $\mathbf{D}$, with the last row set to one for the unconditional bias.

(b) Set up the $N \times K$ *target matrix* $\mathbf{T}$ using the K-dimensional targets from the training data. The matrix will contain exactly the target values in the case of linear output activation functions. In all other cases the target matrix entries have to be calculated by *approximately inverting* the output activation function as shown in Table A.1. Note that the argument of the $ln()$ function must not be zero or very close to zero, which was here taken care of by using a *safe* logarithm function

$$\begin{array}{llll} \text{IF} & x > floor & \text{THEN} & ln_{safe}(x, floor) = ln(x) \\ \text{ELSE} & & & ln_{safe}(x, floor) = ln(floor) \end{array}$$

with $10^{-3} < floor < 10^{-6}$.

Table A.1: Inversion of output activation functions.

Output activation	Inversion function
linear	y
sigmoid	$ln_{safe}(y) - ln_{safe}(y-1)$
tanh	$(ln_{safe}(y+1) - ln_{safe}(y-1))/2$
exp	$ln_{safe}(y)$
softmax	$ln_{safe}(y)$

Combinations of output activation functions and their inversion functions for improved neural network initialization.

(c) Solve the normal equations

$$(\mathbf{D}^T\mathbf{D})\mathbf{W}^T = \mathbf{D}^T\mathbf{T}$$

to get the *weight matrix* $\mathbf{W}$ with the initial weights for the output layer. The recommended way to avoid round-off errors and to deal with nearly singular matrices is singular value decomposition (SVD), but the normal equations can also be solved with faster but more dangerous methods (LU decomposition or Cholesky decomposition) (Press et al., 1992) with respect to their round-off erros.

Appendix B

List of Publications

Journal & Book Chapter

(1) Mike Schuster, "Memory-efficient LVCSR search using a one-pass stack decoder", submitted to *Computer, Speech and Language* in August 1998

(2) Mike Schuster, "Neural Networks for Speech Processing", *Encyclopedia of Electrical and Electronic Engineering*, John Wiley & Sons, invited overview article, reviewed, to appear Feb. 1999.

(3) Mike Schuster, Kuldip K. Paliwal, "Bidirectional recurrent neural networks", *IEEE Transactions on Signal Processing*, November, Vol. 45, pages 2673–2681, 1997.

Journal (co-author)

(4) Toshiaki Fukada, Mike Schuster & Yoshinori Sagisaka, "Phoneme boundary estimation using bidirectional recurrent neural networks and its application", (深田 俊明, Mike Schuster, 匂坂 芳典 : "双方向リカレントニューラルネットワークを用いた音素境界推定とその応用"), *IEICE Transactions*, J81-D-II, 7, pages 1481–1490, July 1998, in Japanese.

International Conference & Workshop

(5) Mike Schuster, "Nozomi - a fast, memory-efficient stack decoder for LVCSR", *Proceedings of the International Conference on Speech and Language Processing*, Sydney, Australia, pages 1835–1838, 1998.

(6) Mike Schuster, "Incorporation of HMM output constraints in hybrid NN/HMM systems during training", *Proceedings of the European Conference on Speech Communication and Technology*, Rhodos, Greece, pages 2843–2847, 1997.

(7) Mike Schuster, "Acoustic model building based on non-uniform segments and bi-directional recurrent neural networks", *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Munich, Germany, pages 3249–3252, 1997.

(8) Mike Schuster, "Learning out of time series with an extended recurrent neural network", *Proceedings of the IEEE Neural Network Workshop for Signal Processing*, Kyoto, Japan, pages 170–179, 1996.

(9) Mike Schuster, Gerhard Rigoll, "Fast on-line video image sequence recognition with statistical methods", *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Atlanta, GA, Vol. VI, pages 3450–3453, 1996.

International Conference & Workshop (co-author)

(10) J. Picone, S. Pike, R. Reagan, T. Kamm, J. Bridle, L. Deng, Z. Ma, H. Richards & M. Schuster, "Initial evaluation of hidden dynamic models on conversational speech ", *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Phoenix, TX, to appear, 1999.

(11) Zhengyou Zhang, Michael Lyons, Michael Schuster & Shigeru Akamatsu, "Comparison between geometry-based and gabor-wavelets-based facial expression recognition using Multi-Layer-Perceptron", *Proceedings of the 3rd IEEE International Conference on Automatic Face and Gesture Recognition*, Nara, Japan, pages 454–459, 1998.

