Graz University of Technology
Signal Processing and Speech Communication Laboratory

Master of Science Thesis

Variational Bayesian Reservoir Computing and its Applications to Handwriting Recognition

by

Christoph Zechner

Advisor: Gernot Kubin
Co-Advisor: Dmitriy Shutin (Princeton University)

Graz, 2010

Deutsche Fassung:

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)

Englische Fassung:

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)

Abstract - Reservoir computing principles - and in particular - Echo State Networks (ESNs) have been shown to work well for many different applications, such as chaotic time series prediction, speech recognition or adaptive control. However, in some cases standard ESNs show only poor learning performance. For this reason, several model extensions to classical ESN learning have been proposed in the literature, such as filter neurons or tunable delay & sum readouts, to name just a few.
Furthermore, ESN learning algorithms require appropriate regularization techniques, especially if large reservoirs are used. Clearly, ESN learning algorithms, which can also deal with tunable model extensions as well as an automatic regularization would be of general interest. However, jointly solving both requirements turns out to be analytically complex and thus, this work utilizes the variational Bayesian (VB) inference framework, to obtain approximate learning algorithms for three different ESN models. Using real-world handwriting trajectories, this thesis tries to find out, if and how the VB algorithms can improve ESN learning performance.

Kurzfassung - Es wurde bereits mehrfach gezeigt, dass zahlreiche komplizierte Probleme, wie beispielsweise die Vorhersage chaotischer Systeme, Spracherkennung oder Adaptive Regelung mithilfe von Reservoir Computing Verfahren - und im Speziellen Echo State Networks (ESNs) - gelöst werden können. Tatsächlich kann es jedoch in manchen Fällen vorkommen, dass der klassische ESN-Ansatz nur wenig zufriedenstellende Ergebnisse liefert. Als Abhilfe wurden einige Erweiterungen zum Standard ESN-Modell, wie zum Beispiel Filterneuronen oder trainierbare Delay & Sum Readouts vorgeschlagen.
Besonders wenn große Reservoirs zum Einsatz kommen, stellt sich zusätzlich heraus, dass Regularisierung eine zentrale Rolle für erfolgreiches Lernen mithilfe von ESNs spielt. Es liegt auf der Hand, dass Methoden, welche sowohl eine automatische Regularisierung als auch das Lernen von erweiterten Modellparametern unterstützen, von besonders großem Interesse sind. Unglücklicherweise sind derartige kombinierte Lernprobleme analytisch komplex, weshalb in dieser Arbeit approximative ESN - Lernalgorithmen demonstriert werden, die auf dem Variational Bayesian (VB) Inference Konzept basieren. Des Weiteren soll am Beispiel von Trajektorien-basierter Handschrifterkennung herausgefunden werden, ob und wie ESNs durch die vorgestellten VB-Algorithmen verbessert werden können.

Contents

Acknowledgements

First of all, I would like to thank Dmitriy Shutin for the brilliant supervision of this thesis. He always found time for intense discussions and support. Furthermore, he enabled me to get started with scientific working, already during my undergraduate studies. During our joint work, I learned so much from Dmitriy and I really hope that our collaboration will persist as long as possible.

I would also like to give many thanks to Prof. Gernot Kubin, as he always provided great support for this thesis as well as during my overall master studies. For instance, he helped me to make it possible to visit a scientific conference in March 2010. Additionally, he aided and encouraged me very much during application for my future PhD studies.

Furthermore, I would like to thank my family, my girlfriend and also all of my friends for being behind me all the time. They constantly gave me optimal support during my work and studies.

Christoph Zechner
Graz, July 2010

Chapter 1

Introduction

In the past decades machine learning and probabilistic inference principles have advanced to one of the most important research areas within the field of natural science and engineering. A vast number of modern developments utilize intelligent or adaptive methods, such as cellular phones, Internet search engines or navigation systems - to name just a few. Previous research has already found lots of highly developed and useful methods, which can be used for solving complex learning and inference tasks. However, many practical problems still suffer from computational complexity or are mathematically intractable, so that fundamental research in these fields is still of great interest.

Even if both learning and inference problems can be described in a similar manner within the common framework of probability theory, a general distinction between them makes sense. While machine learning typically aims to find models with a specific behavior based on any synthetic or real world data, probabilistic inference deals with estimating parameters of a well-known *model*, based on noisy *observations*. It becomes clear that in general, machine learning is somehow more *artificial* than the problem of estimating quantities. In probabilistic inference, parameters often find physical or mathematical interpretations, which is reflected by the underlying model. Thus, estimating parameters is always limited to the quality of the corresponding model. In contrast, complex learning algorithms are often based on so-called *black-box* mechanisms, where model parameters do not necessarily need an interpretation and are simply tuned such as to meet an arbitrary optimality criterion (*training*). In general, such methods are used, if only little structural knowledge of the underlying system is available. For a good and detailed introduction into basic machine learning and inference theory, the reader should refer to [1] and [2].

Neural networks (*NNs*), which can be considered as typical black-box models, gained a lot of popularity as they were shown to work well for several practical regression and classification problems, such as speech and phoneme recognition [3, 4], pattern classification [5] or

adaptive control [6]. [5, 1].
In its most simple form, the neural network does not have any recursive dependencies between any of the neurons (*feed forward neural network*, *FFNN*, [5]). More complex network structures, such as the family of *recurrent neural networks* (*RNNs*), have recursive connections between the neurons (i.e. *loops*). As a result, when using general RNNs, the resulting input-output relations are described by nonlinear dynamical functions, which are known to be difficult from a mathematical point of view. Especially network training turns out to be essentially more complex as in the simpler FFNN case. [7, 8]

1.1 Reservoir Computing

The fact that RNN training is a demanding problem, led to a novel neural information processing concept called *reservoir computing*, which was independently explored in [9] (*Echo State Networks*, *ESNs*) and [10] (Liquid State Machines, *LSMs*). In both cases RNNs are used to create a large number of nonlinear dynamical base functions, given a certain input. An additional output stage then constructs the output as a combination of these base functions. The key difference to classical RNNs is that the complex problem of learning recurrent neuron weights is omitted and only the weights in the output stage are trained. In the ESN case, where the output is computed as a linear combination of the echo states followed by a static nonlinearity, the network can be trained by solving a simple least squares problem[1]. A more detailed overview about reservoir computing can be found in [11], covering liquid - and echo state principles as well as other reservoir based RNN learning concepts. In the following chapters, this thesis will focus on ESN-type reservoir computing principles. [9, 10, 11]

It has been shown that reservoir computing principles can handle many complex machine learning problems such as nonlinear / chaotic time series prediction [9], adaptive nonlinear system identification [12] or symbol classification tasks [13]. Additionally, Maass *et al.* have proved that LSMs are in theory *universal approximators* under some conditions (see [10]). However, all reservoir computing concepts have in common that the learning performance strongly depends on the underlying reservoir, which is typically constructed randomly or by applying heuristics [9, 11].

An important property of any nonlinear dynamical system is its memory capacity, quantifying the influence of the input history on the systems output at a certain time instance. A theoretical analysis of the standard ESN's short term memory capacity is given in [14]. Furthermore, it was shown in [15] that the effective memory capacity for a given ESN can be successfully extended by introducing additional time delays in the ESN's output stage (i.e. ESNs with delay & sum readouts). The authors show that several learning problems - such as the prediction of the *Mackey-Glass* system - can be significantly improved by also optimizing these time delays, for instance by applying the *expectation-maximization* (*EM*) algorithm

[1] Additionally, the least squares solution requires the static nonlinearity to be invertible.

[16]. Chapter 4 will also use ESNs with tunable delay & sum readouts as an extension to standard ESN learning.

While the training of ESNs can also be considered as a deterministic optimization problem, applying the EM algorithm requires a well-defined statistical environment. For that reason it sometimes might be useful, to reformulate a learning problem in terms of estimating the model parameters such as to *approximate* the desired behavior as close as possible. In the following chapters, this work will concentrate on developing Bayesian inference algorithms for reservoir computing principles, so an adoption of the estimation-type perspective on the general problem of learning seems natural. As already discussed in the beginning of this chapter, the reader should keep in mind that in general, in case of performing black-box learning, the estimated parameters do not necessarily have a (simple) physical interpretation, if any.

Another important aspect in machine learning tasks is the problem of *regularization*. Especially, if the input data is noisy and the learning model is complex enough, *overfitting* occurs. As one is looking for a trained model, which best describes the target function, regularization becomes necessary to avoid "learning the noise". [5, 1]
Especially when working with ESNs, also numerical issues might require regularization concepts [11]. Typically, ESNs are trained by computing linear least squares solutions and thus, one has to deal with (maybe large) matrix inversions. Practically, these matrices might be ill-conditioned (i.e., be close to singular), such that estimation results become inaccurate. In such cases regularization is the only way to obtain stable solutions. In section 2.1.1 a Bayesian formulation of regularization will be briefly discussed and further details regarding ESN-specific regularization techniques will be provided in section 3.2.3. [1]

1.2 Variational Inference

In contrast to classical *frequentist* statistics, Bayesian estimation techniques assume model parameters to be random, by introducing *prior* distributions for them [2]. Instead of analyzing *likelihood* functions, *posterior* distributions over those parameters can be obtained via Bayes' law. However, in many practical scenarios it is tedious or even impossible to calculate the posterior distribution analytically. To handle such situations, several approximate inference algorithms have been proposed. [2, 1]
Basically, existing methods can be divided into two classes (see [1] or [17]): stochastic and deterministic approximation algorithms. Stochastic algorithms typically try to draw samples from the target posterior distribution and perform inference based on the resulting density representation. Such types of inference algorithms are often referred to as *sampling* or *Monte Carlo* algorithms.
The second approach tries to find analytical expressions to approximate a target probability density function. Well-known examples representing this class of algorithms are for in-

stance the *Laplace* approximation, the *expectation - propagation* algorithm or the *variational Bayesian* (*VB*) inference framework. The latter will play a central role within this thesis, as it will be used to obtain approximate solutions for advanced ESN learning problems. Chapter 2 provides a brief introduction to the variational Bayesian inference framework. In chapter 3, variational Bayesian algorithms for three constructive ESN learning scenarios will be derived, allowing automatic Bayesian regularization as well as tunable ESN model extensions. [1, 18]

1.3 Handwriting Recognition

Classification of handwritten symbols and characters became one of the central problems in the area of pattern recognition. Especially with the invention of PDA and tablet computers, the development of novel information processing concepts came into focus of computer and information science. Regarding handwritten symbol classification, two contrary concepts can be identified [19]: *off-line* and *on-line* methods. Clearly, this distinction is made with respect to the acquisition of the input data instead of the methodology used for classification. Off-line classification is performed after the whole symbol or text has been acquired. Often, gray-scale or color images of the written characters are used for recognition. It seems natural that for this reason, especially algorithms from computer graphic and vision are used for preprocessing and feature extraction. In a second step any classifier can be used for prediction (i.e. Support Vector Machines, Neural Networks, ...). [19]
In contrast, on-line handwriting recognition aims to classify symbols while the user is writing, which seems to be a more complex task due to different writing speeds, strong ambiguities between parts of the symbols and so on. Typically, the data is represented by a trajectory (i.e. *dynamic handwriting data*), describing the pen movement over time and thus, additional features can be used for classification (e.g., velocities, pen acceleration,...). On-line handwriting systems can act on different levels of abstraction, regarding the input data. For instance classifiers can directly work with the captured trajectories, as it is typically done in combination with *dynamic time warping* algorithms [20], [21]. Several other techniques try to extract certain features from the original data to find more compact descriptions of the handwriting. For instance an automatic speech recognition system was successfully applied to on-line handwriting data in [22]. Another approach is to find analytical models for trajectory representation as described in [23].
In general, any classifier which can handle sequence labeling tasks can be used for the recognition of handwriting trajectories. A common classifier used in the literature is the hidden Markov model (see, e.g., [22]).

Beside *handwriting recognition*, also the identification of the writer is a common task in pen-based information processing. Even if writer identification also has to perform some kind of class-labeling with respect to the input data, the preprocessing as well as classification methods can be quite different. As in handwriting recognition tasks, identification concepts can be divided into off-line [24, 25] and on-line [26, 27] algorithms. [26]

As echo state networks were shown to work well for typical sequence labeling tasks (see, e.g., [28]), section 4 of this work aims to find out if and how ESNs can be used to classify handwriting trajectories, whereas the problem of writer identification is not treated in this work. For the former task, the advanced ESN learning algorithms from section 3 will be tested against two independent handwriting data sets from the *UCI Machine Learning Repository* [29] and using two different classification schemes. The reader should note that this thesis does not aim to find a handwriting classifier which outperforms all previous methods. Instead, a compact case study concerning ESN based handwriting recognition is provided. On the other hand, the real-world handwriting data is used to evaluate variational Bayesian ESN learning.

Chapter 2

The Variational Bayesian Inference Framework

In statistics, parameter estimation deals with the problem of inferring model quantities from noisy observations. In the following, $\boldsymbol{y}$ will denote the observed data set (i.e., the model output) and the model quantities of interest (i.e., parameters) will be collected in the vector $\boldsymbol{\theta}$. Often, the underlying relation between the observed data $\boldsymbol{y}$ and the parameters $\boldsymbol{\theta}$ is known and thus one aims to find the set of parameters $\hat{\boldsymbol{\theta}}$, which *best describes* a particular realization of $\boldsymbol{y}$. Clearly, the procedure of estimating parameters always introduces uncertainty due to the observation noise perturbing $\boldsymbol{y}$. Basically, there exist two main classes of estimation techniques. The first one is referred to as *frequentist* estimation and corresponds to the classical view of probability theory. The second approach tries to find parameters based on *Bayes' law* and thus somehow stands in conflict with the classical frequentist approach. In Bayesian estimation, overall parameter distributions are inferred from the observation, instead of single point estimates. As a consequence, useful information regarding the statistics of the estimate is implicitly obtained. Also typical machine learning problems such as model selection or regularization can be realized via Bayesian inference techniques. As a consequence, it seems rather natural that for this thesis a Bayesian treatment of parameter inference would make sense. [2], [17], [1]

2.1 Introduction to Bayesian Inference

In contrast to classical frequentist statistics, where the model parameters $\boldsymbol{\theta}$ are considered to be deterministic quantities of the underlying model, Bayesian inference also allows stochastic relations by assuming $\boldsymbol{\theta}$ to be random. It is clear that this formulation of parameter estimation can handle a much broader class of problems as the relation between the observable variable $\boldsymbol{y}$ and $\boldsymbol{\theta}$ is simply specified by their joint probability density function $p(\boldsymbol{y}, \boldsymbol{\theta})$ and thus, can handle a deterministic as well as a statistical treatment of $\boldsymbol{\theta}$. The joint density can be written

as

$$p(\boldsymbol{y}, \boldsymbol{\theta}) = p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}), \tag{2.1}$$

where $p(\boldsymbol{y}|\boldsymbol{\theta})$ denotes the conditional density of $\boldsymbol{y}$ given $\boldsymbol{\theta}$ and the second term $p(\boldsymbol{\theta})$ models the *prior* distribution of $\boldsymbol{\theta}$. In a similar manner, this factorization can be done with respect to $p(\boldsymbol{y})$:

$$p(\boldsymbol{y}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}|\boldsymbol{y})p(\boldsymbol{y}) \tag{2.2}$$

Thus, by setting equation 2.1 equal to 2.2, one obtains

$$p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y})}, \tag{2.3}$$

which is known as Bayes law. Note, that the quantity $p(\boldsymbol{\theta}|\boldsymbol{y})$ now denotes the *posterior* distribution over $\boldsymbol{\theta}$, given the observation $\boldsymbol{y}$. In equation 2.3, $\boldsymbol{\theta}$ is considered as a *hidden* variable, characterized by its prior distribution $p(\boldsymbol{\theta})$. Furthermore, $p(\boldsymbol{y}|\boldsymbol{\theta})$ is often called the *likelihood* function, as it describes "how likely" $\boldsymbol{y}$ is seen under a particular realization of $\boldsymbol{\theta}$. [2, 1]

Using equation 2.3, estimates $\hat{\boldsymbol{\theta}}$ can be obtained by evaluation of the posterior function. A simple and intuitive approach is to perform a *maximum a posteriori* (*MAP*) estimation [2]:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{arg\,max}\, p(\boldsymbol{\theta}|\boldsymbol{y}) \\ &= \underset{\boldsymbol{\theta}}{arg\,max}\, p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \end{aligned} \tag{2.4}$$

Note, that the denominator $p(\boldsymbol{y})$ in equation 2.3 can be neglected as it does not depend on the optimization parameter $\boldsymbol{\theta}$. In many cases, closed form solutions for the MAP estimate can be found by setting the derivative of the objective function to zero. Typically, this is done in the log-domain, to obtain simpler analytical expressions:

$$\frac{\partial}{\partial \boldsymbol{\theta}}(\ln p(\boldsymbol{y}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})) = \mathbf{0} \tag{2.5}$$

In general, the MAP estimate has to be found using numerical optimization techniques. [2, 1]

2.1.1 Bayesian Regularization

Performing Bayesian inference requires knowledge of the underlying statistical model in terms of $p(\boldsymbol{y}|\boldsymbol{\theta})$, $p(\boldsymbol{\theta})$ and $p(\boldsymbol{y})$. Given a likelihood function, which describes the relation between the observed data $\boldsymbol{y}$ and realizations of the hidden data $\boldsymbol{\theta}$, it is clear that the resulting estimate $\hat{\boldsymbol{\theta}}$ depends on the choice of the distribution $p(\boldsymbol{\theta})$. Especially when performing machine learning tasks, prior knowledge of $\boldsymbol{\theta}$ is often not available as the model parameters do not correspond to quantities from reality. In contrast, parameter priors can be seen as additional

constraints on the parameter values, which eventually allow to implement regularization to prevent overfitting. [1]
In this section the problem of finding appropriate prior distributions over $\boldsymbol{\theta}$ is briefly discussed using the Bayesian inference framework.

