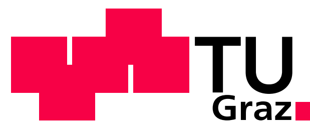
Master's Thesis

Sequence labeling with neural networks: Connectionist Temporal Classification revisited

---

Institute for Theoretical Computer Science
Graz University of Technology



Submitted by:
Matthias Zöhrer

Supervisor:
O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang Maass

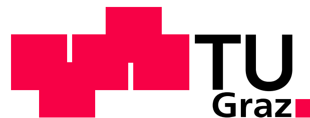
March 28, 2013

Masterarbeit

Sequence labeling with neural networks: Connectionist Temporal Classification revisited

---

Institut für Grundlagen der Informationsverarbeitung
Technische Universität Graz



Vorgelegt von:
Matthias Zöhrer

Betreuer:
O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang Maass

28. März 2013

Abstract

Connectionist Temporal Classification (CTC) is a supervised learning technique enabling to learn *input* $\mapsto$ *target* relations of sequences. The correct labeling of the target symbols does not have to be known explicitly, as the most probable mapping is calculated by the output layer. This characteristic allows to solve a wide spectrum of state-of-the-art industrial problems, which could only be solved by Hidden Markov Models (HMM) so far. Possible examples are handwriting- and speech-recognition problems, object recognition tasks or any other kind of sequence labeling problem. Any learning model, trainable with a gradient based learning rule, can be used to learn the predicted target labels of a CTC output layer. This work puts its focus on this interesting topic of machine learning. Firstly, learning effects of Recurrent Neural Networks (RNN) trained with a CTC objective are analyzed. Secondly, due to the need of more abstract data representation Deep Architectures are discussed. Thirdly, a general concept of learning Deep Recurrent Architectures based on unsupervised greedy layer-wise pre-training, is developed. It is shown that these Deep Recurrent Architectures show a superior recognition performance on a sequential object recognition task and are able to outperform various neural classifiers used for comparison.

Kurzfassung

Connectionist Temporal Classification (CTC) ist eine *supervised* Lerntechnik, welche im Stande ist, *input* $\mapsto$ *target* Relationen in Sequenzen zu lernen. Dabei muss das exakte *target labeling* im Vorhinein nicht bekannt sein, da CTC das wahrscheinlichste Mapping berechnet. Diese Charakteristik erlaubt eine große Anzahl industrieller Probleme zu lösen, welche früher ausschließlich mit Hilfe von Hidden Markov Modellen lösbar waren. Beispiele sind Sprach- und Handschrifterkennung oder jegliche Formen von zeitlicher Objekterkennung. CTC kann auf jedes Klassifkationsmodell, das auf Gradientenverfahren basiert, angewandt werden. Diese Arbeit befasst sich mit rekurrenten neuronalen Netzwerken (RNN), welche mit Hilfe einer CTC *objective* trainiert werden. Hierbei wird ein Schwerpunkt auf die Analyse von Gradientenmethoden gesetzt. Es wird angenommen, dass hierarchische RNNs in der Lage sind, Input-Daten besser zu abstrahieren. Da diese Modelle mit bekannten Optimierungsverfahren nur schwer lernbar sind, wird eine neue Methode, welche auf unsupervised pretraining basiert, entwickelt. Der neue hierarchische rekurrente Classifier wird in einem Objekterkennungsproblem, dem Erkennen von Handschrift, getestet. Hier kann das neu entwickelte Modell ein besseres Ergebnis als vergleichbare neuronale Classifier erzielen.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

date signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen / Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am Unterschrift

Danksagung

Mein besonderer Dank gilt...

Wolfgang Maass, für die Unterstützung dieser Arbeit und die Möglichkeit an aktueller Forschung teilzunehmen.

Der TUG und dem Institut für Grundlagen der Informationsverarbeitung für die finanzielle Unterstützung und die Möglichkeit der Teilnahme am Self-Organized Recurrent Neural Learning for Language Processing Projekt (ORGANIC).

Nikos Chararas, und weiteren Mitarbeitern des zentralen Informatik Dienstes der TUG (ZID) für das zur Verfügungstellen von GPU Clustern und die professionelle Unterstützung in Softwarefragen.

Meinen Eltern, Siegfried und Elfriede, die mir dieses Studium ermöglicht und stets meine Interessen und Talente gefördert haben.

Meinem Bruder, Siegfried, für viele unterhaltsame Stunden und aufmunternde Worte.

All meinen Freunden, Kollegen und Verwandten, die nie aufgehört haben mich aufzumuntern und zu unterstützen.

Graz, im Oktober 2012 Matthias Zöhrer

diese Arbeit ist in
englischer Sprache verfasst

Contents

List of Figures XVII

List of Tables XIX

List of Algorithms XXI

List of Abbreviations XXIII

1. Introduction 1
- 1.1. Problem Definition . . . 1
- 1.2. Contributions . . . 3
- 1.3. Overview of Thesis . . . 4

2. Mathematical backgrounds 7
- 2.1. Supervised Learning . . . 7
- 2.2. Pattern Classification . . . 7
  - 2.2.1. Probabilistic Classification . . . 7
  - 2.2.2. Training Probabilistic Classifiers . . . 8
- 2.3. Sequence Labeling . . . 10
  - 2.3.1. Sequence Labeling Problem . . . 10
  - 2.3.2. Sequence Processing Queues . . . 12
- 2.4. Connectionist Temporal Classification . . . 13
  - 2.4.1. Constructing the Classifier . . . 13
  - 2.4.2. CTC Forward-Backward Algorithm . . . 14
  - 2.4.3. CTC Objective Function . . . 17
  - 2.4.4. CTC Decoding . . . 18
- 2.5. CTC Models . . . 20
  - 2.5.1. Recurrent Neural Networks . . . 21
  - 2.5.2. Deep Belief Networks . . . 24
- 2.6. Gradient Learning Methods for CTC Models . . . 30
  - 2.6.1. Stochastic Gradient Descent . . . 31
  - 2.6.2. Resilient Backpropagation . . . 32
  - 2.6.3. Stochastic Diagonal Levenberg-Marquard Method . . . 33

2.6.4. Hessian Free Method 34

3. Optimized Learning of Recurrent Models **43**
3.1. Defining a Classifier Landscape 43
3.2. Exploring Optimizations for a CTC Classifier 44

4. Toy Experiments **47**
4.1. Classifying concatenated MNIST Digits 47
4.1.1. Data Preparation 48
4.1.2. Comparison of Gradient Based Optimization Algorithms 49
4.1.3. Comparison of Gradient Based Optimization Algorithms for Hierarchical Recurrent Architectures 54
4.1.4. Generative pre-training for Hierarchical Recurrent Architectures 58
4.1.5. Comparison of Neural Network Architectures 60

5. Conclusion and Future Work **65**

A. Appendix **69**
A.1. Software Framework 69

Bibliography **71**

List of Figures

1.1. Machine learning problems 2
1.2. Sequence learning tasks CTC networks can be applied to 3

2.1. Labeling problem 11
2.2. Sequence processing queue 12
2.3. CTC forward path calculation [Graves, 2012] 16
2.4. Neural classifiers 20
2.5. Backpropagation through time 22
2.6. Vanishing gradient problem 23
2.7. Deep Belief Network 24
2.8. Restricted Boltzmann Machine 26
2.9. Block Gibbs Sampling in a RBM 28
2.10. Optimization algorithms for neural classifiers 30
2.11. Gradient descent with momentum 32
2.12. Conjugate gradient method 35
2.13. Optimization in a long narrow valley 37

3.1. Definition of a classifier 43

4.1. MNIST digits 47
4.2. Concatenated MNIST digits 48
4.3. RNN-CTC architecture with 40 frames MNIST training example 49
4.4. Test error curves of the algorithms (MNIST) 53
4.5. HRNN-CTC architecture with 40 frames MNIST training example 54
4.6. Comparing the error curves of a RNN and a HRNN (MNIST) . 57
4.7. Error curves of the DBN-RNN (MNIST) 59
4.8. Network models 60
4.9. Error curves of different network architectures (MNIST) 63

5.1. Improving optimization methods 66
5.2. Improving objectives 67

A.1. Software features 69

List of Tables

4.1. Model parameters of the RNN 50
4.2. Algorithm parameters of the RNN 50
4.3. Test error of different algorithms (RNN) (MNIST) 51
4.4. Model parameters of the HRNN 55
4.5. Algorithm parameters of the HRNN 55
4.6. Test error of different algorithms (HRNN) (MNIST) 56
4.7. Comparison of a RNN and HRNN (MNIST) 57
4.8. Test error of a HRNN with 1000x3 neurons (MNIST) 57
4.9. Model and training parameters of the DBN-RNN 58
4.10. Errors of the DBN-RNN (MNIST) 59
4.11. Model parameters of different network architectures 61
4.12. Errors of different network architectures (MNIST) 62

List of Algorithms

1. Backpropagation through time algorithm 23
2. Linear CG pseudo code 35
3. Linear PCG pseudo code 40
4. Hessian Free pseudo code 42

List of Abbreviations

BPTT Backpropagation through time

BRNN Bidirectional recurrent neural network

CTC Connectionist temporal classification

DARPA Defense Advanced Research Projects Agency

DBN Deep belief network

EM Expectation maximization

GPU Graphical processing unit

HF Hessian free

HMM Hidden markov model

LSTM Long short term memory

MIT Massachusetts Institute of Technology

MLP Multilayer perceptron

MLSTM Multidimensional long short term memory

RBM Restricted Boltzmann Machine

RBM Mean Covariance Restricted Boltzmann Machine

RNN Recurrent Neural Network

RPROP Resilient backpropagation

1. Introduction

1.1. Problem Definition

Recognizing speech, handwriting and movies are difficult problems for every artificial system to solve. Several methods have been developed, using different classification techniques. Despite the fact that recognition results have constantly been improved, the problem is still not completely solved. Looking at these sequence learning tasks in a more generalized way we can say that very often a labeling problem has to be dealt with. The difficulty occurring in this sort of problem set is the classification of unlabeled data, often referred to as the chicken-egg problem. This means that on the one hand, it is very hard to classify sequences, if there is no label information available and on the other hand, it is hard to label a signal, if the sequence has not been classified yet. However, if we are able to find a good way to classify and label signals, we will be able to solve any open task in the described problem class.

Hidden Markov Models (HMM) [Rabiner, 1989] were one of the first approaches to solve sequence labeling problems. These models are widely used for industry purposes and generate good results on various data sets. One problem of HMM systems is that they require a significant amount of task specific knowledge, which is modeled in the structure of the HMM [Graves et al., 2006]. Other technologies, such as neural network architectures, do not need this sort of prior knowledge, since they simply depend on input-output signal relations using non-linear mappings. This leads to the assumption that neural nets could be a more general, powerful mechanism to solve this problem [Martens, 2011a].

[Graves et al., 2006] showed that labeling unsegmented data with recurrent neural networks is possible. In this approach a new technique called Connectionist Temporal Classification (CTC) [Bridle, 1990] was used, which allowed to train and test unlabeled sequences with neural net technology. Using a bidirectional long-short-term-memory recurrent neural net (BLSTM) their architecture outperformed HMMs on several handwriting datasets [ICDAR,

2012]. In a recent paper the algorithm was applied to a phoneme recognition problem [Graves, 2011].

This work has its main focus on sequence learning with recurrent neural networks. The state-of-the-art technique CTC, used to learn and classify speech and handwriting, is analyzed. It evolves around four main problems in machine learning [Bengio, 2012], shown in figure 1.1.

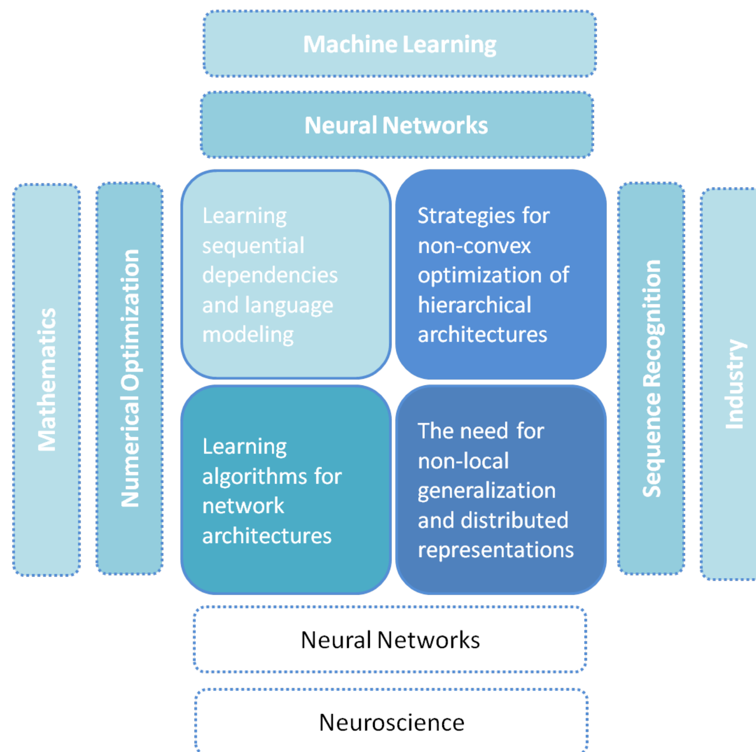


Figure 1.1.: Machine learning problems

The labeling problem solved with Connectionist Temporal Classification, addresses the sequence and language modeling task. Gradient based algorithms like steepest descent with momentum [Phansalkar and Sastry, 1994] or resilient backpropagation [Riedmiller and Braun, 1993] and the Hessian Free Method [Martens, 2011b] solve the non convex of the learning task and are used as learning algorithms for the underlying neural network. With the help of hierarchical architectures a more generalized distribution of data should be found.

Improving existing strategies to solve the tasks mentioned above, should help to solve a great variety of sequence learning problems shown in figure 1.2 frequently occurring in industrial challenges and real world scenarios.

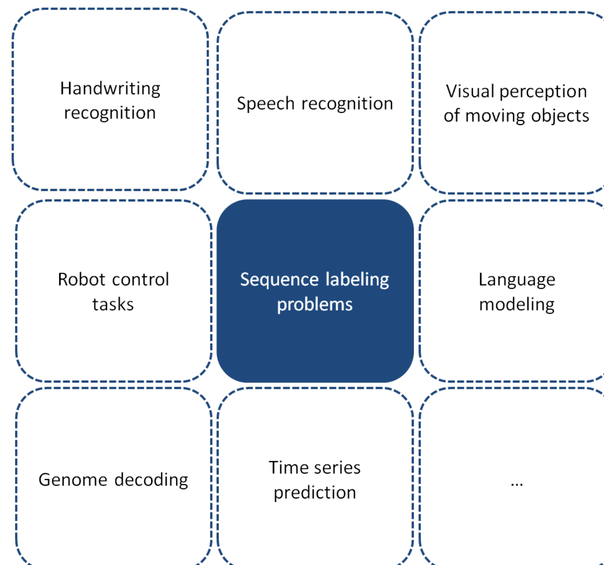


Figure 1.2.: Sequence learning tasks CTC networks can be applied to

1.2. Contributions

In [Graves et al., 2006] an output layer, known as **Connectionist Temporal Classification**, is introduced that allows RNNs to be trained directly for sequence labeling tasks with unknown input-label alignments. The introduced CTC-BLSTM architecture is able to solve the same learning tasks as a HMM, and even outperforms this model on various data sets. Because of the good results it is worth exploring Connectionist Temporal Classification in detail.

In this work various **gradient based learning** methods are evaluated on a **RNN** network with a CTC output layer [Rumelhart et al., 1986], [Graves et al., 2006]. **Learning effects** like the **vanishing gradient problem** are **analyzed** and methods to overcome this problem are discussed.

Hierarchical architectures have superior learning capabilities when it comes to object recognition tasks. A common way to train these models is **greedy**

layer-wise pre-training [Hinton and Salakhutdinov, 2006b]. The use of this generative pre-training of feed-forward architectures leads to significant performance increases, and is extended in this work to recurrent models.

The performance of the underlying model and algorithms is evaluated on the **MNIST** digit database [Lecun and Cortes, 2012]. The RNN model is also compared to other network architectures like Multilayer Perceptrons (MLP) [Rosenblatt, 1958], Convolutional Neural Networks [LeCun et al., 1999] and Long Short Term Memory Networks (LSTM) [Gers et al., 2002] and Deep Belief Networks (DBN) [Mohamed et al., 2009].

A **GPU based framework** based on python and Theano [Bergstra et al., 2010] was built to test various network architectures with CTC. The mathematical expression compiler Theano allows to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays on the graphic processor unit and speeds up the simulation time. By doing computations on a GPU another big problem in machine learning, simulating big datasets in adequate time can be solved.

1.3. Overview of Thesis

The chapters of this work are grouped into four parts: Mathematical background material is presented in **chapter 2**. **Chapter 3** puts forward a hypothesis for optimized learning of recurrent network models on sequence learning tasks. **Chapter 4** encompass toy experiments and experimental results which verify the made assumption of the previous chapter. **Chapter 5** summarizes main achievements gained in this study and give a future outlook. Apart from that a description of the software framework, used to test various models in the toy experiments of section 4, can be found in **Appendix A**.

Although several neural network architectures are used to compare learning capabilities on sequence labeling tasks, the main focus of this study lies on **supervised learning** methods and **learning effects** of a **standard recurrent architecture** (RNN), which is described in detail in chapter 2.5.1. One reason for this decision is the need of storage capabilities of temporal patterns when learning sequential data dependencies.