(12) T. Fukada, S. Aveline and M. Schuster, "Segment boundary estimation using recurrent neural networks", *Proceedings of the European Conference on Speech Communication and Technology*, Rhodos, Greece, pages 2839–2842, 1997.

(13) Gerhard Rigoll, Andreas Kosmala & Mike Schuster, "A new approach to video image sequence recognition based on statistical methods", *Proceedings of the IEEE International Conference on Image Processing*, Lausanne, Switzerland, 1996.

(14) Gerhard Rigoll, Andreas Kosmala & Mike Schuster, "High performance gesture recognition using probabilistic neural networks and hidden Markov Models", *Proceedings of the 5th IEEE International Workshop on time varying image processing and moving image recognition*, Florence, Italy, 1996.

Conference & Workshop

(15) Mike Schuster, "Evaluation of a stack decoder on a Japanese Newspaper Dictation Task", *IEICE Meeting*, Tokyo, Japan, pages 33–40, December 1998.

(16) Mike Schuster, “DARPA Switchboard Workshop 1998 – Impressions and Results”, *IEICE Meeting*, Tokyo, Japan, invited paper, pages 49–54, December 1998.

(17) Mike Schuster, “Evaluation of a stack decoder on a Japanese Newspaper Dictation Task”, *Proceedings of the Acoustical Society of Japan Meeting Fall 98*, Yamagata, Japan, pages 141–142, 1998.

(18) Mike Schuster, “「のぞみ」 – A fast, memory-efficient one-pass stack decoder”, *Proceedings of the Acoustical Society of Japan Meeting Spring 98*, Yokohama, Japan, pages 155–156, 1998.

(19) Mike Schuster, “Acoustic model building based on non-uniform segments and bidirectional recurrent neural networks”, *Proceedings of the Acoustical Society of Japan Meeting Spring 97*, Kyoto, Japan, pages 101–102, 1997.

(20) Mike Schuster, “Bidirectional recurrent neural networks for speech recognition”, *Proceedings of the Acoustical Society of Japan Meeting Fall 96*, Okayama, Japan, pages 77–78, 1996.

(21) Mike Schuster, “Bidirectional recurrent neural networks for speech recognition”, *IEICE Meeting*, Tokyo, Japan, pages 7–12, October 1996.

(22) Mike Schuster, “Fast k-means vector quantizer for very large amounts of data”, *Proceedings of the Acoustical Society of Japan Meeting Spring 96*, Tokyo, Japan, pages 107–108, 1996.

Conference & Workshop (co-author)

(23) 深田 俊明, Sophie Aveline, Mike Schuster & 匂坂 芳典, “リカレントニューラルネットワークを用いた音素境界推定と音声認識への応用”, 信学技報, SP97-15, pages 41–48, June 1997, in Japanese.

(24) 深田 俊明, Sophie Aveline, Mike Schuster & 匂坂 芳典, “リカレントニューラルネットワークを用いたセグメント境界推定”, 音響学会講論集, 3-6-8, pages 103–104, March 1997, in Japanese.

Technical Report

(25) Mike Schuster, “Memory-efficient LVCSR search using a one-pass stack decoder”, *Technical Report TR-IT-0272*, ATR Interpreting Telecommunications Laboratories, Kyoto, Japan, 1998.

(26) Mike Schuster, “Bidirectional recurrent neural networks”, *Technical Report TR-IT-0273*, ATR Interpreting Telecommunications Laboratories, Kyoto, Japan, 1998.

Technical Report (co-author)

(27) Petra Phillips and Mike Schuster, “Adaptation of BRNN Speech Recognition Systems”, *Technical Report TR-IT-0261*, ATR Interpreting Telecommunications Laboratories, Kyoto, Japan, 1998.

(28) Harald Singer, Masahiro Tonomura, Qiang Huo, Jun Ishii, Toshiaki Fukada & Michael Schuster, “Baseline Acoustic Models for the Spoken Language Database (SDB/SLDB)”, *Technical Report TR-IT-0206*, ATR Interpreting Telecommunications Laboratories, Kyoto, Japan, 1997.

Bibliography

Alleva, F. (1997). Search organization in the Whisper continuous speech recognition system. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 295–302, Santa Barbara, CA.

Alleva, F., Huang, X., and Hwang, M. (1996). Improvements on the pronunciation prefix tree search organization. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 133–136, Atlanta, GA.

Aubert, X., Dugast, C., Ney, H., and Steinbiss, V. (1994). Large vocabulary continuous speech recognition of Wall Street Journal data. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume II, pages 129–132, Adelaine, Australia.