Typically, the structure of a density function can be controlled by certain shape parameters. For instance, a one dimensional Gaussian distribution is fully described by its mean and variance. In such cases, the notation of the prior distribution can be extended as

$$p(\boldsymbol{\theta}) \equiv p(\boldsymbol{\theta}|\boldsymbol{\alpha}),$$

where $\boldsymbol{\alpha}$ is a vector, containing so-called *hyperparameters*, e.g., mean and variance in the example above. Within a fully Bayesian treatment, hyperparameters are also considered to be random and thus require a corresponding *hyperprior* $p(\boldsymbol{\alpha})$ [1]. The joint probability density over all variables is given by

$$p(\boldsymbol{y}, \boldsymbol{\theta}, \boldsymbol{\alpha}) = p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\alpha})p(\boldsymbol{\alpha}),$$

and furthermore, assuming $\boldsymbol{\alpha}$ to be known, the posterior distribution over the parameters $\boldsymbol{\theta}$ can be written as

$$p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{\alpha}) = \frac{p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{\alpha})}{p(\boldsymbol{y}|\boldsymbol{\alpha})}. \tag{2.6}$$

Note that varying $\boldsymbol{\alpha}$ directly influences the structure of the parameter prior $p(\boldsymbol{\theta}|\boldsymbol{\alpha})$ and thus the estimation results of $\boldsymbol{\theta}$. It is obvious that estimation of $\boldsymbol{\theta}$, coupled with an automatic determination of the hyperparameters would be of great interest. Note that in a strict sense, priors should not depend on the observed data, which stands in conflict with inferring the hyperparameters from $\boldsymbol{y}$. In machine learning, however, an automatic determination of appropriate priors could be used as an elegant mechanism to obtain regularized solutions. This method is often referred to as empirical Bayes [1]. The joint posterior over $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$ can be found by applying the factorization

$$p(\boldsymbol{\theta}, \boldsymbol{\alpha}|\boldsymbol{y}) = p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{\alpha})p(\boldsymbol{\alpha}|\boldsymbol{y}), \tag{2.7}$$

where

$$p(\boldsymbol{\alpha}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{\alpha})p(\boldsymbol{\alpha})}{p(\boldsymbol{y})}. \tag{2.8}$$

Equation 2.8 denotes the posterior over $\boldsymbol{\alpha}$ given $\boldsymbol{y}$ and the term $p(\boldsymbol{y}|\boldsymbol{\alpha})$ in the numerator is known as the *marginal likelihood* or *evidence function*, which is often used to estimate the hyperparameters $\boldsymbol{\alpha}$. Unfortunately, equations 2.7 and 2.8 cannot be evaluated analytically for many practical problems and thus, appropriate approximation techniques have to be utilized (e.g., the *evidence approximation*). [1, 18]

2.2 Generalizing Bayesian Inference

As already discussed in the previous sections, an analytical evaluation of the posterior function is often not tractable. In such cases, approximate inference algorithms can be used to obtain estimates. In this section, an analytical inference framework is introduced, which combines exact as well as approximate parameter estimation under one global optimality criterion. For simplification reasons, the parameters $\boldsymbol{\theta}$ and hyperparameters $\boldsymbol{\alpha}$ are no longer modeled explicitly but are collected in a single set of hidden variable $\boldsymbol{z}$.

2.2.1 The Variational Free Energy

Given a probabilistic model $p(\boldsymbol{y}, \boldsymbol{z})$, it can easily be shown that the logarithm of the evidence function $\ln p(\boldsymbol{y})$ satisfies the decomposition [18]

$$\ln p(\boldsymbol{y}) = \mathcal{L}\left[q(\boldsymbol{z})\right] + \mathbb{D}_{KL}\left[q(\boldsymbol{z}) \| p(\boldsymbol{z}|\boldsymbol{y})\right] \tag{2.9}$$

for any valid probability density function $q(\boldsymbol{z})$. In equation 2.9, $\mathcal{L}\left[q(\boldsymbol{z})\right]$ forms a lower bound on $\ln p(\boldsymbol{y})$ and is given by

$$\mathcal{L}\left[q(\boldsymbol{z})\right] = \int q(\boldsymbol{z}) \ln \frac{p(\boldsymbol{y}, \boldsymbol{z})}{q(\boldsymbol{z})} d\boldsymbol{z}. \tag{2.10}$$

The second term denotes the *Kullback-Leibler*(*KL*) divergence between the density $q(\boldsymbol{z})$ and the posterior $p(\boldsymbol{z}|\boldsymbol{y})$ and is defined as

$$\mathbb{D}_{KL}\left[q(\boldsymbol{z}) \| p(\boldsymbol{z}|\boldsymbol{y})\right] = \int q(\boldsymbol{z}) \ln \frac{q(\boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{y})} d\boldsymbol{z}. \tag{2.11}$$

The KL divergence measures the dissimilarity between two probability density functions and is often referred to as the *relative entropy* [17]. Clearly, $\mathbb{D}_{KL}\left[q \| p\right]$ is not symmetric with respect to q and p and hence does not define a valid metric. However, the divergence is only zero, if q and p are identical. It follows that in this particular case, the density function $q(\boldsymbol{z})$ is equal to the posterior function $p(\boldsymbol{z}|\boldsymbol{y})$ and furthermore, the lower bound $\mathcal{L}\left[q(\boldsymbol{z})\right]$ becomes tight (i.e., equals the log-evidence function). Thus, the model posterior over $\boldsymbol{z}$ given the observation $\boldsymbol{y}$ can be found by maximizing the lower bound with respect to $q(\boldsymbol{z})$. The quantity

$$\mathcal{F}\left[q(\boldsymbol{z})\right] = \int q(\boldsymbol{z}) \ln \frac{q(\boldsymbol{z})}{p(\boldsymbol{y}, \boldsymbol{z})} d\boldsymbol{z} = -\mathcal{L}\left[q(\boldsymbol{z})\right] \tag{2.12}$$

is the negative lower bound $\mathcal{L}\left[q(\boldsymbol{z})\right]$ and is often called the *variational free energy*[1] [17]. Clearly, minimizing $\mathcal{F}\left[q(\boldsymbol{z})\right]$ with respect to the density $q(\boldsymbol{z})$ is identical to maximizing $\mathcal{L}\left[q(\boldsymbol{z})\right]$ and thus,

$$\hat{q}(\boldsymbol{z}) = \underset{q(\boldsymbol{z}) \in \mathcal{Q}}{arg\,min}\, \mathcal{F}\left[q(\boldsymbol{z})\right], \tag{2.13}$$

[1] The term variational free energy originates from statistical physics [18].

where $\mathcal{Q}$ is the set of all valid density functions over $\boldsymbol{z}$. Without further restrictions on $\mathcal{Q}$, the estimated density $\hat{q}(\boldsymbol{z})$ will exactly match the true posterior $p(\boldsymbol{z}|\boldsymbol{y})$. It is important to see that in its most general formulation, variational Bayesian inference produces exact representations for the posterior functions given a particular observation $\boldsymbol{y}$. Obviously, this formulation is useless if one tries to find appropriate estimates of the hidden variables $\boldsymbol{z}$ if the true posterior is too complex as to evaluate it analytically. In such cases, the goal is to find densities of simple structure, which approximate the true posterior as well as allow straightforward evaluation to obtain estimates. Using the variational inference framework, this can be done by restricting the density $q(\boldsymbol{z})$ to be of special structure. One intuitive approach is to use parameterized functions and to estimate these parameters instead of searching for free form solutions, as it will be seen later in this chapter. [1, 18, 17]

2.2.2 The Mean Field Approximation

The most common restriction on $q(\boldsymbol{z})$, which was shown to have several useful properties is given by the assumption that each of the hidden variables $z_i \in \boldsymbol{z}$ is statistically independent from each other. As a consequence, $q(\boldsymbol{z})$ fully factorizes and can be rewritten as

$$q(\boldsymbol{z}) = \prod_{z_i \in \boldsymbol{z}} q(z_i). \tag{2.14}$$

Using this factorization (i.e., the *mean field factorization*), the variational free energy is separately minimized with respect to each of the q-factors on the right-hand side of equation 2.14. In many cases, it makes sense to keep the statistical dependencies between some of the hidden variables. For instance, when estimating a vector quantity, it might be useful to jointly estimate each of the vector entries, which would be realized by assuming a non-factorizing density for this quantity. For this reason, a more general formulation of the mean field approximation as described in [1] will be used for further considerations. The approximating density can then be written as

$$q(\boldsymbol{z}) = \prod_{i=0}^{I-1} q(\boldsymbol{z}_i). \tag{2.15}$$

In equation 2.15, I denotes the number of disjoint and statistically independent subsets $\boldsymbol{z}_i \in \boldsymbol{z}$ with $\boldsymbol{z}_i \cap \boldsymbol{z}_k = \varnothing$ for $i \neq k$. Obviously, equation 2.15 provides a more flexible restriction on $q(\boldsymbol{z})$ as it also allows to model statistical dependencies, if required. [1]

Basically, there are two possibilities to obtain the variational q-factors. In many cases, the free energy minimization allows to find unconstrained free-form solutions. By doing so, the structure of the q-factor is determined automatically, as it will be discussed later in this chapter. In some cases, however, it makes sense to additionally restrict the q-factors to be of a certain class of parameterized functions. For instance one can assume the q-factors to be Gaussian, such that the posterior would be approximated by minimizing the variational free

energy with respect to each factors mean and variance (see section 2.2.3).

Considering the single factor $q(\boldsymbol{z}_j)$, the variational free energy can be rewritten as

$$\begin{aligned}\mathcal{F}[q(\boldsymbol{z})] &= \int q(\boldsymbol{z}) \ln \frac{q(\boldsymbol{z})}{p(\boldsymbol{y},\boldsymbol{z})} d\boldsymbol{z} \\ &= \int q(\boldsymbol{z}_j) \int q(\boldsymbol{z}_{i\neq j}) \ln \left[\frac{q(\boldsymbol{z}_{i\neq j})q(\boldsymbol{z}_j)}{\ln p(\boldsymbol{y},\boldsymbol{z})}\right] d\boldsymbol{z}_{i\neq j} d\boldsymbol{z}_j \\ &= \int q(\boldsymbol{z}_j)\Big[\ln q(\boldsymbol{z}_j) - \mathbb{E}_{i\neq j}[\ln p(\boldsymbol{y},\boldsymbol{z})]\Big] d\boldsymbol{z}_j + const \\ &= -\mathcal{H}[q(\boldsymbol{z}_j)] - \mathbb{E}_{\boldsymbol{z}}[\ln p(\boldsymbol{y},\boldsymbol{z})] + const,\end{aligned} \tag{2.16}$$

where $\boldsymbol{z}_{i\neq j} = \cup_{i\neq j}\boldsymbol{z}_i$, $\mathcal{H}[q(\boldsymbol{z}_j)]$ denotes the entropy of the density function $q(\boldsymbol{z}_j)$ and $\mathbb{E}_{i\neq j}[\ln p(\boldsymbol{y},\boldsymbol{z})]$ is the expectation over $\ln p(\boldsymbol{y},\boldsymbol{z})$, taken with respect to $q(\boldsymbol{z}_i), \forall i \neq j$. Thus minimizing the variational free energy with respect to a factor $q(\boldsymbol{z}_j)$ can be done by minimizing the term

$$\hat{\mathcal{F}}[q(\boldsymbol{z})] = -\mathcal{H}[q(\boldsymbol{z}_j)] - \mathbb{E}_{\boldsymbol{z}}[\ln p(\boldsymbol{y},\boldsymbol{z})] \tag{2.17}$$

or equivalently

$$\hat{\mathcal{F}}[q(\boldsymbol{z})] = \mathbb{D}_{KL}[q(\boldsymbol{z}_i)\|\hat{p}(\boldsymbol{y},\boldsymbol{z})]. \tag{2.18}$$

with $\ln \hat{p}(\boldsymbol{y},\boldsymbol{z}) = \mathbb{E}_{i\neq j}[\ln p(\boldsymbol{y},\boldsymbol{z})] + const$. A similar notation and further details can be found in [1].
Equations 2.17 and 2.18 provide a starting point for further task-specific derivations. The latter is typically used, if one searches for free-form solutions, as the optimal factor $\hat{q}(\boldsymbol{z}_j)$ can be analytically determined as [1]

$$\ln \hat{q}(\boldsymbol{z}_j) = \mathbb{E}_{i\neq j}[\ln p(\boldsymbol{y},\boldsymbol{z})] + const. \tag{2.19}$$

If equation 2.19 cannot be evaluated for any reason, the q-factors can be restricted to parameterized analytical forms (e.g., Gaussian densities). In such cases, equation 2.17 can be used to find shape parameters such as to minimize the variational free energy and thus minimize the dissimilarity between the true and the approximate density. Furthermore, it is important to note that generally, the computation of each factor requires knowledge of all other factors. Due to these implicit dependencies, the factors have to be estimated in an iterative fashion. [1, 18]

2.2.3 A Simple Example

Now, a simple example should demonstrate the variational Bayesian approach for parameter estimation. Therefore, the simple linear model from equation 2.20 is used.

$$\boldsymbol{y} = \boldsymbol{w}^T \boldsymbol{X} + \boldsymbol{\xi} \tag{2.20}$$

The $1 \times N$-dimensional output $\boldsymbol{y}$ is constructed as a sum of P weighted base functions, collected in the data matrix $\boldsymbol{X}$ of size $P \times N$ plus a $1 \times N$-dimensional Gaussian noise vector $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. The hidden data vector $\boldsymbol{w}$ of length P has a corresponding prior distribution $p(\boldsymbol{w}) = N(\mathbf{0}, \alpha^{-1}\boldsymbol{I})$, where α can be seen as an inverse variance. Furthermore, α is assumed to be deterministic and known for this example. The corresponding graphical model[2] is illustrated in figure 2.1.

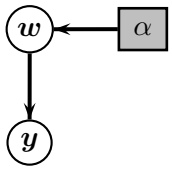


Figure 2.1: A simple graphical model.

