Diplomarbeit im Fach Informatik
Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. H. Ney

Optimization of Hidden Markov Models and Neural Networks

1. Dezember 2011

vorgelegt von:
Patrick Doetsch
Matrikelnummer 266481

Gutachter:
Prof. Dr.-Ing. H. Ney
Prof. Dr. rer. nat. T. Seidl

Betreuer:
Dipl. Inform. C. Plahl
Dipl. Inform. P. Dreuw

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Textauszüge und Grafiken, die sinngemäß oder wörtlich aus veröffentlichten Schriften entnommen wurden, sind durch Referenzen gekennzeichnet.

Aachen, 1. Dezember 2011

Patrick Doetsch

Acknowledgements

I would like to thank my family and friends who supported me in this work. Special thanks go to my father who made it possible for me to study at my favorite university.

I would also like to thank Prof. Dr. Ing. Hermann Ney for the possibilities at the Chair of Computer Science 6 of the RWTH Aachen University and especially for the opportunity writing a diploma thesis about this very recent and interesting topic. Further, I would like to thank Prof. Dr. Thomas Seidl, who kindly accepted to co-supervise the work.

Finally, I would like to thank my supervisors Christian Plahl and Philippe Dreuw for helping me with practical aspects of different stages of this work, proof-reading and constructive discussions about the topic.

Abstract

In this diploma thesis, combination methods for hidden markov models (HMMs) and artificial neural networks (ANNs) are examined and evaluated on several offline handwriting recognition tasks. ANN based features have been applied to automatic speech recognition and are an important component of state-of-the-art systems. The main focus of this work is the integration of recurrent neural networks (RNNs), specifically the Long-Short-Term-Memory (LSTM) which recently lead to significant improvements in several handwriting recognition tasks.

A model combination of HMMs and ANNs following two different approaches is proposed, where a previously trained HMM is used to create a labeling of the input features in order to train an ANN. The choice of the input features as well as the initial labeling play an important role and the effect is analyzed in an experimental setup. Further, the model complexity of different ANN types is considered and evaluated on real world handwriting recognition tasks. Moreover, this work investigates the use of probabilistic labelings and extends the ANN based systems by an alternating optimization procedure referred to as *realignment*. This procedure replaces the Connectionist Temporal Classification (CTC) output layer, which enables an RNN to perform sequence learning without HMMs.

The results show large improvements on two different feature sets for each of three different databases with Arabic, English and French handwriting. The combination methods with ANN based features show a superior performance compared to conventional HMMs and outperform the best published results on the English and Arabic database.

Kurzfassung

In dieser Diplomarbeit werden Methoden zur Kombination von Hidden Markov Modellen (HMMs) und künstlichen neuronalen Netzen (NNs) untersucht und auf die Handschriftenerkennung angewendet. Merkmalsvektoren basierend auf NNs werden in der automatischen Spracherkennung angewendet und sind ein wichtiger Bestandteil aktueller Erkennungssysteme geworden. Der Hauptfokus dieser Arbeit liegt auf der Integration von rekurrenten neuronalen Netzen (RNNs), insbesondere den Long-Short-Term-Memory (LSTM) Netzen, welche zu signifikanten Verbesserung in der Handschriftenerkennungs geführt haben.

HMMs und NNs werden auf zwei unterschiedliche Arten kombiniert. Um das NNs zu trainieren zunächst eine Klassenzugehörigkeit mittels eines HMMs erzeugt. Die erstellte Klassenzugehörigkeiten spielen eine entscheidende Rolle für das Training der NNs. Die Klassifikationsrate des Netzes kann durch die Änderung der Topologie der Netze weiter gesteigert werden. Untersucht wurden hierarchische Ansätze und die rekurrenten Verbindungen der Netze. Außerdem wird in dieser Arbeit die Verwendung probabilistischer Klassenzugehörigkeiten untersucht und die NN basierten Systeme werden durch einen alternierenden Optimierungsprozess erweitert, welcher *Realignment* genannt wird. Dieser Prozess ersetzt die Connectionist Temporal Classification (CTC) Ausgangsschicht, durch die es ermöglicht wird Sequenzlernen unabhängig von HMMs durchzuführen.

Die Ergebnisse zeigen große Verbesserungen auf jeweils zwei verschiedenen Merkmalssätzen auf drei verschiedenen Datensätzen mit Segmenten in arabischer, englischer und französischer Handschrift. Die Kombinationsmethoden mit NN basierten Merkmalen führen im Vergleich zu üblichen HMM basierten Systemen zu überragenden Ergebnissen und erzielen bessere Fehlerraten als die besten publizierten Ergebnisse auf dem englischen und arabischen Datensatz.

Contents

1 Introduction

Optical character recognition (OCR) treats the problem of translating scanned images of handwritten or printed text into machine-encoded text. The benefits from this procedure include compact storing and electronical investigation of text in images like searching for words or phrases, and enables the application of high-level techniques such as machine translation or text mining.

Due to the variation of the image data arising from the image generation process, statistical methods have become the accepted approach to handle the OCR problem. The idea is to fit a mathematical model to a representative collection of samples drawn from the underlying distribution of the data to be recognized. This will be further explained in Section 1.2. While recognition systems for printed text can be calibrated to read a specific font, unconstrained handwriting recognition additionally introduces variations resulting from different writers and handwriting styles. Considering the image as two-dimensional input signal, the OCR problem is related to the problem of automatic speech recognition (ASR). In ASR hidden markov models (HMMs) with Gaussian mixtures are used. HMMs and the corresponding algorithms are explained in Chapter 2.

In the last decades, artificial neural networks (ANNs) also became very popular in speech and handwriting recognition. They overcome several drawbacks of hidden markov models and can learn transformations of arbitrary complexity through high-order interpolation of simple basis functions called *units*. This property makes ANNs equally suitable for preprocessing and classification purposes and enables a rich field of application. While Feed-forward networks (MLPs) map some fixed size input vector to a fixed size output vector, recurrent neural networks (RNNs) further extend this connectionist approach by introducing self connections of units, allowing the model to map complete input sequences to output sequences. One particular recurrent network structure is the long-short-term-memory (LSTM), which supports the RNN in handling long-term dependencies. The LSTM is the RNN topology mainly considered in this thesis. The advantages and disadvantages of different ANN topologies are described in Chapter 3.

In handwriting recognition the sequence of input signals extracted from the image data has to be mapped to an output sequence of words defined by the correct transcription of the text in the image. Since RNNs are only able to map sequences to other sequences of the same length, they require a final decoding step to find the most likely word sequence associated with the classified label sequence. This

task can reliably be performed by HMMs, where efficient decoding algorithms are known. In order to perform the decoding directly in the RNN framework, a special output layer has been invented, the connectionist temporal classification (CTC) layer. CTC behaves very similar to the HMM decoding algorithm and will be described in Section 3.3.4. The goal of this thesis is to discuss both approaches in detail and to compare their performance on real-world handwriting recognition tasks.

1.1 State-of-the-art

Historically HMMs are the most used model for handwriting recognition because of their success in ASR and their efficient training and search algorithms. Modern HMM systems are based on discriminatively trained Gaussian mixture models [Dreuw & Heigold$^+$ 11a]. Recently Bernoulli based HMMs with an appropriate feature extraction lead to promising results [Gimenez & Khoury$^+$ 10].

Feed-forward neural networks The integration of MLPs into HMMs for ASR started in the late 80's [Lippmann 89], when first approaches were published estimating the emission probabilities or transition probabilities of an HMM using MLPs [Martens 94]. In [Bengio & Mori$^+$ 91] a global method was proposed to compute an error gradient of the Gaussian model, serving as error signal for the MLP backpropagation procedure. Using MLPs directly for sequence learning without integrating them into HMMs was first realized by Time-Delay Neural Networks, where the temporal sequence is converted into a spatial sequence over units [Franzini & Lee$^+$ 90].

Many state-of-the-art handwriting recognition systems make use of MLPs as state emission probability estimator in a hybrid HMM approach [Dreuw & Doetsch$^+$ 11, Espana-Boquera & Castro-Bleda$^+$ 11] or as feature extractor to train a tandem HMM system [Hermansky & Ellis$^+$ 00]. In [Espana-Boquera & Castro-Bleda$^+$ 11] MLPs have been additionally used to either perform several preprocessing steps directly or to decide if a particular preprocessing step should be performed.

Recurrent neural networks RNNs for sequence learning were first introduced by [Jordan 89] and contained self-connected units with a fixed weight of 1.0. [Elman 90] presented a network structure with a special context layer. The context layer had connections from and to a hidden layer, where the connections arriving at the context layer again had a fixed weight of 1.0. Another structure called *echo state network* was proposed by [Jaeger 04], where a sparsely connected random hidden layer with fixed weights was used, while only the weights arriving at the output layer were trained. [Bengio & Simard$^+$ 94] recognized, that the memory provided

by the recurrent connections is not able to cope with long time lags, so RNNs remained unattended in handwriting recognition for a long time.

[Hochreiter & Schmidhuber 97] introduced the LSTM network to address this problem and applied it to many theoretical problems with long-term dependencies. In [Graves & Schmidhuber 05] bidirectional LSTM networks were introduced and applied to ASR for phoneme classification and for recognition in a hybrid HMM approach [Graves & Fernández$^+$ 05]. Together with the CTC output layer by [Graves & Fernández$^+$ 06] large improvements in handwriting recognition were reported [Graves & Bunke$^+$ 08, Graves & Schmidhuber 08, Graves 09].

1.2 Statistical Pattern Recognition

The implementation of a pattern recognition system on a computer requires to define a computable function, which assigns a specific target variable to a given input pattern. If the target variable is defined over a real valued vector space, the underlying task is called *regression* problem. Using a discrete and finite set of target variables or *classes* is defined as *classification* problem. Since every sentence to be recognized by a handwriting recognition system is a concatenation of elements of a discrete set of words (*vocabulary*), the problem belongs to the category of classification. In this case, the computable function stated above is called *decision function* $g(x_1^T, w_1^N)$. It can always be defined such that the corresponding *decision rule* $r(x_1^T)$ assigns the sentence which maximizes the decision function for the given input pattern x_1^T:

$$r(x_1^T) = \arg\max_{w_1^N}\{g(x_1^T, w_1^N)\} \tag{1.1}$$

From a statistical standpoint the optimal decision function maximizes the posterior probability of the target sentence given the input pattern. This is defined by Bayes decision rule:

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

Estimation of a model which optimizes Equation 1.2 requires to provide a probability distribution for every possible input pattern x_1^T. This is infeasible or even impossible for most real-world pattern recognition task, where the domain of input variables has a very large or even infinite size. On the other hand many non-probabilistic approaches, where the decision function is based on Equation 1.1, are able to transform their output in a way that it satisfies the conditions of a posterior

probability distribution. This means they provide a score for each class which is greater or equal to zero and sums up to one. These models discriminate between all competing classes by calculation the decision boundaries directly. Therefore, these models are called *discriminative* models.

In order to stay in a strict probabilistic framework Bayes theorem can be applied to obtain:

$$\arg\max_{w_1^N}\{p(w_1^N|x_1^T)\} \quad = \quad \arg\max_{w_1^N}\left\{\frac{p(x_1^T|w_1^N)\cdot p(w_1^N)}{p(x_1^T)}\right\} \tag{1.3}$$

This formulation requires to estimate a probability distribution for every possible target sentence. Chapter 2 explains how to break this down to word specific models. Since the denominator $p(x_1^T)$ is independent of the sentence, it can be removed from Equation 1.3 and this leads to the final decision rule:

$$\arg\max_{w_1^N}\{p(w_1^N|x_1^T)\} \quad = \quad \arg\max_{w_1^N}\{ \underbrace{p(x_1^T|w_1^N)}_{\text{observation model}} \cdot \underbrace{p(w_1^N)}_{\text{language model}} \} \tag{1.4}$$

Bayes theorem decomposed the original formulation into two models which can be estimated independently of each other. The term $p(x_1^T|w_1^N)$ depends on the input pattern to be classified given the currently hypothesized sentence and is therefore called *observation model*. The term $p(w_1^N)$ is independent of the input pattern and provides a prior probability of the sentence hypothesized by the observation model. In the context of natural language processing this prior model is called *language model*. Although the language model is very important for any state-of-the-art handwriting recognition system, this thesis will not discuss it in further detail.

Since the observation model in Equation 1.4 only depends on the currently hypothesized target sentence, the parameter estimation does not rely on any other sentence. This leads to class-specific models, where the decision boundaries are implicitly estimated by the intersection of the class-distributions. This distinguishes the approach from the discriminative models with Equation 1.2 as decision rule. A class-specific model on the other hand makes it possible to generate artificial input patterns by sampling from it and therefore those models are called *generative* models.

The common approach to estimate the parameters of a pattern recognition system is to learn them from data consisting of several segments. These segments are assumed to be independent and identically distributed (i.i.d.). This means that they are drawn from the same distribution and that they are collected independent of each other. The parameter estimation procedure is called *training*, while the application to some data is called *testing*.

A large and representative amount of training data is important to cover all the variation of the data and it is crucial to perform the testing on data that was not used in the training phase. Especially models like ANNs, that are able to approximate very complex functions, quickly specialize on small training sets. In this case the relative number of errors (*error rate*) evaluated on the training data decays, while the error on unseen data will grow. This effect is called *overfitting* and the only method to avoid it is to evaluate the model frequently on data not involved in the training procedure. Doing this on the predefined data for testing is still a problem, because the system is optimized to model the characteristics of the test data, especially if the number of segments is small. For this reason it is recommended to define an additional *validation set*, which consists of segments that are neither in the training data nor in the test data, and to use it as stopping criterion for the training procedure. Another approach is to include a *regularization term* in the error criterion. This term adds constraints on the solution by penalizing solutions with undesired behavior.

When multiple words appear in a segment, the error calculation cannot be done by simple matching. The possibly different lengths of the recognized word sequence and the reference word sequence require these strings to be aligned to each other, such that the matching afterwards leads to a minimal error. This is realized by the Levenshtein distance [Hunt 90], where the minimal number of edit operations is calculated to transform one sequence into the other. An edit operation corresponds to a deletion, insertion or substitution. The Levenshtein distance is a nonlinear optimization problem and can efficiently be solved using dynamic programming. Normalizing the counts of the edit operations gives the final word error rate:

$$WER = \frac{\#\text{deletions} + \#\text{insertions} + \#\text{substitutions}}{\#\text{reference words}} \tag{1.5}$$

1.3 Large Vocabulary Recognition System

In addition to the components of all pattern recognition system, speech and handwriting recognition require further considerations in order to be applicable to large scale vocabulary problems, as they occur in real world tasks. Taking the words as target classes leads to an infeasible amount of parameters, since usually there are not enough observations of each word for a robust estimation. For this reason the words are broken down into subunits. In handwriting recognition these subunits correspond to *characters*. In order to obtain the final word from the recognized sequence of characters the system needs a *pronunciation lexicon*, which contains possible writing variants of each word. Although there is no pronunciation in handwriting, the name is convenient due to the relation to ASR. Therefore, the evaluation of a handwriting recognition system is reported by a word error rate

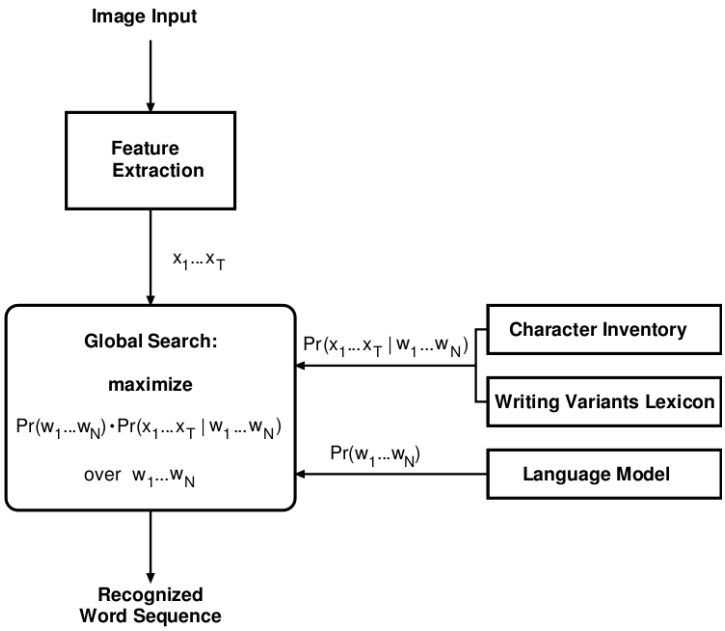


Figure 1.1: Overview of a handwriting recognition system. Important features of the raw image data are extracted to obtain a compact representation. The observation dependent model is used to calculate word hypotheses for all writing variants, which is further weighted by the language model.

(WER) and a character error rate (CER) correspondingly. The overall recognition system is depicted in Figure 1.1.

2 Hidden Markov Models

Using Bayes decision rule as starting point, the observation model should estimate the probability of observing an input pattern sequence x_1^T given the target word sequence w_1^N. Hidden markov models treat this problem by representing each word by a stochastic finite state machine:

$$p(x_1^T|w) = \sum_{s_1^T} p(x_1^T, s_1^T|w) \tag{2.1}$$

Since the states are not observable from the data, they are called *hidden*. The final model for the target sentence w_1^N is constructed by concatenating the corresponding single-word HMMs. Equation 2.1 shows that the whole sequence of states has to be considered to obtain the true probability observing the input pattern sequence x_1^T given the word w. Using the chain rule, this can be reformulated in the following way without loss of generality:

$$p(x_1^T, s_1^T|w) = \prod_{t=1}^{T} p(x_t, s_t|x_1^{t-1}, s_1^{t-1}, w) \tag{2.2}$$

The markov assumption of the HMM reduces the dependency to the current and preceding state. It takes into account that the states are an abstract concept and not observable:

$$\prod_{t=1}^{T} p(x_t, s_t|x_1^{t-1}, s_1^{t-1}, w) \stackrel{!}{=} \prod_{t=1}^{T} p(x_t, s_t|s_1^{t-1}, w) \tag{2.3}$$

$$\stackrel{!}{=} \prod_{t=1}^{T} p(x_t, s_t|s_{t-1}, w) \tag{2.4}$$

$$= \prod_{t=1}^{T} \underbrace{p(s_t|s_{t-1}, w)}_{\text{transition probability}} \cdot \underbrace{p(x_t|s_t, s_{t-1}, w)}_{\text{emission probability}} \tag{2.5}$$

Equation 2.3 decomposes the observation model for a word into a *transition probability* and an observation dependent *emission probability*. The transition probability describes the probability of moving from state at time $t-1$ to state at time t, while

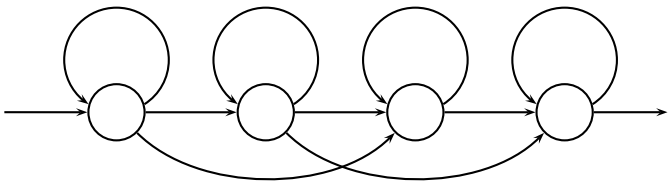


Figure 2.1: Common topology of an HMM system with a loop, forward and skip transition.

the emission probability describes the probability observing x and is conditioned on the current and previous state. The topology of the hidden markov model is task-dependent. Figure 2.1 shows the topology of a state machine associated with a word HMM used in this work. It consists of a forward, loop and skip transition, which is a common choice for the observation model in speech and handwriting recognition.

With the markov assumptions, calculating the probability of an input pattern x_1^T for the HMM of word w corresponds to marginalizing over all possible state sequences:

$$p(x_1^T|w) = \sum_{s_1^T} \prod_{t=1}^{T} \left[p(s_t|s_{t-1}, w) \cdot p(x_t|s_t, s_{t-1}, w) \right] \tag{2.6}$$

The sum in Equation 2.6 is often approximated by the state sequence with maximal probability:

$$p(x_1^T|w) = \max_{s_1^T} \left\{ \prod_{t=1}^{T} \left[p(s_t|s_{t-1}, w) \cdot p(x_t|s_t, s_{t-1}, w) \right] \right\} \tag{2.7}$$

Since in practice one state sequence often turns out to contain the major part of the probability mass, this approximation usually does not influence the final result. Equation 2.7 makes it possible to take the negative logarithm to obtain a better numerical stability. Due to the monotony of the logarithm the final result of the decision rule is not affected by the transformation. Since the emission probability depends on the current state only, the final formulation becomes:

$$p(x_1^T|w) \cong \max_{s_1^T} \left\{ \sum_{t=1}^{T} \left[-\big(\log p(s_t|s_{t-1}, w) + \log p(x_t|s_t, w) \big) \right] \right\} \tag{2.8}$$

In order to calculate the probability of a given word sequence, the corresponding single word HMMs are concatenated to one super-HMM. Especially in a large

vocabulary recognition task this leads to a combinatorial explosion and calculating an optimal solution becomes infeasible. State-of-the-art systems apply efficient heuristic search algorithms, which are not part of this work.

2.1 Model Components

The decomposition into a transition probability and an emission probability builds the two components to be estimated. Although it is possible to estimate the transition probabilities from the training data, the specific choice of the hidden markov model topology used here allows to define them based on the number of states the transition is moving from the current state:

$$p(s|s') = \begin{cases} q(s-s') & s \in \{s'+0, s'+1, s'+2\} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

The state distance function $q(\Delta s)$ is called *time distortion penalty* and the three cases where Equation 2.9 assigns a non zero value correspond to the loop, forward and skip transition respectively. The transition penalties are set manually for each dataset.