Bahl, L. R., de Souza, P. V., Gopalakrishnan, P. S., Nahamoo, D., and Picheny, M. (1991). Decision trees for phonological rules in continuous speech using decision trees. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 185–188, Toronto, Canada.

Bahl, L. R., de Souza, P. V., Gopalakrishnan, P. S., Nahamoo, D., and Picheny, M. (1992). A fast match for continuous speech recognition using allophonic models. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages I–17–I–20, San Francisco, CA.

Bahl, L. R., de Souza, P. V., Gopalakrishnan, P. S., Nahamoo, D., and Picheny, M. (1993). Word lookahead scheme for cross-word right context models in a stack decoder. In *Proceedings of the European Conference on Speech Communication and Technology*, volume II, pages 851–854, Berlin, Germany.

Battiti, R. (1992). First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method. *Neural Computation*, 4:141–166.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estaimation of probabilistic functions of Markov processes. *Inequalities*, 3:1–8.

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41:164–171.

Bell, T. C. (1990). *Text Compression.* Prentice-Hall, Englewood Cliffs, NJ.

Bengio, Y. and Frasconi, P. (1996). Input-output HMMs for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, Berlin, Germany.

Beyerlein, P. and Ullrich, M. (1995). Hamming distance approximation for a fast log-likelihood computation for mixture densities. In *Proceedings of the European Conference on Speech Communication and Technology*, volume II, pages 1083–1086, Madrid, Spain.

Bishop, C. M. (1994). Mixture density networks. Technical Report NCRG/94/004, Neural Computing Research Group, Aston University, Birmingham, England.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, England.

Bourlard, H. and Morgan, M. (1994). *Connectionist Speech Recognition – A Hybrid Approach.* Kluwer Academic Press, Boston, MA.

Bourlard, H. and Wellekens, C. (1990). Links between Markov models and multilayer perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12):1167–1178.

Bridle, J. S. (1989). Probabilistic interpretation of feed-forward classification network outputs, with relationships to statistical pattern recognition. In Fougelman-Soulie, F. and Herault, J., editors, *Neurocomputing: Algorithms, Architectures and Applications*, volume F68 of *NATO ASI Series*, pages 227–236. Springer-Verlag, Berlin, Germany.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms.* MIT Press, Cambridge, MA.

Duda, R. O. and Hart, P. E. (1974). *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, NY.

Fahlmann, E. F. (1988). An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, PA.

Franco, H., Cohen, M., Morgan, N., Rumelhart, D., and Abrash, V. (1994). Context-dependent connectionist probability estimation in a hybrid Hidden Markov Model - Speech Recognition. *Computer, Speech and Language*, 8:211–222.

Fritsch, J. (1997). Context dependent hybrid HME/HMM speech recognition using polyphone clustering decision trees. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume III, pages 1759–1762, Munich, Germany.

Fritsch, J. (1998a). ACID/HNN: Clustering hierarchies of neural networks for context-dependent connectionist acoustic modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 505–508, Seattle, WA.

Fritsch, J. (1998b). Applying Divide and Conquer to Large Scale Pattern Recognition Tasks. In Orr, G. B. and Müller, K.-R., editors, *Neural networks: Tricks of the trade*, volume 1524 of *Lecture Notes in Comuter Science*, pages 315–342. Springer-Verlag, Berlin, Germany.

Fukada, T., Paliwal, P., and Sagisaka, Y. (1998). Model parameter estimation for mixture density polynomial segment models. *Computer, Speech and Language*, 12(3):229–246.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition.* Academic Press, San Diege, CA, second edition.

Gauvain, J. L., Lamel, L. F., G., A., and M., A.-D. (1994). The LIMSI continuous speech ditation system: Evaluation on the ARPA Wall Street Journal task. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 557–560, Adelaide, Australia.

Giles, C. L., Kuhn, G. M., and Williams, R. J. (1994). Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks*, 5(2):153–156.

Gish, H. (1990). A probabilistic approach to the understanding and training of neural network classifiers. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume III, pages 1361–1364, Albuquerque, NM.

Gish, H. and Ng, K. (1996). Parametric trajectory models for speech recognition. In *Proceedings of the International Conference on Spoken Language Processing*, volume I, pages 466–469, Philadelphia, PA.

Gopalakrishnan, P. S. (1995). A tree search strategy for large vocabulary continuous speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 572–575, Detroit, MI.

Gopalakrishnan, P. S. and Bahl, L. R. (1996). Fast matching techniques. In Lee, C.-H., Soong, F. K., and Paliwal, K. K., editors, *Automatic Speech and Speaker Recognition, Advanced Topics*, pages 413–428. Kluwer Academic Publishers, Boston, MA.