The joint density can be written as

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}), \tag{2.21}$$

which is proportional to the posterior $p(\boldsymbol{w}|\boldsymbol{y})$ and could be directly evaluated in this simple scenario. However, the variational inference framework shall be used to demonstrate the principle. For this example, the variational free energy is given by

$$\begin{aligned} \mathcal{F}\left[q(\boldsymbol{w})\right] &= -\mathcal{H}\left[q(\boldsymbol{w})\right] - \mathbb{E}_{\boldsymbol{w}}\left[\ln p(\boldsymbol{y}, \boldsymbol{w})\right] \\ &= -\mathcal{H}\left[q(\boldsymbol{w})\right] - \mathbb{E}_{\boldsymbol{w}}\left[\ln p(\boldsymbol{y}|\boldsymbol{w})\right] - \mathbb{E}_{\boldsymbol{w}}\left[\ln p(\boldsymbol{w})\right]. \end{aligned} \tag{2.22}$$

Defining $q(\boldsymbol{w}) = N(\boldsymbol{w}|\hat{\boldsymbol{w}}, \boldsymbol{S}_w)$, whereas $N(\boldsymbol{w}|\hat{\boldsymbol{w}}, \boldsymbol{S}_w)$ denotes a Gaussian distribution over $\boldsymbol{w}$ with the mean $\hat{\boldsymbol{w}}$ and the covariance matrix $\boldsymbol{S}_w$, the optimal factor $q(\boldsymbol{w})$ is found by minimizing the variational free energy with respect to the variational parameters $\hat{\boldsymbol{w}}$ and $\boldsymbol{S}_w$. Neglecting all terms independent from $\hat{\boldsymbol{w}}$, equation 2.22 can be written as

$$\begin{aligned} \mathcal{F}\left[q(\boldsymbol{w})\right] &= \mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{w}^T\right] \boldsymbol{X}\boldsymbol{y}^T\boldsymbol{\Sigma}^{-1} - \frac{1}{2}\mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{w}^T\boldsymbol{X}\boldsymbol{\Sigma}^{-1}\boldsymbol{X}^T\boldsymbol{w}\right] + \frac{\alpha}{2}\mathbb{E}_{\boldsymbol{w}}\left[\boldsymbol{w}^T\boldsymbol{w}\right] + const. \\ &= \hat{\boldsymbol{w}}^T\boldsymbol{X}\boldsymbol{\Sigma}^{-1}\boldsymbol{y}^T - \frac{1}{2}\hat{\boldsymbol{w}}^T\boldsymbol{X}\boldsymbol{\Sigma}^{-1}\boldsymbol{X}^T\hat{\boldsymbol{w}} + \frac{\alpha}{2}\hat{\boldsymbol{w}}^T\hat{\boldsymbol{w}} + const. \end{aligned} \tag{2.23}$$

Setting the first derivative with respect to $\hat{\boldsymbol{w}}^T$ equal zero, yields the following optimal value for the mean $\hat{\boldsymbol{w}}$

[2] A description of the graphical notation used in this thesis, can be found in section A.1.

$$\hat{\boldsymbol{w}} = \left[\boldsymbol{X}\boldsymbol{\Sigma}^{-1}\boldsymbol{X}^T + \alpha\boldsymbol{I}\right]^{-1}\boldsymbol{X}\boldsymbol{\Sigma}^{-1}\boldsymbol{y}^T. \tag{2.24}$$

The same procedure can be done with respect to the covariance matrix $\boldsymbol{S}_w$. After evaluation of all expectations, the resulting objective function takes the following structure

$$\mathcal{F}\left[q(\boldsymbol{w})\right] = -\frac{1}{2}\ln|\boldsymbol{S}_w| - \frac{1}{2}Tr\left[\boldsymbol{X}\boldsymbol{\Sigma}^{-1}\boldsymbol{X}^T\boldsymbol{S}_w\right] + \frac{\alpha}{2}Tr\left[\boldsymbol{S}_w\right] + const. \tag{2.25}$$

Again, the first derivative with respect to $\boldsymbol{S}_w$ is set to zero and the resulting estimation expression for the covariance matrix is given by

$$\boldsymbol{S}_w = \left[\boldsymbol{X}\boldsymbol{\Sigma}^{-1}\boldsymbol{X}^T + \alpha\boldsymbol{I}\right]^{-1} \tag{2.26}$$

The reader should note that equations 2.24 and 2.26 fully describe the variational distribution $q(\boldsymbol{w})$. Remembering the standard results for Bayesian estimation for a fully Gaussian model, one can see that $q(\boldsymbol{w})$ exactly matches the true posterior $p(\boldsymbol{w}|\boldsymbol{y})$ and thus, the variational inference framework produces exact estimates due to this choice of the q factor. [1]

2.2.4 Summary

Summarizing the previous considerations in a few words, the Variational Bayesian inference framework can be seen as an elegant toolbox for combining exact as well as approximate inference under one common optimality criterion. The quality and also the tractability directly depend on the restrictions made on the approximating distribution q. Under the mean field assumption, free form solutions (equation 2.19) can be obtained in many cases and thus the shape of the q-factors is automatically determined. If this is not possible, one can restrict the factors to belong to a certain class of densities (e.g. Gaussian, Dirac-Delta) and try to estimate the parameters of the approximating distribution such as to minimize the variational free energy.

In general, determining a single q-factor requires taking expectations with respect to all other q-factors such that the resulting estimates implicitly depend on each other. Hence, the variational distributions have to be estimated in an iterative manner.

Chapter 3

Variational Bayesian Reservoir Computing

3.1 Introduction

As discussed in chapter 1, recurrent neural network learning turns out to be a non-trivial task. Due to a high dimensional parameter space and the nonlinear dynamic behavior, training of RNNs is computationally complex. For instance, bifurcations can occur during training due to the parameter changes of the nonlinear dynamical network [8]. As a consequence, there is no guarantee that an optimum is found. Another aspect is that the numerical optimization of the network parameters, which recurrently depend on each other requires many computation steps and thus will lead to a large computational effort [11]. The described shortcomings of RNN learning lead to a novel class of RNN learning concepts, referred to as *reservoir computing* (*RC*). [11]

The basic idea of reservoir computing can be described as a simple two-step procedure [9]:

1. Randomly create a large reservoir of *fixed* nonlinear dynamical base functions
2. Try to reproduce a certain input-output behavior as a simple function of the reservoir base functions

This is realized by using a large, randomly constructed RNN, where each neuron is connected to several other neurons and these connections are assumed to be fixed. The collection of the neuron outputs at a certain time step represents the state of the dynamical reservoir. Using the fixed reservoir states one can try to reproduce a certain input-output behavior of an unknown target system. Clearly, if the reservoir is large and it provides base functions which are *diverse* enough, complicated nonlinear dynamic target functions can be modeled. [9, 10]

3.2 Echo State Networks

The fact that Jaeger's version of reservoir computing was developed in terms of classical supervised learning, makes it a very compact but powerful learning tool, which can be used out-of-the-box for typical regression as well as classification tasks. For this reason, this work will utilize the ESN-type formulation of reservoir computing. An illustration of the concept is given in figure 3.1.

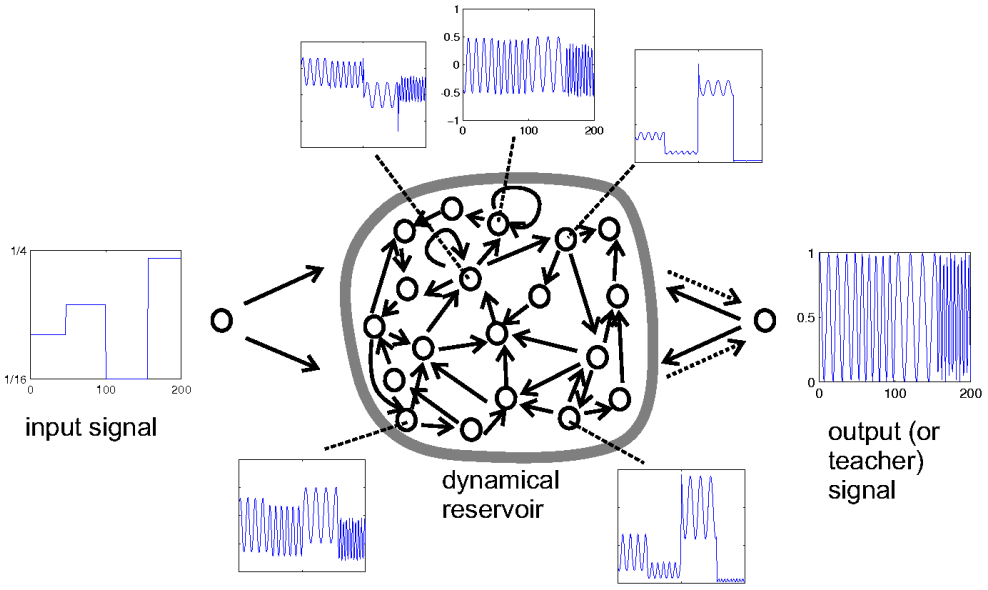


Figure 3.1: Illustration of Jaeger's *Echo State* concept (Source: [30])

The input signal is projected into a high dimensional space using a large, randomly constructed RNN. The neuron outputs form the temporal base functions and are termed the *echo states*. The corresponding mathematical formulation takes the following simple form: considering a K-dimensional input vector $\boldsymbol{u}[n]$, an L-dimensional output $\boldsymbol{y}[n]$ and a reservoir of size M, the ESN functioning is described by two equations

$$\boldsymbol{x}[n] = \boldsymbol{f}\left(\boldsymbol{C}_X^T \boldsymbol{x}[n-1] + \boldsymbol{C}_u^T \boldsymbol{u}[n] + \boldsymbol{C}_y^T \boldsymbol{y}[n-1]\right) \tag{3.1}$$

$$\boldsymbol{y}[n] = \boldsymbol{g}\left(\boldsymbol{W}^T \begin{bmatrix} \boldsymbol{x}[n] \\ \boldsymbol{u}[n] \end{bmatrix}\right) = \boldsymbol{g}\left(\boldsymbol{W}^T \boldsymbol{s}[n]\right). \tag{3.2}$$

The reservoir matrix $\boldsymbol{C}_X$ has dimension $M \times M$ and defines the connectivity of the reservoir. The input weight matrix $\boldsymbol{C}_u$ is of size $K \times M$ and the feedback matrix $\boldsymbol{C}_y$ has dimension $L \times M$. Furthermore, the functions $\boldsymbol{f}$ and $\boldsymbol{g}$ are some static nonlinear functions (e.g., hyperbolic tangent functions) operating on each element of the vector argument and $\boldsymbol{W}$ is the $M + K \times L$-dimensional output weight matrix. The key difference between ESNs and classical RNNs is that all weighting matrices within the state equation 3.1 are defined to be

fixed. By doing so, the echo states can be considered as a dynamical base, which is used to construct the output as a simple linear combination (with an additional static nonlinear function), such that standard solutions for learning linear models can be applied. In section 3.2.2, ESN learning will be introduced but before that, some important constraints on the ESN setup have to be discussed. [9, 11]

3.2.1 ESN Setup and the Echo State Property

As mentioned before, all weighting matrices, that act within the dynamical reservoir (i.e. $\boldsymbol{C}_X$, $\boldsymbol{C}_u$ and $\boldsymbol{C}_y$), are randomly constructed and assumed to be fixed during ESN learning. To obtain base functions which are as diverse as possible, a sparse reservoir matrix $\boldsymbol{C}_X$ should be used. However, good values for the connectivity are quite application dependent. [30]
Another important aspect of ESN learning is the so-called *echo state property* (*ESP*) [9]. The ESP states that the spectral radius of the reservoir matrix $\boldsymbol{C}_X$ must be smaller than one, i.e.

$$\rho(\boldsymbol{C}_X) < 1,$$

to make sure that initial conditions are washed out over time (*fading memory*). Note that this statement has only been proved for zero input signals and reservoirs with hyperbolic tangent functions but in practice it turns out that echo state network learning typically succeeds for any input signal if the reservoir has the ESP. [30, 11]
Typically, also the weighting matrices $\boldsymbol{C}_u$ and $\boldsymbol{C}_y$ are constructed randomly or by using heuristics. Another important aspect is the choice of the nonlinearities $\boldsymbol{g}$ and $\boldsymbol{f}$. As it will be shown section 3.2.2 $\boldsymbol{g}$ is required to be invertible. For more details on ESN configuration issues, the reader should refer to [9], [11] or [14].

3.2.2 Learning

Assuming the reservoir (equation 3.1) to be fixed, only equation 3.2 has to be considered during learning. As the output weight matrix $\boldsymbol{W}$ enters the systems output equation 3.2 in a linear fashion followed by an invertible static nonlinearity, network learning can be casted into a linear least squares problem, acting on the transformed output $\boldsymbol{g}^{-1}(\boldsymbol{y}[n])$. As a Bayesian formulation of ESN learning is a central part of this chapter, reformulating the learning problem into an equivalent estimation problem makes sense. To do so, the transformed output $\boldsymbol{g}^{-1}(\boldsymbol{y}[n])$ is assumed to be perturbed by a random learning error $\boldsymbol{\xi}[n] = [\xi_0[n], \ldots, \xi_{L-1}[n]]^T$.

$$\boldsymbol{y}[n] = \boldsymbol{g}\left(\boldsymbol{W}^T\boldsymbol{s}[n] + \boldsymbol{\xi}[n]\right) \tag{3.3}$$

Assuming the single noise components $\xi_0[n], \ldots, \xi_{L-1}[n]$ - each perturbing one of the L system outputs - to be white and statistically independent from each other, the maximum likelihood solution [2] for $\boldsymbol{W}$ is given by

$$\hat{\boldsymbol{W}} = \left(\boldsymbol{S}\boldsymbol{S}^T\right)^{-1}\boldsymbol{S}\boldsymbol{g}^{-1}\left(\boldsymbol{Y}\right), \tag{3.4}$$

where the $L \times N$ matrix $\boldsymbol{Y}$ contains the length-N observation sequence of $\boldsymbol{y}[n]$ and the $M + K \times N$ matrix $\boldsymbol{S}$ collects the corresponding echo states for time steps $0 \dots N-1$. Using infinitely many observations ($N \to \infty$), equation 3.4 converges toward the *Wiener-Hopf* solution. Note that equation 3.4 could also be interpreted as a MAP estimate using flat (non-informative) priors over $\boldsymbol{W}$. [9, 1, 2]

3.2.3 Regularization

From equation 3.4 one can see that estimating $\boldsymbol{W}$ requires inversion of the matrix $\boldsymbol{S}\boldsymbol{S}^T$. Due to the fact, that the sparse reservoir is created randomly, there is no guarantee that the resulting base functions are linearly independent and thus, $\boldsymbol{S}\boldsymbol{S}^T$ might be ill-conditioned. In such cases, one needs to introduce appropriate regularization techniques. As discussed in section 2.1.1, a Bayesian treatment of regularization can be realized by introducing priors over each output weight vector $\boldsymbol{w}_l$, representing the l-th column in $\boldsymbol{W}$ with $l = 0 \dots L-1$. Typically, Gaussian distributions with zero mean and variance α_l^{-1} are used for each of the output weight vectors, such that $p(\boldsymbol{w}_l|\alpha_l) = N(\boldsymbol{w}_l|\boldsymbol{0}, \alpha_l^{-1}\boldsymbol{I})$. It can be easily verified that the MAP estimate for $\boldsymbol{w}_l$, corresponding to the l-th output $\boldsymbol{y}_l$, is then given by [1]

$$\hat{\boldsymbol{w}}_l = \left(\boldsymbol{S}\boldsymbol{S}^T + \alpha_l \boldsymbol{I}\right)^{-1} \boldsymbol{S} g^{-1}\left(\boldsymbol{y}_l\right), \tag{3.5}$$

whereas $\boldsymbol{y}_l$ denotes the l-th row of the observation matrix $\boldsymbol{Y}$. The prior acts as a quadratic penalization term in the logarithm of the posterior function, which results in the additive term $\alpha_l \boldsymbol{I}$ [1]. Another regularization technique introduces an additive noise term in equation 3.1. However, this would yield solutions similar to equation 3.5. Both methods can be interpreted as Tikhonov regularization techniques, which is reflected by the quadratic penalization term in the resulting objective function. [11, 1]

An alternative regularized ESN learning technique, which is very often used in literature [9, 14, 15] is based on the computation of the *pseudo-inverse*, e.g., by using a *singular value decomposition* (*SVD*) [1]. Compared to the Wiener-Hopf solution, this method is numerically more stable albeit computationally much more demanding [11].

However, the simple Bayesian regularization as well as the SVD-based pseudo-inverse require a manual determination of certain algorithm parameters, specifying the regularization strength. As discussed in section 2.1.1, Bayesian inference allows to estimate the parameters $\boldsymbol{w}_l$, as well as the corresponding regularizer α_l, although an exact analytical computation is often intractable. Later in this section, automatic regularization and simultaneous parameter estimation for ESN learning will be formulated within the variational Bayesian inference framework.

3.2.4 Limitations and Model Extensions

Even if ESNs have been shown to work well for many different tasks, they may also fail in some cases. This section will address limiting aspects for ESNs and will also discuss possible ESN model extensions which can sometimes significantly improve ESN learning. Without loss of generality, this work will consider extended ESN models, which can be written as

$$\boldsymbol{y}_l = \boldsymbol{w}_l^T \boldsymbol{S}(\boldsymbol{\phi}_l) + \boldsymbol{\xi}_l \tag{3.6}$$

with

$$\boldsymbol{S}(\boldsymbol{\phi}_l) = \begin{bmatrix} \boldsymbol{s}_0(\phi_{l,0}) \\ \vdots \\ \boldsymbol{s}_{M+K-1}(\phi_{l,M+K-1}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_0(\phi_{l,0}) \\ \vdots \\ \boldsymbol{x}_{M-1}(\phi_{l,M-1}) \\ \boldsymbol{u}_0(\phi_{l,M}) \\ \vdots \\ \boldsymbol{u}_{K-1}(\phi_{l,M+K-1}) \end{bmatrix} \tag{3.7}$$

and the parameter vector $\boldsymbol{\phi}_l = [\phi_{l,0}, \ldots, \phi_{l,M+K-1}]^T$. From equation 3.7 one can see that the i-th row in the extended state matrix $\boldsymbol{S}(\boldsymbol{\phi}_l)$ represents the transformed base function $\boldsymbol{s}_i(\phi_{l,i})$ corresponding to the l-th output $\boldsymbol{y}_l$.

It has been already discussed in section 1 that the effective memory capacity of an ESN is limited as it was shown in [14]. In his work, the author concludes that the short term memory capacity is bounded by the reservoir size M (under certain conditions). A simple way to boost the effective memory capacity of standard ESNs was proposed in [15]. In this work, the authors introduce ESNs with *delay & sum readouts*, which construct their output as a linear combination of the time shifted base functions. Thus, $\boldsymbol{s}(\phi_{l,i}) \equiv \boldsymbol{s}_i(\tau_{l,i}) = [s_i[-\tau_{l,i}], \ldots, s_i[N - \tau_{l,i} - 1]]$ and $\boldsymbol{S}(\boldsymbol{\phi}_l) \equiv \boldsymbol{S}(\boldsymbol{\tau}_l)$ with $\boldsymbol{\tau}_l = [\tau_{l,0}, \ldots, \tau_{l,M+K-1}]$. Furthermore, they were able to improve the ESN learning performance by optimizing the delay parameters using EM-type algorithms. In section 4, this work will utilize this extension for the recognition of handwriting trajectories.