As far as optimization algorithms are concerned only **gradient based training** methods for training RNNs are examined. Non gradient methods like particle swarm optimization [Kennedy and Eberhart, 1995] or genetic algorithms [Goldberg, 1989] are not taken into account, in order to limit the scope of this work.

Stacking multiple layers of RNNs leads to a **hierarchical recurrent architecture**. This architecture should be able to learn a better representation of the underlying data due the ability of generating more abstract temporal features. However, training of this more advanced model turned out to be a very difficult task. Nevertheless, it is shown that a powerful **second order gradient method** (HF) and **generative pre-training** with the help of **Deep Belief Networks** (DBN), which was formerly only used for feed-forward network architectures is able to solve the learning problem described above leading to an improved recognition rate on the MNIST digit task.

2. Mathematical backgrounds

2.1. Supervised Learning

Machine learning problems where a set of input-target pairs is provided for training are referred to as supervised learning tasks [Graves, 2012]. This differs from reinforcement learning, where only a specific reward is provided for training, and unsupervised learning, where no training signal exists at all and the learning method uncovers the structure of the data by inspection alone.

A supervised learning task consists of a training set $\mathcal{S}$ of input-target pairs (x, z), where x is an element of the input space $\mathcal{X}$ and z is an element of the target space $\mathcal{Z}$, along with a disjoint test set $\mathcal{S}$. Elements of $\mathcal{S}$ are the training examples for a underlying classifier.

2.2. Pattern Classification

Pattern classification, also known as pattern recognition, is one of the most extensively studied areas of machine learning [Bishop, 1996], [Duda et al., 2001]. Pattern classification deals with non-sequential data, nevertheless, much of the practical and theoretical framework underlying it, carries over to the sequential case. It is therefore instructive to briefly review this framework before the term sequence labeling will be explained.

2.2.1. Probabilistic Classification

A pattern classifier is able to map input-features to target classes $h : \mathcal{X} \mapsto \mathcal{Z}$. The input space of $\mathcal{X}$ is a set of real-valued vectors and the target set of $\mathcal{Z}$ consists of discrete classes K [Graves, 2012]. If the classifier is probabilistic,

the conditional probability $p(C_k|x)$ of the $\mathcal{K}$ target classes of a input pattern x is computed and the most probable target is chosen:

$$h(x) = \underset{k}{argmax}\ p(C_k|x) \tag{2.1}$$

2.2.2. Training Probabilistic Classifiers

If a probabilistic classifier yields a conditional distribution $p(C_k|x, \theta)$ over the class labels C_k given input x and parameters θ, the product over the independent and identically distributed (i.i.d.) input-target pairs in the training set S can be written as:

$$p(S|\theta) = \prod_{(x,z)\in S} p(z|x, \theta) \tag{2.2}$$

Applying the Bayes rule leads to:

$$p(\theta|S) = \frac{p(S|\theta)p(\theta)}{p(S)} \tag{2.3}$$

The posterior distribution over classes for some new input x can then be found by the integral:

$$p(C_k|x, S) = \int_\theta p(C_k|x, \theta)p(\theta|S)d\theta \tag{2.4}$$

A common approximation to equation (2.1) is the maximum a priori approximation (MAP), which finds the single parameter vector θ_{MAP} that maximizes $p(\theta|S)$ and uses this to make predictions:

$$p(C_k|x, S) \approx p(C_k|x, \theta_{MAP}) \tag{2.5}$$

$$\theta_{MAP} = \underset{\theta}{argmax}\ p(S|\theta)p(\theta) \tag{2.6}$$

The maximum likelihood (ML) parameter vector θ_{ML} can be defined as:

$$\theta_{ML} = \underset{\theta}{argmax}\ p(S|\theta) = \underset{\theta}{argmax} \prod_{(x,u)\in S} p(z|x,\theta) \tag{2.7}$$

The standard procedure to minimize equation is the computation of the maximum likelihood loss function $\mathcal{L}(S)$ which is defined as the negative logarithm of the probability assigned to S:

$$\mathcal{L}(S) = -ln \prod_{(x,z)\in S} p(z|x) = -\sum_{(x,z)\in S} ln\ p(z|x) \tag{2.8}$$

The function is derived in respect to the model parameters

$$\frac{\partial \mathcal{L}(s)}{\partial \theta} = \sum_{(x,z)\in S} \frac{\partial \mathcal{L}(x,z)}{\partial \theta} \tag{2.9}$$

making the training of the probabilistic classifier possible.

2.3. Sequence Labeling

In the previous chapter a probabilistic classifier was defined. This classifier is able to learn input $\mapsto$ target relations of single patterns. This concept will now be extended to the time domain. A training objective for probabilistic classifier able to learn pattern relations of sequences is presented.

Before directly stepping into the mathematical details of this probabilistic classifier, some basic definitions will be explained. Firstly the term *sequence-labeling*, a problem occurring when learning sequence $\mapsto$ sequence mappings, is defined. Secondly the term *sequence-processing-queue* is explained. This should help to understand the basic principles of sequence classification widely used in machine learning.

2.3.1. Sequence Labeling Problem

The term sequence labeling encompasses all tasks where sequences of data are transcribed with a set of of labels [Graves et al., 2009]. These labels are part of a transcription set of fixed size like the alphabet, words or even sentences. If someone wants to assign a series of acoustic features to spoken words, wants to map a sequence of hand gestures to a sentence (gesture recognition), or predicts protein secondary structures, a sort of sequence labeling has to be done.

Learning labeled sequences is a comparatively easy task to solve for an artificial system, as direct mappings from the input signal to the target signal exist. In the last decade plenty of cleverly designed Deep Learning Models had been able to generate impressive results on image recognition tasks [Le et al., 2012], [Lee et al., 2009] using labeled data.

If the data is unlabeled, the exact location, in other words, the correct alignment between an input frame and the target is not known. Algorithms have to find the most probable alignment. This is possible as time signals as speech or handwriting contain contextual information constrained by the laws of syntax and grammar. Input and target labels form strongly correlated sequences. However, solving a learning problem with unlabeled data is a much harder problem than learning labeled target alignments. In addition, it should be pointed out that a remarkable characteristic of the human brain is, its ability to solve both problems.

Figure 2.1 visualizes this main difference between labeled and unlabeled learning problems. The colored squares on the left side indicate the known input $\mapsto$ target alignment for every time-step, whereas on the right side this information is not given in the dataset.

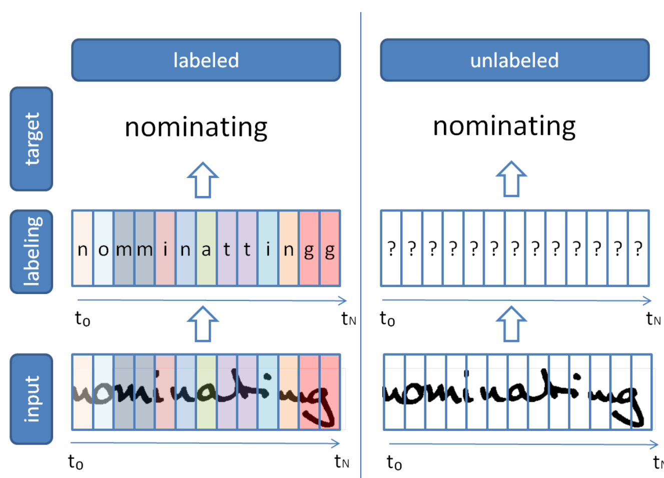


Figure 2.1.: Labeling problem

Two probabilistic models solving the sequence-labeling problem from above can be found in machine learning literature:

Hidden Markov Models: Hidden Markov Models (HMMs) [Rabiner, 1989] are a two stage stochastic process. The first process uses a Markov Chain modeling transition probabilities of hidden states. This process is not visible from outside. The second process emits observations at every time-step t with respect to its emission probabilities. In the case of labeling problems these observations correspond to the sequence of output labels. HMMs are able to correctly label unsegmented data, if equally aligned input sequences are used, or if the emission probabilities are initialized with the mean- and covariance-vectors of the training set [Pfister and Kaufmann, 2008]. This procedure is called *flat start*. Details about this model and the underlying training algorithm (EM) can be found in [Fink, 2008], and will not be presented in this work. The model is able to fully solve the labeling problem mentioned above.

CTC networks: Connectionist Temporal Classification [Graves et al.,

2006] solves the sequence labeling problem in a forward-backward algorithm based approach using neural network architectures. A softmax *readout* connected to a neural network layer selects the most probable output activation at every time-step for classification. In the training case the correct mapping from input- to the target-sequence is computed by calculating the most probable target labeling over the time. The network is trained in respect to this '*observation*' with a gradient based optimization method. Chapter 2.4 provides a mathematical description of this specially designed output layer and training method.

2.3.2. Sequence Processing Queues

When dealing with sequence learning problems very often the input data follows the flow-diagram shown in figure 2.2.

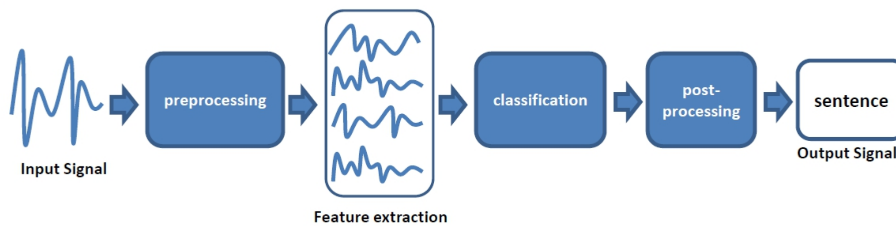


Figure 2.2.: Sequence processing queue

Pre-processing- and feature-extraction-methods extract important features of the data and reassemble the latter to have the correct scale i.e. dimensions. Post-processing methods improve the recognition rates with the help of additional task specific knowledge. In the case of language processing tasks very often the use of dictionaries are a common method to improve recognition results.

In many machine learning applications these typical processing steps are applied. However, in this work neither pre-processing nor post-processing is applied to the data. This guarantees that the result is not distorted by any additional external knowledge. Apart from that the feature extraction is done by the underlying classifier directly.

2.4. Connectionist Temporal Classification

This section defines a probabilistic classifier, known as Connection Temporal Classification [Bridle, 1990]. The classifier is able to extract input $\mapsto$ target relations of unlabeled temporal data with the help of a specially designed output layer connected to an underlying neural network topology. In the training case a forward-backward algorithm computes the most probable path of labels over the target sequence. The network tries to predict the correct labels (*targets*) at every training step and assigns specific output activations to the most probable target label. Features which are not separable are assigned with a '*blank*' label, and do not affect the prediction. Due to the fact that the target labeling is not known, the network will generate noisy (*faulty*) predictions in an early stage of training. However it will improve, its predictions after a couple of training steps.

2.4.1. Constructing the Classifier

CTC uses a specially designed output layer which converts the output activations of the underlying layers to a conditional probability distribution over label sequences [Graves et al., 2009]. Selecting the most probable label at time $t_{0...T}$ will result in a classifier for a given input sequence. The number of CTC outputs L+1 corresponds to the number of possible labels of all sequences plus one '*blank*' unit, indicating '*no label*'. Output activations are normalized with the help of a *softmax* function. [Graves et al., 2009]

$$y_k^t = \frac{e^{a_k^t}}{\sum_k e^{a_k^t}} \tag{2.10}$$

y_k^t is the final output and a_k^t is the activation of the output unit k at time t. These normalized outputs are used to estimate the conditional probability $y_k^t = p(k, t|x)$ of observing a label (or '*blank*') at time t [Graves et al., 2009]. The **conditional probability** $p(\pi|x)$ of observing a particular **path** π through all possible label activations is calculated by multiplying together the probability of a target label and blank labels for every time-step.

$$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{2.11}$$

This is possible as it is assumed that the output probabilities at each time-step are independent of those at other time-steps [Graves, 2012]. The activation $y^t_{\pi_t}$ can be interpreted as the probability that the network will output the label t of path π at time t.

In order to correctly encode the output of a CTC network, repeated labels as well as *blank* labels must be removed. This is done using an operator F. The sum of all paths mapped with B is the **conditional probability** of some **labeling l**[1] [Graves, 2012].

$$p(l|x) = \sum_{\pi \in F^{-1}(\mathbf{l})} p(\pi|\mathbf{x}) \tag{2.12}$$

It is not known where the label of a particular transcription could occur, so the overall sum of the places where an output label appears is calculated. Collapsing together different paths onto the same labeling is what makes it possible for CTC to use unsegmented data. It allows the network to predict the labels without knowing in advance where they occur. A dynamic programming can be used to calculate all possible paths for one input sequence. In case of CTC a graph-based algorithm, which is similar to the HMM forward-backward algorithm, solves this problem [Graves, 2012].

The application of the the operator F^{-1} maps output activations to the correct output label sequence. When training the network the operator F must be applied to the target sequences as well. In this case *blank* labels must be inserted between consecutive target labels. This is needed because F^{-1} would remove repeated labels, otherwise, making it impossible to train repeated tokens. If the network, however, is forced to output *blank* labels in between the repeated tokens, F^{-1} does not remove the repeated label, making it possible to output the label twice [Graves et al., 2009].

2.4.2. CTC Forward-Backward Algorithm

The CTC Forward-Backward Algorithm, a dynamic programming method, is able to quickly calculate the conditional probability $p(l|x)$ of possible label sequences l given the input x [Graves, 2012].

[1] the paths are mutually exclusive

The forward variable $\alpha(t, u)$ is the summed probability of all paths of the length t which are mapped by F and are able to produce the given labels. It is possible that a *blank* label is inserted between two target labels, so all transitions between *blank* and *non-blank* labels are allowed. This also enlarges the label length to $U' = 2U + 1$. U is the length of the original label sequence l. When calculating the forward probability, these additional insertions must be taken into account, so a new operator $V(t, u) = \{\pi \in A'^t : F(\pi) = l_{1:u/2}, \pi_t = l'_u\}$ is defined. All paths of length t that are mapped by F onto the length $u/2$ prefix are taken into account. $A = A \cup \{blank\}$ is the extended alphabet A of the target labels.

The forward probability $\alpha(t, u)$ can now be defined as:

$$\alpha(t, u) = \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y^i_{\pi_i} \tag{2.13}$$

The equation above can be calculated recursively [Graves, 2012]:

$$\alpha(1, 1) = y^1_b \tag{2.14}$$

$$\alpha(1, 2) = y^1_{l_1} \tag{2.15}$$

$$\alpha(1, u) = 0, \forall u > 2 \tag{2.16}$$

$$\alpha(t, u) = y^t_{l'_u} \sum_{i=f(u)}^{u} \alpha(t-1, i) \tag{2.17}$$

$$\text{where } f(u) = \begin{cases} u-1 \text{ if } l'_u = \text{blank or } l'_{u-2} = l'_u \\ u-2 \text{ otherwise} \end{cases} \tag{2.18}$$

The term y^1_b (2.14) stands for the activation of the *blank* label on the output at time-step t_1. The term $y^1_{l_1}$ (2.15) denotes the activation of the first target label on the output at time-step t_1. Equation (2.16) points out that there are only two labels allowed at time-step t_1; a *blank* label (2.15) and the first target label (2.14). Note that a zero probability is assigned to those forward variables that do not have enough time-steps left to complete the sequence.

$$\alpha(t, u) = 0 \ \forall u < U' - 2(T - t) - 1 \tag{2.19}$$

Thinking of a computational implementation, the forward variables could be interpreted as a matrix of the size (S, 2U+1), where S is the input sequence length and U the target length, storing all possible label activations of *blank* and target labels at each time-step. A graphical interpretation is shown in figure 2.3. The black nodes indicate *blank* labels.