The observation dependent emission probabilities are modeled by a suitable probability distribution. Gaussian mixture models (GMMs) are the most common approach. The Gaussian distribution is defined by a mean vector μ describing the point in the observation space the distribution is located at and a covariance matrix Σ describing the statistical variation of the observations. The dimensions of the observation vector are assumed to be statistically independent in the models used in this work. Therefore a diagonal covariance matrix containing only variances σ_d for each dimension d is used. For a robust estimation, the variances are further pooled over all states and words. Using these assumptions, the emission probability of word w in state s for a single Gaussian is given by

$$p(x|s,w) = \frac{1}{\prod_{d=1}^{D}\sqrt{2\pi\sigma_d}} \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_d - \mu_{swd}}{\sigma_d}\right)^2\right) \tag{2.10}$$

The large variability of the data in handwriting recognition leads to a very large variance of a single Gaussian model. To specialize the model the single Gaussians are splitted, resulting in a mixture of single Gaussians localized at different points in the observation space. Each mixture component l takes the form of Equation 2.10. In order to obtain a proper normalization the mixture model performs a linear combination of the components with weights $p(l|s,w)$ that satisfy the conditions of

a probability distribution:

$$p(x|s,w) = \sum_l p(l|s,w)p(x|s,w,l) = \sum_l c_{sl}\mathcal{N}(x|\mu_{swl},\sigma^2) \tag{2.11}$$

2.2 Training

Using the assumptions and simplifications described in the previous section, the free parameters to be estimated are a globally pooled diagonal covariance matrix and a mean vector for each mixture component together with its mixture weight. The number of mixture components is chosen manually during the training procedure by successively splitting the mixture component with largest variance starting from a single Gaussian.

A common criterion for the parameters of the Gaussian mixture model is to choose them such that the likelihood of the training data is maximized. This means that there are no other parameters which result in a larger probability, when the training data x_1^T is recognized by the model. Denoting the true set of all parameters by θ and their estimates by $\hat{\theta}$, the *maximum likelihood* criterion can be described by:

$$\begin{aligned}
\log L(\hat{\theta}) &= \log p(x_1^T|w_1^N,\hat{\theta}) \\
&= \log \sum_{\{s_1^T\}} p(x_1^T, s_1^T|w_1^N,\hat{\theta}) \\
&= \log \sum_{\{s_1^T\}} \prod_{t=1}^{T} p(x_t|s_t,\hat{\theta})p(s_t|s_{t-1}) \\
&= \log \sum_{\{s_1^T\}} \prod_{t=1}^{T} \Big[\sum_l \hat{c}_{sl}\mathcal{N}(x_t|\hat{\mu}_{sl},\hat{\sigma}^2)\Big] q(s_t - s_{t-1}) \\
&\approx \max_{\{s_1^T\}} \sum_{t=1}^{T} \log\Big[\sum_l \hat{c}_{sl}\mathcal{N}(x_t|\hat{\mu}_{sl},\hat{\sigma}^2)\Big] + \log q(s_t - s_{t-1})
\end{aligned}$$

Instead of using all mixture components, the maximum or *viterbi approximation* was used as in Equation 2.7. Considering only the mixture component with maximal weight leads to the final criterion with all simplifications:

$$LL(\hat{\theta}) = \max_{\{s_1^T, l_1^T\}} \sum_{t=1}^{T} \big[\log \hat{c}_{sl} + \log \mathcal{N}(x_t|\hat{\mu}_{swl},\hat{\sigma}^2) + \log q(s_t - s_{t-1})\big] \tag{2.12}$$

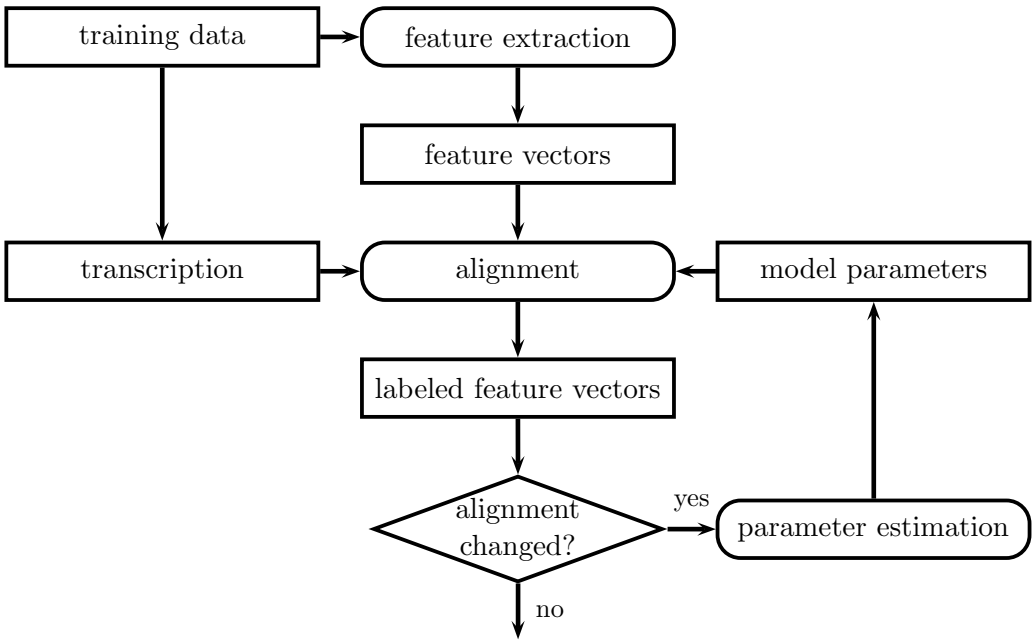


Figure 2.2: HMM training procedure. Once an initial alignment maps the labels of the transcription to the feature vectors, the model parameters and the following alignments can be calculated alternately to obtain the best model.

The log-likelihood function is a convex function and therefore has a unique optimum. This means, once the correct alignment of the observations to the states is given, a closed-form solution for the parameters of the model exists. Unfortunately, this optimal alignment is hidden, and therefore the training procedure consists of repeating two steps in an expectation-maximization fashion:

1. Maximization: Starting from a given alignment computed independently of the model, estimate the model parameters by optimizing the log-likelihood function

2. Expectation: Compute a new alignment with maximal probability using the model parameters of step 1.

There is no guarantee, that the algorithm will converge and due to the unknown alignment, the objective function is not convex anymore. The algorithm therefore can lead to a suboptimal solution, even if it convergences. The following subsections will explain how to perform the parameter estimation and alignment steps. Figure 2.2 illustrates the complete training procedure of an HMM.

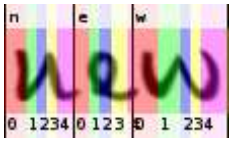


Figure 2.3: Example of an alignment where each character is represented by an HMM with 5 states labeled from $0 \dots 4$. The top line shows the transcription and the central line shows the image data. The bottom line shows the states to which the image slices are aligned.

2.2.1 Parameter Estimation

Assuming the alignment of the observations x_t to the states s_t and the optimal mixture components l_t is known, the maximum likelihood parameters can be obtained by setting the corresponding derivative of the log-likelihood function to zero. For Gaussian distributions it turns out that the empirical mean and the empirical variance maximize the likelihood:

$$\hat{\mu}_{sld} = \frac{\sum_{t=1}^{T} \delta_{s,s_t} \delta_{l,l_t} x_{td}}{N(s,l)} \qquad \hat{\sigma}_d^2 = \frac{1}{T} \sum_{t=1}^{T} \left(x_{td} - \hat{\mu}_{s_t l_t d}\right)^2 \tag{2.13}$$

where the formulas are given for a specific dimension d. The term $\delta(x, x')$ corresponds to the Kronecker delta, which evaluates to 1 on equality of the arguments and to 0 otherwise. $N(s,l)$ is the number of observations aligned to state s and mixture component l. Once the parameters of the single Gaussians are known, the mixture weights can be calculated by counting the number of assigned observations:

$$\hat{c}_{sl} = \frac{N(s,l)}{\sum_l N(s,l)} = \frac{N(s,l)}{N(s)} \tag{2.14}$$

2.2.2 Viterbi Alignment

The estimation of the model parameters requires each observations x_t to be mapped to exactly one state y_s of the HMM for y (see Figure 2.3). A trivial method which is independent of any prior information about the sequences is the linear time alignment, where the observation sequence of length T is linearly stretched or compressed to the length S of the state sequence:

$$s = t \cdot \frac{S}{T} \tag{2.15}$$

One major drawback of this approach is that it does not take different writing variants of the same character into account. In general the time alignment has to

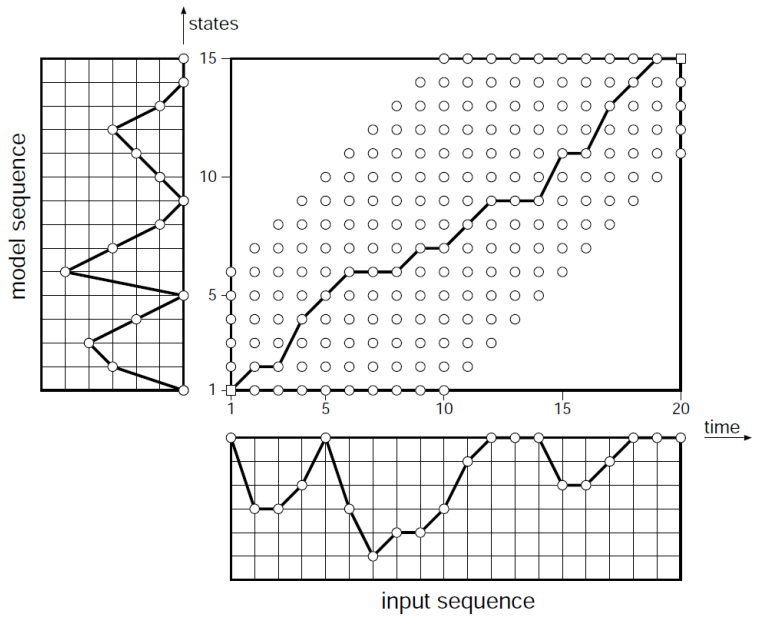


Figure 2.4: Time-Alignment is a nonlinear optimization problem, where each observation (time axis) has to be mapped to a state in the HMM (state axis). The path is constrained by the HMM topology. In the given example each step on the time axis is allowed to make up to three steps on the state axis, which corresponds to the loop/forward/skip model shown in Figure 2.1. Illustration from [Ney & Schlüter 10].

be formulated as a nonlinear optimization problem. Plotting the input observation sequence against the states of the HMM generates a grid where each valid solution is a path w_1^L with $w_l \in (t, s)$ starting in $(0, 0)$ and ending in (T, S) satisfying the HMM topology. See Figure 2.4 for an illustration. For any given distance function $d(x_{t(l)}, y_{s(l)})$ that satisfies the conditions of a metric, an optimal solution is defined as a valid path with minimal cost:

$$\min_{l \mapsto w_l} \left\{ \sum_{l=1}^{L} d(x_{t(l)}, y_{s(t)}) \right\} \tag{2.16}$$

Considering only the mixture component with maximal weight in the GMM and the global pooling of the variances, each state is represented by a single Gaussian. Taking the logarithm of Equation 2.10 reveals that under these conditions the only observation dependent part that remains is a weighted euclidean distance with the

variances as weights:

$$d(x_{t(l)}, y_{s(l)}) = -\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_{t(l)\,d} - \mu_{y_{s(l)\,d}}}{\sigma_d}\right)^2 \tag{2.17}$$

Solving this optimization problem can be done efficiently by observing that any path which is optimal from time $t+1$ to T is still optimal if the predecessor with minimal distance is chosen. This allows the problem to be decomposed into subproblems which can be solved by dynamic programming [Sakoe & Chiba 78].

Since 2.17 depends on the initially unknown model parameters it cannot be used to perform the first alignment step. In this work the training procedure therefore starts with a linear alignment to obtain the first estimates of the model parameters and continues with the nonlinear alignment approach in the following steps.

3 Neural Networks

Inspired from the biological neuronal system of the animal brain, artificial neural networks (ANNs) have evolved to be one of the most commonly used model for pattern recognition and machine learning tasks. A neural network model consists of units with associated *basis* or *activation functions*, which are connected through *weights*. These units are arranged in *layers*. Although ANN structures like the radial basis function networks [Moody & Darken 89] exist where the parameters of the activation functions are estimated during training, the ANNs used in this thesis use fixed activation functions. The connecting weights are the free parameters to be estimated.

Every ANN has an input layer with one unit for each dimension of the observation vector and an output layer with one unit for each target class. Within these layers one or more layers are included, which have no particular desired behavior and have no direct connection to the outer world. For this reason they are called *hidden layers*. Hidden units are able to discover regularities in the data and enrich the family of functions the network is able to approximate.

This chapter starts by introducing preliminaries common to all neural networks used for classification. Afterwards the different network topologies used in this thesis are introduced and the corresponding training procedures are described.

3.1 Preliminaries

Neural networks are a discriminative and non-probabilistic approach. They act as function approximator and therefore naturally perform a regression. In the case of classification the corresponding decision function for an observation vector x and a target class k ideally provides an output value that equals to the posterior probability $p(k|x)$:

$$g(x,k) \stackrel{!}{=} \begin{cases} 1 & x \text{ belongs to class } k \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The criterion used in the training procedure as well as the activation function of the units in the output layer have to be chosen in a way such that the network generates outputs satisfying Equation 3.1.

3.1.1 Error Criterion

During training a specific loss-function is minimized. Traditionally the loss-function is the *squared error criterion*:

$$\sum_{c=1}^{K}[g(x,c)-\delta(k,c)]^2 \tag{3.2}$$

where $\delta(x,y)$ corresponds to the Kronecker function defined in Section 2.2.1. For any given set of training samples $x_1,\ldots,x_N$ the criterion averages over all training samples and taking the limit w.r.t. the number of training samples N leads to the expected value of the squared error over the observation space [Ney 05]:

$$\begin{aligned}
F(g) &= \lim_{N\longrightarrow\infty}\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}[g(x_n,c)-\delta(k_n,c)]^2 && (3.3)\\
&= \int_x dx\sum_{k=1}^{K}p(x,k)\cdot\sum_{c=1}^{K}[g(x,c)-\delta(k,c)]^2 && (3.4)\\
&= \int_x dx\ p(x)\cdot\underbrace{\sum_{k=1}^{K}p(k|x)\cdot\sum_{c=1}^{K}[g(x,c)-\delta(k,c)]^2}_{e(x)} && (3.5)
\end{aligned}$$

The local error $e(x)$ at point x in the observation space can be rewritten to reveal the optimal decision function $g(x,k)$:

$$\begin{aligned}
e(x) &= \sum_{k=1}^{K}p(k|x)\cdot\sum_{c=1}^{K}[g(x,c)-\delta(k,c)]^2 && (3.6)\\
&= \sum_{k=1}^{K}p(k|x)\cdot\left(1-2g(x,k)+\sum_{k=1}^{K}g^2(x,c)\right) && (3.7)\\
&= 1-2\sum_{c=1}^{K}p(c|x)\cdot g(x,c)+\sum_{c=1}^{K}g^2(x,c) && (3.8)\\
&= 1-\sum_{c=1}^{K}p^2(c|x)+\sum_{c=1}^{K}[p(c|x)-g(x,c)]^2 && (3.9)
\end{aligned}$$

As expected the optimal decision function corresponds to the posterior probability distribution. In addition there is a term which is independent of $g(x,k)$ and

which evaluates to zero if and only if the full probability mass is assigned to exactly one class $c^\star$, i.e. $p(c^\star|x) = 1$ and $p(c|x) = 0\ \forall c \neq c^\star$. From an information theoretical perspective this corresponds to a distribution with minimal entropy and this is the case for any given training data where exactly one label is assigned to each training sample. However, Equation 3.6 shows that training can be performed with any posterior distribution and such probabilistic labels are referred to as *soft labels* in this work. Similarly a labeling with a single class per training sample is referred to as *hard labels*.

While the squared error criterion is commonly used in practice, an entropy-based error criterion that considers the network's outputs as independent hypotheses holds at least in theory several advantages when used for classification. Specifically the *cross entropy* is less impacted by outliers and is better suited in estimating small posterior probabilities that often occur if the number of classes is large [Kline & Berardi 05]. It describes an information theoretical distance between the distribution assumed by the network and the distribution provided by the labeling of the training data:

$$-\sum_{c=1}^{K} \delta(k,c) \ln(g(x,c)) \tag{3.10}$$

Although it was shown by [Richard & Lippmann 10] that neural networks minimizing the squared error criterion and cross entropy criterion are capable of accurately estimating posterior probabilities for finite training sets, the cross entropy criterion became the common choice for neural networks applied to classification tasks.

3.1.2 Model Components

The units in every conventional neural network act in exactly the same way by accumulating all inputs x scaled by associated weights α in order to provide an output to other units by applying an activation function $f(z)$. [Rosenblatt 57] introduced this approach for binary classification using a simple thresholding as activation function which follows the hebbian learning theory of animal neuronal firing rates:

$$y = f(z) = \begin{cases} 1 & z + \alpha_0 > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

where $z = \sum_{d=1}^{D} \alpha_d \cdot x_d$ is the dot product of the input and weight vector and α_0 is a bias. This bias value can easily be integrated in the model by adding a *bias unit* that has no inputs and provides a constant output of 1 which is scaled by its

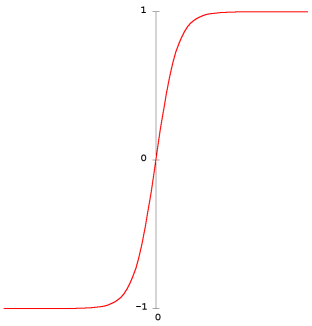
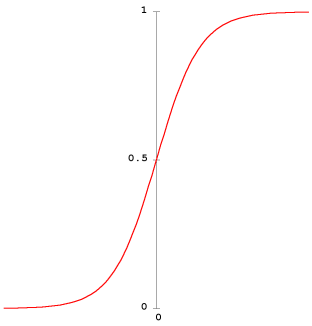
hyperbolic tangent	logistic sigmoid
	
$f_{\tanh}(z) = tanh(z)$	$f_{\text{logistic}}(z) = (1 + \exp(-x))^{-1}$
$\frac{d}{dz} f_{\tanh}(z) = 1 - f_{\tanh}(z)^2$	$\frac{d}{dz} f_{\text{logistic}}(z) = f_{\text{logistic}}(z) \cdot (1 - f_{\text{logistic}}(z))$

Table 3.1: Two differentiable activation functions with their derivative.

weight to the desired value α_0. A neural network without hidden layer, that has a unit for each x_d in the input layer and a single output unit with Equation 3.11 as activation function is called *perceptron*. It models a linear decision boundary shifted by the bias value and therefore it is only able to solve *linearly separable* problems. The calculation of the *net input* z corresponds to the dot product of the inputs and the weights. This can be generalized using kernel methods to allow for non-linear decision boundaries. This method makes the perceptron closely related to support vector machines [Schölkopf & Smola 02]. The objective function is convex and has a closed-form solution.

If more layers of units with non-linear activation functions are introduced the objective function is not convex anymore and gradient based methods need to be applied to find a local optimum. To calculate this gradient the activation functions need to be differentiable. Although there is in general no further restriction, the function usually is still chosen to perform a smooth thresholding. Sigmoidal activation functions are a common choice and in pattern recognition the hyperbolic tangent and logistic sigmoid perform very well. Their derivatives both have a Gaussian shape which is centered at zero. Table 3.1 shows both functions together with their derivatives.

When using the cross entropy error criterion for classification it is useful to constrain the activations of the units in the output layer to be properly normalized. Combining this with sigmoidal functions leads to a multinomial logistic regression model also used in log-linear models like the maximum entropy classifier. In the

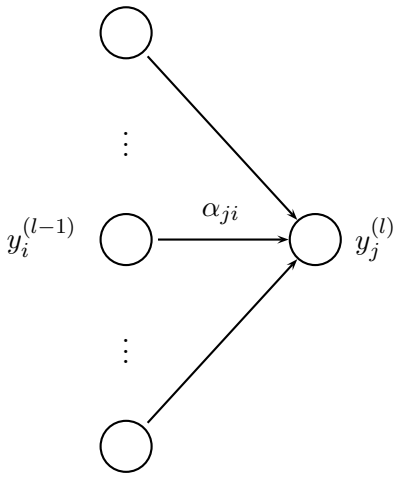


Figure 3.1: General structure of a feed-forward network. A unit in layer l receives only inputs from units in layer $(l-1)$ but not vice versa.

neural network terminology the correspondence to this model is the *softmax activation function*. Since there is one output unit for each class k which activation corresponds to the decision function of observation x w.r.t. k, the posterior estimate of the softmax activation function is given by:

$$p(c|x) = \frac{\exp g(x,c)}{\sum_k \exp g(x,k)} \tag{3.12}$$

3.2 Feed-forward Networks

Arranging units in input, hidden and output layers allows the network to process the information layerwise. The number of layers and the types of connections between their units limits the computational power of the model. A *feed-forward* network is defined as an ANN where the connections between the units do not form a cycle, i.e. all connections arriving at a unit have their origin in a previous layer. Figure 3.1 illustrates this structure. Any network of this kind approximates a function that maps a fixed-size input vector to some fixed-size output vector which dimensionality is defined by the number of targets in the output layer.