Several other extension have been proposed so far. For instance, *leaky integrator neurons* were used in [31] to achieve time warping invariant reservoirs. A generalization of this principle was provided in [32] and [15] with the more generic concept of *filter neurons*. The reader should note that these extensions act within the feedback loop of the reservoir, which makes the estimation of - for instance the filter neuron parameters - quite complex. In some cases, heuristic approximations can reduce the computational effort, while keeping the results accurate enough to boost ESN's learning performance (see [33]). A very good overview about

state-of-the-art ESN techniques is given in [11]. Later in this work, training of the extended ESN parameters $\boldsymbol{\phi}_l$ for $l \in [0, \ldots, L-1]$ will be referred to as *extended ESN learning*.

3.2.5 Classification with ESNs

In [11], two main concepts for ESN-based classification are discussed. The first one uses a single ESN with L outputs, where L denotes the number of class labels. Assuming the current training example to belong to class c, the corresponding target output $y_c[n]$ is set to a certain temporal pattern, indicating the value *true* for this class. All other outputs $y_{l \neq c}[n]$ are set to a contrary sequence, implying that the current sample does not belong these classes. In the most simple case, constants are used (e.g., $+1$ for *true* and 0 for *false*). In the test phase, one decides for the class output, which has the maximum response regarding the positive pattern. This method will be evaluated in section 4.3.2 and will be referred to as *multi-class* ESN learning. [11]
The second scenario uses L independent ESNs, whereas each of them is trained to predict the trajectory of a single letter. During the classification of new samples, the prediction error is calculated for each of the specialized ESNs and the resulting class label is selected, according to the predictor which has performed best. This concept corresponds to the architecture used in section 4.3.3. [11]

3.3 A Variational Bayesian Formulation

In the previous sections, several important aspects of ESN learning have been discussed. A summary is given in table 3.1.

Task	Solution
Estimating $\boldsymbol{W}$	Wiener-Hopf, MAP
Automatic Regularization	Evidence Procedure, Model Selection
Extended ESN Learning	EM-type of algorithms

Table 3.1: Important Aspects of ESN Learning.

In order to handle each of these tasks, one could simply apply a combination of specialized algorithms, but in general, there would be no guarantee that this assembly of algorithms will still be optimal. In terms of Bayesian optimality, a joint realization of all these aspects requires inference of the posterior over the output weights, their corresponding hyperparameters as well as the extended ESN parameters. However, this is mathematically complex and thus, approximate inference algorithms would be of great interest. Using variational Bayesian inference algorithms, all tasks from table 3.1 can be jointly solved within one common optimality framework and thus, the resulting estimates remain interpretable from a statistical point of

view. Starting with a simple scenario, a variational Bayesian solution for learning extended ESNs is successively derived in this section, covering efficient estimation of extended model parameters as well as automatic regularization. Similar results are provided in [33] or [34][1].

For simplicity, ESNs are assumed to have only a single output for the following derivations such that $L = 1$, $\boldsymbol{y}_0 \equiv \boldsymbol{y}$ and $\boldsymbol{\xi}_0 \equiv \boldsymbol{\xi}$. As the classification of on-line handwriting data will require multiple-output ESNs, the algorithms will be extended for the multidimensional case in section 3.3.4, which will be straight forward under some weak conditions. Furthermore, it is assumed that the output non-linearity is a scalar identity transformation. However, any invertible function could be used, resulting in a simple transformation of the observed data, before estimating the model quantities.

3.3.1 Scenario A: Regularized ESN Learning

The first scenario deals with training standard ESNs combined with an automatic regularization. The problem of variational Bayesian learning of a simple linear model was already discussed in section 2.2.3. As mentioned before, ESNs can also be considered as linear systems, assuming the echo state base functions to be fixed. The vector $\boldsymbol{y}$ collecting the scalar output values for $n \in [0 \dots N-1]$ is given as

$$\boldsymbol{y} = \boldsymbol{w}^T \boldsymbol{S} + \boldsymbol{\xi}, \tag{3.8}$$

where $\boldsymbol{S}$ is the $M + K \times N$ state matrix, containing the extended system state at time steps $n \in [0 \dots N-1]$ and $\boldsymbol{\xi}$ models a random learning error with $\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$. In contrast to the simple model from section 2.2.3, the prior distribution over $\boldsymbol{w}$ is defined as $p(\boldsymbol{w}|\boldsymbol{\alpha}) = N(\boldsymbol{0}, \boldsymbol{A}^{-1})$ with $\boldsymbol{A} = \operatorname{diag}\{\alpha_0, \dots, \alpha_{M+K-1}\}$ and $\boldsymbol{\alpha} = [\alpha_0, \dots, \alpha_{M+K-1}]^T$. This means that each of the base functions in $\boldsymbol{S}$ has its own regularization constant such that a more specific regularization becomes possible. Furthermore, we assume $\boldsymbol{\alpha}$ to be random with $p(\boldsymbol{\alpha}) = \prod_{i=0}^{M+K-1} G(\alpha_i|a_i, b_i)$, where $G(\alpha_i|a_i, b_i)$ denotes a Gamma distribution over α_i, given the shape parameters a_i and b_i. The corresponding graphical model is given in figure 3.2.

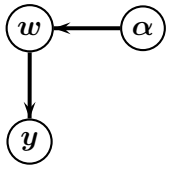


Figure 3.2: Regularized ESN learning.

The probabilistic model is fully specified by its joint density function over all variables, which is given by

[1] The reader should note that the related paper [34] is still in work.

$$p(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\alpha}) = p(\boldsymbol{y}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{\alpha})p(\boldsymbol{\alpha}) \tag{3.9}$$

and the taking the logarithm yields

$$\ln p(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\alpha}) = \ln p(\boldsymbol{y}|\boldsymbol{w}) + \ln p(\boldsymbol{w}|\boldsymbol{\alpha}) + \ln p(\boldsymbol{\alpha}). \tag{3.10}$$

The corresponding log-densities from equation 3.10 are given in table 3.2, whereas terms independent from $\boldsymbol{w}$ as well as $\boldsymbol{\alpha}$ are summarized as constants.

Log-Density	Expression
$\ln p(\boldsymbol{y}\|\boldsymbol{w})$	$-\frac{1}{2}\boldsymbol{w}^T\boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{S}^T\boldsymbol{w} + \boldsymbol{w}^T\boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{y}^T + const$
$\ln p(\boldsymbol{w}\|\boldsymbol{\alpha})$	$\frac{1}{2}\ln\lvert\boldsymbol{A}\rvert - \frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w} + const$
$\ln p(\boldsymbol{\alpha})$	$\sum_i \big((a_i - 1)\ln\alpha_i - b_i\alpha_i\big) + const$

Table 3.2: Log-densities for the model of *Scenario A*.

Note that $\boldsymbol{\alpha}$ is also a hidden variable and has to be estimated in order to obtain automatic regularization. As discussed in section 2.1.1, the posterior over $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ given $\boldsymbol{y}$ cannot be determined analytically for most practical scenarios. However, approximate posterior functions $q(\boldsymbol{w}, \boldsymbol{\alpha})$ can be found by minimizing the variational free energy $\mathcal{F}[q(\boldsymbol{w}, \boldsymbol{\alpha})]$. According to equation 2.15, the density $q(\boldsymbol{w}, \boldsymbol{\alpha})$ is chosen to factorize such that

$$q(\boldsymbol{w}, \boldsymbol{\alpha}) = q(\boldsymbol{w}) \prod_{i=0}^{M+K-1} q(\alpha_i). \tag{3.11}$$

Instead of using parameterized densities for the q-factors, free-form solutions shall be found using equation 2.19. Assuming the factor $q(\boldsymbol{w})$ to be of interest, the optimal factor $\hat{q}(\boldsymbol{w})$ takes the form

$$\begin{aligned}
\ln \hat{q}(\boldsymbol{w}) &= \mathbb{E}_{\boldsymbol{\alpha}}[\ln p(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\alpha})] + const \\
&= \mathbb{E}_{\boldsymbol{\alpha}}[\ln p(\boldsymbol{y}|\boldsymbol{w})] + \mathbb{E}_{\boldsymbol{\alpha}}[\ln p(\boldsymbol{w}|\boldsymbol{\alpha})] + const \\
&= -\frac{1}{2}\boldsymbol{w}^T\boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{S}^T\boldsymbol{w} + \boldsymbol{w}^T\boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{y}^T - \frac{1}{2}\boldsymbol{w}^T\hat{\boldsymbol{A}}\boldsymbol{w} + const \\
&= -\frac{1}{2}\boldsymbol{w}^T\underbrace{\left[\boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{S}^T + \hat{\boldsymbol{A}}\right]}_{\hat{\boldsymbol{C}}_w^{-1}}\boldsymbol{w} - \boldsymbol{w}^T\underbrace{\boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{y}^T}_{\hat{\boldsymbol{C}}_w^{-1}\hat{\boldsymbol{w}}} + const.
\end{aligned} \tag{3.12}$$

with $\hat{\boldsymbol{A}} = \mathbb{E}[\boldsymbol{A}]$. One can see that equation 3.12 is quadratic in $\boldsymbol{w}$ and thus the optimal density $\hat{q}(\boldsymbol{w})$ can be identified as $N(\boldsymbol{w}|\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w)$ with

$$\hat{\boldsymbol{C}}_w = \left[\boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{S}^T + \hat{\boldsymbol{A}}\right]^{-1} \tag{3.13}$$

$$\hat{\boldsymbol{w}} = \hat{\boldsymbol{C}}_w \boldsymbol{S}\boldsymbol{\Sigma}^{-1}\boldsymbol{y}^T. \tag{3.14}$$

With the knowledge, that $\hat{\boldsymbol{w}}$ is Gaussian, this procedure is now repeated with respect to the factors $\hat{q}(\alpha_i), i \in [0, \ldots, M+K-1]$. Again, the logarithm of the optimal free form solution for a single factor $q(\alpha_j)$ is given by

$$\begin{aligned}
\ln \hat{q}(\alpha_j) &= \mathbb{E}_{\boldsymbol{w},\alpha_{i\neq j}}\left[\ln p(\boldsymbol{w}|\boldsymbol{\alpha})\right] + \mathbb{E}_{\alpha_{i\neq j}}\left[\ln p(\boldsymbol{\alpha})\right] + const \\
&= \frac{1}{2}\ln\alpha_j - \frac{1}{2}\hat{c}_{w,j}\alpha_j - \frac{1}{2}\hat{w}_j^2\alpha_j + (a_j - 1)\ln\alpha_j - b_j\alpha_j + const \\
&= (\underbrace{\frac{1}{2} + a_j}_{\hat{a}_j} - 1)\ln\alpha_j - \underbrace{(\frac{1}{2}\hat{c}_{w,j} + \frac{1}{2}\hat{w}_j^2 + b_j)}_{\hat{b}_j}\alpha_j + const,
\end{aligned} \tag{3.15}$$

where $\hat{c}_{w,j}$ is the j-th element on the main diagonal of $\hat{\boldsymbol{C}}_w$. Equation 3.15 is the logarithm of a Gamma distribution and thus, the variational distribution over α_j is found as $\hat{q}(\alpha_j) = G(\alpha_j|\hat{a}_j, \hat{b}_j)$ with

$$\hat{a}_j = \frac{1}{2} + a_j \tag{3.16}$$

$$\hat{b}_j = \frac{1}{2}\left[\hat{c}_{w,j} + \hat{w}_j^2\right] + b_j \tag{3.17}$$

Now, also the term $\hat{\boldsymbol{A}}$ can be evaluated, using the standard results for the expected value of a Gamma distribution, such that

$$\hat{\boldsymbol{A}} = \mathbb{E}\left[\boldsymbol{A}\right] = \mathrm{diag}\left\{\frac{\hat{a}_0}{\hat{b}_0}, \ldots, \frac{\hat{a}_{M+K-1}}{\hat{b}_{M+K-1}}\right\} \tag{3.18}$$

Table 3.3 summarizes the estimation results for the variational densities.

Factor	Structure
$q(\boldsymbol{w})$	$N(\boldsymbol{w}\|\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w)$
$q(\alpha_j)$	$G(\alpha_j\|\hat{a}_j, \hat{b}_j)$

Table 3.3: q factors for *Scenario A*.

It is important to see that calculation of the factor $q(\boldsymbol{w})$ requires knowledge of the factor $q(\boldsymbol{\alpha})$ and vice versa. As described in section 2.2.2 this leads to an iterative estimation procedure, such that one factor is estimated, assuming all other factors to be known and so on.

3.3.2 Scenario B: Extended ESN Learning

In this section, the previous model is extended by additional state parameters according to section 3.2.4. Although this work will focus on delay & sum readouts, the following algorithm will allow any model extensions acting as tunable parameters of the base functions $\boldsymbol{s}_i(\phi_i)$. The corresponding model is given in equations 3.6 and 3.7 with $L = 1$ and $\boldsymbol{\phi}_0 = \boldsymbol{\phi}$.

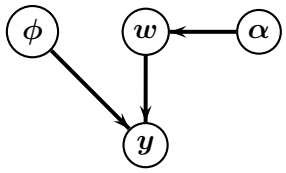


Figure 3.3: Extended ESN learning

The graphical model is illustrated in figure 3.3 and the logarithm of the joint density function over $\boldsymbol{y}$, $\boldsymbol{\phi}$, $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ can be written as

$$\ln p(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\phi}) = \ln p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{\phi}) + \sum_{i=0}^{M+K-1} \ln p(\phi_i) + \ln p(\boldsymbol{w}|\boldsymbol{\alpha}) + \ln p(\boldsymbol{\alpha}). \tag{3.19}$$

Equation 3.19 equals equation 3.10, except the additional terms $\ln p(\phi_i)$, acting as the log-priors of the parameter vector $\boldsymbol{\phi}$ and the fact that $\boldsymbol{\phi}$ also influences the data log-likelihood $\ln p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{\phi})$. As there is no prior information about the parameters of the randomly constructed reservoir functions, it seems natural to choose the priors over ϕ_i to be *non-informative* [1] over a certain interval $\mathcal{D}_\phi$, which means that within the range $\mathcal{D}_\phi$, each value for ϕ_i has the same prior probability. Thus, the parameters ϕ_i are assumed to be uniformly distributed on the interval $\mathcal{D}_\phi$. Clearly, the terms $\ln p(\phi_i)$ can be neglected, when working with the joint density from equation 3.19. The particular log-densities are set up as described in table 3.4.

Log-Density	Expression
$\ln p(\boldsymbol{y}\|\boldsymbol{w}, \boldsymbol{\phi})$	$-\frac{1}{2}\boldsymbol{w}^T\boldsymbol{S}(\boldsymbol{\phi})\boldsymbol{\Sigma}^{-1}\boldsymbol{S}^T(\boldsymbol{\phi})\boldsymbol{w} + \boldsymbol{w}^T\boldsymbol{S}(\boldsymbol{\phi})\boldsymbol{\Sigma}^{-1}\boldsymbol{y}^T + const$
$\ln p(\phi_i)$	$\begin{cases} const & \phi_i \in \mathcal{D}_\phi \\ 0 & else \end{cases}$
$\ln p(\boldsymbol{w}\|\boldsymbol{\alpha})$	$\frac{1}{2}\ln\|\boldsymbol{A}\| - \frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w} + const$
$\ln p(\boldsymbol{\alpha})$	$\sum_i \big((a_i - 1)\ln\alpha_i - b_i\alpha_i\big) + const$

Table 3.4: Log-densities for the model of *Scenario B*.

Very similar to the previous scenario (section 3.3.1), variational inference tries to find the approximate distribution

$$q(\boldsymbol{\phi}, \boldsymbol{w}, \boldsymbol{\alpha}) = q(\boldsymbol{w}) \prod_{i=0}^{M+K-1} q(\alpha_i) q(\phi_i). \tag{3.20}$$

The factors $q(\boldsymbol{w})$ and $q(\alpha_i)$ with $i \in [0, \ldots, M+K-1]$ can be determined the same way as before, i.e. computing free form solutions using equation 2.19. In contrast, unconstrained solutions for the factors $q(\phi_i)$ with $i \in [0, \ldots, M+K-1]$ cannot be determined in general, due to complex dependencies between the base functions and the corresponding parameters. As described in section 2.2.2, these factors can be estimated by restricting them to certain classes of parameterized functions. In this case, Dirac-delta functions[2] are used such that $q(\phi_i) = \delta(\phi_i - \hat{\phi}_i)$ and thus $\hat{\boldsymbol{\phi}} = [\hat{\phi}_0, \ldots, \hat{\phi}_{M+K-1}]$.