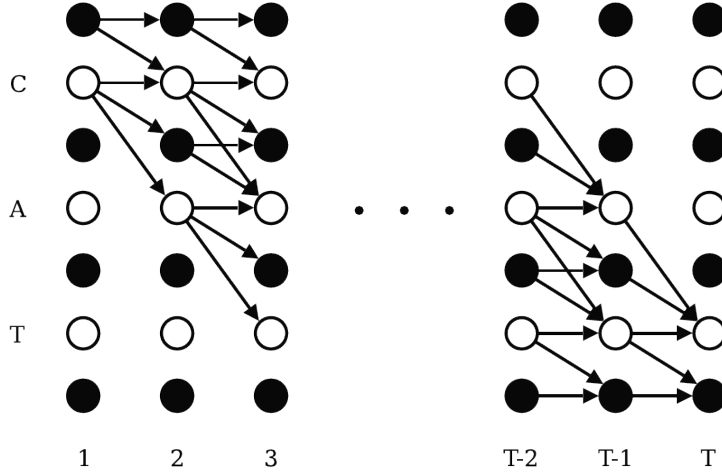


Figure 2.3.: CTC forward path calculation [Graves, 2012]

The backward path calculation is done in a similar way using a matrix of the same size, but storing the summed probabilities of those paths needed to complete a sequence l. The computation is done in reverse order. Again a mapping operator $W(t,u) = \{\pi \in A'^{T-t} : F(\hat{\pi} + \pi) = l \; \forall \hat{\pi} \in V(t,u)\}$ must be defined [Graves, 2012].

$$\beta(t,u) = \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-1} y_{\pi_i}^{t+i} \tag{2.20}$$

The backward path matrix can be initialized as follows:

$$\beta(T, U') = 1 \tag{2.21}$$

$$\beta(T, U'-1) = 1 \tag{2.22}$$

$$\beta(T, u) = 0, \forall u < T' - 1 \tag{2.23}$$

$$\beta(t,u) = \sum_{i=u}^{g(u)} \beta(t+1,i) y_{l'_i}^{t+1} \tag{2.24}$$

$$\text{where } g(u) = \begin{cases} u+1 \text{ if } l'_u = \text{blank or } l'_{u+2} = l'_u \\ u+2 \text{ otherwise} \end{cases} \tag{2.25}$$

All backward variables which are not able to finish the labeling l are assigned with a zero probability [Graves, 2012].

$$\beta(t,u) = 0 \ \forall u > 2t \tag{2.26}$$

$$\beta(t,U'+1) = 0 \ \forall t \tag{2.27}$$

2.4.3. CTC Objective Function

In order to train a network with gradient descent, an objective function and its derivatives with respect to the network outputs must exist [Bishop, 1996]. In a CTC network the objective function is given as the negative log probability of correctly labeling the entire training set S [Graves, 2012].

$$\mathcal{L}(S) = -ln \prod_{(\mathbf{x},\mathbf{z})\in S} p(z|x) = -\sum_{(\mathbf{x},\mathbf{z})\in S} ln\ p(z|x) \tag{2.28}$$

S is the training set, x the input, and z the target sequences. Setting $l = z$ and defining the set $X(t,u) = \pi \in A'^T : \mathcal{F}(\pi) = \mathbf{z}, \pi_t = \mathbf{z'}_u$ the equations (2.11), (2.13) and (2.21) lead to:

$$\alpha(t,u)\beta(t,u) = \sum_{\pi\in X(t,u)} \prod_{t=1}^{T} y_{\pi_t^t} = \sum_{\pi\in X(t,u)} p(\pi|x) \tag{2.29}$$

The equation (2.29) is the portion of the total probability $p(z|x)$ due to those paths going through z'_u at time t. For any t, we can therefore sum over all u to get the loss function [Graves, 2012]:

$$\mathcal{L}(x,z) = -ln \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{2.30}$$

It can be interpreted as the negative log-likelihood of the most probable path in respect to the target labeling.

2.4.4. CTC Decoding

In order to decode the most probable label, two methods are suggested [Graves et al., 2009].

$$l^* = \text{argmax } p(\mathbf{l}|\mathbf{x}) \tag{2.31}$$

The term above is the most probable path in the set of possible label combinations. Finding the *argmax* is known as *best-path* decoding (eq. 2.31). Here the most probable path corresponds to the most probable labeling $l^* \approx \mathcal{F}(l^*)$. This is easy to compute, as it is just the concatenation of the most active network output at every time-step t. However, [Graves, 2012] points out that this does not guarantee to find the most probable labeling.

Another possible decoding procedure is *prefix-search* decoding. Here the output sequence is divided into parts with boundary points of blank labels with high probabilities. The most probable labeling for each section is calculated individually. The search is extended each time-step with those labels who have the largest cumulative probability. The key idea of this method lies in the fact that if the both sides of the boundary points are strongly predicted, it is easier to predict the labels between this boundary. Further details about this search method can be found in [Graves, 2012].

CTC decoding can also be extended to a grammar G, known as *constrained* decoding (eq. 2.32) [Graves, 2012]. In this case the most probable labels of the output path are calculated in respect to this a probabilistic grammar, leading to the following encoding [Graves, 2012]:

$$l^* = \text{argmax } p(\mathbf{l}|\mathbf{x}) * p(\mathbf{l}|\mathbf{G}) \tag{2.32}$$

Calculating the most probable output labeling in respect to the given grammar can be solved with a *token-passing* algorithm iteratively. The method is closely related to the use of language models in HMMs [Young et al., 1989], which, in general, improves the performance of the classifier. Further details

about *constrained* decoding can be found in [Graves, 2012].

In this work an external language model will not be used, as it would bias the result of the underlying classifier. The implemented **models** use **best-path decoding** to generate the output sequences.

2.5. CTC Models

A CTC output layer as described in chapter 2.4 can be connected to any underlying model, which is trainable via a gradient based learning procedure. Although various types of possible universal approximators capable of learning CTC predictions exist, this work has its focus on a specific recurrent architecture defined in [Rumelhart et al., 1986]. Figure 2.4 gives an overview of various network models, like Long Short Term Memory Networks (LSTM) [Gers et al., 2002], Deep Belief Networks (DBN) [Mohamed et al., 2009], Multilayer Perceptrons (MLP) [Rosenblatt, 1958], Convolutional Neural Networks [LeCun et al., 1999] and [Gers et al., 2002], trainable with CTC.

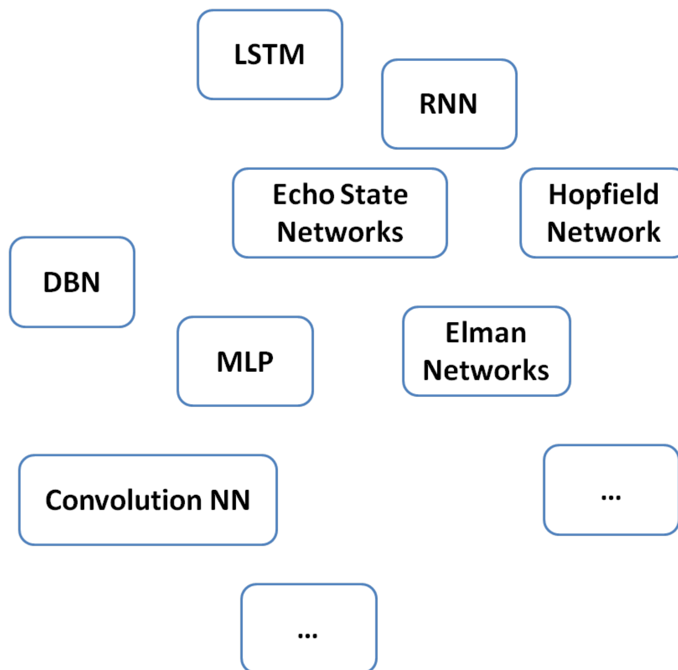


Figure 2.4.: Neural classifiers

In the first part of this chapter the basic mathematical framework of a standard RNN network is explained. In the second part the Deep Belief network, a powerful probabilistic generative model consisting of stacked layers of stochastic, latent variables is defined. Understanding the mathematical principles of both models will help to get a better understanding of probabilistic classification of sequential data and could be seen as a mathematical contribution to the toy experiments in chapter 4

2.5.1. Recurrent Neural Networks

RNNs are able to extract temporal dependencies with the help of its recurrent connections. A standard recurrent architecture as defined in [Rumelhart et al., 1986] has a self connected hidden layer and can be defined with the following equations:

$$t_i = W_{hx}x_i + W_{hh}h_{i-1} + b_h \tag{2.33}$$

$$h_i = e(t_i) \tag{2.34}$$

$$s_i = W_{yh}h_i + b_y \tag{2.35}$$

$$\hat{y}_i = g(s_i) \tag{2.36}$$

W_{hx}, W_{hh} and W_{hy} are the weight matrices of the input-, hidden- and output-layer. The term b_h and b_y denotes the bias terms. g and e are the non-linear activation functions of the network. The values $t_1, t_2, ..., t_T$, $s_1, s_2, ..., s_T \in \Re^k$ are the inputs to the hidden- and output-units and h_i is the hidden activation of the recurrent connection.

A common method to train *input* $\mapsto$ *target* relations of these '*simple*' recurrent networks, in addition to real time recurrent learning ($RTRL$) [Williams and Zipser, 1989] and extended Kalman filters [Williams, 1992], is backpropagation through time ($BPTT$) [Werbos, 1990]. A forward pass, computing the output activations of an RNN using the equations from above, is the same as a forward pass in a feed-forward neural network with the exception of adding the hidden state of the network to the current input. The output $\hat{y}$ is calculated recursively over the input sequence x of the length I.

The recurrent network is literally unfolded over the time, meaning that the recurrent model is converted to a standard feed-forward network architecture in respect to the sequence length. The number of unfolded hidden layers is proportional to the length of the current input signal. Figure 2.5 provides a graphical interpretation of this concept.

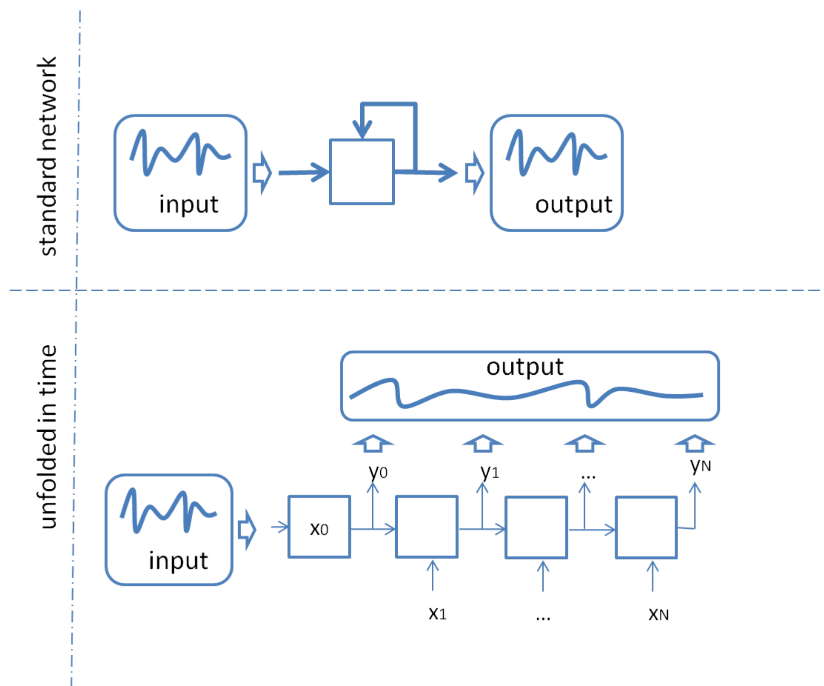


Figure 2.5.: Backpropagation through time

All hidden layers share the same weights. All activations are calculated iteratively, using the current input signal and the activations of the layer below. If the number of layers to store the activations back in time is fixed, the algorithm is called truncated backpropagation through time [Zipser, 1989].

With the help of the BPTT algorithm RNNs can be trained successfully. However when backpropagating the error over the unfolded network, the gradient can get very small or very high. This phenomena is called *vanishing gradient* problem [Hochreiter, 1998], [Bengio et al., 1994] and is a serious problem when learning long time dependencies. A pseudo code of the BBTT algorithm is shown in figure 1.

If the gradient of the objective function vanishes over the time, the error signal gets lost. The network is not able to learn the signal dependencies because of insufficient weight change. Figure 2.6 illustrates the *vanishing gradient* problem. The diagram shows a recurrent network unrolled in time, where the units are shaded according to how sensitive they are to the input at time 1 (black is high and white is low). The influence of the first input decays exponentially over time.

```
back_propagation_through_time()
    % a ... input
    % y ... output
    % x ... current context
    % t ... time
    % k ... unfolding depth (truncation)
    % N ... sequence length
    unfold the network to contain k instances of f
    do until stopping criteria is met:
        for t from 0 to N - 1
            Set the network inputs to x, a[t], a[t+1], ..., a[t+k-1]
            p = forward-propagate input over network
            e = y[t+k] - p % compute the error
            back-propagate error across the unfolded network
            update all the weights in the network
            average the weights
            x = f(x)
```

Algorithm 1: Backpropagation through time algorithm

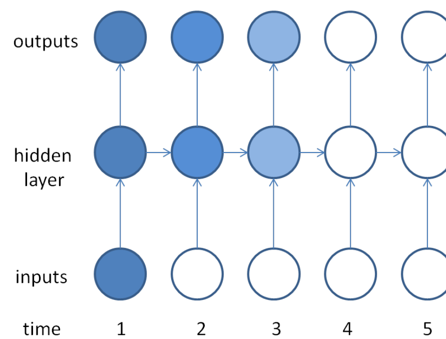


Figure 2.6.: Vanishing gradient problem

2.5.2. Deep Belief Networks

A Deep Belief Network (DBN) is a probabilistic generative model that is composed of multiple layers of stochastic, latent variables [Salakhutdinov and Murray, 2008]. It is a graphical model which learns to extract a deep hierarchical representation of training data. A single layer of a Deep Belief network consists of a Restrictive Boltzmann Machine (RBM) [Hinton and Salakhutdinov, 2006b], which is a sub-model of the class of Energy Based Models. If multiple Restrictive Boltzmann Machines (RBM) are stacked and trained in a greedy manner they form a Deep Belief Network (DBN) Figure 2.7 visualizes a Deep Belief Network.

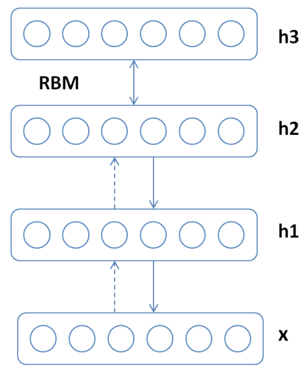


Figure 2.7.: Deep Belief Network

Energy Based Models

Energy based models associate a scalar energy to each configuration of their internal variables. Learning corresponds to modifying that energy function in a way that its shape has desirable properties. Energy-based probabilistic models define a probability distribution through an energy function. It can be defined as follows [Hinton, 2005], [Hinton, 2010]:

$$p(x) = \frac{e^{-E(x)}}{Z} \tag{2.37}$$

The term Z is normalization factor and is also called the partition function by

analogy with physical systems.

$$Z = \sum_{x} e^{-E(x)} \tag{2.38}$$

If an energy based model has some non observed variables h the energy based function will change to the form [Hinton, 2005]:

$$P(x) = \sum_{h} P(x,h) = \sum_{h} \frac{e^{-E(x,h)}}{Z} \tag{2.39}$$

The free energy is defined as follows

$$\mathcal{F}(x) = -\log \sum_{h} e^{-E(x,h)} \tag{2.40}$$

The negative log-likelihood gradient of the data x (2.37) has the form [Bengio et al., 2007]

$$\Delta\theta = E_M \frac{\partial \mathcal{F}(x)}{\partial \theta} - E_D \frac{\partial \mathcal{F}(x)}{\partial \theta} \tag{2.41}$$

The first term represents an expectation of the partial derivative over the model distribution and the second an expectation over the data. The positive and the negative phase of both terms do not refer to the sign of each term, but rather reflect their effect on the probability density defined by the model. The positive term increases the probability of training data (reducing the corresponding free energy), while the negative term decreases the probability of samples generated by the model.

While the second term is straightforward to compute, the first term is usually difficult, since it is often intractable to integrate over the model distribution. The expectation over all possible configurations of the input x (under the distribution M formed by the model) $E_M[\frac{\partial \mathcal{F}(x)}{\partial \theta}]$ has to be calculated. In order to make the computation tractable, sampling can be used. The expectation using a fixed number of model samples is calculated. The gradient is written as:

$$-\frac{\partial \log p(x)}{\partial \theta} \approx \frac{\partial \mathcal{F}(x)}{\partial \theta} - \frac{1}{|\mathcal{N}|} \sum_{\tilde{x} \in \mathcal{N}} \frac{\partial \mathcal{F}(\tilde{x})}{\partial \theta}. \tag{2.42}$$

The elements $\tilde{x}$ of $\mathcal{N}$ are sampled according to M. With the help of sampling the equation (2.42) can almost be solved. The missing ingredient, how to extract the negative particles $\mathcal{N}$, will be defined in the next sections.

Restricted Boltzmann Machines

A Restrictive Boltzmann Machine [Smolensky, 1986] is a particular form of a log-linear Markov Random Field (MRF) for which the energy function is linear in its free parameters. In order to represent complicated distributions some variables of the model are never observed. These variables are called hidden variables. A Restrictive Boltzmann Machine has no hidden $\mapsto$ hidden, and no visible $\mapsto$ visible connections. This is the main difference between a Restrictive Boltzmann Machine and the standard Boltzmann Machine model (BM). A graphical interpretation of a RBM is shown in figure 2.5.2.