The linear perceptron with a single output target as described in the previous section is the simplest kind of feed-forward network. As mentioned it is limited to solve problems that are linearly separable, i.e. where the class-distributions can be separated by a hyperplane in the input vector space. Boolean function like

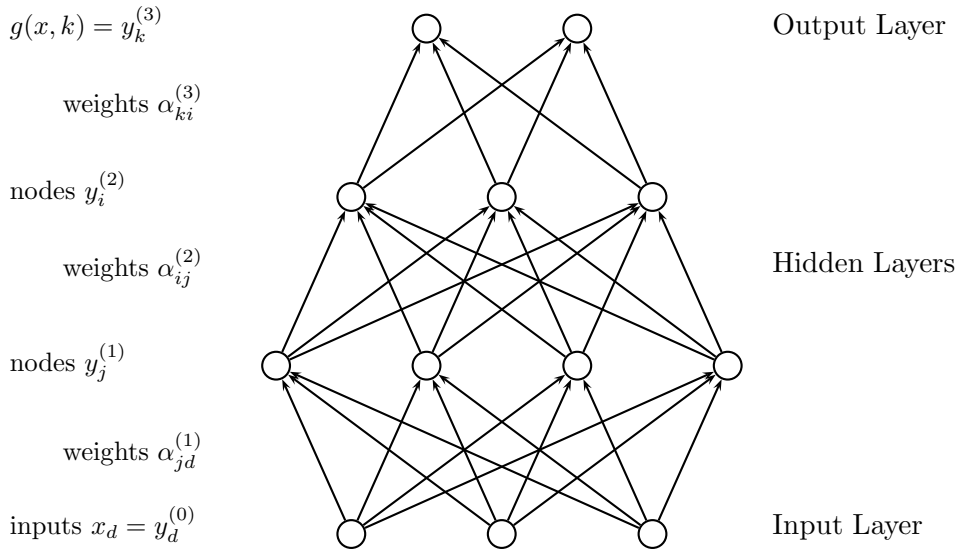


Figure 3.2: Example of an MLP architecture with 2 hidden layers.

the logical AND are linearly separable. An example for a binary function that is not linearly separable is the XOR function, which evaluates to 1 if and only if exclusively one input dimension is 1. However, recently it has been shown that *parallel perceptrons* with a single layer consisting of a finite number of n perceptrons without lateral connections can compute every continuous function $g : \mathbb{R}^d \to [-1, 1]$ [Auer & Burgsteiner$^+$ 08]. This result promotes parallel perceptron to *universal approximators* on the interval $[-1, 1] \subset \mathbb{R}$.

3.2.1 Multilayer Perceptron

The older and even more important insight came from [Hornik 91], who showed that neural networks are universal approximators for the complete set of functions in the vector space $\mathbb{R}^K$ for K output units. He recognized that it is not the specific choice of the activation function but a single hidden layer with a sufficient number n of hidden units that enriches the family of functions. The following *universal approximation theorem* gives a formal definition for a D dimensional input space and a single linear output unit k which can be easily extended to multiple output units [Hornik 91]:

Theorem 1
Let $f(z)$ a non-constant, bounded and monotonically increasing continuous function. For any compact subset $X \subseteq \mathbb{R}^D$ the space of continuous functions on X

is denoted by $C(X)$. *Then* $\forall g \in C(X), \forall \epsilon > 0 : \exists n \in \mathbb{N}, \alpha_{ji}, \alpha_{kj}, \alpha_0 \in \mathbb{R}, j \in \{1 \dots n\}, i \in \{1, \dots D\}$:

$$G(X) = \sum_{j=1}^{n} \alpha_{kj} f\left(\sum_{i=1}^{D} \alpha_{ji} x_i + \alpha_0\right)$$

as an approximation of the function $g(X)$*; that is*

$$||g(X) - G(X)|| < \epsilon \tag{3.13}$$

The left side of inequality 3.13 describes a measure for the closeness between the functions like e.g. the squared error described in Section 3.1.1. The extension of the perceptron by introducing hidden units leads to the *multilayer perceptron* (MLP) which became one of the most popular classifiers in pattern recognition. Figure 3.2 gives an example of an MLP. Although the universal approximation theorem is of great importance in theory it does not provide any information how many hidden units are required in the hidden layer for a given task. Further, due to the non-convexity of the optimization problem, even a sufficient number of hidden units does not guarantees that the data driven training procedure converges to a fixed point in the parameter space that approximates the function of interest with arbitrary precision. In practice this means that the number of hidden units has to be tuned empirically and that additional hidden layers can lead to a better convergence behavior in the optimization process.

Contextual Information

MLPs naturally approximate functions that classify a given input vector into a set of classes. This makes them incapable for general sequence learning tasks where the mapping from frames of the observed sequence to the symbols of the output sequence usually is unknown. For this reason they are combined with HMMs, which are able to provide this mapping by aligning the elements of the input sequence to the states associated with an element of the output sequence. Further, they are *local in time*, which means that the output at a frame at time step t exclusively depends on the input given at that time step. However, it is obvious that contextual information is important or even required to make reliable predictions of the target symbol. Since MLPs are local in time, the integration of contextual information is not trivial and still an open issue.

The easiest way to provide contextual information to an MLP is to transform the temporal sequence into a spacial one by increasing the number of units in the input layer in order to feed in data of future and past context of the current frame. Since the sequences to be learned vary in length, the only way to provide this information is to apply a window of size $2k + 1$ to the observation sequence, such

that the posterior probability estimated for the label at state s_t of the HMM for an observation x_t at time step t can be written as:

$$p(s_t|x_t) = p(s_t|x_{t-k}, \ldots, x_{t+k}) \tag{3.14}$$

Non-symmetric windows reaching in the past or future are also possible but not used in this work. Giving temporal patterns a spatial interpretation has several drawbacks. First, with each additional input unit the number of weights in a fully connected ANN increases by the size of the first hidden layer. This prohibits arbitrary large windows because a robust estimation of these weights would require a much larger amount of data that is often not available.

Second, it is usually not possible to consider the complete input sequence. The requirement of a fixed window has no selective relation to the context actually required for the current frame, i.e. in most cases either irrelevant information is provided if the window is too large or relevant information is missing if the window is too small. Further, a suitable border condition has to be applied to account for the start and the end of the input sequence.

Finally, temporal displacements are considered as spacial displacements. Consider e.g. the temporal sequences $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$, which are temporal displaced versions of the same basic pattern. The spatial interpretation, however, leads to completely different activations of the input layer of the ANN and therefore the learning procedure would either specialize on a single appearance or generalize on all three, while none of these cases would correspond to the actually desired similarity [Elman 90].

Hierarchical Multilayer Perceptron

The first problem described above can be addressed without changing the feed-forward structure of the MLP. The idea is to build a cascade of multiple MLPs with different context windows, where the posterior estimates obtained from the output layer of each neural network in the hierarchy is used as additional input for the next neural network in the hierarchy [Valente & Vepa$^+$ 07]. These cascades are called *hierarchical multilayer perceptrons*. In this work hierarchical MLPs with two cascades are used, where the first MLP does not use any temporal context at all to calculate posterior estimates which are localized in time. These localized posteriors are concatenated with the original input data and fed into a second MLP which makes use of contextual information by applying a window. Figure 3.3 shows this kind of hierarchical MLP.

With hierarchical neural networks the input layer of the first network remains small which enables a robust training of the weights, while the second network benefits from the highly class-specific information encoded in the output of the first network. The training procedure can therefore specialize on the localized posteriors

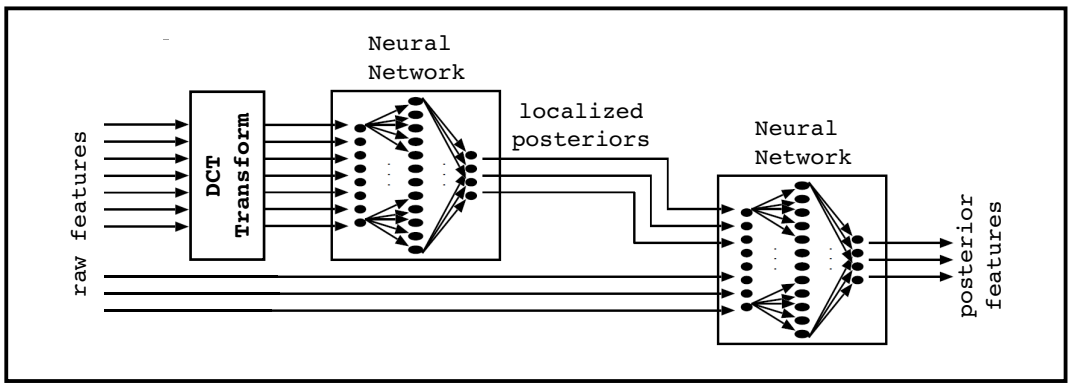


Figure 3.3: Example of a hierarchical multilayer perceptron with 2 cascades.

and additionally it can utilize the contextual information, if required. This simplifies the convergence to a fixed point in the parameter space.

Further, to improve the posterior estimates of the first network, temporal patterns are extracted from the image data and used as input [Hermansky & Sharma 98]. A temporal pattern corresponds to a vector centered at each pixel that contains elements of the row of the pixel in the original image. This vector is interpreted as temporal sequence and a discrete cosine transformation is applied to perform a decorrelation of the neighboring pixels. In the target image the pixel is then represented by a selection of the resulting coefficients. Note that this procedure does not increase the number of frames in the image but the dimensionality of each frame. The more general case of using a PCA window reaching over all rows in the image did not lead to a better performance. See Figure 3.4 for an illustration.

3.2.2 Training

In the learning phase of a neural network the weights are optimized to reach a point in the space spanned by the weights that leads to minimal error when the given training set $(x_1, c_1), \ldots, (x_n, c_n), \ldots, (x_N, c_N)$ is processed by the network. The most widely used method to perform the training for MLPs is *error back propagation*. In this method a stochastic gradient is calculated based on the derivative of the activation function w.r.t. the objective function defined by the error criterion. The errors are propagated back through all layers in order to adjust the weights. It was already described by [Bryson & Ho 69] and first applied to neural networks by [Werbos 74]. However, attention was first paid to it through the work of [Rumelhart & Hinton$^+$ 88]. The training procedure described below performs a weight adjustment after each training sample, while other methods like e.g. RPROP [Riedmiller & Braun 93] follow the idea of *batch learning* and perform a weight ad-

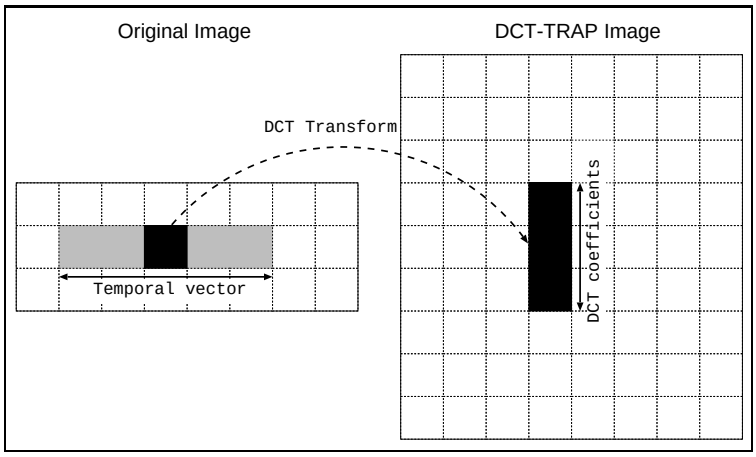


Figure 3.4: Illustration of DCT-TRAP transformation. In this example, a five-dimensional temporal vector centered at each pixel is extracted and mapped to the first three coefficients of its discrete cosine transformation.

justment only once for the complete training set. These methods are much faster in training but need a longer time to converge and often the resulting model shows a bad performance compared to methods that adjust the weights after each training sample.

The error backpropagation algorithm can be divided into two phases performed for each sample in the training set in one or more *epochs* with a runtime and memory complexity of $O(W)$ per epoch in the number of weights W:

- Forward Pass: Successively calculate unit activations up to the output layer
- Backward Pass: Find the derivative of objective function w.r.t. the weights and update the weights

Denoting the local error for a training sample (x_n, c_n) by E_n the global error over the complete training set is denoted by E and can be defined as:

$$E = \sum_{n=1}^{N} E_n \tag{3.15}$$

The local error itself is defined by the error criterion. Since this work deals with classification problems, the cross entropy error is used in this section. The derivation of error backpropagation using the squared error criterion with logistic

sigmoid output activation functions can be found in Appendix 7.1. Using the cross entropy error the objective function defined over all training samples is:

$$E = -\sum_{n}^{N}\sum_{k=1}^{K}\delta(k, c_n)\ln y_k^{(L)} \tag{3.16}$$

where L is the index of the output layer, $y_k^{(L)}$ the activation of the kth output unit and $\delta(k, c_n)$ is the Kronecker delta. Finding the derivative of the weights in the preceding layers is done by a repeated application of the chain rule for partial derivatives. Note that the outer sum in the objective function allows for an independent optimization of the training samples using the local error E_n. First, the error backpropagation equations are derived for the output layer and using this result the equations for the hidden layers and the weight update rule can be derived.

Output Layer

Differentiating the local cross entropy error E_n gives for the output layer:

$$\frac{\partial E_n}{\partial y_k^{(L)}} = -\frac{\delta(k, c_n)}{y_k^{(L)}} \tag{3.17}$$

Since the softmax activation function defined in Equation 3.12 depends on all units in the output layer the derivative w.r.t. the net input $z_k^{(L)} = \sum_j \alpha_{ji}^{(L)} \cdot y_j^{(L-1)}$ is given by the chain rule:

$$\frac{\partial E_n}{\partial z_k^{(L)}} = \sum_{k'=1}^{K}\frac{\partial E_n}{\partial y_{k'}^{(L)}}\frac{\partial y_{k'}^{(L)}}{\partial z_k^{(L)}} \tag{3.18}$$

Differentiating the softmax activation function results in [Bishop 06]:

$$\frac{\partial y_{k'}^{(L)}}{\partial z_k^{(L)}} = y_k^{(L)}\delta(k, k')^{(L)} - y_k^{(L)}y_{k'}^{(L)} \tag{3.19}$$

Substitution 3.19 and 3.17 into 3.18 gives:

$$\frac{\partial E_n}{\partial z_k^{(L)}} = \sum_{k'=1}^{K} -\frac{\delta(k', c_n)}{y_{k'}^{(L)}} \cdot \left[y_k^{(L)}\delta(k, k') - y_k^{(L)}y_{k'}^{(L)}\right] \tag{3.20}$$

$$= y_k^{(L)}\sum_{k'=1}^{K}\delta(k', c_n) - \sum_{k'=1}^{K}\frac{y_k}{y_{k'}}\delta(k', c_n)\delta(k, k') \tag{3.21}$$

For hard targets only one k' can be equal to the label c_n. Therefore it follows that $\sum_{k'}^{K} \delta(k', c_n) = 1$. Further the right sum of the difference in Equation 3.21 evaluates to zero in all cases except if $k = k' \wedge k' = c_n$ and therefore:

$$\frac{\partial E_n}{\partial z_k^{(L)}} = y_k^{(L)} - \delta(k, c_n) \tag{3.22}$$

Hidden Layer

Let $\delta_j^{(l)} = \frac{\partial E_n}{\partial z_j^{(l)}}$ for some arbitrary unit j in layer $1 \leq l \leq L$. Using the chain rule this evaluates for the last hidden layer to:

$$\delta_i^{(L-1)} = \frac{\partial E_n}{\partial y_i^{(L-1)}} \frac{\partial y_i^{(L-1)}}{\partial z_i^{(L-1)}} \tag{3.23}$$

$$= \frac{\partial y_i^{(L-1)}}{\partial z_i^{(L-1)}} \sum_{k=1}^{K} \frac{\partial E_n}{\partial z_k^{(L)}} \frac{\partial z_k^{(L)}}{\partial y_k^{(L)}} \tag{3.24}$$

Differentiating the activation function $f_i\big(z_i^{(L-1)}\big)$ of unit i and the net input $z_i^{(L-1)}$ w.r.t. the output activations leads to:

$$\delta_i^{(L-1)} = f_i'\big(z_j^{(L-1)}\big) \sum_{k=1}^{K} \delta_k^{(L)} \alpha_{ki}^{(L)} \tag{3.25}$$

Then the derivatives for any previous layer l can be calculated recursively:

$$\delta_j^{(l)} = f_j'\big(z_j^{(l)}\big) \sum_i \delta_i^{(l+1)} \alpha_{ij}^{(l+1)} \tag{3.26}$$

Finally the derivatives w.r.t. the network weights are given by:

$$\frac{\partial E_n}{\partial \alpha_{ij}^{(l)}} = \frac{\partial E_n}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial \alpha_{ij}^{(l)}} = \delta_i^{(l)} y_j^{(l-1)} \tag{3.27}$$

Weight Update Rule

Given the derivatives w.r.t. the weights α_{ij} the learning is performed in a gradient descent manner. The update rule adjusts the weights in negative slope of the

gradient given by the derivative scaled by a *learning rate* $0 \leq \gamma \leq 1$. The weight update for epoch p is:

$$\Delta\alpha_{ij}^{(l)}(p) = -\gamma \frac{\partial E_n}{\partial \alpha_{ij}^{(l)}(p)} \tag{3.28}$$

Gradient descent can easily get stuck in local minima. In order to speed up the convergence and to escape local minima, a momentum term is included in the update rule [Plaut & Nowlan$^+$ 86]. An additional parameter $0 \leq m \leq 1$ is introduced that takes the previous weight update into account:

$$\Delta\alpha_{ij}^{(l)}(p) = m\Delta\alpha_{ij}^{(l)}(p-1) - \gamma \frac{\partial E_n}{\partial \alpha_{ij}^{(l)}(p)} \tag{3.29}$$

The learning rate usually is set to perform a fixed-size step in all epochs. Convergence is detected by evaluating the network on a separate validation set after each epoch to avoid overfitting. In addition, the learning rate can be adjusted after each epoch. The idea is to use a large learning rate at the beginning of the training and to reduce it after the net starts to stabilize. An example for a learning rate schedule is to start with a constant learning rate and reduce it exponentially once the reduction of the error on the validation set is small. This method was originally implemented in the Quicknet[1] neural network software, where it is known as *newbob* learning rate schedule. In this work, the Quicknet software is used for experiments with feed-forward ANNs.

3.3 Recurrent Networks

As already mentioned in Section 3.2.1 there exists no natural way to introduce contextual information when using MLPs instead of coding the information into the feature vector. Therefore, the only way to provide information about future and past frames is to mix up the spatial interpretation of the units in the input layer with the temporal interpretation of the neighborhood. To overcome this drawback the network requires dynamic properties which are responsive to temporal sequences, i.e. it must be given *memory* [Elman 90]. This is accomplished by allowing cyclical connections leading to *recurrent neural networks* (RNNs).

The extension from MLPs to RNNs has larger consequences on the modeling than one might suggest. While MLPs map a fixed-size input vector to a fixed-size output vector, RNNs model a sequence-to-sequence mapping using, at least in theory, the complete history of previous frames to the output of the current frame. Further, an equivalent result to the universal approximation theorem for MLPs is that

[1] http://www.icsi.berkeley.edu/Speech/qn.html

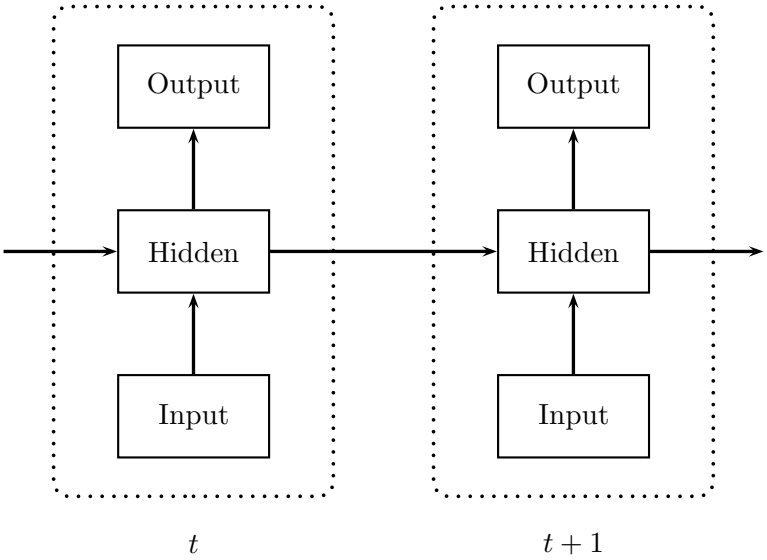


Figure 3.5: A recurrent neural network unfolded in time. The recurrent connections provide information across the time axis.

RNNs with a single hidden layer and a sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping with arbitrary accuracy [Hammer 00].