Similar to the previous scenario (3.3.2), the optimal log-factor $\hat{q}(\boldsymbol{w})$ is found as

$$\begin{aligned}
\ln \hat{q}(\boldsymbol{w}) &= \mathbb{E}_{\boldsymbol{\alpha},\boldsymbol{\phi}} \left[\ln p(\boldsymbol{y}|\boldsymbol{w})\right] + \mathbb{E}_{\boldsymbol{\alpha}} \left[\ln p(\boldsymbol{w}|\boldsymbol{\alpha})\right] + const \\
&= -\frac{1}{2} \boldsymbol{w}^T \boldsymbol{S}(\hat{\boldsymbol{\phi}}) \boldsymbol{\Sigma}^{-1} \boldsymbol{S}^T(\hat{\boldsymbol{\phi}}) \boldsymbol{w} + \boldsymbol{w}^T \boldsymbol{S}(\hat{\boldsymbol{\phi}}) \boldsymbol{\Sigma}^{-1} \boldsymbol{y}^T - \frac{1}{2} \boldsymbol{w}^T \hat{\boldsymbol{A}} \boldsymbol{w} + const \\
&= -\frac{1}{2} \boldsymbol{w}^T \underbrace{\left[\boldsymbol{S}(\hat{\boldsymbol{\phi}}) \boldsymbol{\Sigma}^{-1} \boldsymbol{S}^T(\hat{\boldsymbol{\phi}}) + \hat{\boldsymbol{A}}\right]}_{\hat{\boldsymbol{C}}_w^{-1}} \boldsymbol{w} - \boldsymbol{w}^T \underbrace{\boldsymbol{S}(\hat{\boldsymbol{\phi}}) \boldsymbol{\Sigma}^{-1} \boldsymbol{y}^T}_{\hat{\boldsymbol{C}}_w^{-1} \hat{\boldsymbol{w}}} + const,
\end{aligned} \tag{3.21}$$

and can be identified as a Gaussian with $\hat{q}(\boldsymbol{w}) = N(\boldsymbol{w}|\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w)$ with

$$\hat{\boldsymbol{C}}_w = \left[\boldsymbol{S}(\hat{\boldsymbol{\phi}}) \boldsymbol{\Sigma}^{-1} \boldsymbol{S}^T(\hat{\boldsymbol{\phi}}) + \hat{\boldsymbol{A}}\right]^{-1} \tag{3.22}$$

$$\hat{\boldsymbol{w}} = \hat{\boldsymbol{C}}_w \boldsymbol{S}(\hat{\boldsymbol{\phi}}) \boldsymbol{\Sigma}^{-1} \boldsymbol{y}^T. \tag{3.23}$$

The factor $\ln \hat{q}(\alpha_j)$ does not depend on the parameter vector $\boldsymbol{\phi}$ and thus, the variational distribution $\hat{q}(\alpha_j)$ as well as the matrix $\hat{\boldsymbol{A}}$ will be the same as in section 3.3.1.

For estimation of the factor $q(\phi_k)$, equation 2.17 is used as a starting point for further derivations. As the entropy $\mathcal{H}\left[q(\phi_k)\right]$ is constant, the optimal parameter $\hat{\phi}_k$ is found by maximizing the expected logarithm of the joint density function $\mathbb{E}\left[\ln p(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\phi})\right]$.

$$\begin{aligned}
\hat{\phi}_k &= \underset{\tilde{\phi}_k \in \mathcal{D}_\phi}{arg\,max}\, \mathbb{E}\left[\ln p(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\phi})\right] \\
&= \underset{\tilde{\phi}_k \in \mathcal{D}_\phi}{arg\,max}\, \left[\mathbb{E}_{\boldsymbol{w},\boldsymbol{\phi}} \left[\ln p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{\phi})\right] + \mathbb{E}_{\phi_k} \left[\ln p(\phi_k)\right]\right]
\end{aligned} \tag{3.24}$$

With $p(\phi_k)$ being flat over $\mathcal{D}_\phi$ and using some standard identities from section A.2, equation 3.24 can be expressed as

[2] Choosing Dirac-delta distributions yields point estimates $\hat{\phi}_i$ of the variational parameter ϕ_i [18].

$$\hat{\phi}_k = \underset{\tilde{\phi}_k \in \mathcal{D}_\phi}{arg\,max} \left[\ln p(\boldsymbol{y}|\hat{\boldsymbol{w}}, \tilde{\boldsymbol{\phi}}) - \frac{1}{2} Tr \left[\hat{\boldsymbol{C}}_w \boldsymbol{S}(\tilde{\boldsymbol{\phi}}) \boldsymbol{\Sigma}^{-1} \boldsymbol{S}^T(\tilde{\boldsymbol{\phi}}) \right] \right], \tag{3.25}$$

with $\tilde{\boldsymbol{\phi}} = [\hat{\phi}_0, \ldots, \tilde{\phi}_k, \ldots, \hat{\phi}_{M+K-1}]$. Note that equation 3.25 has to be solved numerically in general. As mentioned before, this work will use delay & sum readouts as a powerful extension to the standard ESN model. The variational expressions described above can be directly used for estimating the time delays τ_i with $i \in [0, \ldots, M+K-1]$.

The resulting approximating densities are given in table 3.5.

Factor	Structure
$q(\boldsymbol{w})$	$N(\boldsymbol{w}\|\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w)$
$q(\alpha_j)$	$G(\alpha_j\|\hat{a}_j, \hat{b}_j)$
$q(\phi_k)$	$\delta(\phi_k - \hat{\phi}_k)$

Table 3.5: q factors for *Scenario B*.

3.3.3 Scenario C: The VBSAGE Algorithm

The algorithm derived in the previous section, estimates the parameters ϕ_k for $k = [0, \ldots, M+K-1]$ independently from each other, which is a direct consequence of the mean field approximation. From the graphical model in figure 3.3 it can be seen that the parameters ϕ_k conditionally depend on each other given the observation $\boldsymbol{y}$ (*explaining away*, [1]). As a consequence, the factorization in equation 3.20 will yield only rough approximations of the true posterior. Intuitively, one could try to estimate $q(\boldsymbol{\phi})$ at once, which is simply done by discarding the full factorization $q(\boldsymbol{\phi}) = \prod_{i=0}^{M+K-1} q(\phi_i)$. Even if this strategy yields much better approximation results in general, it has less practical relevance, as jointly optimizing $\{\phi_0, \ldots, \phi_{M+K-1}\}$ is extremely complex from a computational viewpoint.

An alternative approach for efficient estimation of high dimensional hidden data is to utilize EM-type algorithms [16]. In particular, this section will deal with the Space Alternating Generalized EM algorithm (*SAGE*, [35, 36]). A variational Bayesian formulation of the SAGE algorithm is provided in [37][3] and is referred to as the *VBSAGE* algorithm. Also VBSAGE-based algorithms for extended ESN learning can be found in [33] and [34].

Compared to the classical EM algorithm, only a subset of the parameters $\boldsymbol{\phi}$ is estimated during one iteration. By varying the parameter subset (i.e. *space alternations*), each of the

[3]The reader should note that this work has not been published yet.

parameters in $\boldsymbol{\phi}$ can be estimated using the SAGE algorithm. Formally, this can be realized by introducing the concept of *admissible* hidden data, represented by a latent variable $\mathcal{H}$ in a graphical model, allowing to estimate the parameter subset of interest, given $\mathcal{H}$. The reader should consult [35] or [37] for a detailed theoretical description of the SAGE and VBSAGE-type algorithms.

For the extended ESN learning problem, the VBSAGE algorithm can be simply used within the previous VB algorithm, by rearranging the graphical model from figure 3.3 in terms of an artificial hidden variable $\boldsymbol{h}_k$. This variable denotes the admissible hidden data for estimating the parameter ϕ_k of the k-th base function $\boldsymbol{s}_k(\phi_k)$. The set $\overline{\mathcal{K}}$ denotes all indices $i \in [0, \ldots, M+K-1]$ except the index k itself. The final graphical model is given in figure 3.4.

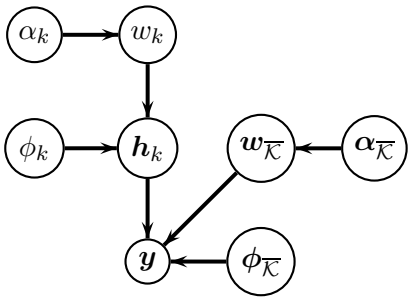


Figure 3.4: Extended ESN learning using the VB-SAGE algorithm.

Clearly, also the data model will change according to the hidden data $\boldsymbol{h}_k$, acting as the k-th summand in the output equation. The modified output $\boldsymbol{y}$ can be written as

$$\begin{aligned} \boldsymbol{y} &= \sum_{i \in \overline{\mathcal{K}}} (w_i \boldsymbol{s}_i(\phi_i) + \boldsymbol{\xi}_i) + \boldsymbol{h}_k \\ &= \sum_{i \in \overline{\mathcal{K}}} w_i \boldsymbol{s}_i(\phi_i) + \boldsymbol{h}_k + \boldsymbol{\xi}_{\overline{\mathcal{K}}} \end{aligned} \tag{3.26}$$

with $\boldsymbol{\xi}_{\overline{\mathcal{K}}} = \sum_{i \in \overline{\mathcal{K}}} \boldsymbol{\xi}_i$ and

$$\boldsymbol{h}_k = w_k \boldsymbol{s}_k(\phi_k) + \boldsymbol{\xi}_k. \tag{3.27}$$

In equations 3.26 and 3.27, $\boldsymbol{\xi}_i, \forall i \in \overline{K}$ and $\boldsymbol{\xi}_k$ denote noise terms, each perturbing a single base function. As the overall learning error in the observed output $\boldsymbol{y}$ is again assumed to be a Gaussian random vector $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, the single noise components are also Gaussian and must decompose such that $\boldsymbol{\xi}_{\overline{\mathcal{K}}} + \boldsymbol{\xi}_k = \boldsymbol{\xi}$. Furthermore, the corresponding covariance matrices sum up as $\boldsymbol{\Sigma}_{\overline{\mathcal{K}}} + \boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$ and additionally, this work assumes $\boldsymbol{\Sigma}_k = \beta\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}_{\overline{\mathcal{K}}} = (1-\beta)\boldsymbol{\Sigma}$. [33, 34]
Under these assumptions, the probability density function for the hidden data vector $\boldsymbol{h}_k$ is given as

$$p(\boldsymbol{h}_k|w_k,\phi_k) = N(\boldsymbol{h}_k|w_k\boldsymbol{s}_k(\phi_k),\boldsymbol{\Sigma}_k) \tag{3.28}$$

and in a similar fashion, the density function over the output can be written as

$$p(\boldsymbol{y}|\boldsymbol{h}_k,\boldsymbol{w}_{\overline{\mathcal{K}}},\boldsymbol{\phi}_{\overline{\mathcal{K}}}) = N\Big(\boldsymbol{y}|\sum_{i\in\overline{\mathcal{K}}} w_i\boldsymbol{s}_i(\phi_i)+\boldsymbol{h}_k,\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}\Big). \tag{3.29}$$

The rest of the model remains the same as in the previous scenario. The logarithm of the joint density over all variables takes the simple form

$$\begin{aligned}\ln p(\boldsymbol{y},\boldsymbol{h}_k,\boldsymbol{w},\boldsymbol{\alpha},\boldsymbol{\phi}) = \ln p(\boldsymbol{y}|\boldsymbol{h}_k,\boldsymbol{w}_{\overline{\mathcal{K}}},\boldsymbol{\phi}_{\overline{\mathcal{K}}}) + \ln p(\boldsymbol{h}_k|w_k,\phi_k)\\ + \sum_{i=0}^{M+K-1} \ln p(\phi_i) + \ln p(\boldsymbol{w}|\boldsymbol{\alpha}) + \ln p(\boldsymbol{\alpha}).\end{aligned} \tag{3.30}$$

and the corresponding log-densities are summarized in table 3.6.

Log-Density	Expression
$\ln p(\boldsymbol{y}\|\boldsymbol{h}_k,\boldsymbol{w}_{\overline{\mathcal{K}}},\boldsymbol{\phi}_{\overline{\mathcal{K}}})$	$-\frac{1}{2}\sum_{i\in\overline{\mathcal{K}}} w_i^2\boldsymbol{s}_i(\phi_i)\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{s}_i^T(\phi_i) - \sum_{i\in\overline{\mathcal{K}}} w_i\boldsymbol{s}_i(\phi_i)\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{h}_k^T$ $+\sum_{i\in\overline{\mathcal{K}}} w_i\boldsymbol{s}_i(\phi_i)\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{y}^T - \frac{1}{2}\boldsymbol{h}_k\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{h}_k^T + \boldsymbol{h}_k\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}\boldsymbol{y}^T + const$
$\ln p(\boldsymbol{h}_k\|w_k,\phi_k)$	$-\frac{1}{2}\boldsymbol{h}_k\boldsymbol{\Sigma}_k^{-1}\boldsymbol{h}_k^T + w_k\boldsymbol{s}_k(\phi_k)\boldsymbol{\Sigma}_k^{-1}\boldsymbol{h}_k^T - \frac{1}{2}w_k^2\boldsymbol{s}_k(\phi_k)\boldsymbol{\Sigma}_k^{-1}\boldsymbol{s}_k^T(\phi_k) + const$
$\ln p(\phi_i)$	$\begin{cases} const & \phi_i\in\mathcal{D}_\phi \\ 0 & else \end{cases}$
$\ln p(\boldsymbol{w}\|\boldsymbol{\alpha})$	$\frac{1}{2}\ln\|\boldsymbol{A}\| - \frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w} + const$
$\ln p(\boldsymbol{\alpha})$	$\sum_i \big((a_i-1)\ln\alpha_i - b_i\alpha_i\big) + const$

Table 3.6: Log-densities for the model of *Scenario C*.

Including the hidden variable $\boldsymbol{h}_k$, the approximate density $q(\boldsymbol{h}_k,\boldsymbol{\phi},\boldsymbol{w},\boldsymbol{\alpha})$ is assumed to factorize as

$$q(\boldsymbol{h}_k,\boldsymbol{\phi},\boldsymbol{w},\boldsymbol{\alpha}) = q(\boldsymbol{h}_k)q(\boldsymbol{w})\prod_{i=0}^{M+K-1} q(\alpha_i)q(\phi_i). \tag{3.31}$$

As in section 3.3.2, the factors $q(\phi_i)$ are restricted to be Dirac-delta distributions. Note that the modification of the model in terms of the hidden data variable $\boldsymbol{h}_k$ does not change the structure regarding the node $\boldsymbol{\alpha}$. For this reason, also the factors $q(\alpha_i)$ remain the same as in section 3.3.2 and can directly be used for the further calculations. In this particular case, it makes sense to start with the calculation of the factor for the hidden data variable $q(\boldsymbol{h}_k)$.

$$\begin{aligned}\ln \hat{q}(\boldsymbol{h}_k) &= \mathbb{E}_{\boldsymbol{w},\boldsymbol{\phi}}\left[\ln p(\boldsymbol{y}|\boldsymbol{h}_k, \boldsymbol{w}_{\overline{\mathcal{K}}}, \boldsymbol{\phi}_{\overline{\mathcal{K}}}) + \ln p(\boldsymbol{h}_k|w_k, \phi_k)\right] \\ &= -\sum_{i\in\overline{\mathcal{K}}} \hat{w}_i \boldsymbol{s}_i(\hat{\phi}_i)\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{h}_k^T - \frac{1}{2}\boldsymbol{h}_k\left[\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1} + \boldsymbol{\Sigma}_k^{-1}\right]\boldsymbol{h}_k^T + \boldsymbol{h}_k\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{y}^T \\ &\quad + \hat{w}_k\boldsymbol{s}_k(\hat{\phi}_k)\boldsymbol{\Sigma}_k^{-1}\boldsymbol{h}_k^T + const\end{aligned}$$