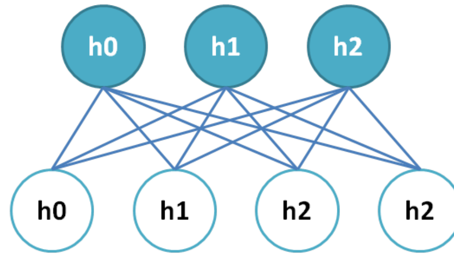


Figure 2.8.: Restricted Boltzmann Machine

The energy function of a Restricted Boltzmann Machine is defined as [Larochelle and Bengio, 2008]:

$$E(v, h) = -b'v - c'h - h'Wv \tag{2.43}$$

The parameter W represents the weight connections between the hidden and visible units. The values b and c are the offsets of the visible and hidden units respectively. The energy formula of a RBM can be written as:

$$F(v) = -b'v - \sum_i log \sum_{h_i} e^{h_i(c_i + W_i v)} \tag{2.44}$$

Due to the fact that the hidden and visible units of the RBM are conditionally independent, the energy formula of equation (2.44) can be rewritten to the form [Larochelle and Bengio, 2008]:

$$p(h|v) = \prod_i p(h_i|v) \tag{2.45}$$

$$p(v|h) = \prod_j p(v_j|h) \tag{2.46}$$

As stated before, a common form of a RBM is a binary Restricted Boltzmann Machine. In this case all the visible and hidden units are of the form $v, j \in 0, 1$. The neuron activation function can be defined as [Larochelle and Bengio, 2008]:

$$P(h_j|v) = sigm(c_i + W_i v) \tag{2.47}$$

$$P(v_j|h) = sigm(b_i + W'_j h) \tag{2.48}$$

As a consequence the free energy function can be simplified to the following equation:

$$F(v) = -b'v - \sum_i log(1 + e^{c_i + W_i v}) \tag{2.49}$$

Combining eqs. (2.42) with (2.49), the following log-likelihood gradients for an RBM with binary units can be obtained:

$$-\frac{\partial \log p(v)}{\partial W_{ij}} = E_v[p(h_i|v) \cdot v_j] - v_j^{(i)} \cdot sigm(W_i \cdot v^{(i)} + c_i) \tag{2.50}$$

$$-\frac{\partial \log p(v)}{\partial c_i} = E_v[p(h_i|v)] - sigm(W_i \cdot v^{(i)}) \tag{2.51}$$

$$-\frac{\partial \log p(v)}{\partial b_j} = E_v[p(v_j|h)] - v_j^{(i)} \tag{2.52}$$

Training Restricted Boltzmann Machines

In order to generate samples of $p(x)$ Gibbs sampling can be used. In general a Markov Chain with using Gibbs sampling as an operator is run to convergence. When sampling the joint of N random variables $S = (S_1, ..., S_N)$, N sampling sub-steps of the form $S_i \sim p(S_i|S_{-i})$, where S_{-i} contains the $N-1$ other random variables in S excluding S_i, are needed [Hinton, 2005].

Regarding RBMs, S consists of the set of visible and hidden units. However, since they are conditionally independent, block Gibbs sampling is performed. In this case, the visible units are sampled simultaneously given fixed values of the hidden units, and vice versa.

One step in the Markov chain is thus taken as follows:

$$h^{(n+1)} \sim sigm(W'v^{(n)} + c) \tag{2.53}$$

$$v^{(n+1)} \sim sigm(Wh^{(n+1)} + b) \tag{2.54}$$

The parameter $h^{(n)}$ refers to the set of all hidden units at the nth step of the Markov chain. Values of $h_i^{(n+1)}$ are randomly chosen to be 1 (versus 0) with probability $sigm(W_i'v^{(n)} + c_i)$, and similarly, $v_j^{(n+1)}$ is randomly chosen to be 1 (versus 0) with probability $sigm(W_{.j}h^{(n+1)} + b_j)$.

The sampling procedure can be visualized graphically for better understanding [Hinton, 2005]:

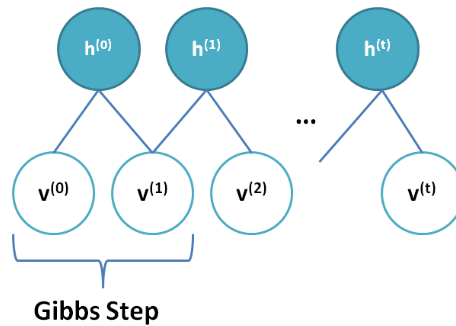


Figure 2.9.: Block Gibbs Sampling in a RBM

For the case $t \to \infty$ the samples $(v^{(t)}, h^{(t)})$ are guaranteed to be accurate samples of $p(v, h)$.

However in theory, each parameter update in the learning process would require running one such chain to convergence, leading to a prohibitively expensive learning mechanism. As a consequence, several algorithms have been devised for RBMs, in order to efficiently sample from $p(v, h)$ during the learning process.

One of the most popular algorithms is Contrastive Divergence (CD-k). Two tricks are involved to speed up the sampling process in this case:

- **(i)** The Markov chain is initialized with a training example in order to get a close estimate of $p_{train}(v)$ for $p(v)$
- **(ii)** CD stops the sampling process before the chain has converged. The samples are obtained after only k-steps of Gibbs sampling, a trick which works surprisingly well in practice.

Training Deep Belief Networks

Deep Belief Networks are trained in greedy layer-wise unsupervised procedure, which is applied to DBNs with RBMs as the building blocks for each layer [Hinton and Salakhutdinov, 2006b], [Hinton et al., 2006]. The process is defined as follows:

- **(i)** Train the first layer as an RBM modeling the raw input $x = h^{(0)}$ as its visible layer.
- **(ii)** Use the first layer to obtain, representing a model of the input as data for the second layer. This representation can be either the mean activations $p(h^{(1)} = 1|h^{(0)})$ or samples of $p(h^{(1)}|h^{(0)})$.
- **(iii)** Train the second layer as an RBM, taking the new input data as training examples.
- **(iv)** Iterate **(ii)** and **(iii)** for the desired number of layers, each time propagating upward either samples or mean values.
- **(v)** Fine-tune all the parameters of this deep architecture with respect to a supervised training criterion.

If a single Markov chain is not initialized again at the chain start and the chain is not restarted for each observed example, the training procedure is called persistent contrastive divergence [Tieleman, 2008]. The general intuition behind this idea is that if parameter updates are small enough compared to

the mixing rate of the chain, the Markov chain should be able to “catch up” to changes in the model.

2.6. Gradient Learning Methods for CTC Models

In order to learn sequences with a CTC output layer, a learning method with the capability of handling noisy target predictions is needed. [Graves and Schmidhuber, 2005] and [Graves et al., 2009] showed that RNN- and LSTM-CTC networks can be successfully trained with stochastic gradient descent [Bishop, 1996] or resilient backpropagation [Riedmiller and Braun, 1993]. These algorithms overcome local minima and are able to handle faulty target predictions generated by the CTC output layer at the beginning of the training due to its adaptive step-size. However, various other gradient based learning methods are able to solve the optimization problem of a CTC network. Figure 2.10 visualizes possible algorithms in respect to their computational complexity.

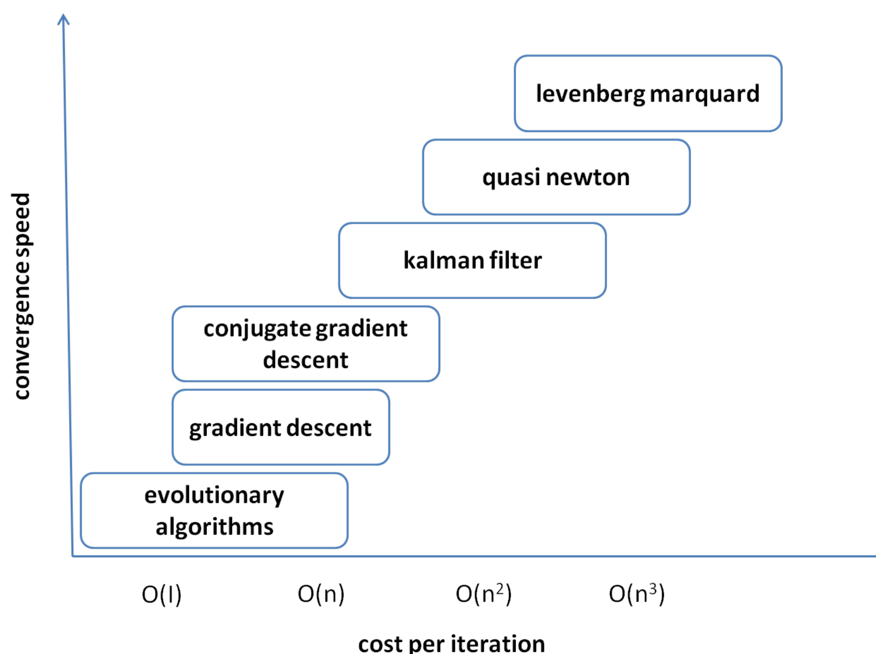


Figure 2.10.: Optimization algorithms for neural classifiers

The next chapters provide the mathematical background of selected gradient based training methods. Chapter 2.6.1 and 2.6.2 provide information about stochastic gradient descent and the resilient backpropagation algorithm; two first order gradient methods. In chapter 2.6.3 and 2.6.4 the Stochastic Diagonal Levenberg-Marquard Method [LeCun et al., 1998a] and the Hessian Free

Method [Martens, 2011b], two powerful second order gradient methods are analyzed. Positive and negative characteristics and their learning capabilities when applied to CTC networks are discussed. Details about the performance of these algorithms and the underlying network model can be found in chapter 4.1.

2.6.1. Stochastic Gradient Descent

Stochastic gradient descent is a common technique to optimize objective functions in machine learning problems. For a specific parameter set θ of a model the given loss function $f(\theta)$ is minimized. This is done with an iterative stochastic way, where data-samples are picked randomly from a common training set. This stands in contrast to batch-learning where the gradient of the loss function given the weights is calculated in minibatches. The general update rule for the model parameters for stochastic gradient descent is:

$$\theta^{k+1} = \theta^k - \eta \nabla f(\theta^k) \tag{2.55}$$

The algorithm makes small steps downward on an error surface defined by a loss function of some parameters θ. The value η is the learning rate, controlling the step-size along the current error direction. Although stochastic gradient descent is a very simple approach solving the mostly non-convex optimization problem of the underlying model the approach is highly efficient, when it comes to convergence speed. Although the gradient estimates are very noisy the algorithm usually converges much faster than batch gradient methods [LeCun et al., 1998b]. Stochastic learning very often results in better solutions, a phenomena which also occurs when testing CTC models in chapter 4.1. This has to do with noisy gradient estimates which might result in a better parameter update, as a better minimum than the current one will be discovered [LeCun et al., 1998b]. Apart from that, stochastic learning methods are able to track changes in the data. The downside of the on-line learning technique should be pointed out as well. In very complex models and complex learning tasks stochastic gradient descent might not be able to find a suitable solution of the underlying problem in appropriate time. In this case the fluctuations of the weight updates are so dramatic that the algorithm will converge very slowly to a minimum, or not at all. This especially happens when descending long narrow valleys on the error surface [Martens, 2011b].

Figure 2.11.: Gradient descent with momentum

Stochastic gradient descent can be accelerated with the help of a momentum term. The use of a momentum term should help to overcome local minima [Bishop, 1996] and is an extension of the standard gradient descent learning algorithm described above. The update equation to compute a parameter change of θ with the help of an objective function f is defined as:

$$\Delta\theta^{k+1} = \alpha\Delta\theta^{k} + (1-\alpha)\nabla f(\theta^{k}) \tag{2.56}$$

The idea of this version of gradient descent is to compute on-the-fly (*on-line*) a moving average of the past gradients, and use this moving average instead of the current example's gradient, in the update equation. The momentum tern α is a hyper-parameter that controls how much weight is given to older parameter updates. The update equation of stochastic gradient descent with momentum is the same as defined in equation (2.55).

2.6.2. Resilient Backpropagation

Resilient backpropagation [Riedmiller and Braun, 1993] evaluates the sign of the last gradient and adapts its step-size according to this direction. It is a batch update algorithm, meaning that good learning results will be achieved with small minibatches. The parameter update is done according to the formula:

$$\gamma^{k+1} = \begin{cases} min(\gamma^{k} \cdot \eta^{+}, \gamma_{max}) & \text{if } \nabla f(\theta^{k}) \cdot \nabla f(\theta^{k-1}) > 0 \\ max(\gamma^{k} \cdot \eta^{-}, \gamma_{min}) & \text{if } \nabla f(\theta^{k}) \cdot \nabla f(\theta^{k-1}) < 0 \\ \gamma^{k} & \text{other} \end{cases}$$

where $0 < \eta^{-} < 1 < \eta^{+}$

The parameters $\eta^+, \eta^-, \gamma_{min}, \gamma_{max}$ define the positive and negative step values and their maxima and minima. The change of parameters for the objective f is calculated as:

$$\Delta \theta^k = -\gamma^k sgn(\nabla f(\theta^k)) \tag{2.57}$$

[Igel et al., 2000] defines four versions of the resilient backpropagation algorithm, namely *rprop+*, *rprop-*, *irprop+*, *irprop-*[2]. Rprop+ is very similar to the original Reidmiller implementation, with the exception of reverting the previous iteration's parameter change if the sign of the gradient changes in the current iteration. In the *irprop-* algorithm no parameter backtracking is used. Furthermore, the last gradient is set to zero when the gradient changes its sign. The parameter update of θ is done in the same way as standard gradient descent $\theta^{k+1} = \theta^k + \Delta\theta^k$.

2.6.3. Stochastic Diagonal Levenberg-Marquard Method

The Stochastic Diagonal Levenberg-Marquard Method is a second order gradient method which calculates the diagonal Hessian through a running estimate of the second derivative (Hessian) with respect to each parameter θ. The estimates of the derivatives are used to calculate (scale) the learning rate of each individual parameter. The equation defining the specific learning rates is [LeCun et al., 1998b], [LeCun et al., 1999]:

$$\eta_{ki} = \frac{\epsilon}{\frac{\partial^2 L}{\partial^2 \theta_{ki}} + \mu} \tag{2.58}$$

The parameter ϵ is the global learning rate. The value ϵ is the current learning rate for each individual parameter and μ is a regularization parameter preventing η_{ki} of becoming too high. The running estimate of the second order derivative is calculated with the equation [LeCun et al., 1998b]:

$$\langle \frac{\partial^2 L}{\partial^2 \theta} \rangle_{s+1} = (1-\gamma) \langle \frac{\partial^2 L}{\partial^2 \theta} \rangle_{s-1} + \gamma \langle \frac{\partial^2 L}{\partial^2 \theta} \rangle_s \tag{2.59}$$

[2] rprop+, irprop- are implemented in the software framework found in appendix A

The value γ is a small amount of "memory" storing the old values of the Hessian. [LeCun et al., 1998b] notes that the Stochastic Diagonal Levenberg-Marquard Method is about three times faster than standard stochastic gradient descent, due to the clever on-line adaption of the individual learning rates.

The update rule of the parameter *theta* is the same as the update rule in standard stochastic gradient descent, defined in equation (2.55).

2.6.4. Hessian Free Method

The Hessian Free Method is a second order gradient method developed by [Martens, 2010]. The optimization algorithm combines the truncated Newton Method [Nocedal and Wright, 2000] using the Gauss-Newton approximation of the Hessian matrix, with a Levenberg Marquard damping heuristic [Marquardt, 1963].

The Hessian Free Method was first applied to Deep Auto-Encoders [Martens, 2010] and outperforms pre-trained Auto-Encoders described in [Hinton and Salakhutdinov, 2006a]. In another approach the Hessian Free technique was successfully applied to recurrent neural networks. [Martens, 2011b] showed that the combination of the Hessian Free Method and the use of a additional regularization term, called *structural damping*, helps to solve long time dependency problems and overcomes the vanishing gradient problem in RNNs.

Truncated Newton Method

The Truncated Newton Method [Nocedal and Wright, 2000] belongs to the group of second order optimization methods. It is an inexact Newton approach with the goal of keeping the computational cost as small as possible. The central idea of this method is that the objective f can be locally approximated around $\theta \in R^N$ up to the second order with the Taylor expansion [Martens, 2010]:

$$f(\theta + p) \approx q_\theta(p) \equiv f(\theta)^T + \nabla f(\theta)^T p + \frac{1}{2} p^T B p \tag{2.60}$$

$B = H(\theta) = \nabla f(\theta)^2$ is the Hessian of f given θ. The value p is the search direction. Considering the right-hand side of the above equation as a quadratic

with constant coefficients, the equation defined above can be rewritten the Newton criteria for optimization [Nocedal and Wright, 2000]:

$$\nabla^2 f(\theta)p = -\nabla f(\theta) \tag{2.61}$$

Very often the Linear Conjugate Gradient Method (CG) is used to solve the equation defined above. This method was proposed by Hestenes and Stiefel in the 1950s as an iterative method for solving linear systems with positive definite coefficient matrices. A pseudo code of a linear CG[3] is shown below [Nocedal and Wright, 2000]:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat
    $\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$
    $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$
    $\beta_k := \frac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$
    $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
    $k := k + 1$
    if r_{k+1} is sufficiently small then **exit** loop **end if**
end repeat
result is x_{k+1}

Algorithm 2: Linear CG pseudo code

CG iteratively updates the parameter θ of an objective function f by computing the search direction p and updating the parameter θ as $\theta + \alpha p$. The exact solution of the Newton direction p_k^N might not be calculated, as the CG iteration is stopped if a defined stopping criteria (cf. chapter 2.6.4) is met. Global convergence will be guaranteed if search direction p_k is a descent direction, which will be true if the Hessian $\nabla^2 f(\theta)$ of the objective f is positive definite [Nocedal and Wright, 2000]. The convergence speed can be improved with the help of preconditioning (cf. chapter 2.6.4).

[3] In the case of Truncated Newton the given system equation $Ax = b$ must be replaced with the defined Newton parameter of (2.61)

A graphical interpretation of CG is shown in figure 2.12. The left side shows a standard steepest descent optimization, the right side shows the CG-method.