Different ways of recurrent connections are possible and were used to construct a various number of RNN models, including Jordan networks [Jordan 89], Elman networks [Elman 90], and echo state networks [Jaeger 04]. In this thesis the recurrent connections are restricted to self connected hidden layers. Assuming an RNN with a single hidden layer, this means that the input layer is connected by feed-forward connections to the hidden layer and that the hidden layer is connected by feed-forward connections to the output layer, while in addition recurrent connections exist from the hidden layer to itself. Unfolding a network in time leads to connections from the units in the hidden layer at a time step t to the units in the same hidden layer at time step $t + 1$. The recurrent connections therefore introduce the desired temporal property providing a natural way treating time without making contextual information an explicit part of the input. Figure 3.5 visualizes the RNN structure used in this work unfolded in time.

While the RNN model fits much better to sequence learning tasks like handwriting recognition, they also can not be directly applied when the length of the output differs from the length of the input sequence. An RNN still estimates one output vector for each input vector. Therefore an alignment procedure has to be performed in order to obtain a direct mapping between the observation vectors and the output symbols. As in the case of MLPs this is done in this work using HMMs, such that

the RNN is trained using the state sequence as labels to which the observations were aligned.

The forward pass of an RNN is very similar to the forward pass of MLPs. The net input for a unit j in hidden layer l at time step t is calculated by the weighted sum of the activations of the previous layer at the current time step t but additionally considers the activations of the same hidden layer at time step $t-1$:

$$z_j^{(l)}(t) = \sum_i \alpha_{ji}^{(l-1)} y_i^{(l-1)}(t) + \sum_i \alpha_{ji}^{(l)} y_i^{(l)}(t-1) \tag{3.30}$$

Note the change in notation of the net input $z_j^{(l)}(t)$ and the unit activations $y_i^{(l)}(t)$ denoting their dependence on the time step t.

3.3.1 Contextual Information

The dependence on the previous time step recursively reaches to the begin of the input sequence and provides contextual information about the complete history to the network. Therefore, the posterior estimate at time step t for state s_t given observation x is:

$$p(s_t|x) = p(s_t|x_1, \ldots, x_t) \tag{3.31}$$

Contextual information about frames occurring after the current frame can also be of importance for a reliable prediction on the current frame. In handwriting recognition, where the label of a particular state in the HMM usually corresponds to a character, the history ending in the current frame covers the character to be estimated in most cases only partially. Characters, where the flow in writing is interrupted, often show a temporal displacement of their separated parts like e.g. the dot over an *'i'*. Further, the information about a matching parenthesis can be beneficial for the prediction of the corresponding opening parenthesis. [Rosenbaum & Cohen$^+$ 07] believe that human cognition also has a "memory for what has happened" as well as a "memory for what is about to happen", although there is of course no supervision for the latter.

While frames in the past are provided by the recurrent connections, no contextual information of future frames is taken into account by RNNs when the output for the frame at time step t is calculated. A trivial solution is to provide the information by a window containing future frames, i.e. to transform the temporal sequence into a spatial one as for MLPs. Again, this leads to the same disadvantages described in Section 3.2.1.

Another approach is to delay the output label of the frame at time step t by a specific amount of time steps Δt [Robinson 94a]. In this case the contextual

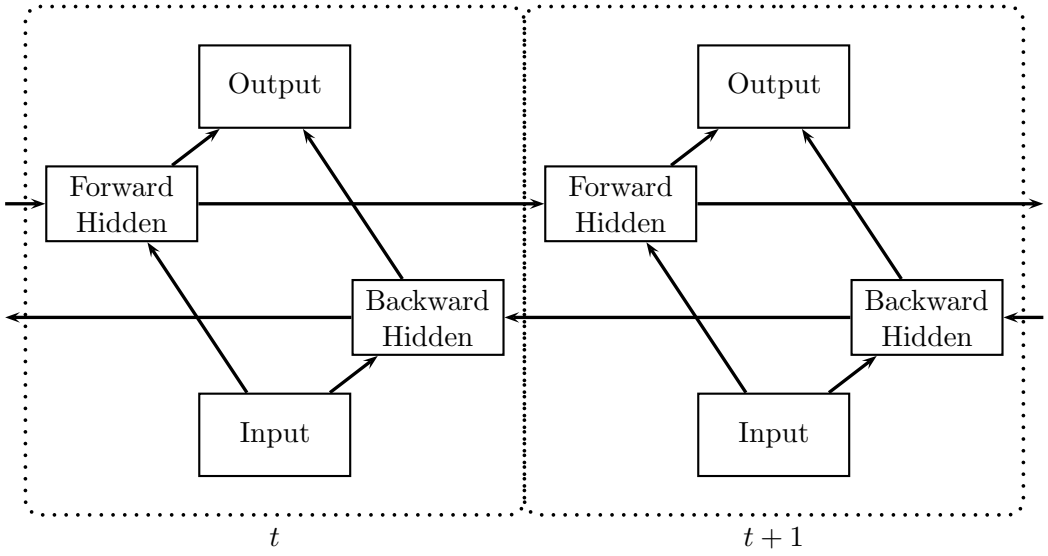


Figure 3.6: Two separate hidden layers scan the input sequence in opposite directions and are connected to the same output layer.

information becomes:

$$p(s_t|x) = p(s_t|x_1, \ldots, x_t, \ldots, x_{t+\Delta t}) \tag{3.32}$$

While this approach was successfully applied to phoneme recognition in ASR, the additional parameter is task dependent and needs to be tuned empirically.

In order to use all available information of the observation sequence, it is possible to use two separate networks scanning the sequence in opposite directions. In this case a *forward RNN* is applied to the observation sequence in its original direction and a *backward RNN* is applied to the same observation sequence in reversed order. Each of the RNNs delays the output labels using the method described above [Robinson 94b]. However, a strategy is required to merge the resulting estimates.

Bidirectional Recurrent Neural Networks

A more elegant approach to provide contextual information about the full sequence is the *bidirectional recurrent neural network* (BRNN) [Schuster & Paliwal 97].

The units in the recurrent hidden layer are splitted into a part responsible for the temporal dynamics of the observation sequence in forward direction (*forward states*) and a part responsible for the temporal dynamics in backward direction (*backward states*). The forward and backward states are completely independent of each other. No connections starting in a forward state ends in a backward state and vice versa, while both hidden layers are connected to the same output layer. Figure

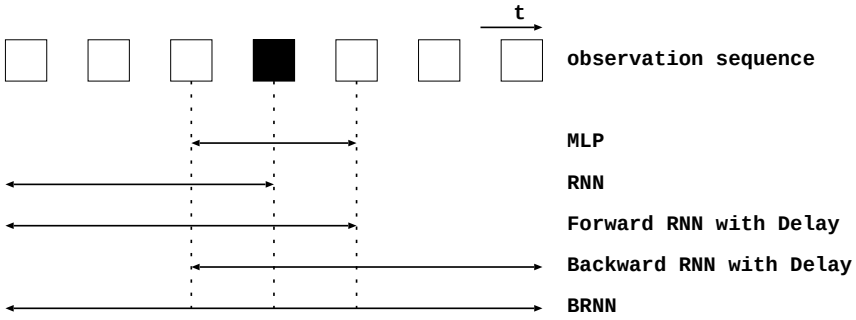


Figure 3.7: Illustration of contextual information used by different types of ANNs.

3.6 visualizes this RNN structure. Using the complete contextual information leads to following equation calculating the posterior estimate of the label of state s_t in the HMM given an observation sequence x:

$$p(s_t|x) = p(s_t|x_1, \ldots, x_t, \ldots, x_T) \tag{3.33}$$

The forward pass of BRNN hidden layers is the same as in the unidirectional case except that the observation sequence is presented in its original and reversed order to the hidden layers containing the forward and backward states respectively. The output layer is not updated until both hidden layers have processed the entire observation sequence. The backward pass needs to be performed for the complete observation sequence in the output layer to obtain the $\delta_k(t)$ values for each output unit k at each time step t, which then can be fed back to the separate hidden layers in opposite directions. The training procedure for RNNs is described in the next section.

Although most of the approaches treating contextual information discussed above show a good performance in practice, the BRNN outperforms the other techniques in most cases and is used in all experiments for the final results of this thesis. See Figure 3.7 for a summary of the different ways to handle contextual information in ANNs.

3.3.2 Training

As depicted in Figure 3.5 an RNN can be unfolded in time. For any observation sequence of finite length T this results in a structure that has only feed-forward connections with some connections reaching towards the output layer of the current time step t and some connections reaching *through time* towards the last time step T. This can be noticed in the forward pass shown in Equation 3.30, where the recurrent units collect their input just as in the feed-forward case but additionally take their activations of the last time step into account. The activation functions

applicable to RNNs are the same as for MLPs and the same error criterion can be used in the output layer. The partial derivatives w.r.t. the output units are calculated as described in Section 3.2.2. Therefore the error signals $\delta_k(t)$ for each output unit k at each time step t are assumed to be given and this section directly starts finding the formulas to obtain the derivatives w.r.t. the network weights $\alpha_j^{(l)}$ of a unit j in the recurrent hidden layer l.

To calculate these derivatives efficiently, two algorithms were devised with different properties concerning their complexity in time and space. The *backpropagation through time* (BPTT) algorithm [Werbos 02, Robinson 94a] is the direct extension of error backpropagation for RNNs and has a good runtime complexity, while the *real time recurrent learning* (RTRL) algorithm [Robinson & Fallside 87] is able to calculate the error gradient after each time step and therefore it is beneficial in complexity concerning memory [Williams & Zipser 89]. Due to its relation to the standard error backpropagation procedure this section will focus on BPTT. See Appendix 7.2 for the equations using RTRL.

Similar to backpropagation in MLPs, BPTT is performed by a repeated application of the chain rule. Considering a unit j of the last hidden layer $L-1$, which is connected to other units in this layer and to the output layer, the derivative $\delta_j^{(L-1)}(t)$ at time step t with activation function $f_j(z_j^{(L-1)}(t))$ for the net input $z_j^{(L-1)}(t)$ is:

$$\delta_j^{(L-1)}(t) = f_j'(z_j^{(L-1)}(t))\Big(\sum_k \delta_k^{(L)}(t)\alpha_{ij}^{(L)} + \sum_i \delta_i^{(L-1)}(t+1)\alpha_{ij}^{(L-1)}\Big) \tag{3.34}$$

Then, the derivatives for any previous layer l can be calculated recursively:

$$\delta_j^{(l)}(t) = f_j'(z_j^{(l)}(t))\Big(\sum_i \delta_i^{(l+1)}(t)\alpha_{ij}^{(l+1)} + \sum_i \delta_i^{(l)}(t+1)\alpha_{ij}^{(l)}\Big) \tag{3.35}$$

The $\delta_j^{(l)}(t)$ terms are calculated backwards in time starting at the last time step $t = T$, with the starting condition $\delta_j^{(l)}(T+1) = 0\ \forall j$.

The time-invariant derivatives w.r.t. the network weights are obtained by accumulating the time dependent $\delta_j^{(l)}(t)$ terms over the complete sequence:

$$\frac{\partial E_n}{\partial \alpha_{ij}^{(l)}} = \sum_{t=1}^{T} \frac{\partial E_n}{\partial z_j^{(l)}(t)} \frac{\partial z_j^{(l)}(t)}{\partial \alpha_{ij}^{(l)}} = \sum_{t=1}^{T} \delta_i^{(l)}(t) y_j^{(l-1)}(t) \tag{3.36}$$

The weight update rule is the same as for MLPs and is usually applied with a momentum term as described in Section 3.2.2.

Compared to RTRL, BPTT is very efficient with a runtime complexity of $O(n^2)$ per time step using n fully connected units and an observation sequence of length

T. However, as long as the length of the observation sequence is unknown the required memory is also unbounded with a complexity of $O(Tn + n^2)$. According to [Schmidhuber 89], a recurrent neural network algorithm is *local in space* if the update complexity per time step and weight does not depend on the size of the network. Further, an algorithm is *local in time* if its memory requirements do not depend on the length of the input sequence. Using these characterizations, BPTT is local in space but not in time. If the correlation between the observation sequence and the error signals are known not to exceed a delay of more than τ time steps, a way to overcome this drawback is to send those error signals not farther than τ time steps backwards in time [Williams & Peng 90], leading to an algorithm called *truncated* BPTT.

In the RNN software used in this thesis both the BPTT and RTRL algorithm can be used for training. All experiments in the result section are performed using the BPTT algorithm.

3.3.3 Long-Short-Term Memory

Short-term dependencies are given if there is no important correlation between the target at the output layer and the observed input for large time windows. These dependencies can be learned with any gradient descend based method. Learning long-term dependencies with an RNN using gradient descent based methods like BTPP or RTRL requires the error signals $\delta_j^{(l)}(t)$ for a unit j in layer l at time step t to remain stable while they are propagated back in time. However, without severe restrictions on the activation functions it is hard for gradient based methods to converge to a fixed point in the weight space. The following section will sketch this problem. Then, as a solution, the Long-Short-Term-Memory RNN is introduced and described.

Vanishing Gradient Problem

Assuming a fully connected RNN with n units discards the layerwise network structure and according to Equation 3.35 the error signal is given by:

$$\delta_j(t) = f_j'(z_j(t))\Big(\sum_i \delta_i(t+1)\alpha_{ij}\Big) \tag{3.37}$$

Propagating back an error signal from unit u at time step t to another unit v at time step $t-q$ leads to following scaling of the error [Hochreiter 98]:

$$\frac{\partial \delta_v(t-1)}{\partial \delta_u(t)} = \begin{cases} f_v'(z_v(t-1))\alpha_{vu} & q = 1 \\ f_v'(z_v(t-q))\sum_{l=1}^{n} \frac{\partial \delta_v(t-q+1)}{\partial \delta_u(t)}\alpha_{vl} & q > 1 \end{cases} \tag{3.38}$$

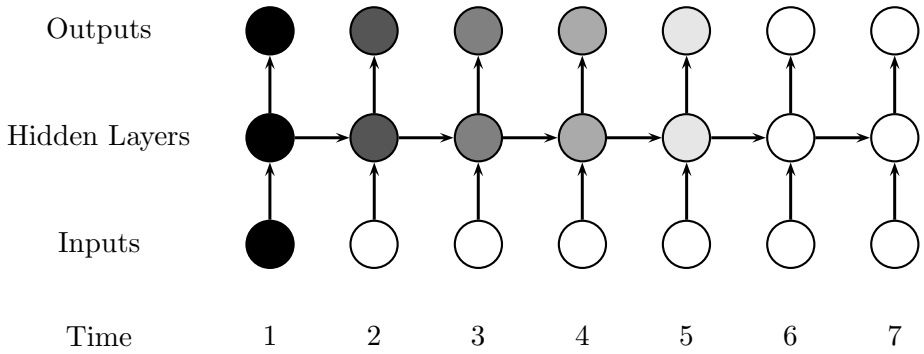


Figure 3.8: Gradient vanishes rapidly over time [Graves 09].

In the case of $q = 1$ all terms in Equation 3.37 that do not depend on u vanish. For a time lag $q > 1$ this result is applied recursively, so the total error scaling with $l_q = v$ and $l_0 = u$ is:

$$\frac{\partial \delta_v(t-1)}{\partial \delta_u(t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1}=1}^{n} \prod_{m=1}^{q} f'_{l_m}(z_{l_m}) \alpha_{l_m l_{m-1}} \tag{3.39}$$

Let $\rho(m, l_m, l_{m+1}) = |f'_{l_m}(z_{l_m}(t-m))\alpha_{l_{m-1}l_m}|$ of Equation 3.39 be the error back flow. Note that the terms may have different signs, so a larger number of units increases the absolute value of the error back flow only in expectation. If $\rho(m, l_m, l_{m+1}) < 1.0$ for all m the largest product of Equation 3.39 decreases exponentially with the time lag q and the error back flow vanishes. On the other hand, if $\rho(m, l_m, l_{m+1}) > 1.0$ for all m the largest product of Equation 3.39 increases exponentially and the error back flow blows up.

Using logistic sigmoids as activation function, f_{l_m} leads to a maximal value in the derivation f'_{l_m} of 0.25. For a non-zero constant c_{l_m} at time step l_m the derivative $|f'_{l_m}(z_{l_m})|$ takes on maximal values where

$$\alpha_{l_m l_{m-1}} = \frac{\coth(\frac{1}{2} z_{l_m})}{c_{l_m}} \tag{3.40}$$

goes to zero for $|\alpha_{l_m l_{m-1}}| \to \infty$ and is less than 1.0 for $|\alpha_{l_m l_{m-1}}| < 4.0$, so the error back flow vanishes [Hochreiter & Schmidhuber 97]. Increasing the learning rate also leads to no benefit, because inputs with a less distance in time still have a greater influence on the output of the current time step.

This property of the error gradient and the inability of BPTT to handle it was recognized by [Bengio & Simard$^+$ 94] and is referred to as *vanishing gradient problem*. A detailed analysis can be found in Sepp Hochreiter's diploma thesis (1991). See Figure 3.8 for an illustration.

A possibility to overcome the vanishing gradient problem is to use non-gradient based training algorithms like simulated annealing [Kirkpatrick & Gelatt$^+$ 83] or discrete error backpropagation [Bengio & Simard$^+$ 94]. However, gradient descend methods show a superior performance in practice. In order to stay in a gradient descend framework the system has to learn selectively, i.e. it has to focus on *relevant* inputs in time. This idea lead to the method of *history compression* [Schmidhuber 92], where redundant information is detected and removed and learning is focused on *unexpected* inputs by successively training multiple RNNs in a hierarchy and to collapse it into a single network. Although defining a proper hierarchy is often problematic and the algorithm remained unattended in practice, the idea of introducing a selective behavior w.r.t. the inputs is also the basis for the LSTM.

Gating Units

In an LSTM network the hidden units are replaced by *memory cells*. The idea of these cells is to protect the error back flow by multiplicative gating units, which allow the network to store and access information selectively. In order to enforce a constant error back flow for potentially infinite time lags w.r.t. the recurrent connections, it follows from the last section that

$$f_j'(z_j(t))\alpha_{jj} = 1.0 \tag{3.41}$$

The integration of Equation 3.41 reveals that the activation function has to be linear. Constant error back flow for non-recurrent connections is achieved by the gating units that scale the signal flowing in and out of this self-connected recurrent unit called *cell state*. The complex of a memory cell consisting of a cell state with its gating units is depicted in Figure 3.9. The input gate collects inputs from other memory cells and the internal cell state to calculate a unit activation which is used to scale the net input arriving at the memory cell with a specific activation function g. Therefore, the network can learn to ignore incoming information by scaling it to zero. Likewise, the output gate scales the information leaving the cell by an activation function h to protect other cells from irrelevant input. The central element of a memory cell is the *constant error carousel* (CEC), a self-connected unit with a fixed weight of 1.0 which is further protected by the forget gate, which allows the memory cell to reset its internal state.

Error signals inside the CEC cannot vanish or blow up due to the linearity of the activation function. However, the error signal flowing back through the output gate can superimpose the cell state, so the output gate has to learn which errors to trap in the CEC. Further, the input gate has to learn when to release errors. If the sequences are long, the internal cell state activation may grow indefinitely

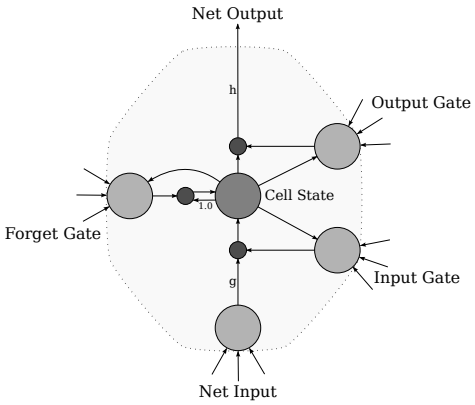


Figure 3.9: LSTM memory cell with an input, output and forget gate. The gating units and the net input unit receive incoming weights from outside the cell and the cell state. The output of these units is scaled by multiplicative units shown as small dark circles here [Hochreiter & Schmidhuber 97].

[Gers & Schmidhuber$^+$ 00], so it is required to reset it which is achieved by the forget gate.

LSTM networks have been successfully applied to many theoretical tasks where long time lags are present and that could not be solved by traditional RNNs [Hochreiter & Schmidhuber 97]. Additionally, LSTM networks have been applied to real-world tasks like speech and handwriting recognition [Graves & Schmidhuber 05, Graves & Bunke$^+$ 08]. All RNN experiments in this thesis are performed using the LSTM structure.

Training

In its first formulation the training was performed with a truncated version of BPTT, where the BPTT part was truncated after one time step. The motivation was that the time dependencies would be dealt by the recurrent connection in the memory cells and not by the surrounding flow of activation [Graves 09]. Initially it could only be proven for the truncated BPTT version that the error flow remains constant [Hochreiter & Schmidhuber 97]. On the other hand the exact error gradient is more accurate than the truncated gradient and can be checked numerically. For this reason this section describes the forward and backward pass of the exact gradient.