Gray, R. M. (1984). Vector quantization. *IEEE ASSP Magazine*, pages 4–29.

Hamming, R. W. (1986). *Coding and Information Theory.* Prentice-Hall, Englewood Cliffs, NJ.

Hertz, J. A., Krogh, A., and Palmer, R. P. (1991). *Introduction to the theory of neural computation.* Addison-Wesley, Redwood City, CA.

Hetherington, I. L., Phillips, M. S., Glass, J. R., and Zue, V. W. (1993). A^* word network search for continuous speech recognition. In *Proceedings of the European Conference on Speech Communication and Technology*, volume III, pages 1533–1537, Berlin, Germany.

Huang, X. D., Ariki, Y., and Jack, M. A. (1990). *Hidden Markov Models for Speech Recognition.* Edinburgh University Press, Edinburgh, England.

Jacobs, R. A. (1995). Methods for combining experts' probability assessments. *Neural Computation*, 7(5):867–888.

Jelinek, F. (1997). *Statistical Methods for Speech Recognition.* MIT Press, Cambridge, MA.

Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical Mixtures of Experts and the EM algorithm. *Neural Computation*, 6(2):181–214.

Kawahara, T., Kobayashi, T., Takeda, K., Minematsu, N., Itou, K., Yamamoto, M., Utsuro, T., and Shikano, K. (1998). Sharable software repository for {Japanese} large vocabulary continuous speech recognition. In *Proceedings of the International Conference on Spoken Language Processing*, volume VII, pages 3257–3260, Sidney, Australia.

Kershaw, D. J., Hochberg, M. M., and Robinson, A. J. (1995). Context-dependent classes in a hybrid recurrent network-hmm speech recognition system. Technical Report CUED/F-INGENG/TR217, Cambridge University Engineering Department, Cambridge, England.

MacKay, D. J. C. (1999). Information theory, inference and learning algorithms. Draft 1.9.0, unpublished manuscript for a book (http://wol.ra.phy.cam.ac.uk/mackay).

McCullagh, P. and Nelder, J. A. (1989). *Generalized linear models.* Chapman & Hall, London, England, second edition.

McLachlan, G. J. and Krishnan, T. (1997). *The EM Algorithm and Extensions.* John Wiley & Sons, New York, NY.

Mildenberger, O. (1992). *Informationstheorie und Codierung.* Vieweg Verlag, Braunschweig, Germany. in German.

Murveit, H., Butzberger, J., Digalakis, V., and Weintraub, M. (1993). Large vocabulary dictation using SRI's Decipher^{TM} speech recognition system: Progressive search techniques. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume II, pages 319–322, Minneapolis, MN.

Neukirchen, C. and Willett, D. (1997). personal communication.

Ney, H. (1993). Search strategies for large-vocabulary-continuous-speech recognition. In Rubio Ayuso, A. and Lopez Soler, J., editors, *Speech Recognition and Coding – New Advances and Trends*, pages 210–225. NATO Advanced Studies Institute, Bubion, Spain, June-July 1993, Springer, Berlin.

Ney, H. and Aubert, X. (1996). Dynamic programming search: From digit strings to large vocabulary speech recognition. In Lee, C.-H., Soong, F. K., and Paliwal, K. K., editors, *Automatic Speech and Speaker Recognition, Advanced Topics*, pages 385–412. Kluwer Academic Publishers, Boston, MA.

Ney, H. and Ortmanns, S. (1997). Progress in dynamic programming search for LVCSR. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 287–294, Santa Barbara, CA.

Nilsson, N. J. (1971). *Problem Solving Methods of Artificial Intelligence.* McGraw Hill, New York, NY.

Odell, J. J. (1995). *The Use of Context in Large Vocabulary Speech Recognition.* PhD thesis, Cambridge University, Cambridge, England.

Ortmanns, S., H., N., and Aubert, X. (1997). A word graph algorithm for large vocabulary continuous speech recognition. *Computer, Speech and Language*, 11:43–72.

Ostendorf, M. (1996). From HMMs to segment models: Stochastic modeling for csr. In Lee, C.-H., Soong, F. K., and Paliwal, K. K., editors, *Automatic Speech and Speaker Recognition, Advanced Topics*, pages 185–210. Kluwer Academic Publishers, Boston, MA.

Paul, D. (1991). Algorithms for an optimal $A^{\star}$ search and linearizing the search in the stack decoder. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 693–696, Toronto, Canada.