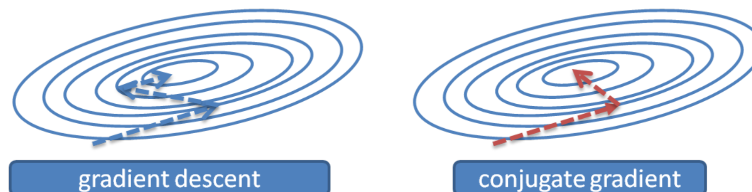


Figure 2.12.: Conjugate gradient method

Gauss-Newton approximation

The truncated Newton method is able to optimize a objective function if the Hessian of this function is positive semidefinite [Nocedal and Wright, 2000]. One way of overcoming the obstacle of indefinite Hessians is the use of the Gauss Newton approximation of the Hessian matrix. In this case, the standard Newton equation is replaced with the term:

$$J_k^T J_k p_k^{GN} = -J_k^T r_k \tag{2.62}$$

The Gauss-Newton matrix is positive definite if the loss function is convex [Chapelle and Erhan, 2011]. J, the Jacobian, is the partial derivative of the underlying objective function. Modifying the terms above also leads to a surprising number of advantages over the plain Newton Method. Firstly, the use of the approximation $\nabla^2 f_k \simeq J_k^T J_k$ saves the trouble of computing the individual Hessians, which might be computational expensive. Secondly, the vector-matrix product of p and the Gauss-Newton Matrix can be calculated efficiently using the **R operator** [Pearlmutter, 1994], [Schraudolph, 2002]. The Gauss-Newton approximation of the Hessian matrix is used in the HF algorithm [Martens, 2010].

Tikhonov regularization

[Martens, 2010] applied the truncated Newton Method when defining the HF algorithm. Apart from that an **additional regularizer** known as **Tikonov damping** [Tikhonov and Arsenin, 1977] was used to damp the Hessian matrix.

This is necessary, as in practice when using large values of p, the approximation of f might not be very trustworthy. A Tikonov regularizer, a method to recondition the Hessian matrix is defined as [Martens, 2010]:

$$B_n(d) = H(\theta_n)d + \lambda d \tag{2.63}$$

The parameter d is the descent direction. Figure 2.13 visualizes this descent direction on a low curvature. The use of this additional curvature information is controlled by the defined damping parameter λ of the Tikonov regularization. The parameter regularizes the sensitivity of the new descent direction [Martens, 2010].

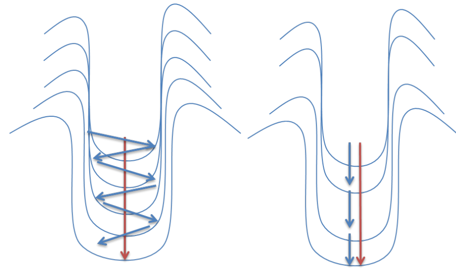


Figure 2.13.: Optimization in a long narrow valley

In the initial Hessian Free implementation a Levenberg-Marquardt damping heuristic for adjusting λ is used [Martens, 2010]. It is defined as:

$$\text{if } \rho < \frac{1}{4} : \lambda \leftarrow \frac{3}{2}\lambda \text{ elif } \rho > \frac{3}{4} : \lambda \leftarrow \frac{2}{3}\lambda \tag{2.64}$$

The value ρ is the reduction ratio, a scalar quantity which attempts to measure the accuracy of the locally approximated objective q_θ (eq. 2.60) and the true objective. It is defined as [Martens, 2010]:

$$\rho = \frac{f(\theta + \rho) - f(\theta)}{q_\theta(\rho) - q_\theta(0)} \tag{2.65}$$

Finding the correct λ and ρ is very important for finding the correct solution of the optimization problem. If λ has a value of zero, the optimum of the quadratic function approximation is inside the trust region, in which case a trust-region-scaled Newton step is performed. If λ is greater than zero when the optimum is at the boundary of the trust region, in which case the scaled

Newton step is too long to fit in the trust region a quadratically-constrained optimization is done. Summarizing the things above large λ values indicate optimization difficulties [Martens, 2011b]. Negative values indicate the special case of an indefinite Hessian matrix [Gay, 1983].

Structural damping

When applying the original definition of the Hessian Free Method to recurrent neural networks, the Tikhonov regularizer defined in chapter 2.6.4 did not seem to achieve good results on **certain** long-term dependency problems. [Martens, 2011b] introduced a new additional regularizer which helped to reduce the effect of vanishing gradients of RNNs. This regularizer is called *structural damping* and adds a penalty to the recurrent neural network's cost function if there are **high fluctuations** in the **hidden activation** terms of the model.

The key idea of this method is that for certain small changes in the parameter θ of the models there can be large and highly non-linear changes in the hidden state sequence h. This leads to a bad local approximation of $f(\theta)$ (eq. 2.60). The parameter λ of the Levenberg Marquard damping strategy will get very high in order to compensate this behavior. This is an unwanted side effect, as high λ reduce the optimization to a first order method, leading to a poor performance of HF [Martens, 2011b].

Structural damping, when applied to recurrent neural network, improves the learning of long term dependencies [Martens, 2011b]. The penalty term which is added to the cost function of the model can be defined as the difference between two state vectors, in this case, the hidden states of a recurrent neural network. A common measure depending on the appropriate distributional assumptions on the input given the code, e.g., is the traditional squared error [Bishop, 1996]

$$L(\mathbf{x}, \mathbf{z}) = ||\mathbf{x} - \mathbf{z}||^2, \tag{2.66}$$

or, when dealing with vectors of bit probabilities, the cross entropy [Bishop, 1996]:

$$\mathrm{H}(p, q) = -\sum_{x} p(x)\, \log q(x) + (1 - t) \cdot log(1 - x) \tag{2.67}$$

In this work the cross entropy was used as a reconstruction measure.

Preconditioning

Preconditioning is a method to accelerate CG [Nocedal and Wright, 2000]. The increased speedup is achieved by a linear change of variables x̂=Cx for some matrix C. The transformed quadratic objective of $q(\theta)$ (eq. 2.60) is defined as:

$$\hat{\theta}(\hat{x}) = \frac{1}{2}\hat{x}^{-T}AC^{-1}\hat{x} - (C^{-1}b)^T \tag{2.68}$$

leading to a system equation which should be easier to solve [Martens, 2010].

The term $\mathbf{M} = \mathbf{C^T C}$ is called **preconditioner**. There are various types of different preconditioners. [Martens, 2010] used the diagonal Fisher information matrix to substitute the diagonal elements of the Gauss-Newton matrix. This preconditioner, denoted as *martens* preconditioner in future, is defined as:

$$M = [diag(\sum_{i=1}^{D} \nabla f_i(\theta) \odot \nabla f_i(\theta)) + \lambda I]^{\alpha} \tag{2.69}$$

$\odot$ is element-wise product f_i is the value of the objective function at training-case i and the exponent α is a scalar value < 1, used to suppress extreme values.

Another preconditioner used in the experimental part in chapter 4.1 is the '*Jacobi*' preconditioner. This preconditioner is defined as:

$$C_{ij} = \begin{cases} A_{ii} & \text{, if } i = j \\ 0 & \text{, otherwise} \end{cases} \tag{2.70}$$

It uses the diagonal entries of a given matrix A. Details how to efficiently compute the values of the diagonal elements of the Gauss-Newton matrix for the jacobi preconditioner can be found in [Chapelle and Erhan, 2011].

Figure 3 shows a pseudo code of the CG method, adapted for preconditioning [Nocedal and Wright, 2000]:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{z}_0 := \mathbf{M}^{-1}\mathbf{r}_0$
$\mathbf{p}_0 := \mathbf{z}_0$
$k := 0$
repeat
  $\alpha_k := \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{z}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$
  $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
  $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
  if r_{k+1} is sufficiently small then **exit** loop **end if**
  $\mathbf{z}_{k+1} := \mathbf{M}^{-1}\mathbf{r}_{k+1}$
  $\beta_k := \frac{\mathbf{z}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{z}_k^{\mathrm{T}}\mathbf{r}_k}$
  $\mathbf{p}_{k+1} := \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$
  $k := k + 1$
end repeat
result is x_{k+1}

Algorithm 3: Linear PCG pseudo code

Termination conditions for CG

Terminating a CG run after a couple of CG steps plays an important role when it comes to finding the optimal CG solution. [Martens, 2010] defines the following termination criteria for CG:

$$i > k \text{ and } \phi(x_i) < 0 \text{ and } \frac{\phi(x_i) - \phi(x_i - k)}{\phi(x_i)} < k * \epsilon \qquad (2.71)$$

The scalar k determines how many steps into the past we look in order to compute the estimate of the current per-iteration reduction rate. The value i is the index of the current iteration and ϕ is the quadratic $\phi(x_i) = \frac{1}{2}x^T Ax - b^T x$ where $A = B$ and $b = -\nabla f(\theta)$ [Martens, 2010]; in other words the current solution of CG using minimizing the objective (2.60). Typical value for ϵ is 0.0005 and k=max$(10, 0.1 * i)$. This stopping criteria and the same parameters were also used in the HF algorithm developed for CTC networks.

CG iteration backtracking

The Hessian Free Method optimizes the value of p in respect to the 2nd order model $q_\theta(p)$. These improvements do not necessarily reflect the value of $f(\theta+p)$ [Martens, 2010]. If the current solution of a CG step is worse then the previous solution, the parameter values are backtracked until a lower value of $f(\theta + p)$ than the current one has been found. A backtracking method, also known as *line-search*, is implemented in the HF algorithm.

Algorithm definition

Summarizing the algorithmic and mathematical subtleties of the HF method, figure 4 shows a **pseudo code** of the implemented **HF algorithm** adapted for solving sequence labeling problems with CTC. The algorithm of [Martens, 2010] was slightly changed to fit into a CTC framework and follows the definition of [Chapelle and Erhan, 2011]. The method calculates the gradient and the Gauss-Newton matrix over a single minibatch in one CG run. A CTC prediction of the current minibatch is calculated before applying the HF algorithm to the model. The starting vector in each new CG step is initialized with the previous search direction. If the objective function value increases this starting vector is reset to zero. A CG stopping criteria as defined in [Martens, 2010], also taking a CG iteration maximum into account, was implemented. The *structural damping* regularizer, as defined in chapter 2.6.4 does not appear in the algorithm definition since it is added to the cost function of the network directly. Details about the algorithm performance can be found in the experimental part in chapter 4.1.

```
% Parameters
g ... gradient
f ... objective
d ... search direction
p ... parameter update
t ... target
λ ... damping factor
μ ... structural damping factor
ρ ... Levenberg-Marquardt reduction ratio
θ ... parameters of the model
B ... damped Gauss-Newton matrix

% Initialization
λ ← 1 (or some other initialization)
d ← 0
μ ← λ * μ_0 (updated with λ)

% Loop over mini-batches
for minibatch in train-batches

    % CTC
    t = calculate CTC target prediction

    % Tikhonov damping
    g = ∇f(θ)
    B(d) = H(θ)d + λd

    % Truncated Newton
    −p = solve(B, −g) using CG

    % Backtracking
    if f(θ + p) > f(θ) then
        Go back to the previous step
    end if

    % Levenberg Marquardt heuristic
    ρ ← (f(θ+p) − f(θ)) / (q_θ(p) − q_θ(0))
    λ ← (3/2)λ if ρ < 1/4
    λ ← (2/3)λ if ρ > 3/4

    % Parameter update
    θ ← θ + p
```

Algorithm 4: Hessian Free pseudo code

3. Optimized Learning of Recurrent Models

3.1. Defining a Classifier Landscape

In the previous chapters an objective function, a model, and various training algorithms have been defined. A classifier for solving the sequence labeling problem can now be described with the following graphic of figure 3.1. In case of this work, the objective function which is minimized is the negative log-likelihood of the CTC output layer described in chapter 2.4.

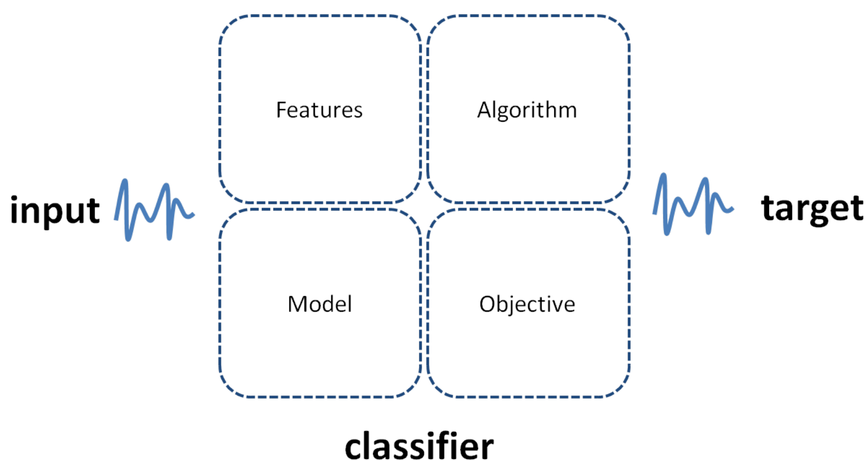


Figure 3.1.: Definition of a classifier

The model which is used is a RNN (2.5.1). The optimization methods described in chapter 3.1 can be used to learn input $\mapsto$ target mappings. Features extracted from the input sequence will be generated by the model itself.

3.2. Exploring Optimizations for a CTC Classifier

In the last decades very powerful mechanisms to classify patterns have been developed. The two models which stand out the most are the Deep Convolutional Network [LeCun et al., 1999] and the Deep Belief Network [Hinton and Salakhutdinov, 2006b].

The first model is a variant of a Multilayer Perceptron and is inspired by the visual cortex [Hubel and Wiesel, 1968]. It exploits spatial local correlation in the data with the help of local convolutional filter operations. Another important concept is non-linear down-sampling (max-pooling), an operation which reduces the computational complexity for upper layers.

The second architecture, the Deep Belief Network (chapter 2.5.2), models the input distribution $P(x)$ in a pre-training phase. This is done with the help of a layer-wise greedy training algorithm (chapter 2.5.2). The learned deep representations allow a higher level of abstraction of the data, leading to a good initialization of the network weights. Modeling $P(x)$ seems to be helpful when learning $P(y|x)$ as a very powerful classifier is obtained after fine-tuning the model.

Basically, it seems to be helpful to use a form of Deep Architecture for CTC labeling problems as a very generalized representation of the data is learned, leading to better recognition rates in the end. Nevertheless, a few tricky things have to be considered: The training of (hierarchical) Recurrent Neural Networks is a very difficult problem due to the vanishing gradient problem (chapter 2.5.1). Learning $P(y|x)$ directly might not be possible although various algorithms to overcome this problem have been presented (chapter 3.1). Using a non-recurrent model is not a solution either, as the memory mechanisms of a RNN are needed to learn correlations of input frames over the time. If the direct learning of $P(y|x)$ is not possible, the use of Deep Belief Networks might be the method of choice due to the good weight initialization obtained by unsupervised pre-training. This might help to overcome local minima, even if Recurrent Neural Networks are used instead of standard Feed-Forward Networks.

The next chapters analyze learning effects of RNNs with a CTC objective systematically. First gradient based algorithms (chapter 3.1) are compared in

order to find a suitable training method for the underlying problem. Next learning effects of hierarchical RNNs are analyzed. This section again compares the presented gradient algorithms using a more complex model. The next section covers generative pre-training of hierarchical RNNs, in order to simplify the highly non-convex problem. The made assumption about Deep Architectures is verified here. Chapter 4.1.5 compares the obtained results with different state-of-the-art neural classifiers.

In order to use a DBN for stacked hierarchical RNNs, the following training procedure was developed:

(i) pre-train the stacked RBMs in a greedy manner (*e.g. CD*)
(ii) replace the input weights of each RNN with the weights of the RBM
(iii) initialize the recurrent weights of each RNN with small gaussians
(iv) fine-tune the model with stochastic gradient descent (or any other suitable training algorithm)

4. Toy Experiments

4.1. Classifying concatenated MNIST Digits

This chapter presents the results of various toy experiments with CTC networks carried out with the MNIST database [Lecun and Cortes, 2012].

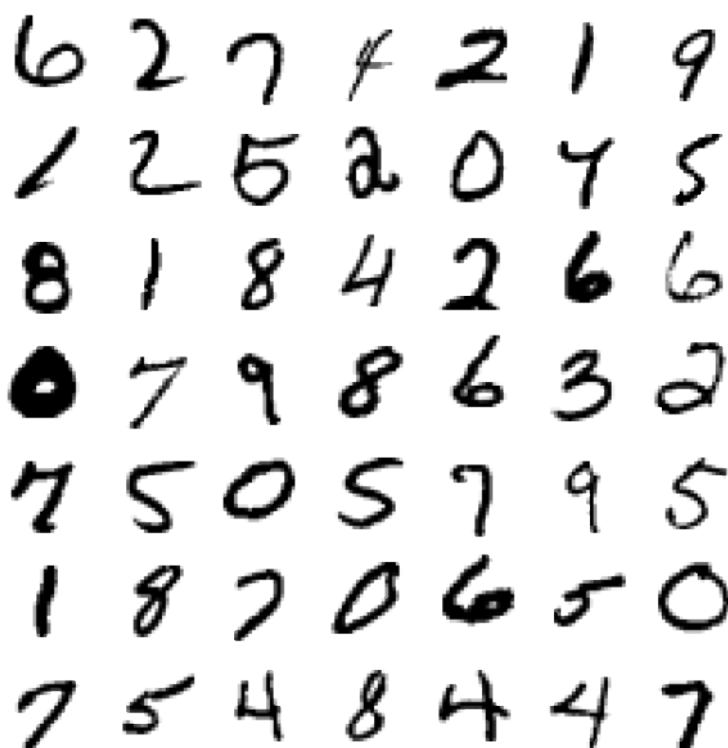


Figure 4.1.: MNIST digits

In section 4.1.2 the optimization algorithms presented in chapter 3.1 are analyzed in detail using a RNN with a CTC output layer. This simple recurrent model is extended to a hierarchical architecture and possible training methods based on the assumptions made in chapter 3.2 are evaluated. Section

4.1.5 compares the performance of a hierarchical RNN optimized by generative pre-training (chapter 2.5.2) and fine-tuned with stochastic gradient descent (chapter 2.6.1), with other classifiers on the MNIST database.