The forward pass is similar to traditional RNNs, i.e. the update equations are recursively applied for the full sequence of length T starting at time step $t = 1$. In the backward pass the derivatives are calculated starting at $t = T$ and propagated

back in time to the start of the sequence. Let the input, output and forget gate activation in layer l in memory cell j at time step t be denoted by $I_j^{(l)}(t)$, $O_j^{(l)}(t)$ and $F_j^{(l)}(t)$ respectively. Further, let $c_j^{(l)}(t)$ be the cell state activation and $y_j^{(l)}(t)$ be the output of the memory cell. The forward pass corresponds to calculating the activations in following order:

Input Gate: $$I_j^{(l)}(t) = \sum_i \alpha_{I_j i}^{(l-1)} y_i^{(l-1)}(t) + \sum_i \alpha_{I_j i}^{(l)} y_i^{(l)}(t-1) + \alpha_{I_j c_j}^{(l)} c_j^{(l)}(t-1)$$

Forget Gate: $$F_j^{(l)}(t) = \sum_i \alpha_{F_j i}^{(l-1)} y_i^{(l-1)}(t) + \sum_i \alpha_{F_j i}^{(l)} y_i^{(l)}(t-1) + \alpha_{F_j c_j}^{(l)} c_j^{(l)}(t-1)$$

Cell State: $$z_{c_j}^{(l)} = \sum_i \alpha_{c_j i}^{(l-1)} y_i^{(l-1)}(t) + \sum_i \alpha_{c_j i}^{(l)} y_i^{(l)}(t-1)$$

$$c_j^{(l)}(t) = f(F_j^{(l)}(t)) c_j^{(l)}(t-1) + f(I_j^{(l)}(t)) g(z_{c_j}^{(l)})$$

Output Gate: $$O_j^{(l)}(t) = \sum_i \alpha_{O_j i}^{(l-1)} y_i^{(l-1)}(t) + \sum_i \alpha_{O_j i}^{(l)} y_i^{(l)}(t-1) + \alpha_{O_j c}^{(l)} c_j^{(l)}(t)$$

Cell Output: $$y_j^{(l)}(t) = f(O_j^{(l)}(t)) h(c_j^{(l)}(t))$$

In the backward pass the error back flow $\delta_{I_j}^{(l)}(t)$, $\delta_{O_j}^{(l)}(t)$, $\delta_{F_j}^{(l)}(t)$ for the input, output and forget gate respectively have to be computed. In order to simplify notation, let $\epsilon_j^{(l)}(t) = \frac{\partial E_n}{\partial y_j(t)}$ and $\epsilon_{c_j}^{(l)}(t) = \frac{\partial E_n}{\partial c_j^{(l)}(t)}$ for the local error E_n of sample n. As in the case for traditional RNNs, the partial derivatives of the output layer are computed as in the MLP case and therefore are assumed to be given. The error back flow is then computed in the following order:

Cell Output: $$\epsilon_j^{(l)}(t) = \sum_i \alpha_{kj} \delta_k^{(l+1)}(t) + \sum_i \alpha_{ij} \delta_i^{(l)}(t+1)$$

Output Gate: $$\delta_{O_j}^{(l)}(t) = f'(z_{O_j}^{(l)}(t)) h(c_j^{(l)}(t)) \epsilon_j^{(l)}(t)$$

Cell State:

$$\begin{aligned} \epsilon_{c_j}^{(l)}(t) &= y_{O_j}^{(l)}(t) h'(c_j^{(l)}(t)) \epsilon_{c_j}^{(l)}(t) + y_{F_j}^{(l)}(t+1) \epsilon_{c_j}^{(l)}(t+1) \\ &+ \alpha_{I_j c_j} \delta_I^{(l)}(t+1) + \alpha_{F_j c_j} \delta_{F_j}^{(l)}(t+1) + \alpha_{O_j c_j} \delta_{O_j}^{(l)}(t) \end{aligned}$$

$$\delta_{c_j}^{(l)}(t) = y_{I_j}^{(l)}(t) g'(z_{c_j}^{(l)}) \epsilon_{c_j}^{(l)}(t)$$

Forget Gate: $$\delta_{F_j}^{(l)}(t) = f'(z_{F_j}^{(l)}(t)) c_j^{(l)}(t-1) \epsilon_{c_j}^{(l)}(t)$$

Input Gate: $$\delta_{I_j}^{(l)}(t) = f'(y_{I_j}^{(l)}(t)) g(y_{c_j}^{(l)}(t)) \epsilon_{c_j}^{(l)}(t)$$

3.3.4 Connectionist Temporal Classification

The output sequence of any RNN mapping has always the length of the input sequences. This has two major consequences. First, a segmentation of the training data has to be given, which is unknown. Second, the transformation of the output posterior stream to the most likely word hypothesis has to be estimated by an external decoding step. Both steps can be performed by HMMs. However, Chapter 2 showed that several dependency assumption are made when HMMs are used and that a naturally non-discriminative training is performed.

Motivated by the inability of any ANN to perform the segmentation and decoding step, [Graves 09] invented the Connectionist Temporal classification (CTC) layer, which is a specific output layer suitable for any kind of RNN. It allows the RNN to make label predictions at any point in the observation sequence as long as the complete output sequence remains correct. This property removes the requirement of an alignment and the localized framewise probabilities are replaced by probability estimations concerning the complete label sequence. Only the mapping from the label sequence to the word with highest probability using a dictionary or language model still has to be performed externally.

This section begins with a description how the mapping from the posterior sequence to a label sequence is performed by CTC and briefly sketches the training and decoding procedure of CTC networks. A detailed analysis can be found in Alex Graves dissertation [Graves 09].

Sequence Mapping

The central element of a CTC layer is a softmax output layer containing a unit for each possible label from a character lexicon L and an additional unit for the *blank* label, i.e. this unit is trained to have maximal activation if no label was observed. For $L' = L \cup \{blank\}$ and an observation sequence x_1^T of length T the probability of an output sequence $\pi \in {L'}^T$ for conditionally independent labels is given by

$$p(\pi|x_1^T) = \prod_{t=1}^{T} y_{\pi_t}^{(L)}(t) \tag{3.42}$$

Although CTC works together with unidirectional RNNs, Equation 3.42 is conditioned on the entire observation sequence and therefore bidirectional RNNs are preferred.

Considering the fact that the length of the output label sequence is always smaller or equal to the length of the input label sequence, the CTC layer has to perform a *many-to-one* map $\mathcal{B} : {L'}^T \mapsto L^{\leq T}$. This is done in two steps:

1. Remove repeated labels
2. Remove blanks

This leads to many possible labelings π that result in the same output sequence, like e.g.

$$\mathcal{B}(a - bb - bc-) = \mathcal{B}(a - b - bc-) = abbc$$

where $-$ denotes the blank symbol. The posterior probability of any output labeling $l \in L^{\leq T}$ then corresponds to the marginalization over all these possible labelings $\pi \in \mathcal{B}^{-1}(l)$:

$$p(l|x_1^T) = \sum_{\pi \in \mathcal{B}^{-1}(l)} p(\pi|x_1^T) \tag{3.43}$$

Through this marginalization no alignment is required, because no mapping from input observations x_t to labels $y_k^{(L)}(t)$ is considered anymore.

Training

The sum over all labelings in Equation 3.43 obviously leads to a combinatorial explosion, so it is infeasible to solve it in a trivial way. Therefore, a dynamic programming procedure is applied. This makes the training algorithm very similar to the general training procedure of HMMs. The idea is that iteratively picking the best label results in an optimal labeling as long as the prefix of that labeling is optimal. This leads to the *Forward-Backward* algorithm, which was originally introduced by [Rabiner 89] for HMMs.

For a labeling l, the *forward variable* $\alpha_t(s)$ is defined as the marginalization over all length t labeling prefixes that are mapped to a prefix of l with length $\frac{1}{2}s$ using mapping $\mathcal{B}$:

$$\alpha_t(s) = \sum_{\pi: B(\pi_1^t) = l_1^{\lfloor s/2 \rfloor}} \prod_{t'=1}^{T} y_{\pi_{t'}}^{(L)} \tag{3.44}$$

The *backward variables* $\beta_t(s)$ correspondingly are defined as the marginalization of all labelings suffixes starting at time step t that are mapped by $\mathcal{B}$ onto the suffix of l starting at label $\frac{1}{2}s$:

$$\beta_t(s) = \sum_{\pi: \mathcal{B}(\pi_t^T) = l_{s/2}^{|l|}} \prod_{t'=t+1}^{T} y_{\pi_{t'}}^{(L)}(t') \tag{3.45}$$

The forward and backward variables can be calculated recursively [Graves 09]. Given the values for $\alpha_t(s)$ and $\beta_t(s)$ for all t and s the objective function for the CTC layer can be defined using the principle of maximum likelihood. For a training observation sequence x_1^T with the corresponding transcriptions w_1^N, let ${w'}_1^N$ be the transcription with blanks placed between all symbols and at the begin and the end of the transcription. The maximum likelihood criterion was introduced in Section 2.2, while the parameters $\hat{\theta}$ now correspond to the network weights:

$$-\log L(\hat{\theta}) \quad = \quad \log p(x_1^T | w_1^N, \hat{\theta}) \tag{3.46}$$

Differentiating Equation 3.46 w.r.t. the output activation of unit k in the softmax output layer leads to

$$-\frac{\partial \log L(\hat{\theta})}{\partial y_k^{(L)}(t)} = -\frac{1}{p(w_1^N | x_1^T)} \frac{\partial p(w_1^N | x_1^T)}{\partial y_k^{(L)}(t)} \tag{3.47}$$

Defining the set $lab(w_1^N, k) = \{s : {w'}_1^N = k\}$ it turns out that the partial derivative of the posterior probability $p(w_1^N | x_1^T)$ w.r.t. the output activations of the softmax layer is represented by the forward and backward variables [Graves 09]:

$$\frac{\partial p(w_1^N | x_1^T)}{\partial y_k^{(L)}(t)} = \frac{1}{y_k^{(L)}(t)} \sum_{s \in lab(w_1^N, k)} \alpha_t(s) \beta_t(s) \tag{3.48}$$

Applying the chain rule for partial derivatives and substituting the partial derivation of the softmax layer w.r.t. the net input showed in Equation 3.19 finally results in

$$-\frac{\partial \log L(\hat{\theta})}{\partial y_k^{(L)}(t)} = y_k^{(L)}(t) - \frac{1}{p(w_1^N | x_1^T)} \sum_{s \in lab(w_1^N, k)} \alpha_t(s) \beta_t(s) \tag{3.49}$$

After calculating the gradient given in Equation 3.49 the normal BPTT algorithm can be applied.

Decoding

Finding the most probable labeling for a new observation sequence is a hard problem and as for HMMs no tractable exact decoding algorithm is known. One simple approach is to apply the viterbi approximation, i.e. to consider only the labeling with maximal probability by concatenating the labels of the output units with maximal activation at each time step.

A better approximation is a best-first search through the prefix tree of labelings, leading to an algorithm called *Prefix Search Decoding*. It is out of the scope of this thesis, but a description can be found in [Graves 09].

3.3.5 RNN Software

In this thesis, all RNN based experiments were performed with the RNNLIB[2] software. It was originally developed by Alex Graves in association with his dissertation. The software supports traditional RNNs with self-connected units and the LSTM structure. As part of this thesis the implementation was extended by following features:

Parallel Training Two different parallelization techniques were implemented. The first method is based on the RPROP batch learning training procedure. In every epoch the forwarding of all training segments is distributed on an arbitrary number of machines. The number of parallel jobs is only limited by the number of segments contained in the training set. Each machine writes the derivatives w.r.t. the weights in a file. Afterwards, these files are combined on a single machine in order to perform the weight update.

The other method uses the multicore architecture of modern machines in order to perform a local parallelization. It is not restricted to batch learning. Instead, it forwards several segments referred to as *mini-batches* in parallel and recombines the derivatives to perform a weight update afterwards. As long as the size of the mini-batches is small compared to the number of segments in the training set, the quality of the weight estimation is not affected by this method. All experiments conducted in the result section are performed with this method using a mini-batch of size four, leading to $\frac{N}{4}$ weight updates per epoch for a training set of size N.

Soft Labels In order to use a posterior distribution over the labels instead of a hard target, the integration of those posteriors was implemented into the RNNLIB software. The extension requires to use the cross-entropy error criterion and the BPTT training algorithm. The generation of soft labels will be explained in Section 4.3.2.

Feature Extraction While the original implementation provides only access to the activations of the output layer units, experiments have been conducted using unit activations of an inner hidden layer. Therefore, the extraction of hidden layer unit activations has been implemented in the RNNLIB software. This method is described in Section 4.2. Further, the activations extracted from the output layer

[2] http://rnnl.sourceforge.net

or an hidden layer can be transformed by the natural logarithm and are written in an XML syntax, which can directly be parsed by the HMM software used in this thesis.

4 Model Combination

While the CTC output layer of RNNs provides an elegant way to apply neural networks directly to sequence learning tasks like handwriting recognition, its central components rely on the same algorithms used in HMMs. For this reason, there is no explicit argument for a superior performance of CTC networks, while HMMs are a well studied approach and provide several possibilities to make use of the ANN outputs. As already mentioned in the previous chapters, HMMs are used in two phases of the training procedure. First, they perform the segmentation of the data that is required to obtain a mapping from frames to labels used in the training of the ANN. Secondly, they are used as decoder for the output activations of the ANN, i.e. the HMM performs the mapping from output activations to the final word sequence. This alternating procedure is depicted in Figure 4.1.

Many other attempts were made in order to introduce the discriminative training of ANNs into the modeling capability of HMMs. Some early approaches tried to emulate HMMs with specific ANN structures like e.g. the *Alphanets* [Bridle 90]. Although theoretical interesting, these approaches did not allow to overcome the limitations of HMMs and therefore were not able to outperform them in practice. [Bengio & Mori$^+$ 91] used a global optimization criterion for both models by training the ANN to perform a feature transformation that is more suitable for the state-distribution of the HMM. While it seems beneficial to formulate an analytically motivated training scheme without alternating the optimization process of both models, in this case the ANN does not act as classifier but as regression model. Due to the approximation ability of ANNs, their application in preprocessing the features seems to be natural and recently such an approach lead to significant

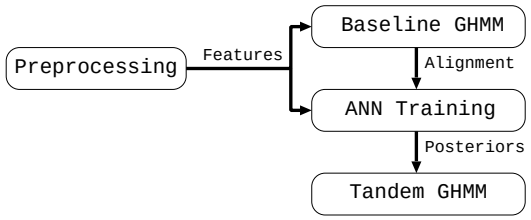


Figure 4.1: HMMs are used to train a baseline system to generate a labeling used in the ANN training. The posterior estimates are finally decoded by an HMM.

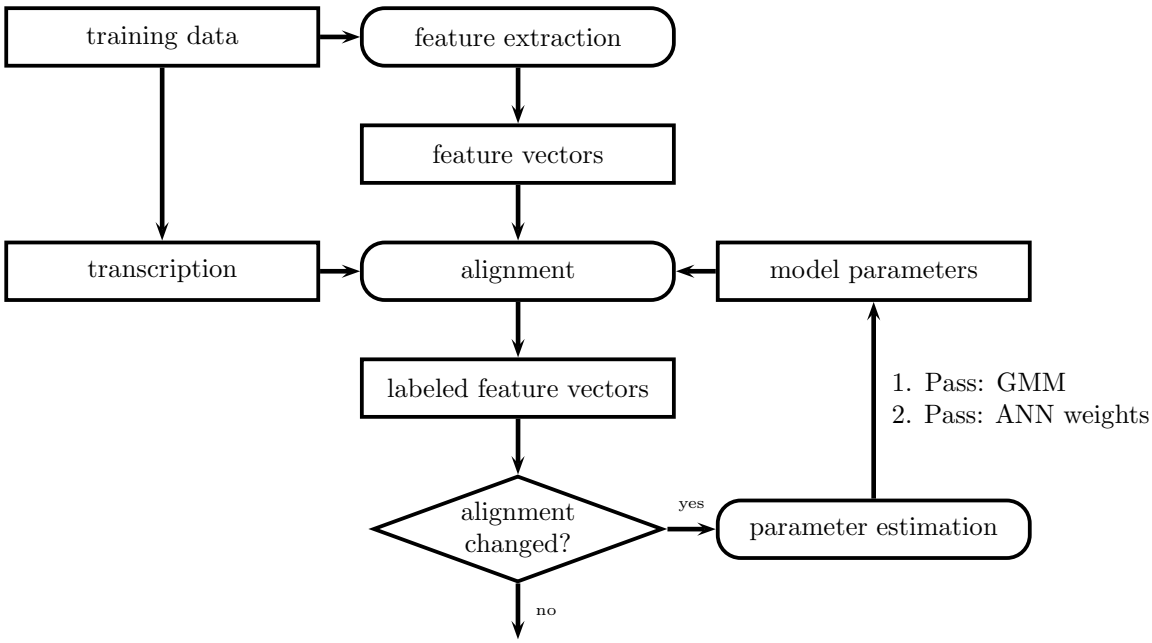


Figure 4.2: Illustration of the hybrid recognition approach. First, the original features extracted from the image data are used to obtain an initial alignment used in the ANN training procedure. Afterwards, the posterior estimates are used as scaled emission probabilities in the already trained HMM.

improvements in handwriting recognition [Espana-Boquera & Castro-Bleda$^+$ 11]. Without a global gradient provided by an HMMs the criterion has to be defined externally and can usually not be derived from the given transcription of the training data.

In this thesis two approaches were used that rely on an alternating optimization of the ANN as frame-wise classifier and the HMM as decoder of the ANN classifications. In the following sections these two approaches are described and their theoretical limitations are discussed.

4.1 Hybrid Approach

Introduced by [Bourlard & Morgan 93], the hybrid approach tries to model the observation dependency of a given HMM directly using ANNs. The observation de-

pendent components of an HMM are the state emission probabilities. Since HMMs are a generative approach, these components model the probability of observing the input vector x given the label at the current state s. On the other hand, ANNs are a discriminative approach and model the posterior probability of observing a particular label at state s given the input vector x. The connection between these probabilities is given by the Bayes theorem:

$$p(x|s) = \frac{p(s|x) \cdot p(x)}{p(s)} \tag{4.1}$$

The state prior $p(s)$ can be estimated by the relative frequency of assigned observations in the HMM, but the observation prior $p(x)$ in the numerator of Equation 4.1 cannot be derived from the model and therefore is discarded. This leads to scaled likelihoods that are not normalized. An additional exponential term κ is further introduced to control the sharpness of the estimates. This leads to

$$p(x|s) \approx \frac{p(s|x)}{p(s)^{\kappa}} \tag{4.2}$$

It is worth mentioning that the same effect of the κ term can be obtained by adjusting the TDPs of the HMM. [Bourlard & Wellekens 90] have shown that the state posteriors estimated by ANNs could directly be used in discriminant HMMs, which leads to the question why it is required to scale the posteriors by the inverse state prior. [Bourlard & Morgan 93] indeed measured a better performance on frame level with state posteriors but a huge degradation of the performance on word level. An explanation for this effect is that the prior probabilities imposed by the topology of the HMM do not match the prior probabilities of the training data.

Figure 4.2 illustrates the overall training procedure. The benefits of the hybrid approach are a simple implementation and that an already trained HMM can be used. Assuming an alignment to be given by some other instance and the transition probabilities of the HMM to be defined as fixed TDPs, no HMM training is required at all because there is no Gaussian estimation. An obvious drawback are the scaled likelihoods. Further, the HMM uses the probabilities estimated by the ANN directly and is therefore sensitive to poor estimations for particular frames. Nevertheless hybrid HMM/ANN systems achieve large improvements in practice on character level [Espana-Boquera & Castro-Bleda^{+} 11] and especially on context-dependent states of Gaussian HMMs [Seide & Gang^{+} 11, Sainath & Kingsbury^{+} 11].

4.2 Tandem Approach

In order to reduce the sensitivity to poor posterior estimations on frame level, [Hermansky & Ellis^{+} 00] introduced a tandem approach to use ANN outputs in

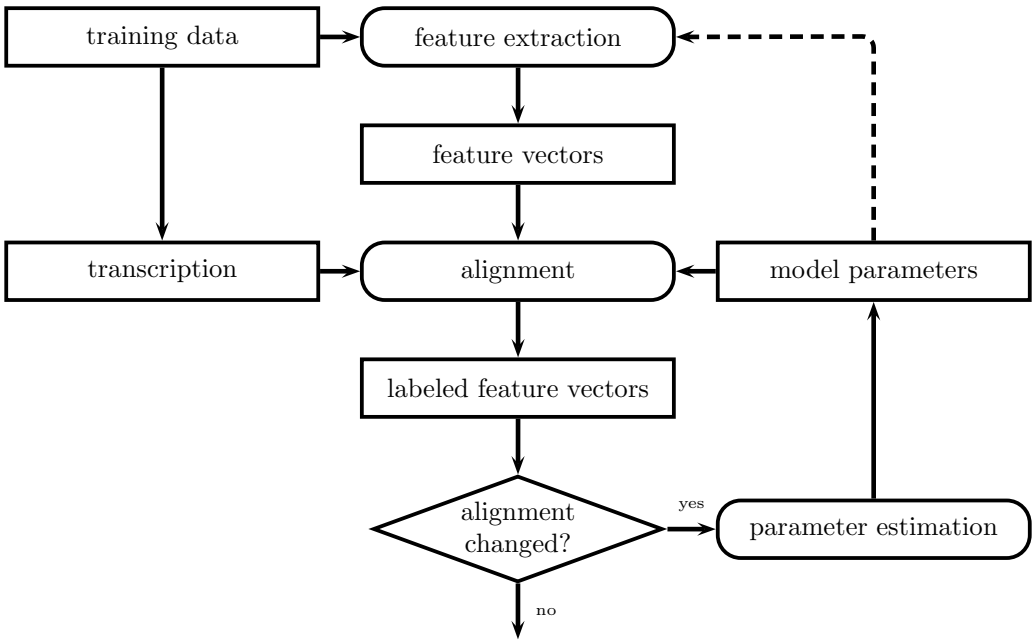


Figure 4.3: Illustration of the tandem approach. In the first pass the original features extracted from the image data are used to obtain an initial alignment used in the ANN training procedure. After the first pass the posterior estimates of the ANN are used as feature stream in order to train a new GMM based system.