By adding and subtracting the missing k-th summand of the sum in equation 3.32, one obtains

$$\begin{aligned}\ln \hat{q}(\boldsymbol{h}_k) &= -\boldsymbol{h}_k\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{S}^T(\hat{\boldsymbol{\phi}})\hat{\boldsymbol{w}} - \frac{1}{2}\boldsymbol{h}_k\underbrace{\left[\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1} + \boldsymbol{\Sigma}_k^{-1}\right]}_{\hat{\boldsymbol{S}}_h^{-1}}\boldsymbol{h}_k^T \\ &\quad + \hat{w}_k\boldsymbol{s}_k(\hat{\phi}_k)\left[\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1} + \boldsymbol{\Sigma}_k^{-1}\right]\boldsymbol{h}_k^T + \boldsymbol{h}_k\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}\boldsymbol{y}^T + const \\ &= -\frac{1}{2}\boldsymbol{h}_k\hat{\boldsymbol{S}}_h^{-1}\boldsymbol{h}_k^T \\ &\quad + \boldsymbol{h}_k\Big(\hat{\boldsymbol{S}}_h^{-1}\boldsymbol{s}_k^T(\hat{\phi}_k)\hat{w}_k + \boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\left(\boldsymbol{y}^T - \boldsymbol{S}^T(\hat{\boldsymbol{\phi}})\hat{\boldsymbol{w}}\right)\Big) + const \\ &= -\frac{1}{2}\boldsymbol{h}_k\hat{\boldsymbol{S}}_h^{-1}\boldsymbol{h}_k^T + \\ &\quad + \boldsymbol{h}_k\hat{\boldsymbol{S}}_h^{-1}\underbrace{\left(\boldsymbol{s}_k^T(\hat{\phi}_k)\hat{w}_k + \hat{\boldsymbol{S}}_h\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\left(\boldsymbol{y}^T - \boldsymbol{S}^T(\hat{\boldsymbol{\phi}})\hat{\boldsymbol{w}}\right)\right)}_{\hat{\boldsymbol{h}}_k^T} + const,\end{aligned} \tag{3.32}$$

where $\hat{w}_k = \mathbb{E}[w_k]$ and $\hat{\boldsymbol{w}} = \mathbb{E}[\boldsymbol{w}]$. From equation 3.32 the optimal factor $\hat{q}(\boldsymbol{h}_k)$ can be determined as a Gaussian distribution with $q(\boldsymbol{h}_k) = N(\boldsymbol{h}_k|\hat{\boldsymbol{h}}_k, \hat{\boldsymbol{S}}_h)$ with

$$\hat{\boldsymbol{S}}_h = \left[\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1} + \boldsymbol{\Sigma}_k^{-1}\right]^{-1} \tag{3.33}$$

$$\hat{\boldsymbol{h}}_k = \hat{w}_k\boldsymbol{s}(\hat{\phi}_k) + \left(\boldsymbol{y} - \hat{\boldsymbol{w}}^T\boldsymbol{S}(\hat{\boldsymbol{\phi}})\right)\hat{\boldsymbol{S}}_h\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1} \tag{3.34}$$

and noting that $\boldsymbol{\Sigma}_{\overline{\mathcal{K}}} = (1-\beta)\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}_k = \beta\boldsymbol{\Sigma}$ one finds

$$\hat{\boldsymbol{S}}_h = \beta(1-\beta)\boldsymbol{\Sigma} \tag{3.35}$$

$$\hat{\boldsymbol{h}}_k = \hat{w}_k\boldsymbol{s}(\hat{\phi}_k) + \beta\left(\boldsymbol{y} - \hat{\boldsymbol{w}}^T\boldsymbol{S}(\hat{\boldsymbol{\phi}})\right). \tag{3.36}$$

Given the admissible hidden data $\boldsymbol{h}_k$, the optimal values for the parameters ϕ_k can be easily obtained. As in the scenario of section 3.3.2, the optimal factor $\hat{q}(\phi_k)$ is found by maximizing the expected logarithm of the joint density function with respect to $\tilde{\phi}_k$:

$$\begin{aligned}\hat{\phi}_k &= \underset{\tilde{\phi}_k\in\mathcal{D}_\phi}{arg\,max}\,\mathbb{E}\left[\ln p(\boldsymbol{y}, \boldsymbol{h}_k, \boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\phi})\right] \\ &= \underset{\tilde{\phi}_k\in\mathcal{D}_\phi}{arg\,max}\,\left[\ln p(\hat{\boldsymbol{h}}_k|\hat{w}_k, \tilde{\phi}_k) - \frac{1}{2}\hat{c}_{w,k}\boldsymbol{s}_k(\tilde{\phi}_k)\boldsymbol{\Sigma}_k^{-1}\boldsymbol{s}_k^T(\tilde{\phi}_k)\right]\end{aligned} \tag{3.37}$$

When comparing equations 3.37 and 3.25 from the previous section, one can observe interesting similarities between the resulting estimation expressions for ϕ_k. The difference is that using the VBSAGE algorithm, the influence of all other base functions and their parameters is absorbed into the calculation via the admissible hidden data vector $\boldsymbol{h}_k$. Clearly, this ends up with a more efficient estimation of ϕ_k, as it avoids several large matrix multiplications. One could also question if the VBSAGE based algorithm from this section allows better approximation results, as $\boldsymbol{h}_k$ enforces conditional independence between ϕ_k and $\boldsymbol{\phi}_{\overline{\mathcal{K}}}$ given $\boldsymbol{y}$, which corresponds to the chosen factorization of $q(\boldsymbol{\phi})$. However, this is an interesting but non-trivial question and thus, a detailed theoretical as well as numerical comparison of these algorithms should be topic of future research.
Equation 3.37 can be further simplified, if the model is extended by simple delay & sum readouts, as the terms $\boldsymbol{s}_k(\tilde{\phi}_k)\boldsymbol{\Sigma}_k^{-1}\boldsymbol{s}_k^T(\tilde{\phi}_k)$ is asymptotically invariant against time shifts. As a consequence, determining $\hat{\phi}_k$ reduces to maximizing the weighted cross-correlation between $\boldsymbol{s}_k(\tilde{\phi}_k) \equiv \boldsymbol{s}_k(\tilde{\tau}_k)$ and $\hat{\boldsymbol{h}}_k$ with respect to the time lag $\tilde{\tau}_k$.

To complete the derivation of the VBSAGE-based ESN learning algorithm, the free form solution for $\boldsymbol{w}$ is calculated. Again, the expected logarithm of the joint density function is the key quantity, when determining the optimal free form solution for $q(\boldsymbol{w})$:

$$\begin{aligned}
\ln \hat{q}(\boldsymbol{w}) &= \mathbb{E}_{\boldsymbol{h}_k,\boldsymbol{\phi}_{\overline{\mathcal{K}}}}\left[\ln p(\boldsymbol{y}|\boldsymbol{h}_k,\boldsymbol{w}_{\overline{\mathcal{K}}},\boldsymbol{\phi}_{\overline{\mathcal{K}}})\right] + \mathbb{E}_{\boldsymbol{h}_k,\phi_k}\left[\ln p(\boldsymbol{h}_k|w_k,\phi_k)\right] \\
&\quad + \mathbb{E}_{\boldsymbol{\alpha}}\left[\ln p(\boldsymbol{w}|\boldsymbol{\alpha})\right] \\
&= -\frac{1}{2}\sum_{i\in\overline{\mathcal{K}}} w_i^2 \boldsymbol{s}_i(\hat{\phi}_i)\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{s}_i^T(\hat{\phi}_i) - \sum_{i\in\overline{\mathcal{K}}} w_i \boldsymbol{s}_i(\hat{\phi}_i)\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\hat{\boldsymbol{h}}_k^T \\
&\quad + \sum_{i\in\overline{\mathcal{K}}} w_i \boldsymbol{s}_i(\hat{\phi}_i)\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}^{-1}\boldsymbol{y}^T + w_k \boldsymbol{s}_k(\hat{\phi}_k)\boldsymbol{\Sigma}_k^{-1}\hat{\boldsymbol{h}}_k^T - \frac{1}{2}w_k^2 \boldsymbol{s}_k(\hat{\phi}_k)\boldsymbol{\Sigma}_k^{-1}\boldsymbol{s}_k^T(\hat{\phi}_k) \\
&\quad - \frac{1}{2}\boldsymbol{w}^T\hat{\boldsymbol{A}}\boldsymbol{w}
\end{aligned} \tag{3.38}$$

One can see that equation 3.38 also depends on the expected value of the admissible hidden data vector $\hat{\boldsymbol{h}}_k$. As the weights $\boldsymbol{w}$ are estimated jointly (which is reflected by the choice of a non-factorizing density $q(\boldsymbol{w})$), the admissible data vector $\boldsymbol{h}_k$ becomes redundant. To get rid of this term, the joint density of $\boldsymbol{y}$ and $\boldsymbol{h}_k$, appearing as $p(\boldsymbol{y},\boldsymbol{h}_k|\boldsymbol{w},\boldsymbol{\phi}) = p(\boldsymbol{y}|\boldsymbol{h}_k,\boldsymbol{w}_{\overline{\mathcal{K}}},\boldsymbol{\phi}_{\overline{\mathcal{K}}})p(\boldsymbol{h}_k|w_k,\phi_k)$ is marginalized with respect to $\boldsymbol{h}_k$ such that

$$p(\boldsymbol{y}|\boldsymbol{w},\boldsymbol{\phi}) = \int p(\boldsymbol{y},\boldsymbol{h}_k|\boldsymbol{w},\boldsymbol{\phi})d\boldsymbol{h}_k. \tag{3.39}$$

As both $p(\boldsymbol{y}|\boldsymbol{h}_k,\boldsymbol{w}_{\overline{\mathcal{K}}},\boldsymbol{\phi}_{\overline{\mathcal{K}}})$ as well as $p(\boldsymbol{h}_k|w_k,\phi_k)$ are Gaussian, standard results for marginal Gaussian distributions can be applied. The densities can be written as

$$\begin{aligned}
p(\boldsymbol{h}_k|w_k,\phi_k) &= N\Big(\boldsymbol{h}_k|w_k\boldsymbol{s}_k(\phi_k),\boldsymbol{\Sigma}_k^{-1}\Big) \\
p(\boldsymbol{y}|\boldsymbol{h}_k,\boldsymbol{w}_{\overline{\mathcal{K}}},\boldsymbol{\phi}_{\overline{\mathcal{K}}}) &= N\Big(\boldsymbol{y}|\boldsymbol{h}_k\boldsymbol{M}+\boldsymbol{v},\boldsymbol{\Sigma}_{\overline{\mathcal{K}}}\Big),
\end{aligned}$$

with

$$\boldsymbol{M} = \boldsymbol{I} \text{ and} \tag{3.40}$$

$$\boldsymbol{v} = \sum_{i \in \overline{\mathcal{K}}} w_i \boldsymbol{s}_i(\phi_i). \tag{3.41}$$

Now, the marginal distribution $p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{\phi})$ is given as [1]

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{\phi}) &= N\Big(\boldsymbol{y}|w_k \boldsymbol{s}_k(\phi_k)\boldsymbol{M} + \boldsymbol{v}, \boldsymbol{\Sigma}_{\overline{\mathcal{K}}} + \boldsymbol{M}\boldsymbol{\Sigma}_k\boldsymbol{M}^T\Big) \\ &= N\Big(\boldsymbol{y}|\boldsymbol{w}^T\boldsymbol{S}(\boldsymbol{\phi}), \boldsymbol{\Sigma}_{\overline{\mathcal{K}}} + \boldsymbol{\Sigma}_k\Big) \\ &= N\Big(\boldsymbol{y}|\boldsymbol{w}^T\boldsymbol{S}(\boldsymbol{\phi}), \boldsymbol{\Sigma}\Big) \end{aligned} \tag{3.42}$$

Observing that equation 3.42 exactly matches the data likelihood from section 3.3.2, also the estimation expressions for the optimal factor $q(\boldsymbol{w})$ are given by equation 3.21 from section 3.3.2. This can also be seen from the resulting graphical model after the marginalization, which is the same as the model depicted in figure 3.3. The estimated q-factors are summarized in table 3.7.

Factor	Structure
$q(\boldsymbol{w})$	$N(\boldsymbol{w}\|\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w)$
$q(\alpha_j)$	$G(\alpha_j\|\hat{a}_j, \hat{b}_j)$
$q(\boldsymbol{h}_k)$	$N(\boldsymbol{h}_k\|\hat{\boldsymbol{h}}_k, \hat{\boldsymbol{S}}_h)$
$q(\phi_k)$	$\delta(\phi_k - \hat{\phi}_k)$

Table 3.7: q factors for *Scenario C*.

Similar results can be found in [34] and also [33].

3.3.4 Extension to Multiple Outputs

So far, a variational Bayesian formulation of ESN learning has been discussed for three different scenarios (sections 3.3.1, 3.3.2 and 3.3.3). The presented algorithms can only handle ESNs with a single output $y[n]$. However, in many practical applications it is required to consider models with multiple outputs, e.g., as in handwritten symbol recognition (see section 4). Within a probabilistic model, the L outputs are perturbed by random learning errors $\xi_l[n]$ with $l \in [0, \ldots, L-1]$, which may also depend on each other in the most general formulation. In this case, also the variational estimation expressions for the model quantities would change accordingly.

However, this work assumes an even simpler, but still plausible case, where the L noise

components, each perturbing one particular output, are modeled to be independent from each other, such that the algorithms from the previous sections can be directly applied.

Chapter 4

Simulations

So far, mathematical descriptions of several ESN-based learning algorithms have been provided. In this section, the discussed algorithms will be evaluated and numerical simulations using MATLAB(2008a, The MathWorks, Natick, MA) [38] will be performed for comparison. Section 4.1 will address important implementation aspects of the VB algorithms from sections 3.3.1 to 3.3.3. In section 4.2, optimality of these algorithms will be verified empirically using synthetic data (i.e., the Mackey-Glass time series). Afterwards, section 4.3 will evaluate the ESN performance using two different learning schemes for real-world handwriting recognition (sections 4.3.2 and 4.3.3). The ESN-based learning algorithms, considered in this section, are summarized below:

Std-ESN: A standard ESN is trained using the SVD-based pseudo-inverse as described in section 3.2.2. For all simulations, MATLAB's default setup for the SVD computation was used.

Ext-ESN: An ESN with delay & sum readouts is trained using the SVD-based pseudo-inverse. The time delays $\boldsymbol{\tau}_l$ are randomly chosen. In many cases, better results can be obtained, if only a particular number of the time delays is set to a value different than zero.

VB-ESN-A: The VB learning algorithm from section 3.3.1. The algorithm combines training of the output weights $\boldsymbol{w}$ with an automatic regularization. In an iterative manner, the weights as well as the hyperparameters $\boldsymbol{\alpha}$ are estimated consecutively.

VB-ESN-B: The VB learning algorithm from section 3.3.2. The output weights $\boldsymbol{w}$, the corresponding hyperparamters $\boldsymbol{\alpha}$ as well as the delay & sum readouts are trained.

VB-ESN-C: The VB learning algorithm from section 3.3.3. The output weights $\boldsymbol{w}$, the corresponding hyperparamters $\boldsymbol{\alpha}$ as well as the delay & sum readouts are trained using a VBSAGE-type algorithm.

Note that for all the following simulations, the covariance matrix of the random learning error is assumed to be $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$.

4.1 Implementation Aspects

It has been discussed in sections 2 and 3, that in general, the variational parameters have to be estimated in an iterative manner as they may implicitly depend on each other. As a consequence, the algorithms will show a cyclic structure, whereas one particular q-factor is re-estimated during a single step, given the most recent estimates of all other q-factors. Even if variational inference does not require any particular order of estimating the variational distributions, the performance may strongly depend on the choice of the update sequence. Furthermore, the resulting estimates may also depend on the initialization. One can immediately see that there are many degrees of freedom when using variational inference algorithms. For this reason, algorithms **VB-ESN-A**, **VB-ESN-B** and **VB-ESN-C**, will be further specified in terms of structure and implementation details. To keep the descriptions as compact as possible, the following considerations will assume ESNs with single outputs. As discussed in section 3.3.4 an extension to multiple outputs is straight forward.

When performing algorithm **VB-ESN-A**, the estimation of $q(\boldsymbol{w})$ requires knowledge of the factors $q(\alpha_j)$ for $j \in [0, \ldots, M+L-1]$ and vice versa. One possible realization of the algorithm is given by algorithm 1.

Algorithm 1 VB-ESN-A

Initialize $\hat{a}_i$, $\hat{b}_i$ for $i \in [0, \ldots, M+L-1]$
repeat
  $\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w \leftarrow$ 3.13, 3.14 {Estimate $q(\boldsymbol{w})$}
  for $j \in [0, \ldots, M+L-1]$ **do**
    $\hat{a}_j, \hat{b}_j \leftarrow$ 3.16, 3.17 {Estimate $q(\alpha_j)$}
  end for
until Convergence

As the factor $q(\boldsymbol{w})$ is estimated first, all other factors (i.e., $q(\alpha_j)$) need to be initialized at the beginning. This could for instance be done randomly or by applying heuristics. The algorithm cycles through the estimation of $q(\boldsymbol{w})$ and $q(\alpha_j)$ until a certain convergence criterion is met. Often, a fixed number of iterations is used or the algorithm is aborted as soon as the free energy does not significantly decrease any more. Of course other criteria for convergence could be used.
In algorithm **VB-ESN-B** also the time delays $\boldsymbol{\tau}$ are trained[1]. The update order shown in algorithm 2 was empirically found to work best for the simulations performed in the context of this thesis.

[1]Note that in algorithms 2 and 3 a general parameter vector $\boldsymbol{\phi}$ is used instead of explicitly modeling the time delays $\boldsymbol{\tau}$.