4.1.1. Data Preparation

The MNIST database of single handwritten digits is a subset of a larger set available from NIST [Lecun and Cortes, 2012]. It consists of a training set of 60,000 digit samples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image of 28×28 pixels.

In order to learn sequence dependencies from the MNIST data set with a CTC network, sequences of concatenated digits are needed. For all experiments 10 concatenated MNIST digits were used as a single sequence. This leads to 6000 training sequences and 1000 test sequences. A single sequence is 280×28 pixels long. Each target sequence consists of 10 digits.

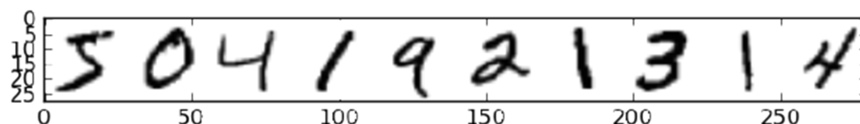


Figure 4.2.: Concatenated MNIST digits

All images were normalized to lie in the range [0-1]. No further preprocessing method, like elastic- or random-distortions [Ciresan et al., 2010], [Lauer et al., 2007] or the addition of Gaussian noise [Graves et al., 2009] was applied to the data.

The normalized Levenshtein distance [Damerau, 1964] was used to measure the difference between the predicted CTC labels and the target labels:

$$\text{error} = 100 * \frac{\sum_{n=0}^{N} \text{lev. distance}}{\sum_{n=0}^{N} \#\text{characters}} \qquad \text{N ... number of sequences} \tag{4.1}$$

4.1.2. Comparison of Gradient Based Optimization Algorithms

This section presents recognition results of different gradient training algorithms, which were also presented in chapter 3.1. All optimization methods were tested on the adapted MNIST dataset on a RNN model with a CTC output layer.

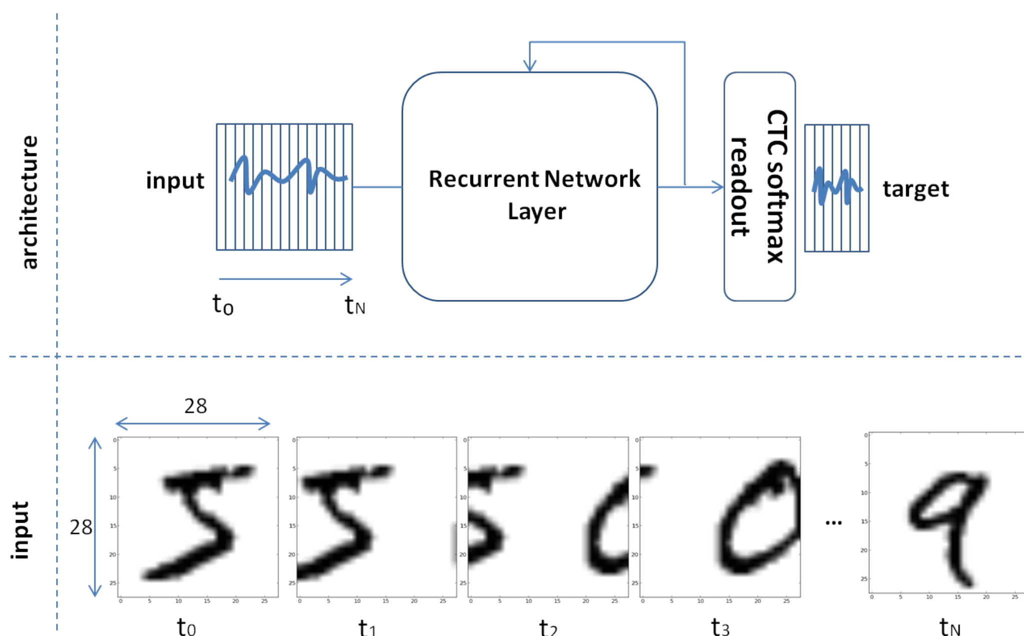


Figure 4.3.: RNN-CTC architecture with 40 frames MNIST training example

Sequences of 10 concatenated MNIST digits were used as a network input. Each sequence was split to chunks of 28 × 28 pixels shifted by 7 pixels per time-step. In this case a training sequence consists of 40 frames and has a corresponding target sequence of the length 10. The training was stopped after the early-stopping criteria was met or the number of epochs exceeded the maximum. A batch size of 500 training examples were used for the batch learning algorithms. This batch size achieved a very stable error curve when testing the algorithms. Table 4.1 summarizes the parameter settings the model. Table 4.2 shows the individual learning parameters for each optimization algorithm.

dataset	parameters	
MNIST	train=5000 sequences	*10 digits/sequence*
	valid=1000 sequences	*10 digits/sequence*
	test=1000 sequences	*10 digits/sequence*
model	**parameters**	
RNN	input=784	28×28 | 7 shift
	hidden=100	
	output=11	10+1 labels
	output layer=CTC	
	weights=*gaussian*	$\mu = 0, \sigma = 0.1$
	activation=*logistic sigmoid*	
	decoding=*best-path*	

Table 4.1.: Model parameters of the RNN

training	parameters	
	early-stopping=50,max-epochs=500,validation-frequency=5	
algorithm	**batch size**	**parameters**
SGD	1	η=0.01
SLM	1	η=0.1
RPROP+	500	α=0.1, η+=1.2, η-=0.5, eta_{max}=0.2, eta_{min}=1e-6
IRPROP-	500	α=0.1, η+=1.2, η-=0.5, eta_{max}=0.2, eta_{min}=1e-6
HF	500	cg-iter-max=5, preconditioner=None, $\mu = 0.123$, $\lambda = 1$

Table 4.2.: Algorithm parameters of the RNN

Table 4.12 lists the average train-, validation- and test-errors of 3 test runs. In every run the same weight initialization was used for every algorithm. This step ensures that bad weight initializations would not bias the result. The training sequences were permuted randomly in every epoch.

When comparing the results of different batch and on-line training methods in table 4.12 it can be seen that the on-line optimization algorithms clearly outperform the batch methods when it comes to convergence speed. The CTC output layer is able to adapt its predictions more often due to the on-line

adapted MNIST (28 × 28 | 7)				
algorithm	**error**			
	train [%]	*valid* [%]	*test* [%]	*epoch* [1]
SGD	0.0±0	2.23±0.11	2.27±0.1	67±20
SLM	0.0±0	2.40±0.12	2.34±0.07	48±37
RPROP+	1.61±1.37	3.89±0.47	3.82±0.4	252±163
IRPROP-	1.66±1.4	3.99±0.43	3.77±0.47	247±185
HF	**0.23±0.15**	**2.52±0.69**	**2.16±0.09**	**203±77**

Table 4.3.: Test error of different algorithms (RNN) (MNIST)

training and the stochastic methods *'follow'* these predictions much faster by changing the weights of the underlying network. This makes stochastic learning algorithms the method of choice for this sort of output layer. Very good results of Stochastic Gradient Descent (SGD) also verify this assumption. The algorithm achieves very good overall performance and is still the state-of-the-art method to beat. The stochastic Levenberg Marquard Method, designed to accelerate SGD, shows a slightly faster convergence rate, but achieves a worse overall result. Nevertheless it might be a fruitful acceleration technique for future learning tasks.

The Hessian Free Method scored the best overall result. This was possible as this method was slightly adapted to a semi-stochastic version in this work and it is worth to discuss these modifications in more detail.

Various HF parameter settings were tested with the goal of improving the algorithm's performance. Most of the findings correspond to the results presented in [Montavon et al., 2012], an excellent summary of learning effects and pitfalls when training neural networks with the Hessian Free Method.

In this experiment the HF algorithm was able to approximate the objective locally with the second order approximation using a batch size of 500 training examples. Increasing the batchsize to values bigger than 500 makes the optimization of the network possible but leads to a worse recognition rates. This has to with bad generalization properties of batch algorithms [LeCun et al., 1998b], [Wilson and Martinez, 2003] and the fact that more CTC updates seem to be preferable when solving sequence labeling problems due to the on-line adaption of the target signals.

Using the original implementation of the HF Method [Martens, 2011b], [Martens, 2010] the use of smaller minibatches leads to a bad local approximation of the

likelihood objective. The HF parameter λ increases, in order to compensate this behavior. Due to the insufficient curvature information the algorithm suffers from bad noisy fluctuations leading to non-convergence of the method in the worst case. This in particular happens when batchsizes smaller than 10 are used. The use of higher μ values $(0, ..., 1)$, affecting the penalty term of the structural damping regularizer, did not help to solve this problem.

In order to handle this problem the hard limit of maximum allowed CG iterations was fixed to a small value. It was verified by experiment that many CG iterations does not seem to to improve the result on the testset when using small minibatches. Restricting the maximum allowed CG iterations to a small value however improves the recognition rate. The limit $CG_{max} = 5$ of maximum allowed CG-iterations in this experiment achieved the best overall result in multiple test runs. Preferable values should be in the range of $CG_{max} = \{2, 10\}$. From the CTC perspective this change of parameters might also prevent the extraction of wrong features at the beginning of the training as wrong predictions might be generated by the CTC output layer. These wrong predictions might be learned too precisely, leading to a worse long term learning effects.

In order to design a semi-stochastic Hessian Free Method, which might be the method of choice in this learning task, two additional things are worth to mention. In [Martens, 2011b] the current CG starting position of CG was initialized with the previous solution. This information sharing contributes to the success of the algorithm. In a semi-stochastic setup, which was also used in this work, the sharing of information might not be preferable, as noisy information might change things to the worse. It might be better to skip or rescale the last solution. However this has to be tested more accurately. Apart from that, the Levenberg Marquard Heuristic defined in [Martens, 2011b] seems be too strict in a semi-stochastic setup. Softening this criteria to $\{ \text{ if } \rho < \frac{1}{4} : \lambda \leftarrow \frac{100}{99}\lambda \text{ elif } \rho > \frac{3}{4} : \lambda \leftarrow \frac{99}{100}\lambda \}$ reduces the influence of the reduction ratio computed from curvature mini-batches, preventing the heavy increase of the λ parameter.

As far as the use of a preconditioner (chapter 2.6.4) is concerned, it can be said that preconditioning did not improve the recognition rate when using recurrent architectures. This was also reported by [Martens, 2011b] and [Montavon et al., 2012]. The use of an additional structural damping regularizer did not seem to influence all optimization methods as well. This might have to do with the absence of long-term pattern dependencies in the data.

Summarizing the things above, stochastic learning algorithms seem to be the method of choice when using CTC models. This finding can be verified when comparing the results gained in this experiment. On-line optimization methods seem to have better generalization properties and are able to adapt the weight vectors much faster than its batch counterparts. In the case of CTC, the noisy gradient estimates of stochastic methods might also help to uncover hidden labels, as the algorithm will adapt the network weights to a single CTC prediction. On the contrary noisy gradient estimates might also prevent the network to learn the underlying training patterns in some cases. Algorithms might get stuck in a local minima, making optimization of the network impossible. Additional drawbacks like the loss of parallel computation on a GPU and the loss of advanced acceleration techniques like conjugate gradient methods must be taken into account as well. A semi-stochastic Hessian Free Method seems to be a promising candidate for future research.

Figure 4.4 visualizes the best individual test run of the algorithms tested in this toy experiment for completness.

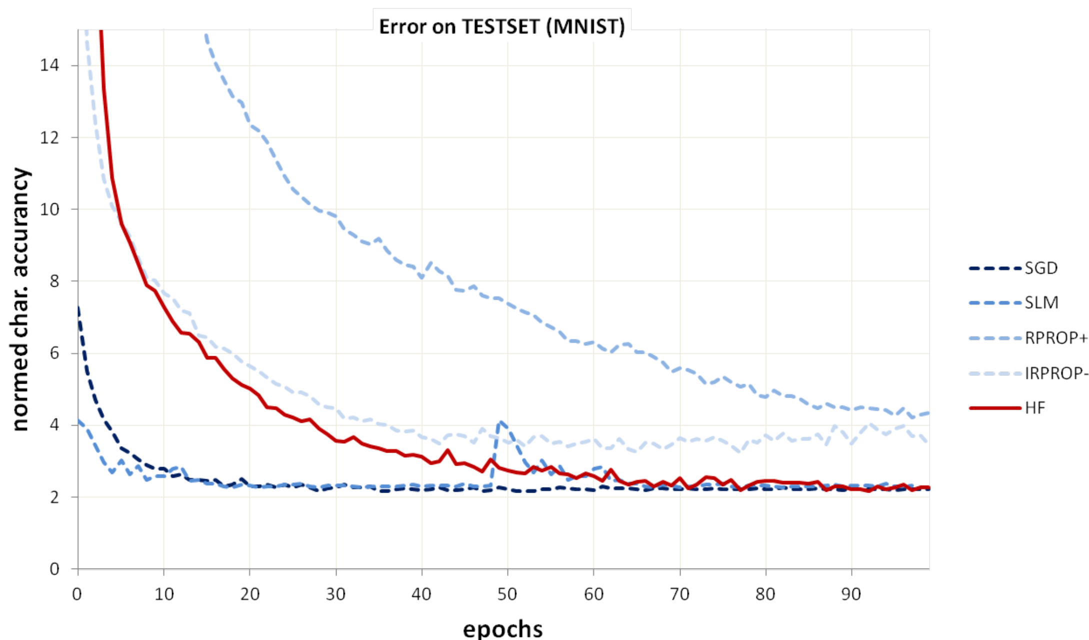


Figure 4.4.: Test error curves of the algorithms (MNIST)

4.1.3. Comparison of Gradient Based Optimization Algorithms for Hierarchical Recurrent Architectures

In the last chapter it was demonstrated that the training of simple recurrent architectures with a CTC objective is possible. The Hessian Free method slightly outperformed other learning algorithms.

In this experiment the same problem is problem is tackled with the help of stacked hierarchical architectures (cp. figure 4.5). It is hoped that, due to the layered structure of the model, more abstract features would be learned, leading to an improved recognition performance on the test set.

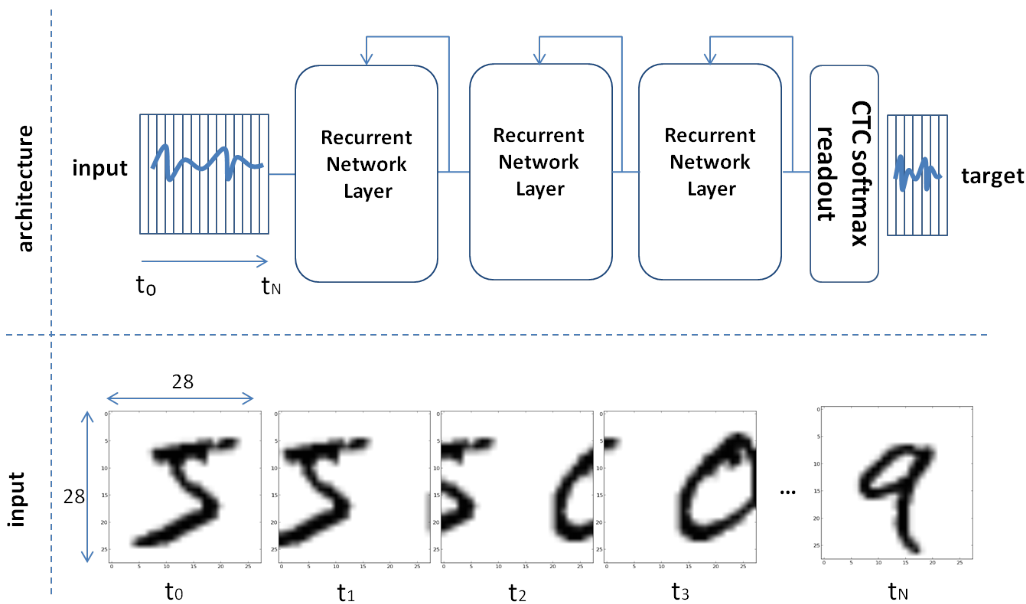


Figure 4.5.: HRNN-CTC architecture with 40 frames MNIST training example

The learning algorithms of chapter 3.1 were applied to a hierarchical RNN with $\{100, 100, 100\}$ layers. Table 4.4 lists the parameters of the used recurrent model. Table 4.5 shows the individual learning parameters for each optimization algorithm. All experiments were carried out 3 times.

dataset	**parameters**	
MNIST	train=5000 sequences	*10 digits/sequence*
	valid=1000 sequences	*10 digits/sequence*
	test=1000 sequences	*10 digits/sequence*
model	**parameters**	
HRNN	input=784	28×28 | 7 shift
	hidden=$\{100, 100, 100\}$	
	output=11	10+1 labels
	output layer=CTC	
	weights=*gaussian*	$\mu = 0, \sigma = 0.1$
	activation=*logistic sigmoid*	
	decoding=*best-path*	

Table 4.4.: Model parameters of the HRNN

training	**parameters**	
	early-stopping=100 max-epochs=500 validation-frequency=5	
algorithm	**batch size**	**parameters**
SGD	1	η=0.01
SLM	1	η=0.1
RPROP+	100	α=0.1, η+=1.2, η-=0.5, eta_{max}=0.2, eta_{min}=1e-6
IRPROP-	100	α=0.1, η+=1.2, η-=0.5, eta_{max}=0.2, eta_{min}=1e-6
HF	100	cg-iter-max=$\{5\}$, precond.=None, $\mu = 0.123$, $\lambda = 1$

Table 4.5.: Algorithm parameters of the HRNN

adapted MNIST (28 × 28 | 7)				
algorithm	**error**			
	train [%]	*valid* [%]	*test* [%]	*epoch* [1]
SGD	100±0	100±0	100±0	100±0
SLM	100±0	100±0	100±0	100±0
RPROP+	100±0	100±0	100±0	100±0
IRPROP-	100±0	100±0	100±0	100±0
HF	**0.19±0.14**	**2.50±0.29**	**2.45±0.32**	**127±21**

Table 4.6.: Test error of different algorithms (HRNN) (MNIST)

Table 4.6 lists the recognition rates of each optimization method. SGD, the Stochastic Diagonal Levenberg Marquard Method and both RPROP variants were not able to solve the optimization problem. Neither the change of the weight initialization scheme to a sparse format allowing 15 random connections per neuron nor the use of different learning rates $\eta = \{0.01, 0.1, 0.5, 1\}$ helped to improve the result. Also attempts to change the batchsize to values of $\{1, 10, 100, 500\}$ and the addition of a structural damping regularizer (chapter 2.6.4) to the cost function were unsuccessful. Nevertheless, the Hessian Free Method was able to successfully solve the learning problem. It is a suitable optimization algorithm for complex neural network models.