HMMs. In this case the output activations of the neural network are considered as new feature-vector for each frame. A completely new HMM is trained on this feature stream. These gaussianized output activations account for a larger variability and enable the HMM to generalize on the data making it less sensitive to outliers or falsely predicted single frames. The training procedure of the tandem approach is depicted in Figure 4.3.

The tandem approach leads to a variety of possibilities the hybrid approach can not provide. While the hybrid approach requires the estimations of the ANN to be state posteriors used as scaled emission probabilities, the only condition on the output activations of the neural network in the tandem approach is that they represent the characteristics of the data w.r.t. the state labels. These characteristics are not only given by the posterior estimates but are encoded in the complete activation map of all hidden and output units in the ANN. Therefore, instead of

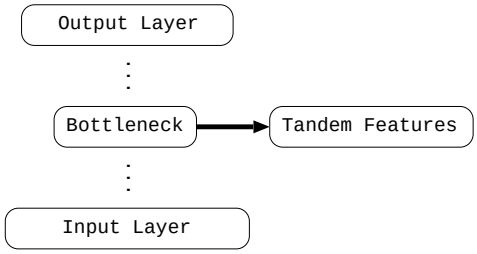


Figure 4.4: Instead of the posterior estimates of the output layer, the activations of an inner hidden layer are used as tandem features. When the inner layer is small the features are referred to as bottleneck features.

the posterior estimates of the output layer, the activations of an inner hidden layer of the ANN can be taken as feature vector for each frame as well. This hidden layer is usually designed to have a small number of units in order to generate a *bottleneck* in the network [Plahl & Schlüter$^+$ 10]. Bottleneck features are the result of a non-linear feature transformation generated by an error criterion that maximizes the correlation of the input features to the output labels of the classification task. Figure 4.4 illustrates this kind of features.

Besides the desired correlation between the labels and the input features, the activations of the ANN are highly correlated to themselves. This correlation should be discarded in order to create a compact feature set. A common way to do this in an unsupervised framework is the *principal component analysis* (PCA) [Dunteman 89]. It performs a transformation of the data distribution into a vector space with diagonal covariance matrix and afterwards reduces the dimensionality by ignoring the components with minimal variance. Especially the diagonalization of the covariance matrix is important due to the diagonal covariance matrix used in the GHMM. Another approach for dimensionality reduction is the *linear discriminant analysis* (LDA) [Mclachlan 04]. The LDA takes the class labels of the samples into account and calculates a mapping of the samples such that the distances between the samples belonging to the same class remains constant while the distances between the samples and the class centers of other classes are maximized. Similar to the PCA estimation this is accomplished by an eigenvalue decomposition. After calculating the eigenvectors the principal axis build the new coordinate system. The dimensionality is reduced based on the magnitude of the corresponding eigenvalues.

Benefits of the tandem approach include the large variation of possibilities of how to use the ANN outputs. Further, it is able to overcome improper estimations for particular frames of the ANN through the generalization of the GMM. They also allow to perform a feature combination and the final HMM can be discriminatively trained to obtain a further improvement. One drawback is that they require a full

HMM training after extracting the ANN outputs and the decorrelation and feature reduction introduce additional parameters which have to be tuned empirically. However, in practice tandem systems seem to be superior to the hybrid approach when applied to handwriting recognition [Dreuw & Doetsch$^+$ 11]. This observation is confirmed by the experiments of this thesis.

4.3 Extensions

4.3.1 Realignment

Both, the hybrid and tandem approach are applied in a two pass scheme: After the alignment is provided by an HMM the ANN system is trained and the decoding is again performed by an HMM. After training it is possible to create a new training alignment and to repeat the process. This procedure is referred to as *Realignment* and was implemented as part of this thesis. Unfortunately it turns out that the quality of the initial alignment has a large influence on the success of the realignment procedure. While an ANN system based on a linear alignment of poor quality can be significantly improved by the realignment procedure, a good starting alignment cannot be outperformed by the realignment procedure and may even lead to an increase of the word error rate. Further details can be found in Section 5.3.2 and possible reasons are given in the conclusions.

4.3.2 Soft Labels

Another extension is to replace the alignment containing only a single label at each frame by a posterior distribution over all labels. These labels are referred to as *soft labels*. Since ANNs always act as function approximator this distribution can easily be integrated in the training criterion as shown in Section 3.1.1. The implementation into the RNN software was done as part of this thesis. To obtain soft labels from the HMM training a lattice is generated for the training set. At each point in the lattice where a label is chosen a normalized confidence measure is extracted w.r.t. all competing hypotheses of choosing another label [Gollan & Bacchiani 08]. This confidence can be used to define a posterior probability of the selected label and a uniform distribution of the remaining probability mass is used for all other labels. This approach is therefore very restricted because only a first-best confidence is given and the result section shows that no significant difference in performance is observed when this kind of soft labels are used.

5 Experimental Results

5.1 Corpora

In order to measure the effect of the neural network based features described in the previous chapters, experiments were conducted on corpora for English, French and Arabic. The English corpus was an open vocabulary text line recognition task, while the Arabic and French corpora were used for isolated word recognition. This section briefly summarizes the corpus statistics of each task. Additional information can be found in Appendix 7.3.

5.1.1 IAM

The IAM database [Liwicki & Bunke 05] is an English continuous sentence recognition task with an open-vocabulary lexicon. It consists of $9,862$ text lines with $5,685$ sentences on $1,539$ pages, where each word is build from 79 symbols consisting of upper- and lowercase characters, punctuation, quotation marks and a white-space model. The database is divided into a training set containing $6,161$, a validation set containing 920 and a test set containing $2,781$ text lines respectively. The vocabulary used in this work contains the 50k most common words of the LOB corpus [Jonas 09] resulting in an OOV rate of 4.01% on the validation set and 3.47% on the test set. The training set contains $4,309$ singletons. See Appendix 7.3.1 for detailed corpus statistics.

5.1.2 RIMES

The RIMES database [Augustin & Carré$^+$ 06] consists of $5,605$ fictional French letters by more than $1,300$ writers. Each word is build from 82 symbols containing upper- and lowercase characters, ligatures, typographical symbols, punctuations and a white-space model. In the presented experiments the training and validation corpus of the ICDAR 2011 competition for isolated word recognition were used with a closed vocabulary containing $5,340$ words. The validation corpus was used as test corpus in the ICDAR 2009 competition and therefore the official results of this competition were used for comparison in this work. The training corpus contains $51,738$ words and the validation corpus contains $7,464$ words. See Appendix 7.3.2 for further statistics.

5.1.3 IFN/ENIT

The IFN/ENIT database [Margner & Abed 10] contains 32,492 Arabic handwritten Tunisian town names by about 1,000 writers with a vocabulary size of 937. A whitespace character and position dependent length modeling of the 28 base characters leads to 121 different character labels [Dreuw & Jonas$^+$ 08]. The database is divided in five disjoint sets, where in this work the sets a-d were used for training and set e for testing. This setup results in 335 singletons. Fold-specific statistics can be found in Appendix 7.3.3.

5.2 Systems and Models

In order to discuss the influence of different types of preprocessing, the neural network based systems have been evaluated on two different feature sets for each corpus. These features are not only used to train the neural network model, but also to train the baseline Gaussian HMM (GHMM) which provides the first alignment required to train the ANN. This section describes the preprocessing steps and the modeling approach of each system.

5.2.1 IAM

Systems on two feature sets labeled *E-1* and *E-2* have been trained with different levels of preprocessing. *E-1* directly used raw slice features with a sliding window of size 7 which were reduced by PCA to 30 components [Dreuw & Doetsch$^+$ 11]. *E-2* used MLP-preprocessed images provided by Salvador España Boquera from the Department of Information Systems and Computing at the Polytechnic University of Valencia [Espana-Boquera & Castro-Bleda$^+$ 11] and were again reduced by PCA to 30 components with a window of size 7. Each character was modeled by a 10-state left-to-right HMM with five separate GMMs resulting in 391 mixtures with about 25k densities and a globally pooled diagonal covariance matrix. The experiments have been performed using a Kneser-Ney smoothed 5gram language model with an LM scaling factor of $\kappa = 30$ and a 50k lexicon.

The LSTM networks for both feature sets consisted of two hidden layers with 100 nodes in the first and 200 nodes in the second hidden layer. While other topologies lead to similar results, this topology turned out to be a reasonable general purpose model and was used in all databases in order to obtain a good comparability. The tandem system trained on *E-2* used the activations of the first hidden layer following the bottleneck approach described in Section 4.2, while the tandem system trained on *E-1* used the log-posterior estimates of the network. Afterwards, the features were reduced by PCA to 20 components with a sliding window of size 7.

For both feature types hierarchical MLPs were trained using a single hidden layer with 1500 nodes in the first network and a single hidden layer with 3000 nodes in the second network. The first network in the cascade was fed with temporal patterns as input as described in Section 3.2.1. The posterior estimates of this network were reduced by LDA to 40 dimensions and provided with a context window of 9 frames on the original features and 5 on the LDA-reduced posterior estimates to the second network leading to an input layer of size 344.

5.2.2 RIMES

The RIMES database was also evaluated on two feature sets with different cascades of preprocessing steps. These features are referred to as *F-1* and *F-2*. The feature set *F-1* used images with three levels of preprocessing. After performing a gray-value normalization, a median filter was applied to clean the images. Finally, a slant correction method as described in [Pastor & Toselli$^+$ 04] was used. Afterwards, the images were scaled to fixed height of 32 pixels and a sliding window of size 14 was applied to extract the features. These features were further reduced by PCA to 30 components. Each characters was modeled by a 12-state left-to-right HMM with six separate GMMs, resulting in 487 mixtures with about 31k densities and a globally pooled diagonal covariance matrix.

F-2 used the same preprocessing steps as *F-1*, but, instead of the simple image scaling, a size normalization procedure following the method described in [Juan & Toselli$^+$ 01] was applied, scaling the images to a fixed height of 48 pixels. A sliding window of size 18 was used to extract the features which were further reduced by PCA to 30 components. The left-to-right HMM for each character consisted of 30 states with 15 separate GMMs yielding $1,216$ mixtures with about 277k densities and a globally pooled diagonal covariance matrix. More details about the systems and the preprocessing can be found in [Pesch 11].

The hierarchical MLP networks on RIMES were build by a single hidden layer with 1000 nodes in the first network, while the hidden layer of the second network in the hierarchy consisted of 2000 nodes. Again, temporal patterns were extracted and fed into the first network. The resulting localized posteriors of the first network were reduced by LDA to 72 dimensions. A window of 5 frames of these localized posteriors together with a window of 9 frames of the original features lead to an input layer of size 630 in the second network.

As on the IAM database two hidden layers with 100 and 200 hidden nodes respectively were used in the LSTM network, but in contrast to IAM the tandem HMM system was trained on the the posterior estimates of the output layer. These posterior estimates were reduced by PCA to 72 components with a window of size 18.

5.2.3 IFN/ENIT

The two feature sets for IFN/ENIT are called *A-1* and *A-2* and differ in their preprocessing used for training. The feature set *A-1* consisted of the original images scaled to a height of 30 pixels without any preprocessing using raw slice features with a sliding window of size 9 that was applied from right to left. Each feature vector was further reduced by PCA to 35 components.

Feature set *A-2* was build by a preprocessing of the images suitable for Bernoulli based HMMs using the method described in [Gimenez & Khoury$^+$ 10], where the center of gravity (COG) of black pixels is calculated for a window which is repositioned such that the COG is in the center of that window. Again a sliding window of size 9 was applied and reduced by PCA to 35 components.

In both systems each character was modeled by a 12 state right-to-left HMM with six separate GMMs resulting in 721 mixtures with about 71k densities. As on the other databases a globally pooled diagonal covariance matrix was used.

On IFN/ENIT the hierarchical MLPs consisted of one hidden layer in each network with 750 hidden units in the first and 1500 hidden units in the second network using DCT-TRAP transformed features for the first network. The posteriors within a window of size 5 of the first network concatenated with a context window of 9 frames of the original features result in an input layer of size 750 in the second network.

Similar to the other databases, the LSTM networks used in both systems were build of two hidden layers with 100 nodes in the first and 200 nodes in the second hidden layer. The tandem system was trained on the posterior estimates and reduced by PCA to 64 components with a PCA window of size 9.

5.3 Experiments

For each system described above experiments were performed with the tandem and hybrid approach using hierarchical MLPs and LSTMs. Realignment was performed only on the system showing the best performance. On IAM and IFN/ENIT additional experiments were conducted analyzing the difference between different types of LSTMs and conventional RNNs. Further, experiments comparing the bidirectional LSTM structure to unidirectional LSTMs were performed and the realignment procedure was evaluated on initial alignments of different quality. For all experiments results are reported by the word error rate (WER) and character error rate (CER).

5.3.1 LSTM Analysis

Examining the requirement of a larger model complexity is crucial because simple models with similar performance should always be preferred. An MLP can be seen as the simplest connectionist model in this work. Considering conventional RNNs as extension to MLPs by recurrent connections, the LSTM structure is an extension of conventional RNNs by introducing gating units with their multiplicative behavior. The performance of conventional RNNs and LSTMs greatly differs when applied to the IAM and to the IFN/ENIT corpus. For this reason experiments were performed on both databases to give a detailed comparison of the results. Note that these experiments were conducted on a subset of the complete training set and that the reported error rates cannot be directly compared to those presented in Section 5.3.2.

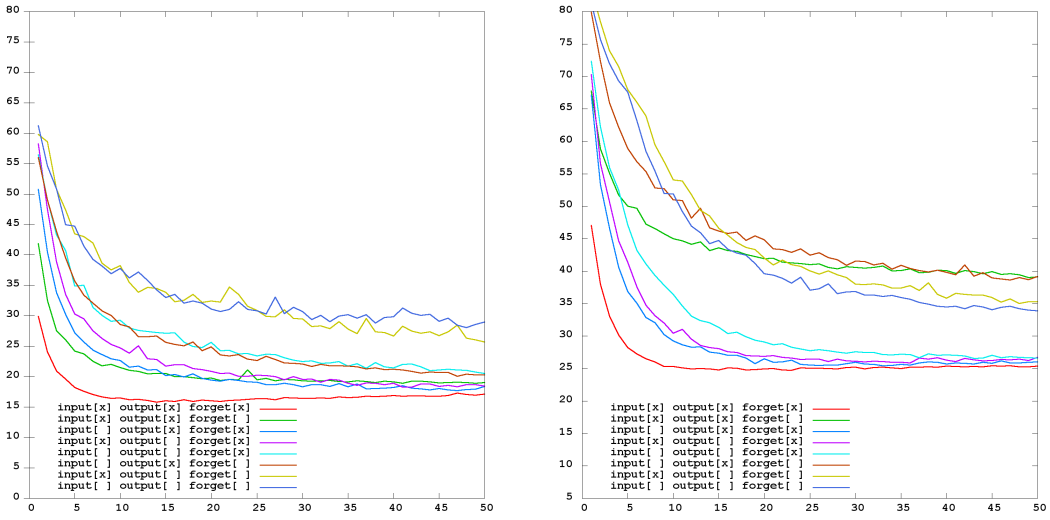
Gating Units

Theoretical evidence for the advantage of LSTMs handling long time lags has been proven by [Hochreiter & Schmidhuber 97] and the superior performance of LSTMs in tasks like handwriting recognition was shown by [Liwicki & Graves$^+$ 07]. The effect of each particular gating units in real world tasks is an interesting aspect and has to be discovered for each corpus separately. Figure 5.1 compares the evolution of the frame error over 50 epochs on IAM and on IFN/ENIT, where each gating unit is either enabled or disabled.

On the IAM database all gating units improve the frame error but the improvement obtained if only the input gate is activated seems negligible. The performance using an output or forget gate exclusively is very similar. Any two simultaneously activated gating units also show comparable results and the best result is obtained when all gating units are enabled. In total the activation of the gating units leads to a reduction of the frame error from 34% to 17%. It is worth mentioning that the convergence of the network with all gating units activated is very smooth compared to the other setups and that the label error already reaches its minimum after about 12 epochs.

On the IFN/ENIT database only the forget gate has a positive impact on the frame error. Deactivating it even increases the frame error above the frame error of the conventional RNN without any gating unit. Although the LSTM memory cell with all gating units still shows the best performance and convergence speed, the difference to all other setups with activated forget gate is small.

After the LSTM training of 50 epochs, tandem systems were trained to see the influence in the final recognition results of the HMM. Table 5.1 shows the WER and CER for each setup for IAM and for IFN/ENIT respectively. On IAM only the output gate improves the WER and if only the input gate is activated the WER is even larger than the WER of the conventional RNN. Using a forget gate exclusively



(a) IAM (b) IFN/ENIT

Figure 5.1: Label error of LSTM training for different variants of memory cells for IAM (left) and IFN/ENIT (right). The x-axis corresponds to the epoch of the neural network training and the y-axis is the label error in percent.

shows no significant difference compared to the conventional RNN. Except for the combination of the output and the forget gate any two simultaneously activated gating units further improve the performance and the setup with all gating units clearly outperforms any other setups concerning the WER, while the best CER is observed when the forget gate is disabled. Overall, the WER is reduced by 2.9% absolute from 26.9% to 24.0% by activating all units, while the CER is reduced by only 1% absolute from 12.3% to 11.3% when the conventional RNN is compared to the LSTM with all gating units enabled.

On IFN/ENIT the gating units increase the WER and CER compared to the conventional RNN with 10.1% WER and 7.9% CER. Here, the best performance is achieved without any gating unit and activating all gating units leads to an absolute increase of 0.8% concerning the WER and 0.6% concerning the CER. Activating the output and forget gate only raises the WER to 25% and the CER to 19.9%. A possible reason is that the error signal trapped in the CEC of the LSTM unit is released too early by the input gate. The large amount of contextual information already encoded in the feature set *A-2* triggers this behavior, which is further enforced by an activated forget gate. This can also be observed in the large increase of the WER when the forget gate is activated exclusively.

In total, it can be observed that the frame error of the LSTM training does not

LSTM gating units			IAM		IFN/ENIT	
input	output	forget	WER [%]	CER [%]	WER [%]	CER [%]
			26.9	12.3	10.1	7.9
×			27.7	12.5	10.3	8.0
	×		25.6	11.5	13.1	10.3
		×	26.9	12.9	18.0	14.4
×	×		25.0	11.0	15.5	12.6
×		×	26.6	12.2	11.5	8.9
	×	×	26.9	12.8	25.0	19.9
×	×	×	24.0	11.3	10.9	8.5

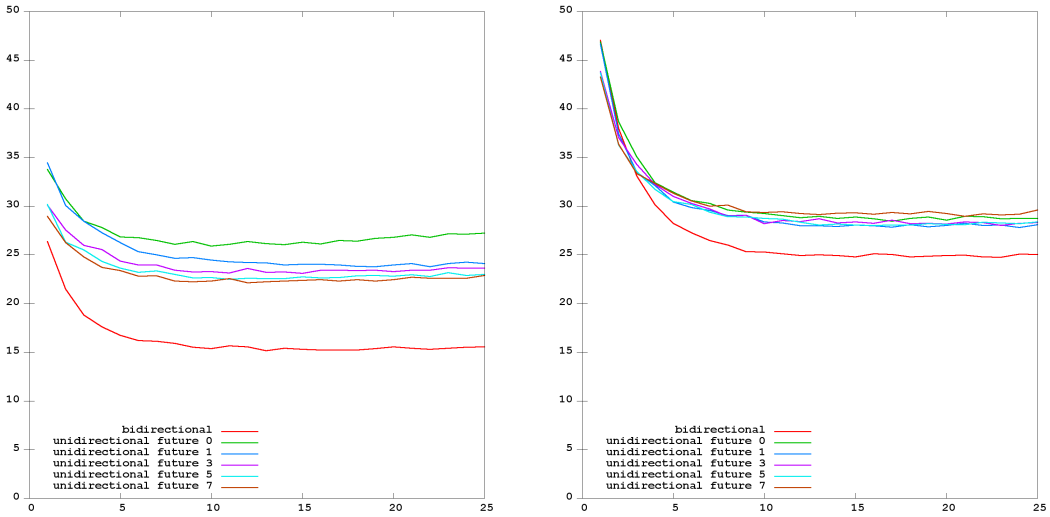
Table 5.1: Evaluation of tandem systems for different variants of LSTM memory cells on IAM and IFN/ENIT. The check boxes indicate whether the corresponding gating unit is activated or deactivated.

transfer to the word error of the tandem system trained on these features. Further, different databases show a different behavior when using gating units. Nevertheless, the LSTM training benefits from all gating units through a better convergence and the final evaluation of the tandem system either outperforms the conventional RNN or shows at least a performance close to it.