Algorithm 2 VB-ESN-B

Initialize $\hat{a}_i$, $\hat{b}_i$, $\hat{\phi}_i$ for $i \in [0, \ldots, M+L-1]$
Initialize $\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w$
repeat
  for $k \in [0, \ldots, M+L-1]$ **do**
    $\hat{\phi}_k \leftarrow 3.25$ {Estimate $q(\phi_k)$}
  end for
  $\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w \leftarrow 3.22, 3.23$ {Estimate $q(\boldsymbol{w})$}
  for $j \in [0, \ldots, M+L-1]$ **do**
    $\hat{a}_j, \hat{b}_j \leftarrow 3.16, 3.17$ {Estimate $q(\alpha_j)$}
  end for
until Convergence

In this case, the factors $q(\boldsymbol{w})$ and $q(\alpha_j)$ have to be set to certain values during the algorithm initialization. Note that the estimation of the time delays $\hat{\tau}_i$ for $i \in [0, \ldots, M+K-1]$ strongly depends on the initialization of the other factors. If $\hat{a}_i$ and $\hat{b}_i$ are chosen such that $\hat{\alpha}_i = \frac{\hat{a}_i}{\hat{b}_i}$ takes a high value, the weight $\hat{w}_i$, corresponding to the i-th base function will be highly regularized during the subsequent calculation of $\hat{\boldsymbol{w}}$. As a consequence, it will be likely that the corresponding time delay $\hat{\tau}_i$ changes during the next update cycle and so on. For the simulations from sections 4.3.2 and 4.3.3 very high initial values for $\hat{\alpha}_i$ will be used (i.e., $10e5 - 10e10$), as the best results have been achieved using this setting.

As algorithm **VB-ESN-C** is quite similar to algorithm **VB-ESN-B**, the previous considerations concerning the update order also hold in this case. The realization of **VB-ESN-C**, which will be used in the following simulations, is given by algorithm 3.

Algorithm 3 VB-ESN-C

Initialize $\hat{a}_i$, $\hat{b}_i$, $\hat{\phi}_i$ for $i \in [0, \ldots, M+L-1]$
Initialize $\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w$
repeat
  for $k \in [0, \ldots, M+L-1]$ **do**
    $\hat{\boldsymbol{h}}_k, \hat{\boldsymbol{C}}_h \leftarrow 3.34, 3.33$ {Estimate $q(\boldsymbol{h}_k)$}
    $\hat{\phi}_k \leftarrow 3.37$ {Estimate $q(\phi_k)$}
  end for
  $\hat{\boldsymbol{w}}, \hat{\boldsymbol{C}}_w \leftarrow 3.22, 3.23$ {Estimate $q(\boldsymbol{w})$}
  for $j \in [0, \ldots, M+L-1]$ **do**
    $\hat{a}_j, \hat{b}_j \leftarrow 3.16, 3.17$ {Estimate $q(\alpha_j)$}
  end for
until Convergence

An important parameter of algorithm **VB-ESN-C** is the noise splitting constant β. For the classical SAGE algorithm it was shown in [35] that fast convergence can be achieved, by assuming that all the learning perturbation comes from the latent variable of interest. This

assumption corresponds to the choice $\beta = 1$ and is adopted for the following simulations.

4.2 Optimality - Free Energy Minimization

In this section, it will be empirically shown that indeed each of the algorithms monotonically decreases the variational free energy over the number of update iterations. Therefore, the variational free energy will be analytically determined up to a constant term which does not depend on the variational parameters. For compactness, the derivations will be skipped as the calculation is long but straight forward using the relation

$$\hat{\mathcal{F}}\left[q(\boldsymbol{z})\right] = \mathbb{E}_{\boldsymbol{z}}\left[q(\boldsymbol{z})\right] - \mathbb{E}_{\boldsymbol{z}}\left[\ln p(\boldsymbol{x}, \boldsymbol{z})\right] + const. \tag{4.1}$$

The ESNs will be trained such as to predict the chaotic *Mackey-Glass* attractor [9] with the delay parameter $\tau_{MG} = 30$. As the only goal of this simulation is to illustrate the optimization progress, a detailed description of the system and also the algorithm configuration is skipped, as the parameters have been tuned to obtain illustrative learning curves instead of optimal performance. Figures 4.1, 4.2 and 4.3 show the variational free energy and the mean squared prediction error over 600 update iterations for algorithms **VB-ESN-A**, **VB-ESN-B** and **VB-ESN-C**.

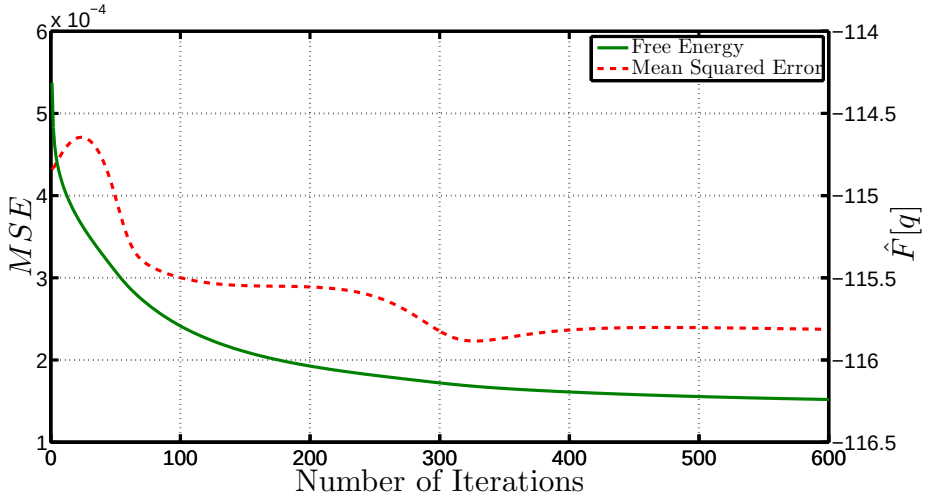


Figure 4.1: Free energy $\hat{\mathcal{F}}\left[q\right]$ and mean squared error over 600 update iterations for **VB-ESN-A**.

From each of the plots, one can clearly see that the variational free energy in fact decreases after each update cycle. In contrast, the mean squared error can also increase during optimization, which is a natural consequence of automatic regularization.

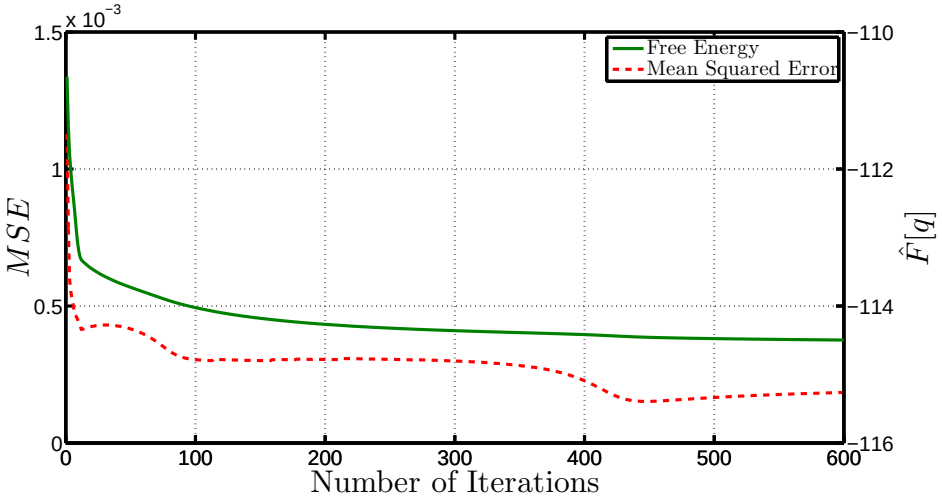


Figure 4.2: Free energy $\hat{\mathcal{F}}[q]$ and mean squared error over 600 update iterations for **VB-ESN-B**.

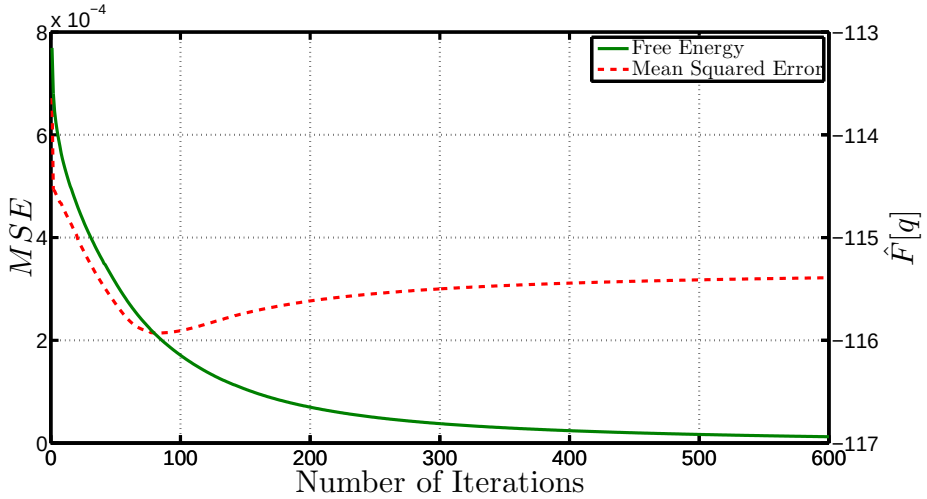


Figure 4.3: Free energy $\hat{\mathcal{F}}[q]$ and mean squared error over 600 update iterations for **VB-ESN-C**.

A detailed performance analysis of ESN-based prediction of the Macky-Glass system using algorithms very similar to **VB-ESN-C** is provided in [34]. The VB algorithms are compared against state-of-the-art ESN learning techniques and it is shown that they can significantly boost ESN's learning performance. Additionally, it is demonstrated that the learned delay parameters strongly correlate with real quantities from the Mackey-Glass equation (i.e. the delay parameter τ_{MG}). Furthermore, the effects of automatic regularization are demonstrated. It was shown that output weights, corresponding to base functions with a delay parameter close to τ_{MG} or zero are typically barely regularized during training. All other base

functions are often suppressed, which is expressed by very large hyperparameters (and as a consequence, weights close to zero). Further details and additional simulation results can be found in [34].

4.3 Handwriting Recognition

In this section, the proposed VB algorithms from sections 3.3 are evaluated using real world handwriting data. First of all, it should be figured out, if reservoir based learning principles - in particular echo state networks - can be used for classification of on-line handwriting data, which has not been reported so far. In the second step, this section tries to find out if, and how the classification performance can be improved, using the proposed algorithms from section 3. Algorithms **Std-ESN**, **Ext-ESN** and **VB-ESN-C** will be considered during this section. The reader might wonder, why the VB learning algorithms from sections 3.3.1 and 3.3.2 are not chosen to be subject of the following simulations. Remembering the results from sections 3.3.2 and 3.3.3, one could state that algorithm **VB-ESN-C** supersedes algorithm 3.3.1 as well as the approach from section 3.3.2, as it allows estimating extended ESN parameters, without neglecting dependencies between them. Additionally, the computational complexity of algorithm **VB-ESN-C** is much smaller than in case of algorithm **VB-ESN-B**. Thus, algorithms **VB-ESN-A** and **VB-ESN-B** can be considered as intermediate stages in formulating a powerful VB learning algorithm (i.e., **VB-ESN-C**), which satisfies all desired requirements.

The simulations provided in this section concentrate on the ESN-based recognition of handwritten letters, given as pen-movement trajectories. Furthermore, only letters, which can be drawn within a single stroke (i.e., continuous trajectory) are considered for simplification reasons. As discussed in section 1, classifiers can directly act on the trajectories themselves or can use a sequence of extracted features for recognition. In the following simulations, the ESNs will act on the raw trajectories (except some simple preprocessing).

4.3.1 Handwriting Data

The simulations will be performed using two different databases, which are freely available. Both of them are part of the *UCI Machine Learning Repository* which can be found at [29]. In the following, the two data sets are described and compared against each other.

The *WILLIAMS* Database:

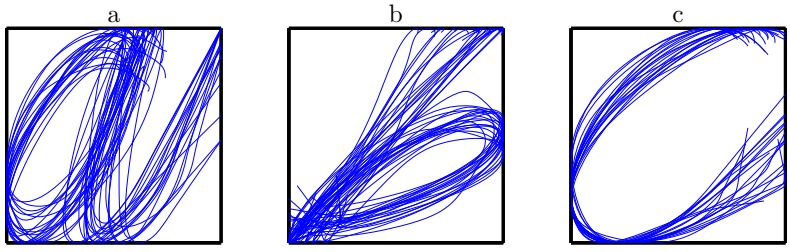
The data set consists of 2585 character instances, represented by their pen trajectories. The database only contains letters, which can be written as a single stroke. Furthermore, the data was captured only from a single writer, which drastically simplifies the recognition task, as the data shows quite small interclass variations. Each trajectory is represented by a three dimensional time series, covering velocities along the x- and y-

position and the pen force. The velocities were calculated by differentiating the original data. Additionally, a Gauss filter was applied to smoothen the data. For further details, the reader should refer to [39].

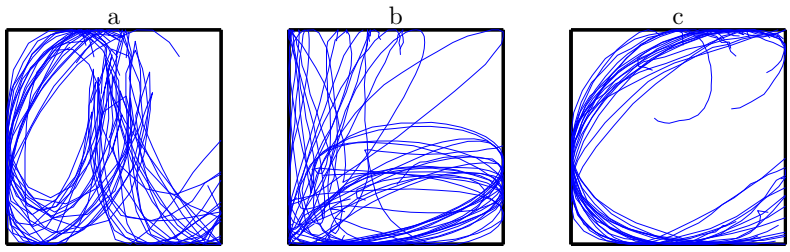
The *UJI* Database:

All together, the database provides 11640 samples consisting of 66 letters and several other symbols, which are not in the focus of this work. The characters were recorded twice for each of the 60 writers. Clearly, the interclass variations will be much higher than for the WILLIAMS database because many different writers have contributed. As a consequence, recognition will be much more challenging as for the WILLIAMS database. Another difference is that trajectories of the *UJI* data set are described with only two features: the absolute x- and y- positions. In [40], the authors claim that duplicate entries might occur in the original data and thus a suitable preprocessing scheme was applied.

To visualize the differences between the *UJI* and the *WILLIAMS* database, 20 instances of three different letters ('a', 'b', 'c') are drawn in figures 4.4a and 4.4b for each database[2], whereas the x- and y-coordinates of each trajectory have been normalized to a common range of $[-1, +1]$.



(a) WILLIAMS Database



(b) UJI Database

Figure 4.4: Handwriting data comparison: 20 samples of letters 'a', 'b' and 'c' are plotted.

[2]The third dimension of the WILLIAMS trajectories was neglected for figure 4.4.

Considering a classification scenario, the UJI database is much more demanding in terms of very diverse and writer-specific realizations of the characters. Clearly, automatic recognition becomes very complex due to strong fluctuations concerning writing speed, size and also the shape of the input letters.

4.3.2 Multi-Class ESNs

This section provides simulation details for handwriting recognition algorithms based on multi-class ESNs. The central problem, which arises when working with this type of classifier is the choice of the temporal reference patterns, indicating the values *true* or *false* for a certain output (i.e., class label). Clearly, it would be beneficial, if the classifier could also be used in an on-line fashion, which means that it would be able to detect and simultaneously classify [31] the symbols while the test person is still writing. From this viewpoint, a reference signal, which indicates the class, as well as the moment of detection would be an intuitive choice. In the following simulations, narrow Gauss pulses were used for the positive patterns. The position parameter of the Gaussian was chosen as $\mu_p = 0.7N_T$, whereas N_T denotes the length of a single trajectory. The corresponding width parameter was set to $\sigma_p^2 = 0.5$. In other words, the output corresponding to the correct class label is trained such as to emit a pulse at 70% of the trajectory length[3]. An illustration of this behavior is demonstrated in figure 4.5, where an ESN with three outputs (corresponding to classes 'a', 'b' and 'c') was trained according to algorithm **VB-ESN-C**. The plot shows the ESN class responses and the ideal reference signals for several test examples from the *WILLIAMS* database.

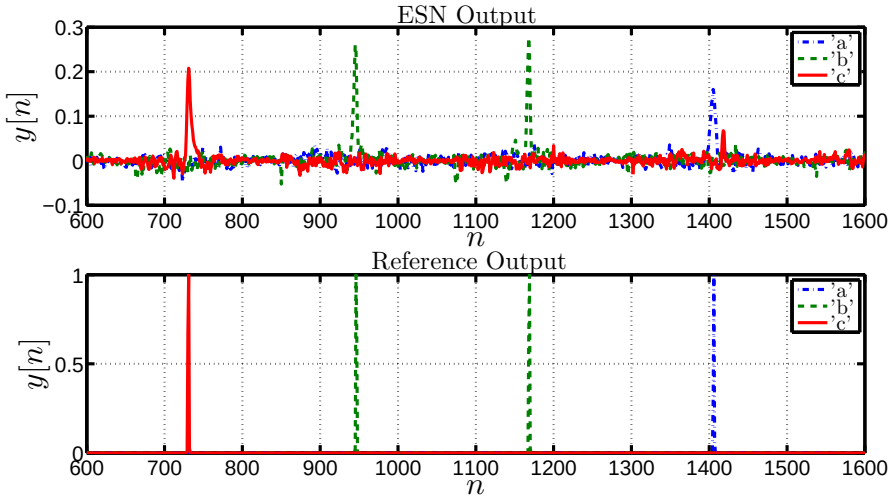


Figure 4.5: Comparison of the ESN output and the reference output for several test examples.