Having a closer look at the test error of the hierarchical RNN in this experiment, it can be seen that the increase of layers does not help to improve the recognition rate on the test set. Table 4.7 and figure 4.6 compare the best test HF result of the previous chapter with the best individual result of this experiment. The increase of the layer size to $\{1000, 1000, 1000\}$ neurons does not help to improve the model's generalization capabilities as well. Table 4.8 verifies this assumption. It seems that additional model side aspects must be taken into account in order to improve the model's performance.

adapted MNIST $(28 \times 28 \mid 7)$					
model	**algorithm**	**error**			
		train [%]	*valid* [%]	*test* [%]	*epoch* [1]
RNN	HF	0.154	2.11	2.07	119
HRNN	HF	0.13	2.44	2.64	129

Table 4.7.: Comparison of a RNN and HRNN (MNIST)

adapted MNIST $(28 \times 28 \mid 7)$					
model	**algorithm**	**error**			
		train [%]	*valid* [%]	*test* [%]	*epoch* [1]
HRNN	HF	0.0	2.13	2.04	113

Table 4.8.: Test error of a HRNN with 1000x3 neurons (MNIST)

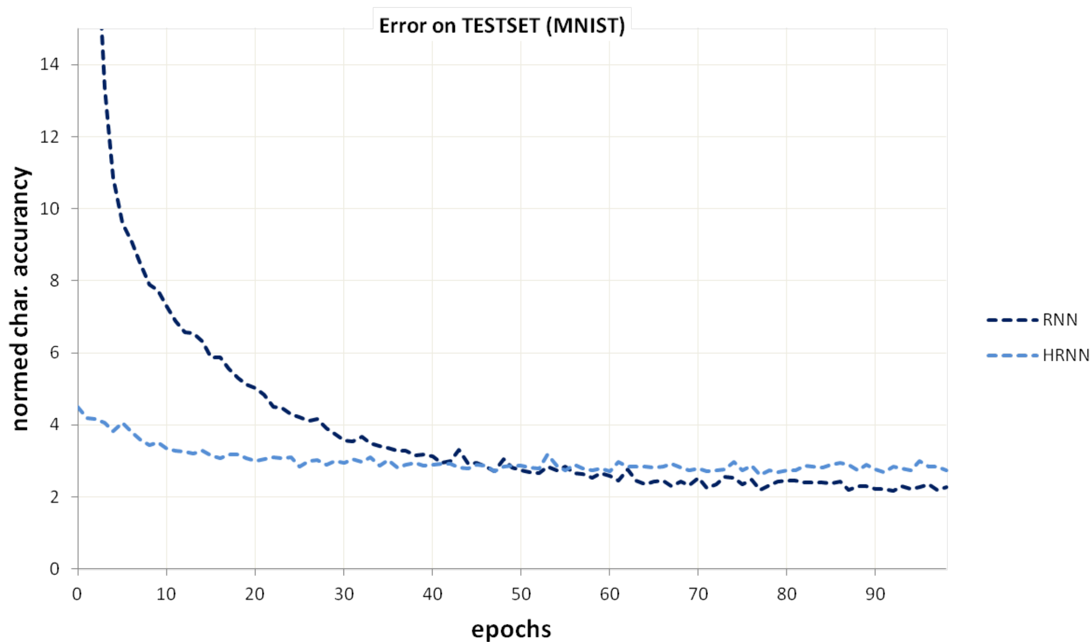


Figure 4.6.: Comparing the error curves of a RNN and a HRNN (MNIST)

4.1.4. Generative pre-training for Hierarchical Recurrent Architectures

In the last experiment it was shown that with the help of the Hessian Free Method it was possible to optimize a Hierarchical Recurrent Network. However the test error did not improve with the help of the layered structure. It seems that additional model side aspects like the use of additional regularizers or sparse connections must be taken into account to further improve the classifier's performance.

In this experiment the learning problem of the previous chapters is divided into two parts. First the data distribution $P(x)$ is learned. Then the learning problem $P(y|x)$ is solved. Learning $P(x)$ can be achieved with the help of the generative pre-training technique for Deep Belief Networks (DBN) (chapter 2.5.2). This technique was adapted for hierarchical RNNs (chapter 3.2).

The input weights of a single RNN layer were exchanged with those of the pre-trained RBM layer of a DBN. The recurrent connections of each RNN were initialized to small gaussians ($\mu = 0, \sigma = 0.1$). Table 4.9 lists the configuration parameters of this model.

dataset	**parameters**		
MNIST	train=5000 sequences	*10 digits/sequence*	
	valid=1000 sequences	*10 digits/sequence*	
	test=1000 sequences	*10 digits/sequence*	
model	**parameters**		
DBN-RNN	**input neurons**	**activation**	**output layer**
	784	sigmoid	CTC
	hidden neurons	**algorithm**	**batchsize**
	{1000,1000,1000}	$\{cd\text{-}k, sgd\}$	$\{10, 1\}$
	output neurons	**parameters**	
	10	$\eta_{pretrain} = 0.01$, $\eta_{finetune} = 0.1$	
training	**parameters**		
	early-stopping=50		
	pretrain-epochs=100		
	max-epochs=500		
	validation-frequency=1		

Table 4.9.: Model and training parameters of the DBN-RNN

After 100 epochs of unsupervised pre-training and the model was fine-tuned with stochastic gradient descent using a early-stopping rate of 50 epochs. Table 4.10 lists the errors on the training-, validation- and test-set respectively.

adapted MNIST (28 × 28 | 7)				
model	**error**			
	train [%]	*valid* [%]	*test* [%]	*epoch* [1]
DBN-RNN	**0.0**	**1.4**	**1.39**	**146**

Table 4.10.: Errors of the DBN-RNN (MNIST)

Pre-training the RNNs with contrastive divergence leads to a preferable weight initialization, which is needed to create a smoother error surface. As a consequence the stochastic gradient descent algorithm used in the fine-tuning phase was able to escape local minima and performed very well. The model outperforms the best RNN of the previous chapter, which was trained directly with the HF method.

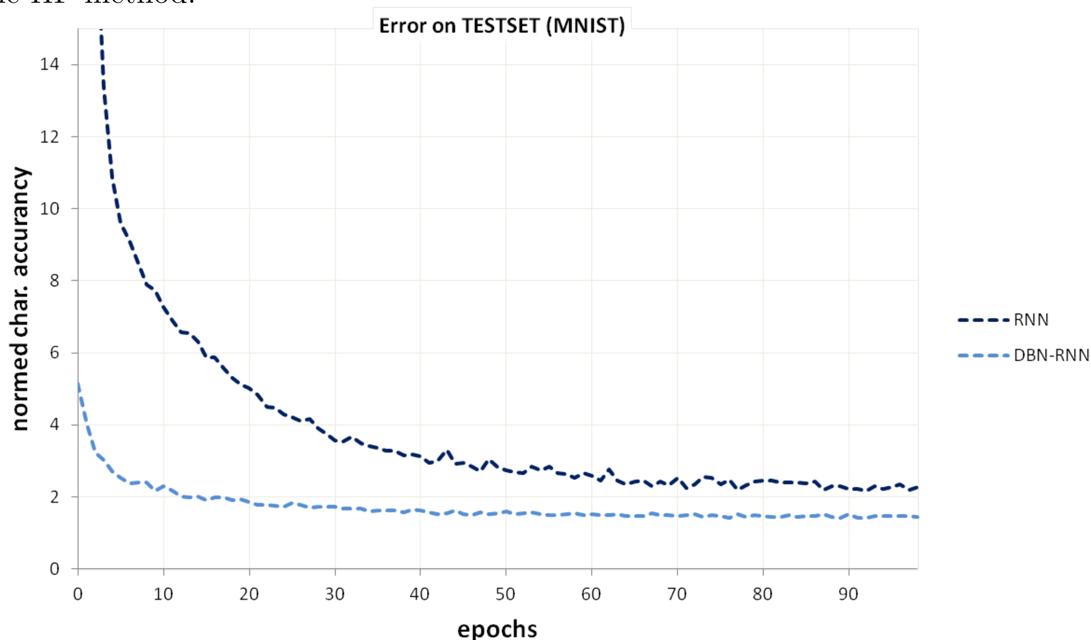


Figure 4.7.: Error curves of the DBN-RNN (MNIST)

Figure 4.7 shows the error curves of both models for comparison. The DBN-RNN was able to solve the learning task fully and clearly outperforms the HRNN.

4.1.5. Comparison of Neural Network Architectures

In order to put this analysis in a bigger context, various neural networks architectures were compared on the modified MNIST database. A Multilayer Perceptron (MLP) [Rosenblatt, 1958], a Convolutional Neural Network [LeCun et al., 1999], a Long Short Term Memory Network (LSTM) [Gers et al., 2002] and a Deep Belief Network (DBN) [Mohamed et al., 2009] were chosen as test candidates. All networks were connected to a CTC output layer.

Figure 4.8.: Network models

In this experiment the same input-window of 28×28 pixels was used again. This input-window was shifted 7 pixels per time-step, leading to an overall sequence length of 40 frames. It is assumed that all classifiers will be able to recover the original digits from the sequence and learn to ignore the noise between the consecutive digits with the help of the CTC layer.

Table 4.12 lists the specific parameters for each neural models which were used in the experiments.

RNN	**input neurons**	**activation**	**output layer**
	784	sigmoid	CTC
	hidden neurons	**algorithm**	**batchsize**
	100	hf	1
	output neurons	**parameters**	
	10	cg-iter-max=3, precond=None, $\mu = 0$, $\lambda = 1$	

LSTM	**input neurons**	**activation**	**output layer**
	784	sigmoid	CTC
	hidden neurons	**algorithm**	**batchsize**
	100	sgd	1
	output neurons	**parameters**	
	10	$\eta = 0.01$	

MLP	**input neurons**	**activation**	**output layer**
	784	sigmoid	CTC
	hidden neurons	**algorithm**	**batchsize**
	{500,500}	sgd	1
	output neurons	**parameters**	
	10	$\eta = 0.01$	

ConvNet	**input neurons**	**activation**	**output layer**
	784	sigmoid	CTC
	hidden neurons	**algorithm**	**batchsize**
	{576,512}	sgd	1
	output neurons	**parameters**	
	10	$\eta = 0.01$	

DBN	**input neurons**	**activation**	**output layer**
	784	sigmoid	CTC
	hidden neurons	**algorithm**	**batchsize**
	{1000,1000,1000}	$\{cd\text{-}k, sgd\}$	$\{10, 1\}$
	output neurons	**parameters**	
	10	epochs$_{pretrain}$=100, k=1, $\eta_{pretrain} = 0.01$, $\eta_{finetune} = 0.1$	

DBN-RNN	**input neurons**	**activation**	**output layer**
	784	sigmoid	CTC
	hidden neurons	**algorithm**	**batchsize**
	{1000,1000,1000}	$\{cd\text{-}k, sgd\}$	$\{10, 1\}$
	output neurons	**parameters**	
	10	epochs$_{pretrain}$=100, k=1, $\eta_{pretrain} = 0.01$, $\eta_{finetune} = 0.1$	

Table 4.11.: Model parameters of different network architectures

Table 4.12 shows the results of the models with a input-frame of 28×28 pixels shifted 7 pixels per time-step. Due to the fact that a single input-frame represents a single MNIST digit, the hierarchical feed-forward classifiers were able to learn the underlying patterns very well. All networks were able to ignore the data between two concatenated digits, meaning that the CTC output layer assigned a 'blank' label to those noisy frames.

adapted MNIST (28×28 | 7)				
model	**error**			
	train [%]	*valid* [%]	*test* [%]	*epoch* [1]
RNN	0.154	2.11	2.07	119
LSTM	0.02	2.95	2.94	143
MLP	0.0	1.92	1.75	112
ConvNet	0.65	5.46	6.27	122
DBN	0.01	1.47	1.41	108
DBN-RNN	**0.0**	**1.4**	**1.39**	**146**

Table 4.12.: Errors of different network architectures (MNIST)

When looking at table 4.12 it is interesting to see that the convolutional neural network, which outperforms other classifiers on the MNIST database in a standard regression task, was not able to achieve very good results on the test set. This might be due to the fact that the convolutional filters have problems of ignoring the noisy patterns between each digit.

As far as the DBN and the developed RNN-DBN are concerned, it can be seen that both models achieved nearly the same results, and outperformed all other models in this experiment. Figure 4.9 shows the error curves of the models.

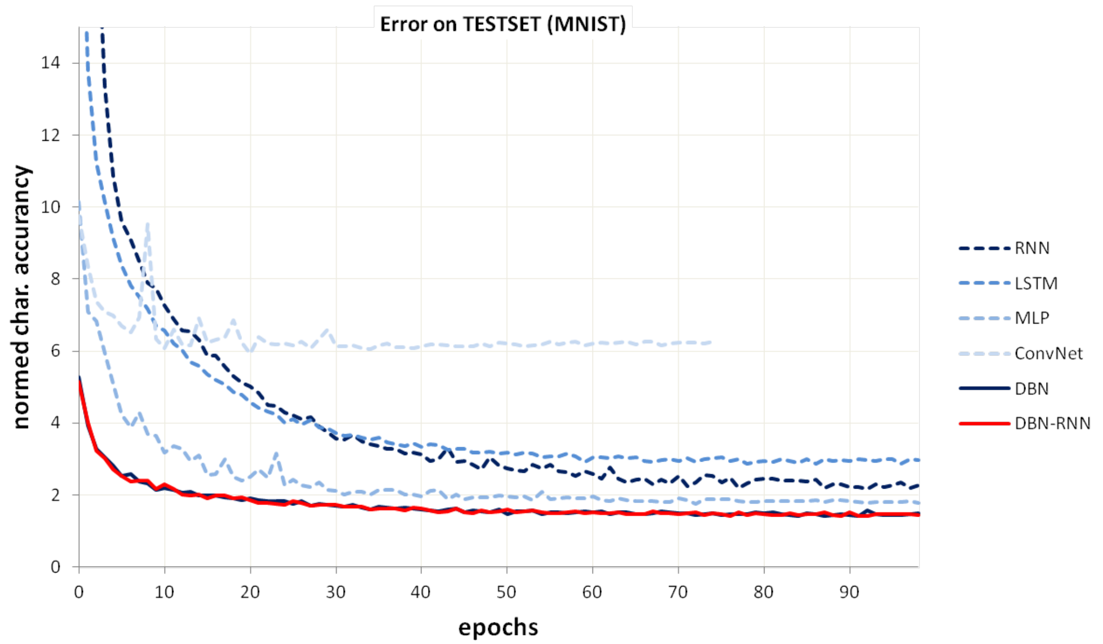


Figure 4.9.: Error curves of different network architectures (MNIST)

5. Conclusion and Future Work

In this work a Recurrent Neural Network with a CTC output layer on top was presented. The specially designed output layer was able to successfully solve the sequence labeling problem with the help of a maximum likelihood objective. As a consequence, the direct training of input $\mapsto$ target relations without knowing the explicit target labeling was possible.

First of all the effects of different gradient based optimization techniques, able to handle noisy target predictions, were analyzed. Gradient descent with momentum, RPROP variants and the Hessian Free Method were used to train a RNN on sequences of concatenated MNIST digits. Although nearly all algorithms were able to learn the training data, the models did not generalize very well, leading to moderate recognition results on the test set.

In a next step, a single recurrent architecture was extended to a hierarchical RNN. It was hoped that due to the new structure of the model, more abstract features would be generated helping to improve the recognition rate. Neither RPROP variants, nor the Stochastic Diagonal Levenberg-Marquard Algorithm were able to solve the learning problem optimally. Instead the Hessian Free Method was able to solve the learning problem. However the recognition rate on the test set did not improve. It seems that additional model side aspects such as a more clever data representation or additional regularizers must be taken into account to further improve the model.

The generative pre-training technique supported by Deep Belief Models could handle this problem. Pre-training a hierarchical recurrent network with contrastive divergence leads to superior recognition results on the underlying test set.

Different network architectures, all connected to a CTC output layer were tested on the adapted MNIST dataset. In fact the developed DBN-RNN model was able to outperform all other classifiers due to its ability to learn a more abstract representation of the data, and its capability of extracting temporal relations with the help of recurrent connections. Both characteristics are needed for challenging future learning tasks.