Contextual Information

Bidirectional RNNs benefit from the time modeling ability concerning both future and past context and therefore they are able to take the complete input sequence into account when estimating the posterior probability of a frame. However, they do this at the cost of a separate hidden layer and the number of weights to be estimated are doubled. For this reason it is necessary to see if this is actually required for handwriting recognition tasks and so experiments were conducted comparing bidirectional RNNs and unidirectional RNNs with different amount of future information encoded in the feature vector.

Figure 5.2 shows the evolution of the label error on IAM and IFN/ENIT using a bidirectional RNN and unidirectional RNNs with windows containing future context of none, one, three, five and seven frames. Concerning the frame error each additional future frame improves the performance on IAM and the main improvement is obtained by the very first future frame, while the difference between five and seven frames is small. On IFN/ENIT the RNN does not benefit much from the future context and all unidirectional setups with future context converge to a similar frame error. A context window of seven frames even evaluates to a slightly



(a) IAM (b) IFN/ENIT

Figure 5.2: Label error of setups using unidirectional LSTM networks with different window sizes over 25 epochs on IAM (left) and IFN/ENIT (right).

higher frame error due to the large number of parameters that need to be estimated. On both corpora the bidirectional RNN clearly outperforms all setups with unidirectional RNNs with an absolute improvement of 8% on IAM and an absolute improvement of 3% on IFN/ENIT.

Compared to the experiments concerning the gating units the frame errors of the experiments here transfer much better to the word and character error rates as shown in Table 5.2. However, large context windows lead to an increase of the WER. For this reason and due to the superiority of the bidirectional approach concerning the frame error the final experiments were performed with BRNNs only.

5.3.2 Experimental Results

IAM

The results for the different input features *E-1* and *E-2* are shown in Table 5.3 and 5.4. For *E-2* the error distribution with deletions, insertions and substitutions can be found in Appendix 7.4.3. After training the tandem systems with the maximum likelihood criterion a discriminative training was applied with the maximum likelihood system as initialization. The discriminative model was trained using the modified minimum phoneme error (M-MPE) criterion [Heigold & Schlüter$^+$ 09, Dreuw & Heigold$^+$ 11b].

	IAM		IFN/ENIT	
future frames	WER [%]	CER [%]	WER [%]	CER [%]
0	25.0	11.9	13.3	10.6
1	24.6	11.6	12.7	9.9
3	24.4	11.4	11.6	9.1
5	25.1	11.9	11.0	8.6
7	25.9	12.2	11.2	8.8

Table 5.2: Evaluation of tandem systems for unidirectional LSTM networks with different window sizes on IAM and IFN/ENIT. The window provides context for future frames only.

E-1	WER [%]		CER [%]	
	Devel	Eval	Devel	Eval
GHMM	31.9	38.9	10.6	14.7
+ M-MPE	24.3	30.0	9.8	13.9
MLP Hybrid	31.2	36.9	10.5	14.2
MLP Tandem	25.7	32.9	10.2	14.4
+ M-MPE	22.6	28.7	9.2	13.1
LSTM Hybrid	23.4	28.1	9.8	13.3
LSTM Tandem	23.0	28.4	9.6	13.2
+ M-MPE	**22.2**	**27.0**	**8.9**	**12.6**

Table 5.3: Results for the feature set *E-1* on the IAM database.

The tandem approach clearly outperforms the hybrid approach on both feature sets. Moreover, the LSTM systems outperform the hierarchical MLP systems. Comparing the M-MPE results on feature set *E-1*, an absolute improvement on the validation set over the GHMM M-MPE baseline of 1.3% WER has been achieved by the MLP tandem system, while the LSTM tandem system has achieved an absolute improvement of 2.2% WER. On the test set the performance achieved by MLPs improves the baseline by 1.3% WER and the improvement of LSTMs is 3% WER. The difference in performance between the MLP tandem system and the LSTM tandem system is only 0.4% WER on the validation set, but 1.7% WER on the test set. A possible reason is a better generalization ability of the LSTM network.

On feature set *E-2* a larger difference between the LSTM and hierarchical MLP

E-2	WER [%]		CER [%]	
	Devel	Eval	Devel	Eval
ML baseline	27.2	34.7	13.6	18.2
M-MPE baseline	24.2	30.3	11.8	16.9
MLP Hybrid	26.5	34.2	11.4	17.0
+ Realignment	26.2	33.1	11.3	16.5
MLP Tandem	24.6	31.0	11.2	16.2
+ M-MPE	22.9	29.0	9.8	14.6
+ Realignment	22.7	28.8	9.9	14.8
LSTM Hybrid	20.6	24.8	7.6	10.7
+ Realignment	22.2	26.2	9.8	12.9
LSTM Tandem	19.4	23.8	7.7	10.8
+ M-MPE	17.9	21.5	6.8	9.5
+ Realignment	**17.4**	**21.4**	**6.6**	**9.5**

Table 5.4: Results for the feature set *E-2* on the IAM database.

can be observed. Here, the absolute improvement over the MLP tandem approach is 8.8% WER. The absolute improvement of the LSTM tandem system over the GHMM M-MPE baseline is 6.32% WER on the validation set and 8.84% WER on the test set. Table 5.4 further shows the results of the realignment procedure applied to all systems. On IAM realignment lead to minor improvements in all setups except the LSTM hybrid approach. However, the difference is small compared to the corresponding system without realignment. Nevertheless the setup with the best performance could further be improved by 0.5% absolute on the validation set and 0.1% on the test set.

The best recognition result is obtained with a realigned M-MPE trained LSTM tandem system, which outperforms the best published WER on the same feature set by 1.6% WER absolute on the validation set and 1.0% WER absolute on the test set. Table 5.5 shows a comparison to other groups.

The performance of the realignment procedure is not as good as one might expect. For this reason an experiment was conducted using either a linear alignment or an alignment obtained from a well tuned M-MPE system as starting point. Table 5.6 shows the result for this experiment with LSTM tandem systems. It can be seen that an initially improper alignment can be improved by the realignment procedure of the LSTM/HMM system. In the end, however, both system show a similar performance.

System	WER [%]		CER [%]	
	Devel	Eval	Devel	Eval
GHMM + M-MPE baseline	24.2	30.3	11.8	16.9
MLP Tandem + M-MPE + Realignment	22.7	28.8	9.9	14.8
LSTM Tandem + M-MPE + Realignment	**17.4**	**21.4**	**6.6**	**9.5**
UPV, HMM/MLP [Espana-Boquera & Castro-Bleda$^+$ 11]	19.0	22.4	-	9.8
TUM, Graves et al., CTC [Liwicki & Graves$^+$ 07]	-	25.9	-	18.2
Dreuw et al., HMM [Dreuw & Doetsch$^+$ 11]	23.7	29.2	11.5	16.3

Table 5.5: Comparison of the best systems on the IAM database with results reported by other groups.

E-1	WER [%]		CER [%]	
	Devel	Eval	Devel	Eval
M-MPE Alignment	22.2	26.2	8.8	12.2
+ Realignment	21.3	26.3	7.0	12.4
Linear Segmentation	27.9	33.3	13.5	17.7
+ Realignment	22.1	27.2	8.4	12.0

Table 5.6: Comparison of the realignment procedure applied to the feature set *E-1* on IAM using a linear segmentation or an M-MPE alignment as initial alignment. In this experiment log-posterior features were extracted from the LSTM.

E-2	WER [%]		CER [%]	
	Devel	Eval	Devel	Eval
Hard labels	20.7	25.3	7.7	10.8
+ M-MPE	20.1	24.6	7.4	10.4
Soft labels	22.1	26.5	8.1	11.4
+ M-MPE	20.8	25.6	7.6	10.8

Table 5.7: Comparison of log-posterior features of an LSTM tandem training with and without soft labels for the feature set *E-2* of the IAM database.

Furthermore, the effect of soft labels as described in Section 4.2 was analyzed on the IAM database. Table 5.7 shows a comparison of LSTM tandem systems with log-posterior features based on feature set *E-2*. Both cases lead to a similar performance especially after the M-MPE training was applied. The initial alignment was based on an M-MPE trained GHMM and the confidences for the single best label was quite large, even on the transition from one label to another. For this reason and since only the single best hypotheses is mapped to a confidence value this result is not surprising.

RIMES

The results for the RIMES database are presented in Table 5.8 and the error distribution with deletions, insertions and substitutions is given in Appendix 7.4.2. On both systems the LSTM model clearly outperforms the MLP model concerning both the hybrid and tandem systems. As seen for the other databases the tandem approach evaluates to better results than the hybrid approach in all cases. The GHMM baseline trained on feature set *F-1* is improved by 12.9% WER absolutely, while the LSTM tandem system trained on feature set *F-2* achieves an absolute improvement of 13% WER to 9.6% WER. This result is close to the best published WER of 8.98% using LSTMs with CTC as shown in Table 5.9. The large gap to other HMM systems with and without MLP based features emphasizes the superiority of the RNN model on this database.

As for the other databases, realignment was applied to the best performing feature set. On RIMES only the hybrid LSTM system was improved. In all other cases the realignment procedure lead to an increase of the WER and CER. Especially the MLP systems show a inferior performance after realignment with an absolute increase of 1.9% WER using the hybrid approach and 1.78% WER using the tandem approach.

Systems		WER [%]	CER [%]
F-1	ML baseline	36.6	24.4
	MLP Hybrid	34.3	22.8
	MLP Tandem	31.2	19.3
	LSTM Hybrid	26.2	18.4
	LSTM Tandem	25.8	17.2
F-2	ML baseline	29.4	13.8
	MLP Hybrid	29.2	13.3
	+Realignment	31.1	15.7
	MLP Tandem	27.4	12.9
	+Realignment	29.2	13.0
	LSTM Hybrid	12.4	8.3
	+Realignment	12.3	8.3
	LSTM Tandem	**9.6**	**4.2**
	+Realignment	10.2	4.7

Table 5.8: Results for the RIMES database using the feature sets *F-1* and *F-2*.

Systems	WER [%]	CER [%]
GHMM baseline	29.4	13.8
MLP Tandem	27.4	12.9
LSTM Tandem	9.6	4.2
TUM, CTC	**9.0**	-
UPV, MLP Hybrid	16.8	-
ParisTech, HMM ensemble	23.7	-
SIEMENS, HMM	26.8	-

Table 5.9: Comparison of the best systems on the RIMES database with results reported by other groups. Error rates by other groups are from the ICDAR 2009 competition [Grosicki & Abed 09].

Systems		WER [%]	CER [%]
A-1	ML baseline	13.1	10.6
	MLP Hybrid	10.3	8.1
	MLP Tandem	5.9	4.7
	LSTM Hybrid	8.7	6.6
	LSTM Tandem	7.2	5.6
A-2	ML baseline	6.4	4.6
	MLP Hybrid	6.6	5.0
	+ Realignment	6.1	4.8
	MLP Tandem	**4.7**	**3.6**
	+ Realignment	4.7	3.7
	LSTM Hybrid	5.7	4.3
	+ Realignment	6.1	4.4
	LSTM Tandem	5.0	3.9
	+ Realignment	5.6	4.3

Table 5.10: Results for the IFN/ENIT database with set a-d as training set and set e as validation set using the feature sets *A-1* and *A-2*.

IFN/ENIT

The results for the IFN/ENIT database are shown in Table 5.10 and the corresponding error distribution is shown in Appendix 7.4.1. For the first time, the MLP tandem system outperforms the LSTM tandem system on both feature sets, while in the hybrid case the LSTM systems outperform the MLPs. The best absolute improvement over the GHMM baseline is 7.2% WER on *A-1* and 1.7% on *A-2* using MLP tandem systems. Both models benefit from the preprocessing method used for *A-2* and the absolute improvement by the MLP tandem system over the LSTM tandem system decreases from 1.3% WER on *A-1* to 0.3% WER on *A-2*. Compared to other systems the MLP tandem system outperforms the BHMM approach by 1.5% WER absolute as shown in Table 5.11.

As on the RIMES database, realignment in general did not lead to a superior performance. Although the MLP hybrid system could slightly be improved by 0.5% WER absolutely, the LSTM systems evaluated to a larger WER and CER after the realignment procedure was applied. Especially the performance of the LSTM tandem system decreased with an absolute difference of 0.6% in terms of

Systems	WER [%]	CER [%]
GHMM baseline	6.4	4.6
MLP Tandem	**4.7**	**3.6**
LSTM Tandem	5.0	3.9
UPV, BHMM [Gimenez & Khoury$^+$ 10]	6.2	-
UPV, HMM [Märgner & Abed 09]	12.3	-

Table 5.11: Comparison of the best systems on the IFN/ENIT database with results reported by other groups.

Systems	WER [%]		
	abcd-e	set f	set s
ICFHR 2010 [Margner & Abed 10]			
RWTH-OCR, MLP Tandem	7.3	9.1	18.9
UPV, BHMM [Gimenez & Khoury$^+$ 10]	6.2	7.8	15.4
CUBS-AMA, HMM	-	19.7	22.1
ICDAR 2011 [Margner & Abed 11]			
RWTH-OCR, MLP Tandem	5.9	7.8	15.5
REGIM, HMM	-	21.0	31.6
JU-OCR, RF	-	36.1	50.3

Table 5.12: Results of the ICDAR 2011 Arabic handwriting recognition.

WER and 0.4% in terms of CER.

The MLP tandem system using *A-1* as feature set was submitted to the ICDAR 2011 Arabic handwriting recognition competition and achieved the best performance on the unknown datasets *f* and *s*. Table 5.12 shows a comparison of the systems submitted to the ICFHR 2010 and ICDAR 2011 competition. Here, the performance compared to the BHMM approach is very similar and all other HMM based systems are clearly outperformed.

6 Conclusions and Outlook

In this work combined optimization methods for ANNs and HMMs were analyzed and evaluated on two different feature sets for each of three different handwriting databases. An HMM was used to obtain a labeling of the input data required to perform the discriminative ANN training. Afterwards, the activations of the ANN were used in an HMM in order to generate the word sequence with maximal probability. Two different approaches, the tandem and hybrid approach, were described to use the ANN activations in the HMM framework. Two different ANN topologies have been introduced, the hierarchical feed-forward MLP and the RNN. While hierarchical MLPs build the basis for experiments with feed-forward networks, the main focus of this work lied on the recurrent LSTM network structure, which extends the traditional RNN model by multiplicative gating units. Since conventional RNNs have problems learning contextual information over long time lags, the effect of these gating units regarding long range contextual information was analyzed. Further, the effect of the labeling was examined by performing experiments on labelings from different stages of the HMM training as well as labelings with a probabilistic interpretation.

6.1 Conclusions

Large improvements over the GHMM baseline could be observed when features derived from the ANN training were included. The discriminative training procedure and powerful approximation ability of ANNs outperform the generative GMM approach. An appropriate preprocessing of the features used in the HMM and ANN training is important and the improvement of the GHMM system obtained by preprocessing also transfers to the ANN based systems. On the other hand, the initial labeling has a large influence on the performance of the system. A partly incorrect alignment lead to less improvements than an alignment that was based on a well tuned HMM system. This difference could be compensated by the realignment procedure, that was introduced in this thesis. Using soft labels instead of hard labels could not improve the system. However, the approach used here had only access to a confidence value of the first-best label. Since the HMM system used to extract the confidence values was a well tuned M-MPE trained HMM system, the confidence values were close to the maximal value and therefore the difference to a hard label alignment was small.

Temporal context information has to be modeled explicitly by the ANN and only RNNs provide a natural way dealing with it. The results reflect this property and the LSTM network outperforms the hierarchical MLP in all cases where long time lags are present. Only the IFN/ENIT database is an exception, where a large amount of contextual information is encoded in the feature vector through the preprocessing steps. Although the absolute difference in the final results is small with an absolute difference of 1.3% on feature set *A-1* and 0.3% on feature set *A-2*, a large variation in performance is observed when particular gating units are enabled or disabled. Especially the error signal leaving the memory cells of the LSTM structure needs to be protected by an activated input gate. Further, the activation of the forget gate leads to a false behavior by reseting the cell state in undesired situations. Although beneficial for the frame error, this leads to an absolute increase of the WER of up to 15.1% compared to the conventional RNN. In contrast, the preprocessing on IAM was not optimized to include contextual information in the framewise feature vectors. Therefore, the LSTM model itself was responsible for handling the contextual information. For this reason, the IAM database benefits from all gating units and shows the best performance when all gating units are activated.

In addition, contextual information about the future in RNNs is important and improvements on the IAM and the IFN/ENIT database could be observed when a particular amount of future frames was provided. However, the best performance was achieved when the whole input sequence is taken into account by an BRNN. The large size of the input layer generated by the window function lead to an improper estimation of the weights and the model could not learn a selective behavior w.r.t. the information of future frames. On the other hand, bidirectional LSTMs provide a natural way to incorporate future context and the gating units can selectively make use of it.

The experiments show a general superiority of the tandem approach over the hybrid approach. Estimating Gaussian models on the basis of ANN activations leads to a better generalization in order to overcome poor estimations for single frames and improve the decoding ability of the HMM. Further, the PCA reduction with the corresponding window function includes contextual information for the HMM. Although the information about the class discrimination is included in the ANN based features, the generative HMM model of the IAM database could be improved by a discriminative M-MPE training. In this sense, the discriminative HMM training starting from the maximum likelihood model is able to discover properties about the class discrimination which were not detected by the ANN.

Compared to the CTC decoding the performance of the combined LSTM/HMM model is only superior in the case of the IAM database, although different feature sets were used. However, concerning the proposed combined ANN/HMM approach the correspondence to CTC training is the realignment procedure. The weak per-

formance of the realignment procedure is remarkable. A possible reason is that the training data used to train the ANN is the same as used in the HMM training. Moreover, the realignment was performed after the ANN stabilized. This can lead to overfitting and the forced alignment performed after the first pass can be stuck in a local optimum. The best way to overcome this problem is to include a regularization term in the optimization criterion of the ANN, instead of using a validation set as single stopping criterion. In addition, the realignment could be performed after each epoch of the ANN training. However, this results in very large training periods of the combined system even for small databases.

6.2 Outlook

Some results of this thesis require a more detailed analysis and the methods can be improved accordingly. Extracting a real state posterior distribution from an HMM instead of using only first-best confidence values can lead to further improvements of the combined systems. This requires advanced techniques in processing the generated lattices and is part of future work.

Further, the CTC networks have to be put in contrast to the combined ANN and HMM approach presented here. A direct integration of the ANN model in the HMM training would lead to a better comparison. The impact of the realignment procedure on the generated alignment has to be analyzed as well as the effect of the ANN training on the labeling of the training data. Instead of using a validation set as stopping criterion, the training procedure of the ANN has to be extended by a regularization term in order to avoid overfitting and to get stuck into local optima.

Another extension related to the ANN training is to use a pretraining procedure [Dahl & Member$^+$ 11], in order to overcome the random weight initialization of the ANN. The pretraining can lead to a better convergence behavior and reproducibility of the experiments.

7 Appendix

7.1 Squared Error Criterion Backpropagation

Using the squared error criterion as starting point to derive the equations of the error backpropagation yields a decomposition of the global error $E = \sum_{n=1}^{N} E_n$ into following local errors:

$$E_n = \frac{1}{2} \sum_{k=1}^{K} \Big[y_k^{(L)} - \delta(k, c_n) \Big]^2 \tag{7.1}$$

The activation function is denoted as $f\big(z_i^{(l)}\big)$ for unit i in layer $1 \leq l \leq L$ with the net input is $z_i^{(l)} = \sum_j \alpha_{ij}^{(l)} y_j^{(l-1)}$, where $y_j^{(l-1)}$ corresponds to the unit activation after the activation function was applied. Here, only the equation for the output layers are derived. Once the formulas for the $\delta_j^{(l)}$ values are known, the backpropagation procedure for the preceding layers is exactly the same as described in Section 3.2.2.