Paul, D. (1992). An efficient $A^{\star}$ stack decoder algorithm for continuous speech recognition with a stochastic language model. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages I–25–I–28, San Francisco, CA.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C.* Cambridge University Press, Cambridge, England, second edition.

Rabiner, L. and Juang, B. H. (1993). *Fundamentals of Speech Recognition.* Prentice-Hall, Englewood Cliffs, NJ.

Ravishankar, M. K. (1996). *Efficient Algorithms for Speech Recognition.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA.

Renals, S. and Hochberg, M. (1995a). Decoder technology for connectionist large vocabulary speech recognition. Technical Report CUED/F-INGENG/TR186, Cambridge University Engineering Deparment, Cambridge, England.

Renals, S. and Hochberg, M. (1995b). Efficient search using posterior phone probability estimates. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 596–599, Detroit, MI.

Renals, S. and Hochberg, M. (1996). Efficient evaluation of the search space using the NOWAY decoder. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 149–153, Atlanta, GA.

Richard, M. D. and Lippman, R. P. (1991). Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural Computation*, 3(4):461–483.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, England.

Robinson, A. J. (1994). An application of recurrent neural nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5(2):298–305.

Robinson, T. (1991). Several improvements to a recurrent error propagation network phone recognition system. Technical Report CUED/F-INFENG/TR82, Cambridge University Engineering Deparment, Cambridge, England.

Robinson, T. and Christie, J. (1998). Time-first search for large vocabulary speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume II, pages 829–833, Seattle, WA.

Robinson, T., Hochberg, M., and Renals, S. (1994). Improved phone modeling with recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 37–40, Adelaine, Australia.

Robinson, T., Hochberg, M., and Renals, S. (1996). The use of recurrent neural networks in continuous speech recognition. In Lee, C.-H., Soong, F. K., and Paliwal, K. K., editors, *Automatic Speech and Speaker Recognition, Advanced Topics*, pages 233–258. Kluwer Academic Publishers, Boston, MA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error backpropagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume I, pages 318–362. MIT Press, Cambridge, MA.

Saito, K. and Nakano, R. (1997). Partial BFGS update and efficient step-length calculation for three-layer neural networks. *Neural Computation*, 9(1):123–141.

Schuster, M. and Paliwal, K. K. (1996). Learning out of time series with an extended recurrent neural networks. In *Proceedings of the IEEE Neural Network Workshop for Signal Processing*, pages 170–179, Kyoto, Japan.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Neural Networks*, 45(11):2673–2681.

Schwartz, R., Nguyen, L., and Makhoul, J. (1996). Multiple-pass search strategies. In Lee, C.-H., Soong, F. K., and Paliwal, K. K., editors, *Automatic Speech and Speaker Recognition, Advanced Topics*, pages 429–456. Kluwer Academic Publishers, Boston, MA.

Senior, A. and Robinson, A. J. (1996). Forward-backward retraining of recurrent neural networks. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, number 8, pages 743–749. MIT Press, Cambridge, MA.

Shepherd, A. (1997). *Second-Order Methods for Neural Networks.* Springer, London, England.

Shimizu, T., Yamamoto, H., Masataki, H., Matsunaga, S., and Sagisaka, Y. (1997). Reduction of number of word hypotheses for large vocabulary continuous speech recognition. *IEICE Transactions*, J79-D-II(12):2117–2124. in Japanese.

Sivia, D. S. (1996). *Data Analysis, A Bayesian Tutorial.* Clarendon Press, Oxford, England.

Soong, F. and Huang, E. (1991). A tree-trellis based fast search for finding the N-best sentence hypotheses in continuous speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, pages 705–708, Toronto, Canada.

Steinbiss, V., Tran, B., and Ney, H. (1994). Improvements in beam search. In *Proceedings of the International Conference on Spoken Language Processing*, pages S36–5.1–S36–5.4, Yokohama, Japan.

Tsoi, A. C. and Back, A. D. (1994). Locally recurrent globally feedforward networks: A critical review of architectures. *IEEE Transactions on Neural Networks*, 5(2):229–239.

Valtchev, V. (1995). *Discriminative Methods in HMM-based Speech Recognition.* PhD thesis, Cambridge University, Cambridge, England.

Waibel, A. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3):328–339.

Young, S. (1996). A review of large vocabulary speech recognition. *IEEE Signal Processing Magazine*, 15(5):45–57.

Young, S., Jansen, J., Odell, J., Ollason, D., and Woodland, P. (1997). The HTK Book (version 2.1). (distributed with the HTK toolkit).