The main emphasis of this work was the analysis of gradient based optimization techniques (figure 5.1) for RNN models with CTC output layers. Although very powerful learning techniques for optimizing these models exist, it was demonstrated that the presented algorithms still have problems optimizing complex layered network structures. Such 'structures' are needed to model levels of abstraction, allowing a system to learn complex functions mapping from an input to a target directly from data, without depending completely on human crafted features.

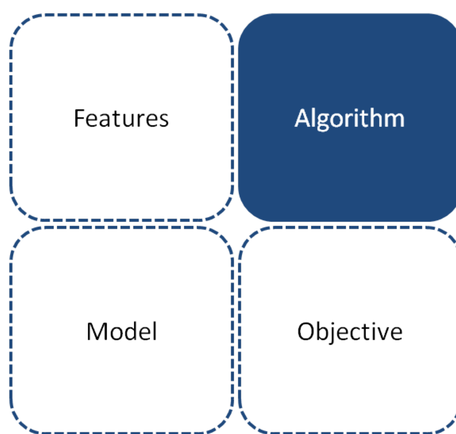


Figure 5.1.: Improving optimization methods

In this work a 'loophole', generative pre-training to escape optimization problems was used. Splitting the learning problem into two parts, modeling $P(x)$ and solving $P(y|x)$ afterwards, seems to be a good solution. Further research should be invested and might be rewarding.

Although the presented architecture (DBN-RNN) outperformed all other models in the experimental part, the model still suffers from several drawbacks, which are worth to discuss.

From a **biological point of view** studies on brain energy expense suggest that neurons encode information in a sparse and distributed way [Attwell and Laughlin, 2001]. The estimated percentage of neurons active at the same time lies between 1 and 4%; a trade-off between richness of representation and small action potential energy expenditure [Lennie, 2003]. In this work a dense initialization scheme was used to initialize the weights of a recurrent neural network, which might not be biological plausible. The use of more a biological inspired neuron models, like the leaky integrate-and-fire (LIF) [Dayan and Abbott, 2001] was not taken into account as well. As far as

generative pre-training is concerned, it must be questioned if this learning concept is biologically plausible at all.

From the **machine learning perspective** it might be possible to enhance the CTC objective (figure 5.2). The maximum likelihood training criteria only tries to optimize the likelihood of the training data, but does not take the model side aspects such as efficient data compression into account. As a consequence, at a certain point of training the model only learns the training data, but does not improve on the test set.

This problem could be partly solved by adding additional regularization terms, like L1- or the L2 norm added to the cost function, however, much more interesting ideas to solve this problem exist. The use of stochastic weight noise [Hinton et al., 2012] helps to improve the recognition rates on the test set. This concept could be further extended. Thinking of a network as a transmission channel, the transmission cost, measured as the Kullback-Leibler divergence between a prior and posterior distribution of the weights, can be minimized. This idea was first published by [Hinton and Van Camp, 1993] and was used as a compression method. [Graves, 2011] extended the concept, deriving a variational inference method for CTC networks.

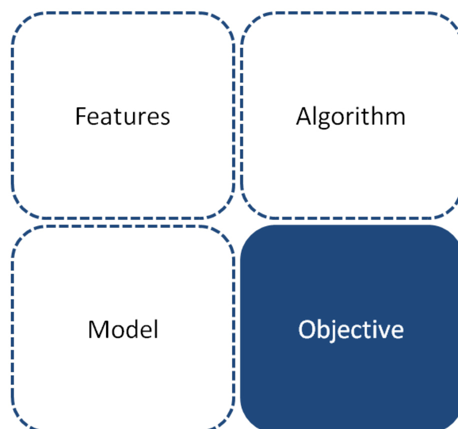


Figure 5.2.: Improving objectives

Optimal compression of data stored in neural networks will not only improve the overall learning performance on a dataset, removing the need of a validation set. It will also have an influence on the structure of the model, as far *pruning* and weight *growing* methods are concerned. This principle follows Occam's razor [Ariew, 1976] and might be an interesting concept for future research.

A. Appendix

A.1. Software Framework

Figure A.1 gives an overview of the current software features of this framework developed in this work. Simulations on various databases with a wide range of network models with different algorithms are possible. All models are connected by default with a CTC output layer.

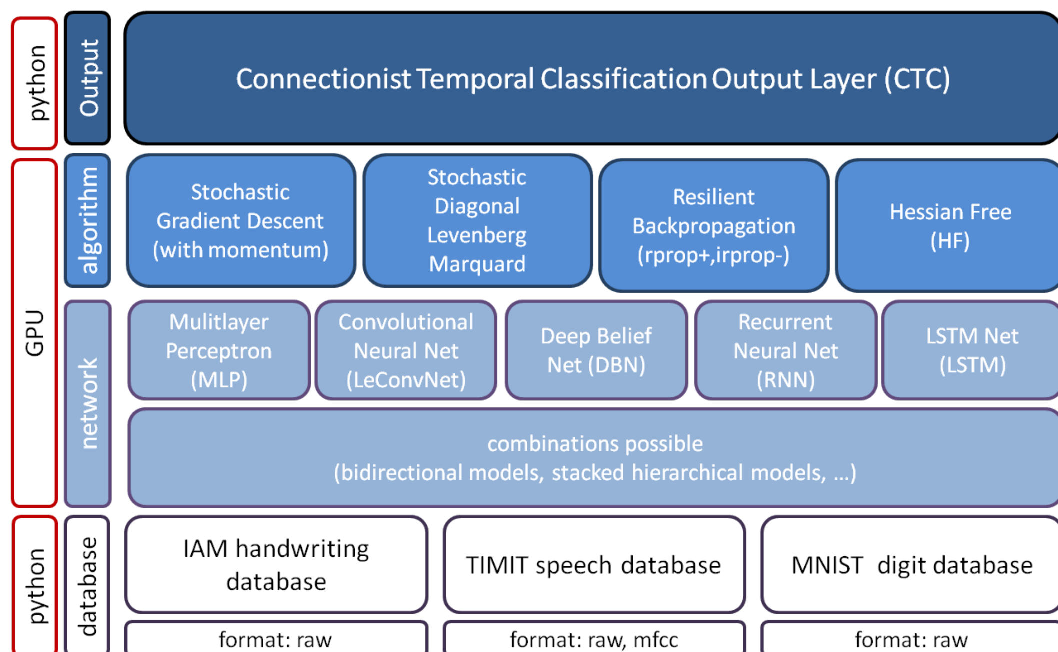


Figure A.1.: Software features

Any experiment can be simulated on a 32/64bit *Linux*, *Windows* and *MacOs* workstation. In order to enable GPU support a NVIDIA graphics card is

needed. All experiments were run with Theano v0.6rc1 which can be downloaded from the corresponding website[1].

In order to run an experiment on a Linux based target matching the following *.bash_profile* entry is needed to correctly initialize the Theano GPU support on Linux based operating systems:

```
THEANO_FLAGS=device=gpu,floatX=float32,nvcc.fastmath=True,mode=FAST_RUN,
force_device=True,cxxflags=/usr/local/cuda/include/
```

Information about the installation process and additional optimization parameters can found at the Theano website[2].

[1] http://deeplearning.net/software/theano/index.html
[2] http://deeplearning.net/software/theano/index.html

Bibliography

[Ariew, 1976] Ariew, R. (1976). *Ockham's Razor: A Historical and Philosophical Analysis of Ockham's Principle of Parsimony*. University of Illinois press, Champaign-Urbana. (Cited on page 67.)

[Attwell and Laughlin, 2001] Attwell, D. and Laughlin, S. B. (2001). An energy budget for signaling in the grey matter of the brain. *Journal of cerebral blood flow and metabolism official journal of the International Society of Cerebral Blood Flow and Metabolism*, 21(10):1133–1145. (Cited on page 66.)

[Bengio, 2012] Bengio, Y. (2012 (accessed Sep. 07, 2012)). Research. http://www.iro.umontreal.ca/∼bengioy/yoshua_en/research.html. (Cited on page 2.)

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Processing*, 19(d):153. (Cited on page 25.)

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. (Cited on page 22.)

[Bergstra et al., 2010] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. (Cited on page 4.)

[Bishop, 1996] Bishop, C. M. (1996). *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1 edition. (Cited on pages 7, 17, 30, 31 and 38.)

[Bridle, 1990] Bridle, J. S. (1990). *Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition*, volume 68, pages 227–236. Springer-Verlag. (Cited on pages 1 and 13.)

[Chapelle and Erhan, 2011] Chapelle, O. and Erhan, D. (2011). Improved Preconditioner for Hessian Free Optimization. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. (Cited on pages 36, 39 and 41.)

[Ciresan et al., 2010] Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep big simple neural nets excel on handwritten digit recognition. *Neural Computation*, 22(12):1–14. (Cited on page 48.)

[Damerau, 1964] Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176. (Cited on page 48.)

[Dayan and Abbott, 2001] Dayan, P. and Abbott, L. (2001). *Theoretical Neuroscience*, volume 60. MIT Press. (Cited on page 66.)

[Duda et al., 2001] Duda, R., Hart, P., and Stork, D. (2001). *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley. (Cited on page 7.)

[Fink, 2008] Fink, A. (2008). *Markov Models for Pattern Recognition From Theory to Applications*. Springer. (Cited on page 11.)

[Gay, 1983] Gay, D. M. (1983). Algorithm 611: Subroutines for unconstrained minimization using a model/trust-region approach. *ACM Trans. Math. Softw.*, 9(4):503–524. (Cited on page 37.)

[Gers et al., 2002] Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002). Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research*, 3(1):115–143. (Cited on pages 4, 20 and 60.)

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley. (Cited on page 5.)

[Graves, 2011] Graves, A. (2011). Practical variational inference for neural networks. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F. C. N., and Weinberger, K. Q., editors, *NIPS*, pages 2348–2356. (Cited on pages 2 and 67.)

[Graves, 2012] Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer. (Cited on pages XVII, 7, 14, 15, 16, 17 and 18.)

[Graves et al., 2006] Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). *Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks*, pages 369–376. ACM. (Cited on pages 1, 3 and 11.)

[Graves et al., 2009] Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868. (Cited on pages 10, 13, 14, 18, 30 and 48.)

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm networks. (Cited on page 30.)

[Hinton, 2010] Hinton, G. (2010). A practical guide to training restricted boltzmann machines a practical guide to training restricted boltzmann machines. *Computer*, 9(3):1. (Cited on page 24.)

[Hinton and Salakhutdinov, 2006a] Hinton, G. and Salakhutdinov, R. (2006a). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507. (Cited on page 34.)

[Hinton and Van Camp, 1993] Hinton, G. and Van Camp, D. (1993). *Keeping neural networks simple by minimizing the description length of the weights*, pages 5–13. ACM Press, New York, NY. (Cited on page 67.)

[Hinton, 2005] Hinton, G. E. (2005). What kind of a graphical model is the brain? *Computer*, pages 1765–1775. (Cited on pages 24, 25, 27 and 28.)

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554. (Cited on page 29.)

[Hinton and Salakhutdinov, 2006b] Hinton, G. E. and Salakhutdinov, R. R. (2006b). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–7. (Cited on pages 4, 24, 29 and 44.)

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580. (Cited on page 67.)

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty Fuzziness and KnowledgeBased Systems*, 6(2):107–115. (Cited on page 22.)

[Hubel and Wiesel, 1968] Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243. (Cited on page 44.)

[ICDAR, 2012] ICDAR (2012 (accessed Sep. 07, 2012)). 10th international conference on document analysis and recognition. http://www.cvc.uab.es/icdar2009/competitions.html. (Cited on page 2.)

[Igel et al., 2000] Igel, C., H, M., and De, M. H. (2000). Improving the rprop learning algorithm. *Symposium A Quarterly Journal In Modern Foreign Literatures*, pages 115–121. (Cited on page 33.)

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *PLoS ONE*, 4(6):1942–1948. (Cited on page 5.)

[Larochelle and Bengio, 2008] Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted boltzmann machines. *Proceedings of the 25th International Conference on Machine Learning (2008)*, pages 536–543. (Cited on pages 26 and 27.)

[Lauer et al., 2007] Lauer, F., Suen, C. Y., and Bloch, G. (2007). A trainable feature extractor for handwritten digit recognition. *Pattern Recogn.*, 40(6):1816–1824. (Cited on page 48.)

[Le et al., 2012] Le, Q. V., Ranzato, M. A., Devin, M., Corrado, G. S., and Ng, A. Y. (2012). Building high-level features using large scale unsupervised learning. *Arxiv preprint arXiv*, 28(4):61–76. (Cited on page 10.)

[LeCun et al., 1998a] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. (Cited on page 30.)

[LeCun et al., 1998b] LeCun, Y., Bottou, L., Orr, G., and Müller, K. (1998b). Efficient backprop. *Neural networks Tricks of the trade*, 1524(3):546–546. (Cited on pages 31, 33 and 51.)

[Lecun and Cortes, 2012] Lecun, Y. and Cortes, C. (1998 (accessed (Sep. 07, 2012))). The mnist database of handwritten digits. (Cited on pages 4, 47 and 48.)

[LeCun et al., 1999] LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). *Object Recognition with Gradient-Based Learning*, pages 319–344. Springer. (Cited on pages 4, 20, 33, 44 and 60.)

[Lee et al., 2009] Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning ICML 09*, 2008(3):1–8. (Cited on page 10.)

[Lennie, 2003] Lennie, P. (2003). The cost of cortical computation. *Current Biology*, 13(6):493–497. (Cited on page 66.)

[Marquardt, 1963] Marquardt, D. W. (1963). An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441. (Cited on page 34.)

[Martens, 2010] Martens, J. (2010). Deep learning via hessian-free optimization. *Proceedings of the 27th International Conference on Machine Learning ICML*, 951:2010. (Cited on pages 34, 36, 37, 39, 40, 41 and 51.)

[Martens, 2011a] Martens, J. (2011a). Generating text with recurrent neural networks. *Neural Networks*, 131(1):1017–1024. (Cited on page 1.)

[Martens, 2011b] Martens, J. (2011b). Learning recurrent neural networks with hessian-free optimization. *Neural Networks*, 144(4):1–8. (Cited on pages 2, 30, 31, 34, 37, 38, 51 and 52.)

[Mohamed et al., 2009] Mohamed, A.-r., Dahl, G., and Hinton, G. (2009). Deep belief networks. *Science*, 4(5):1–9. (Cited on pages 4, 20 and 60.)

[Montavon et al., 2012] Montavon, G., Orr, G. B., and Müller, K.-R., editors (2012). *Neural Networks: Tricks of the Trade, Reloaded*, volume 7700 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2nd edn edition. (Cited on pages 51 and 52.)

[Nocedal and Wright, 2000] Nocedal, J. and Wright, S. J. (2000). *Numerical Optimization*. Springer. (Cited on pages 34, 35, 36, 38 and 39.)

[Pearlmutter, 1994] Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160. (Cited on page 36.)

[Pfister and Kaufmann, 2008] Pfister, B. and Kaufmann, T. (2008). *Sprachverarbeitung: Grundlagen und Methoden der Sprachsynthese und Spracherkennung*. Springer, Berlin. (Cited on page 11.)

[Phansalkar and Sastry, 1994] Phansalkar, V. V. and Sastry, P. S. (1994). Analysis of the back-propagation algorithm with momentum. *IEEE Transactions on Neural Networks*, 5(3):505–506. (Cited on page 2.)

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. (Cited on pages 1 and 11.)

[Riedmiller and Braun, 1993] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the rprop algorithm. *IEEE International Conference on Neural Networks*, 1(3):586–591. (Cited on pages 2, 30 and 32.)

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408. (Cited on pages 4, 20 and 60.)

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. (Cited on pages 3, 20 and 21.)

[Salakhutdinov and Murray, 2008] Salakhutdinov, R. and Murray, I. (2008). On the quantitative analysis of deep belief networks. In *Proceedings of the International Conference on Machine Learning*, volume 25. (Cited on page 24.)

[Schraudolph, 2002] Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738. (Cited on page 36.)

[Smolensky, 1986] Smolensky, P. (1986). *Information processing in dynamical systems: Foundations of harmony theory*, volume 1, pages 194–281. MIT Press. (Cited on page 26.)

[Tieleman, 2008] Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. *Proceedings of the 25th International Conference on Machine Learning (2008)*, volume25:1064–1071. (Cited on page 29.)

[Tikhonov and Arsenin, 1977] Tikhonov, A. N. and Arsenin, V. Y. (1977). *Solutions of ill-posed problems*. V. H. Winston & Sons, Washington, D.C.: John Wiley & Sons, New York. Translated from the Russian, Preface by translation editor Fritz John, Scripta Series in Mathematics. (Cited on page 36.)

[Werbos, 1990] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560. (Cited on page 21.)

[Williams, 1992] Williams, R. J. (1992). *Training recurrent networks using the extended Kalman filter*, volume 4, page 241–246. IEEE. (Cited on page 21.)

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1(2):270–280. (Cited on page 21.)

[Wilson and Martinez, 2003] Wilson, D. R. and Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451. (Cited on page 51.)

[Young et al., 1989] Young, S., Russell, N., and Thornton, J. (1989). Token passing: a simple conceptual model for connected speech recognition systems. Technical report, Cambridge University Engineering Department. (Cited on page 18.)

[Zipser, 1989] Zipser, D. (1989). A subgrouping strategy that reduces learning complexity and speeds up learning in recurrent networks. *Neural Computation*, 1:552–558. (Cited on page 22.)