Differentiating the local squared error results for the output layer:

$$\frac{\partial E_n}{\partial y_k^{(L)}} = y_k^{(L)} - \delta(k, c_n) \tag{7.2}$$

By applying the chain rule the derivative of the local error w.r.t. the net input using the logistic sigmoid activation function in the output units is:

$$\frac{\partial E_n}{\partial z_k^{(L)}} = \frac{\partial E_n}{\partial y_k^{(L)}} \frac{\partial y_k^{(L)}}{\partial z_k^{(L)}} \tag{7.3}$$

Differentiating the logistic sigmoid activation function results in:

$$\frac{\partial y_k^{(L)}}{\partial z_k^{(L)}} = \frac{\exp\big(-z_k^{(L)}\big)}{\Big(1 + \exp\big(-z_k^{(L)}\big)\Big)} = y_k^{(L)}\big(1 - y_k^{(L)}\big) \tag{7.4}$$

Substituting back gives the final formula for $\delta_k^{(L)}$:

$$\delta_k^{(L)} = \frac{\partial E_n}{\partial z_k^{(L)}} = \big(y_k^{(L)} - \delta(k, c_n)\big) \cdot y_k^{(L)}\big(1 - y_k^{(L)}\big) \tag{7.5}$$

7.2 Real Time Recurrent Learning

The real time recurrent learning algorithm updates the weights after each time step t, making it applicable to online learning and further makes the memory required for training independent of the length of the observation sequence, i.e. it is *local in time* [Schmidhuber 89]. The weight update rule for weight α_{ij} at time step t with step size γ concerning the local error $E_n(t)$ corresponds to the sum of the errors w.r.t. the output units [Williams & Zipser 89]:

$$\Delta\alpha_{ij}(t) = -\gamma \frac{\partial E_n(t)}{\partial w_{ij}} = -\gamma \sum_{k=1}^{K} E_k(t) \frac{\partial y_k^{(L)}(t)}{\partial \alpha_{ij}} \tag{7.6}$$

The time dependent partial derivative of the activation $y_k^{(L)}(t)$ w.r.t. the weights α_{ij} of an output unit k with its activation function $f_k(z_k^{(L)})$ is given by:

$$\frac{\partial y_k^{(L)}(t)}{\partial \alpha_{ij}} = f_k'(z_k^{(L)}) \left(\sum_h \alpha_{kh} \frac{\partial y_h^{(L)}(t-1)}{\partial \alpha_{ij}} + \delta(k,i) y_j^{(L)}(t-1) \right) \tag{7.7}$$

To calculate the partial derivatives of Equation 7.7 an auxiliary variable is defined. Let $p_{ij}^k(t) = \frac{\partial y_k^{(L)}(t)}{\partial \alpha_{ij}}$ with the starting condition $p_{ij}^k(0) = 0$. Then

$$p_{ij}^k(t) = f_k'(z_k^{(L)}) \left(\sum_h \alpha_{kh} p_{ij}^k(t-1) + \delta(k,i) y_j^{(L)}(t-1) \right) \; \forall t > 0 \tag{7.8}$$

After each time step the corresponding value for $p_{ij}^k(t)$ can be calculated and used to update the weights according to Equation 7.6. The weight update can also be delayed until the time dependent weight updates for all frames in the sequence were calculated. In this case the global time invariant weight update is done as in BPTT by accumulating the weight updates for each time step:

$$\Delta\alpha_{ij} = \sum_{t=1}^{T} \Delta\alpha_{ij}(t) \tag{7.9}$$

The time complexity of the RTRL algorithm for n fully connected units is $O(n^4)$ per time step, making it inapplicable for large scale network structures. However, the memory complexity with $O(n^3)$ in the number of units does not depend on the length of the observation sequence as it is the case in BPTT. Therefore RTRL is especially useful if the length of the observation sequence is a priori unknown.

7.3 Corpus Statistics

7.3.1 IAM Database

	trainset	validationset	testset
Number of running words	53,884	8,717	25,472
Number of running characters	219,749	33,352	100,762
Number of text lines	6,162	920	2,781
Average number of words/line	8.75	9.47	9.16
Average number of characters/word	4.08	3.82	3.96
Average image width in pixels	1,751	1,740	1,763
Average image height in pixels	123	115	131
Average image aspect ratio	1 : 14.2	1 : 15.1	1 : 13.5

7.3.2 RIMES Database

	trainset	validationset
Number of running words	51,738	7,464
Number of running characters	321,178	46,403
Average number of characters/word	6.2	6.06
Average image width in pixels	187	185
Average image height in pixels	72	71
Average image aspect ratio	1 : 2.59	1 : 2.6

7.3.3 IFN/ENIT Database

	a	b	c	d	e
Number of running words	6,537	6,710	6,477	6,735	6,033
Number of running characters	55,654	57,688	55,864	58,028	47,638
Average number of characters/word	8.51	8.6	8.63	8.62	7.89
Average image width in pixels	420	412	408	396	381
Average image height in pixels	98	97	94	96	93
Average image aspect ratio	1: 4.29	1 : 4.25	1 : 4.32	1 : 4.16	1 : 4.16

7.4 Experimental Results

7.4.1 IFN/ENIT

Systems	Words [%]			Characters [%]		
	Del	Ins	Sub	Del	Ins	Sub
A-1 ML baseline	0.0	0.0	13.1	7.2	0.1	3.3
MLP Hybrid	0.0	0.0	10.3	0.8	0.8	6.5
MLP Tandem	0.0	0.0	5.9	0.5	0.5	3.7
LSTM Hybrid	0.0	0.0	8.7	0.4	0.4	5.8
LSTM Tandem	0.0	0.0	7.2	0.5	0.5	4.6
A-2 ML baseline	0.0	0.0	6.4	1.5	0.1	2.3
MLP Hybrid	0.0	0.0	8.7	0.7	0.7	5.4
+Realignment	0.0	0.0	6.1	0.7	0.7	3.4
MLP Tandem	0.0	0.0	4.7	0.4	0.4	2.9
+Realignment	0.0	0.0	4.7	0.4	0.4	2.9
LSTM Hybrid	0.0	0.0	5.7	0.4	0.4	3.5
+Realignment	0.0	0.0	6.1	0.4	0.4	3.6
LSTM Tandem	0.0	0.0	5.0	0.3	0.3	3.2
+Realignment	0.0	0.0	5.6	0.4	0.4	3.5

Table 7.1: Deletions, insertions and substitution on IFN/ENIT for HMM based systems.

7.4.2 RIMES

Systems	Words [%]			Characters [%]		
	Del	Ins	Sub	Del	Ins	Sub
F-1 ML baseline	0.0	0.0	36.6	18.8	0.1	5.5
MLP Hybrid	0.0	0.0	34.3	11.0	1.0	12.9
MLP Tandem	0.0	0.0	31.2	6.3	3.9	16.9
LSTM Hybrid	0.0	0.0	26.2	8.1	0.0	10.3
LSTM Tandem	0.0	1.2	24.6	5.6	2.6	9.1
F-2 ML baseline	0.0	0.0	29.4	1.9	3.1	8.9
MLP Hybrid	0.0	2.4	26.8	4.2	3.0	6.1
+Realignment	0.0	2.4	28.7	5.1	3.3	7.3
MLP Tandem	0.0	3.2	24.2	4.8	2.9	5.2
+Realignment	0.0	4.6	24.6	2.1	3.7	7.2
LSTM Hybrid	0.0	0.0	12.4	2.3	1.4	4.6
+Realignment	0.0	0.0	12.3	2.3	1.5	4.5
LSTM Tandem	0.0	0.0	9.6	1.2	0.5	2.5
+Realignment	0.0	0.0	10.2	2.8	1.3	0.5

Table 7.2: Deletions, insertions and substitution on RIMES for HMM based systems.

7.4.3 IAM

Systems	Words [%]						Characters [%]					
	Devel			Eval			Devel			Eval		
	Del	Ins	Sub	Del	Ins	Sub	Del	Ins	Sub	Del	Ins	Sub
ML baseline	5.6	5.1	16.5	6.3	5.8	22.6	2.4	2.7	8.5	2.6	6.6	9.9
M-MPE baseline	5.8	4.0	14.4	6.1	5.0	19.2	2.1	2.4	7.1	2.8	4.3	8.8
MLP Hybrid	6.5	4.0	16.0	6.9	5.3	22.0	3.4	2.7	5.3	4.1	4.8	8.1
+ Realignment	8.3	2.7	15.3	8.6	3.5	21.0	3.8	2.3	5.2	4.5	4.2	7.9
MLP Tandem	4.1	3.9	16.6	5.4	4.1	21.4	2.5	2.2	6.6	3.9	3.1	9.1
+ M-MPE	2.8	4.5	15.6	3.6	5.2	20.2	1.7	2.2	5.9	3.4	2.7	8.5
+ Realignment	2.9	4.3	15.5	4.0	4.6	20.2	2.2	1.8	5.8	3.5	2.8	8.5
LSTM Hybrid	5.6	3.3	11.7	6.7	3.6	14.5	2.8	1.5	3.2	3.8	2.1	4.7
+ Realignment	4.6	2.3	15.3	5.8	2.1	18.2	3.4	1.9	4.5	4.2	2.6	6.2
LSTM Tandem	2.4	4.0	13.0	2.9	4.4	16.5	1.1	1.5	5.1	1.8	2.2	6.8
+ M-MPE	2.1	3.9	11.9	2.9	3.9	14.7	1.0	1.2	4.6	1.7	1.8	6.0
+ Realignment	2.0	3.7	11.7	2.6	4.1	14.7	1.0	1.2	4.4	1.6	1.9	6.0

Table 7.3: Deletions, insertions and substitution on IAM for HMM based systems using feature set *E-2*.

List of Figures

List of Tables

Bibliography

[Auer & Burgsteiner$^+$ 08] P. Auer, H. Burgsteiner, W. Maass. A learning rule for very simple universal approximators consisting of a single layer of perceptrons. *Neural Netsworks*, Vol. 21, pp. 786–795, June 2008.

[Augustin & Carré$^+$ 06] E. Augustin, M. Carré, G. E., J. M. Brodin, E. Geoffrois, F. Preteux. Rimes evaluation campaign for handwritten mail processing. In *Proceedings of the Workshop on Frontiers in Handwriting Recognition*, pp. 231–235, October 2006.

[Bengio & Mori$^+$ 91] Y. Bengio, R. D. Mori, G. Flammia, R. Kompe. Global optimization of a neural network - hidden markov model hybrid. *IEEE Transactions on Neural Networks*, Vol. 3, pp. 252–259, March 1991.

[Bengio & Simard$^+$ 94] Y. Bengio, P. Simard, P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 157–166, September 1994.

[Bishop 06] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Bourlard & Morgan 93] H. A. Bourlard, N. Morgan. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[Bourlard & Wellekens 90] H. Bourlard, C. J. Wellekens. Links Between Markov Models and Multilayer Perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 12, pp. 1167–1178, 1990.

[Bridle 90] J. S. Bridle. Alpha-nets: a recurrent neural network architecture with a hidden markov model interpretation. *Speech Commun.*, Vol. 9, pp. 83–92, January 1990.

[Bryson & Ho 69] E. Bryson, Y. C. Ho. *Applied optimal control: optimization, estimation, and control*. Blaisdell Publishing Company, 1969.

[Dahl & Member+ 11] G. E. Dahl, S. Member, D. Yu, S. Member, L. Deng. Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. *Processing*, Vol. 99, pp. 1–13, 2011.

[Dreuw & Doetsch+ 11] P. Dreuw, P. Doetsch, C. Plahl, G. Heigold, H. Ney. Hierarchical hybrid MLP/HMM or rather MLP features for a discriminatively trained Gaussian HMM: A comparison for offline handwriting recognition. In *International Conference on Image Processing*, 2011.

[Dreuw & Heigold+ 11a] P. Dreuw, G. Heigold, H. Ney. Confidence- and margin-based MMI/MPE discriminative training for off-line handwriting recognition. *International Journal on Document Analysis and Recognition*, Vol. 14, No. 3, 2011.

[Dreuw & Heigold+ 11b] P. Dreuw, G. Heigold, H. Ney. Confidence and margin-based mmi/mpe discriminative training for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, Vol. 14, No. 3, pp. 273–288, April 2011. DOI 10.1007/s10032-011-0160-x The final publication is available at www.springerlink.com.

[Dreuw & Jonas+ 08] P. Dreuw, S. Jonas, H. Ney. White-space models for offline Arabic handwriting recognition. In *International Conference on Pattern Recognition*, pp. 1–4, Tampa, Florida, USA, Dec. 2008.

[Dunteman 89] G. H. Dunteman. *Principal Component Analysis*. Sage Publications, 1989.

[Elman 90] J. L. Elman. Finding structure in time. *Cognitive Science*, Vol. 14, No. 2, pp. 179–211, 1990.

[Espana-Boquera & Castro-Bleda+ 11] S. Espana-Boquera, M. Castro-Bleda, J. Gorbe-Moya, F. Zamora-Martinez. Improving offline handwritten text recognition with hybrid HMM/ANN models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 33, No. 4, pp. 767–779, april 2011.

[Franzini & Lee+ 90] M. Franzini, K.-F. Lee, A. Waibel. Connectionist viterbi training: a new hybrid method for continuous speech recognition. In *International Conference on Acoustics, Speech, and Signal Processing*, pp. 425 –428 vol.1, apr 1990.

[Gers & Schmidhuber+ 00] F. A. Gers, J. A. Schmidhuber, F. A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, Vol. 12, pp. 2451–2471, October 2000.

[Gimenez & Khoury+ 10] A. Gimenez, I. Khoury, A. Juan. Windowed Bernoulli mixture HMMs for Arabic handwritten word recognition. In *Proceedings of the 2010 12th International Conference on Frontiers in Handwriting Recognition*, pp. 533–538, 2010.

[Gollan & Bacchiani 08] C. Gollan, M. Bacchiani. Confidence scores for acoustic model adaptation. In *ICASSP*, pp. 4289–4292. IEEE, 2008.

[Graves & Bunke+ 08] A. Graves, H. Bunke, S. Fernandez, M. Liwicki, J. Schmidhuber. Unconstrained online handwriting recognition with recurrent neural networks. In *Advances in Neural Information Processing Systems*, Vol. 20. MIT Press, 2008.

[Graves & Fernández+ 05] A. Graves, S. Fernández, J. Schmidhuber. Bidirectional LSTM networks for improved phoneme classification and recognition. In *International Conference on Artificial Neural Networks*, pp. 799–804, 2005.

[Graves & Fernández+ 06] A. Graves, S. Fernández, F. Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, pp. 369–376, 2006.

[Graves & Schmidhuber 05] A. Graves, J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM networks. In *Proceedings of international Joint conference on Neural Networks*, 2005.

[Graves & Schmidhuber 08] A. Graves, J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Neural Information Processing Systems*, pp. 545–552, 2008.

[Graves 09] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Ph.D. thesis, 2009.

[Grosicki & Abed 09] E. Grosicki, H. E. Abed. ICDAR 2009 handwriting recognition competition. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, pp. 1398–1402, 2009.

[Hammer 00] B. Hammer. *Learning with recurrent neural networks*. Lecture notes in control and information sciences Nr 254. Springer, 2000.

[Heigold & Schlüter+ 09] G. Heigold, R. Schlüter, H. Ney. Modified mpe/mmi in a transducer-based framework. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 3749–3752, Taipei, Taiwan, April 2009.

[Hermansky & Ellis+ 00] H. Hermansky, D. Ellis, S. Sharma. Tandem connectionist feature stream extraction for conventional HMM systems. In *International Conference on Acoustics, Speech and Signal Processing*, pp. 1635–1638, 2000.

[Hermansky & Sharma 98] H. Hermansky, S. Sharma. TRAPS - classifiers of temporal patterns. In *Proceedings of 5th International Conference on Spoken Language Processing*, pp. 1003–1006, 1998.

[Hochreiter & Schmidhuber 97] S. Hochreiter, J. Schmidhuber. Long short-term memory. *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

[Hochreiter 98] S. Hochreiter. Recurrent neural net learning and vanishing gradient, 1998.

[Hornik 91] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, Vol. 4, No. 2, pp. 251–257, 1991.

[Hunt 90] M. J. Hunt. Figures of merit for assessing connected-word recognisers. *Speech Communication*, Vol. 9, No. 4, pp. 329–336, 1990.

[Jaeger 04] H. Jaeger. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science*, Vol. 304, pp. 78–80, 2004.

[Jonas 09] S. Jonas. Improved modeling in handwriting recognition. Master's thesis, RWTH Aachen University, June 2009.

[Jordan 89] M. I. Jordan. Serial order: A parallel, distributed processing approach. In *Advances in Connectionist Theory: Speech*. 1989.

[Juan & Toselli+ 01] A. Juan, A. H. Toselli, J. Domnech, J. González, I. Salvador, E. Vidal, F. Casacuberta. Integrated handwriting recognition and interpretation via finite-state models. *International Journal of Pattern Recognition and Artifial Intelligence*, Vol. 2004, pp. 519–539, 2001.

[Kirkpatrick & Gelatt+ 83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. Optimization by simulated annealing. *Science*, Vol. 220, pp. 671–680, 1983.

[Kline & Berardi 05] M. Kline, L. Berardi. Revisiting squared-error and cross-entropy functions for training neural network classifiers. *Neural Computation*, Vol. 14, pp. 310–318, December 2005.

[Lippmann 89] R. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, Vol. 1, pp. 1–38, 1989.

[Liwicki & Bunke 05] M. Liwicki, H. Bunke. Iam-ondb - an on-line english sentence database acquired from handwritten text on a whiteboard. In *Proceedings of 8th International Conference on Document Analysis and Recognition*, Vol. 2, pp. 956–961. IEEE, 2005.

[Liwicki & Graves$^+$ 07] M. Liwicki, A. Graves, H. Bunke, J. Schmidhuber. A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks. In *Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.

[Märgner & Abed 09] V. Märgner, H. E. Abed. ICDAR 2009 Arabic Handwriting Recognition Competition. pp. 1383–1387, 2009.

[Margner & Abed 10] V. Margner, H. E. Abed. ICFHR 2010 - Arabic handwriting recognition competition. *International Conference on Frontiers in Handwriting Recognition*, Vol. 0, pp. 709–714, 2010.

[Margner & Abed 11] V. Margner, H. E. Abed. ICDAR 2011 - Arabic handwriting recognition competition. *International Conference on Document Analysis and Recognition*, Vol. 0, 2011.

[Martens 94] J.-P. Martens. A connectionist approach to continuous speech recognition. In *Progress and Prospects of Speech Research and Technology: Proceedings of the CRIM/FORWISS Workshop*, pp. 26–33. Infix, St. Augustin, 1994.

[Mclachlan 04] G. J. Mclachlan. *Discriminant Analysis and Statistical Pattern Recognition (Wiley Series in Probability and Statistics)*. Wiley-Interscience, Aug. 2004.

[Moody & Darken 89] J. Moody, C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, Vol. 1, No. 2, pp. 281–294, 1989.

[Ney & Schlüter 10] H. Ney, R. Schlüter. Introduction to automatic speech recognition. In *Lecture Slides*, 2010.

[Ney 05] H. Ney. Pattern recognition and neural networks. In *Lecture Slides*, 2005.

[Pastor & Toselli$^+$ 04] M. Pastor, A. Toselli, E. Vidal. Projection profile based algorithm for slant removal. In *Image Analysis and Recognition*, Vol. 3212, pp. 183–190. 2004.

[Pesch 11] H. Pesch. Advancements in latin script recognition. Master's thesis, RWTH Aachen University, Nov. 2011.

[Plahl & Schlüter$^+$ 10] C. Plahl, R. Schlüter, H. Ney. Hierarchical bottle neck features for LVCSR. pp. 1197–1200, Makuhari, Japan, Sept. 2010.

[Plaut & Nowlan+ 86] D. Plaut, S. Nowlan, G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Carnegie Mellon University, Pittsburgh, PA, 1986.

[Rabiner 89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pp. 257–286, 1989.

[Richard & Lippmann 10] M. D. Richard, R. P. Lippmann. Neural network classifiers estimate bayesian a posteriori probabilities. *Neural Computation*, Vol. 3, No. 4, pp. 461–483, Dec. 2010.

[Riedmiller & Braun 93] M. Riedmiller, H. Braun. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *IEEE international conference on neural networks*, pp. 586–591, 1993.

[Robinson & Fallside 87] A. J. Robinson, F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University, London, 1987.

[Robinson 94a] A. Robinson. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 298–305, 1994.

[Robinson 94b] A. Robinson. IPA: Improved phone modeling with recurrent neural networks - Robinson. In *Acoustics, Speech, and Signal Processing, International Conference on*, Vol. 1, pp. 37–40. IEEE, 1994.

[Rosenbaum & Cohen+ 07] D. Rosenbaum, R. Cohen, S. Jax, D. Weiss, R. Vanderwel. The problem of serial order in behavior: Lashley's legacy. *Human Movement Science*, Vol. 26, No. 4, pp. 525–554, Aug. 2007.

[Rosenblatt 57] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[Rumelhart & Hinton+ 88] D. E. Rumelhart, G. E. Hinton, R. J. Williams. *Learning representations by back-propagating errors*, pp. 696–699. MIT Press, Cambridge, MA, USA, 1988.

[Sainath & Kingsbury+ 11] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak. Making Deep Belief Networks effective for Large Vocabulary Continuous Speech Recognition. In *Automatic Speech Recognition and Understanding*, Hawaii, USA, Dec. 2011.

[Sakoe & Chiba 78] H. Sakoe, S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 26, No. 1, pp. 43–49, 1978.

[Schmidhuber 89] J. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, Vol. 1, pp. 403–412, 1989.

[Schmidhuber 92] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, Vol. 4, pp. 234–242, 1992.

[Schölkopf & Smola 02] B. Schölkopf, A. J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.

[Schuster & Paliwal 97] M. Schuster, K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, Vol. 45, No. 11, pp. 2673 –2681, nov 1997.

[Seide & Gang$^+$ 11] F. Seide, L. Gang, Y. Dong. Conversational Speech Transcription using context-dependent Deep Neural Network. pp. 437–440, Florence, Italy, Aug. 2011.

[Valente & Vepa$^+$ 07] F. Valente, J. Vepa, C. Plahl, C. Gollan, H. Hermansky, R. Schlüter. Hierarchical neural networks feature extraction for LVCSR system. In *Interspeech*, 0 2007.

[Werbos 74] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University, Cambridge, MA, 1974.

[Werbos 02] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, Vol. 78, No. 10, pp. 1550–1560, Aug. 2002.

[Williams & Peng 90] R. J. Williams, J. Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, Vol. 2, pp. 490–501, 1990.

[Williams & Zipser 89] R. J. Williams, D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, Vol. 1, pp. 87–111, 1989.