Even if the emitted ESN pulses from figure 4.5 have much smaller amplitudes than the ref-

[3]The value of 70% was found empirically.

erence pulses, the ESN can correctly detect letters 'a', 'b' and 'c' for the given examples.

Another important aspect of the handwriting classifier is the way how it decides for the class given the resulting pulse signals. Considering a real-time scenario, it would make sense to decide for a class, if the corresponding output signal exceeds a certain level, acting as a simple pulse detection. Using a decision criterion of this type, the classifier would suggest a class label, every time a pulse was seen on one of the outputs. Clearly, this will always lead to a certain amount of *false alarms* [31], due to the systematic ambiguity of handwritten character trajectories. For instance, identifying the trajectory of letter 'c', one cannot decide, if the writer will proceed to letter 'a'. In case he does, the last part of the trajectory could again lead to a vote for letter 'c' and so on. A typical false alarm example is depicted in figure 4.6. Some time steps after the correct class (i.e., 'a') was indicated, also a pulse for letter 'c' was generated by the ESN.

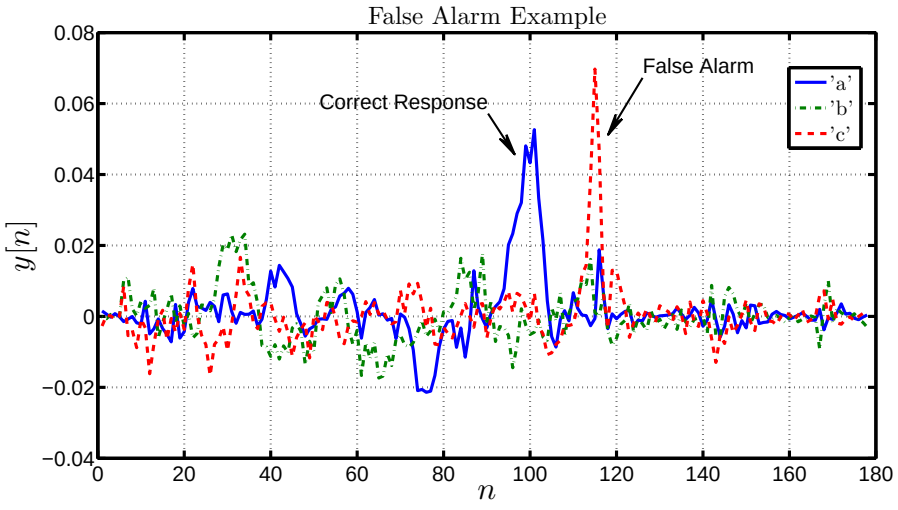


Figure 4.6: Demonstration of the *false alarm* problem.

For the following simulations, it will be necessary to define an appropriate quality measure for the ESN-based classification algorithms. Regarding the ambiguity problem of real-world handwriting data, the question arises, if and how the false alarms should enter the quality measure. Remembering that on-line recognition is subject to causality constraints, the ESN is in fact expected to vote for letter 'c' after observing the first part of the trajectory, also if the writer intends to write letter 'a', for instance. Thus, the corresponding false alarm should not be counted as an error in this case, as even humans would make the same decisions. In contrast, the second false alarm, which might occur after the overall trajectory was seen, should indeed contribute to the error, as in theory, a distinction based on the input history would be possible. In general, the problem of separating systematic from erroneous false alarms strongly depends on the context, in which the trajectory was written. For this reason, this work uses a simple off-line evaluation of the resulting pulse patterns, such that

each character trajectory yields exactly one classification result. In this case, the class label is selected according to the output which shows the maximum value, no matter, when this value has occurred.

For the simulations, ESNs with $N = 200$ neurons were used. A reservoir connectivity of 10% was used and the reservoir matrices were rescaled such that $\rho(\boldsymbol{C}_X) = 0.8$. The input weights $\boldsymbol{C}_U$ were randomly chosen between -1 and $+1$. Output feedback was not necessary for this task and thus, $\boldsymbol{C}_Y$ was set to zero. For algorithms **Ext-ESN** and **VB-ESN-C**, the time delay vectors $\boldsymbol{\tau}_l, \forall l \in [0, \ldots, L-1]$ were initialized randomly, whereas 30% of the vector entries were set to zero and the resulting 70% were uniformly drawn from the interval $[0, \ldots, 100]$. Furthermore, **VB-ESN-C** was trained using 6 iterations and the standard deviation of the white observation noise was assumed to be $\sigma = 10e{-}9$, which resulted in a weakly regularized solution of $\boldsymbol{w}$.
For a compact evaluation of the algorithms, the classification is performed on only five letters (i.e., 'a', 'b', 'c', 'd' and 'e'), instead of the overall set of character, available in the databases. The ESNs were trained using 60% of the character samples and the resulting 40% were used for testing. For each of the algorithms, 50 independent runs have been used. Classification errors were calculated for the entire set of test examples (i.e., E_{Total}) as well as for each class separately (i.e., the class error rates E_a, E_b,...). Furthermore, the error rate E_{Avrg} was calculated as the average over the class error rates. Obviously, E_{Avrg} is a more meaningful quality measure than the total error rate, as it is insensitive against the number of test examples used for the different classes. The results obtained for the WILLIAMS database are given in table 4.1.

Algorithm	E_a	E_b	E_c	E_d	E_e	E_{Total}	E_{Avrg}
	%	%	%	%	%	%	%
Std-ESN	41.85	32.65	21.75	10.85	14.58	24.29	24.34
Ext-ESN	0.00	0.00	26.49	0.19	0.22	3.70	5.38
VB-ESN-C	0.00	0.00	3.14	0.00	0.04	0.43	0.64

Table 4.1: Classification results for the WILLIAMS data set.

Obviously, multi-class ESNs without delay & sum readouts (i.e., algorithm **Std-ESN**) can not properly handle the complexity of dynamic handwriting data, which is indicated by large classification errors, even for the WILLIAMS database. The observed behavior could be explained by the fact that the standard ESNs memory capacity is obviously not large enough for the handwriting recognition problem. Remembering the false alarm discussion from above, ambiguous pulses can be avoided by memorizing previous parts of the trajectory. For instance, an erroneous 'c'-pulse in case of writing letter 'a' can only be avoided if the ESN is able to remember longer traces of the 'a'-trajectory, such that the 'c'-pattern is correctly considered

as a part of letter 'a'. To boost the ESN's memory capacity, the reservoir size could be increased, for instance[4]. Alternatively, delay & sum readouts provide a much simpler and more effective way of sufficiently extending the ESN's memory. The results from table 4.1 show that the performance can be drastically improved by introducing random time delays in the readout stage (i.e., algorithm **Ext-ESN**). Furthermore, it can be seen that letter 'c' is miss-classified most often. Again, the reason for this might be that the trajectory of 'c' often occurs as a part of other trajectories. Note that for algorithm **Ext-ESN**, the computational effort increases only marginal, compared to algorithm **Std-ESN**. For this simulation, algorithm **VB-ESN-C** is found as the clear winner, as it can further decreases the average classification error from 5.38% to 0.64%. One can see from table 4.1 that also for letter 'c', most of the test examples can be correctly classified (96.86% instead of 72.51%). It can be concluded that a significant part of the handwriting ambiguities can be resolved by an adaptation of the time delays, combined with an automatic regularization.

As a second test, the simulations have been repeated for the UJI database. The corresponding results are provided in table 4.2

Algorithm	E_a	E_b	E_c	E_d	E_e	E_{Total}	E_{Avrg}
	%	%	%	%	%	%	%
Std-ESN	22.79	13.21	92.71	88.51	93.87	43.51	46.70
Ext-ESN	25.31	11.27	11.08	72.28	93.51	39.57	42.69
VB-ESN-C	11.30	15.41	4.37	55.31	88.51	31.66	34.98

Table 4.2: Classification results for the UJI data set.

As expected, the obtained classification results get much worse for the UJI data. Even if the ranking of the three algorithms (i.e., **Std-ESN**, **Ext-ESN** and **VB-ESN-C**) remained the same as for the WILLIAMS database, a practical application seems quite unrealistic. Three main problems can be identified using the UJI database. First of all, the set of letters corresponding to a certain class is quite diverse, as instances from many different writers (i.e., 60) have been recorded. The second problem is that the database only provides two examples per writer for a single letter, which is obviously not enough for multi-class ESNs. Probably, the results could be improved by increasing the number of training examples per writer. Additionally, the UJI database does not provide a pen force feature, which seems to be important for a correct character classification. To realize ESN-based classification of dynamic handwriting data also for more complex data sets (e.g., the UJI database), an alternative approach will be investigated during the next section.

[4]This was found during the experiments.

4.3.3 Classification by Prediction

An alternative approach for classification of handwriting trajectories is based on time series prediction, as discussed in section 3.2.5. For each class, one specialized ESN is trained such as to predict the handwriting trajectories as well as possible. Thus the target output is set to $\boldsymbol{y}[n] = \boldsymbol{u}[n + n_0]$ during training. Test examples are simply classified by assigning the class label corresponding to the predictor, which has achieved the smallest mean squared prediction error. The reader should note that this method is not very well suited for performing on-line recognition, as the resulting error signal will fluctuate in general and thus it is hard to identify clear class votes at one certain time step. Of course one could try to find advanced methods for on-line detection of the characters given the prediction error over time. However this is not in the scope of this thesis.

To wash out initial conditions, the input trajectory was repeated R times during prediction. In this case $R = 4$ repetitions have been used. Furthermore, the prediction horizon n_0 was set to 11, which seemed to work best in this case. The ESN setup was almost identical to section 4.3.2 except that the input weights were uniformly drawn between -0.2 and $+0.2$, which turned out to work well for this scenario. For algorithm **VB-ESN-C**, the input variance was assumed to be $\sigma^2 = 10e - 10$. Detailed simulation results are provided only for the UJI database, as even the simple algorithm **Std-ESN** has achieved 0% error rates for the WILLIAMS database and thus there is nothing to improve using the extended learning algorithms. The results obtained for UJI database are summarized in table 4.3.

Algorithm	E_a	E_b	E_c	E_d	E_e	E_{Total}	E_{Avrg}
	%	%	%	%	%	%	%
Std-ESN	2.03	5.80	10.09	4.97	2.51	5.10	5.08
Ext-ESN	5.20	6.79	11.21	5.75	3.48	6.54	6.49
VB-ESN-C	4.53	7.62	11.82	6.02	3.92	6.82	6.78

Table 4.3: Classification results for the UJI data set.

From table 4.3 one can see that the classification of handwriting data based on prediction is in fact able to achieve good results. Obviously, classification becomes much more robust against variations of the input data using ESN-based predictors, each trained for one particular class. This seems rather intuitive, as the specialized ESNs will probably yield the best predictions for letters corresponding to the class for which they were trained, even if the test examples strongly differ from the examples used for training.
Furthermore, a very surprising behavior can be observed. Algorithms **Ext-ESN** and even **VB-ESN-C** are obviously not able to boost the classification performance. In fact, results get even worse, as demonstrated in table 4.3. The problem, when performing classification by prediction is that further decreasing the prediction error (which is done by applying **Ext-**

ESN and **VB-ESN-C**) does not necessarily end up with an improvement of the classification performance. The key criterion for successful recognition is that the predictor corresponding to the true label achieves the smallest prediction error. Furthermore, all other predictors are expected to perform worse. As a consequence, this would ensure large gaps between errors achieved by the best and the next best predictor and furthermore an optimal separability regarding the character classes. It seems that when applying algorithms **Ext-ESN** and **VB-ESN-C** very small errors are achieved also for foreign classes. Clearly, this behavior is not desired, as it systematically worsens the discrimination power of the classifier.

The simulations demonstrate that classification by prediction can handle more complex data sets, as this method only needs a few examples per writer to perform well. The main drawback of this principle is that it cannot be used for on-line handwriting recognition out-of-the-box.

Chapter 5

Conclusion

In this thesis, a variational Bayesian formulation of ESN learning was presented and evaluated using synthetic as well as real-world handwriting data.
After a brief introduction to classical Bayesian inference, the concept of variational Bayesian inference was discussed in chapter 2 and general estimation expressions under certain density factorizations have been provided. The chapter was concluded with a simple example, demonstrating the principle of variational free energy minimization (section 2.2.3).

Chapter 3 started with a compact introduction to reservoir computing and in particular ESNs, covering standard ESN learning (section 3.2.2), limiting aspects (section 3.2.4) and ESN based classification (section 3.2.5). Afterwards, in section 3.3, a variational Bayesian formulation of ESN learning was presented. For three different scenarios, the variational estimation expressions have been successively derived (see sections 3.3.1 - 3.3.3). It was shown that learning extended ESN models (e.g., ESNs with delay & sum readouts), as well as an automatic Bayesian regularization can be simultaneously realized using the concept of variational free energy minimization. Even if performing Bayesian inference is intractable for the original problem, ESNs can be efficiently trained using approximate solutions, without losing optimality. A general formulation of the extended ESN model has been used (section 3.2.4), such that the proposed algorithms can be easily adopted for a large family of ESN model extensions.

Using synthetic data (i.e. the Mackey-Glass time series), it was empirically shown that the VB-ESN learning algorithms discussed in chapter 3 are optimal as they monotonically decrease the free energy over the number of update iterations (section 4.2). In section 4.3, ESN-based recognition of real-world handwriting trajectories (i.e. the WILLIAMS and the UJI database) was evaluated using two different classification scenarios. For the WILLIAMS database it was demonstrated that only multi-class ESNs with delay & sum readouts can handle the complex ambiguities of handwriting trajectories. Furthermore, it was shown that

the performance can be drastically improved by applying the VB learning algorithm from section 3.3.3. Using the UJI database, only poor classification results have been obtained for all of the ESN algorithms under test (error rates above 30%). The reason for this might be that each character was recorded from many different writers only twice, which systematically leads to huge inter-class variations of the data. Obviously, classification based on multi-class ESNs would require larger amounts of training data to properly handle complex data sets as the UJI database.

In contrast, the second classification scenario (i.e. classification by prediction, section 4.3.3) is much more robust against large differences between the training and the test set. One could state that a specialized predictor will probably achieve the smallest prediction error, even if a certain test example - belonging to the class, for which the predictor was trained - strongly differs from the training data. As a consequence, the error rate was significantly reduced for the UJI database using simple ESNs without delay & sum readouts. It seems paradoxical that for this scenario, the classification results became worse when using the advanced learning algorithms. Obviously, the prediction error remains small, even if the input trajectory does not correspond to the class for which the particular ESN was trained, and thus, miss-classifications become more likely. The major disadvantage of classification by prediction is the fact that it can not be directly used for on-line recognition of character trajectories. Furthermore, the simulations from section 4.3.3 have indicated that extending ESNs such as to achieve better predictions does not imply better classification results.

Chapter 6

Future Work

As already mentioned in chapter 3, a detailed comparison of the algorithms from sections 3.3.2 and 3.3.3 would be of great interest. One could assume that the learning algorithm from section 3.3.3 could achieve better results, as the factorization of the approximating density better matches the true model. On the other hand, the admissible hidden data vector is not known and thus, has to be estimated too, which introduces additional uncertainty during estimation. It could even be possible to show that in theory both algorithms converge towards the same solution, which also seems plausible, as both models are exactly the same as soon as the latent variable is marginalized out. One can see that a detailed theoretical analysis would be necessary to give clear statements on the approximation quality of these algorithms.

In chapter 4, initial simulation results for ESN-based classification of dynamic handwriting data have been provided. Using two common classification scenarios, it was shown that ESNs are principally able to classify real-world handwriting trajectories. In fact, several improvements - especially to the multi-class ESN could be thought. For instance, one could try to use additional features or other reference signals than Gauss pulses might perform better as class indicators. Furthermore, the simple threshold in the readout stage of the algorithm could be replaced by a more sophisticated pulse detector. However, one can see that on-line recognition of handwriting trajectories using multi-class ESNs could be tuned on many different stages.
Using classification by prediction, the prediction horizon n_0 plays a central role for the resulting performance. It would make sense to perform a more detailed analysis regarding this parameter. For instance, the optimal value for n_0 might vary from class to class. Thus, additionally optimizing n_0 might achieve better classification results. Another important aspect regarding this scenario is how the winner class is selected, given the prediction errors. In this work, the most simple strategy was used, i.e., deciding for the class corresponding to the predictor with the smallest mean squared prediction error. More sophisticated decision strategies could maybe further improve the discrimination power. As already mentioned, us-

ing classification by prediction in an on-line fashion is naturally more complex, as it requires detecting the character classes from the history of the (fluctuating) prediction error sequence. One could, for instance, try to estimate the error statistics for certain time windows to obtain more robust classification results. However, there would be a trade-off between robustness and time resolution of the resulting decisions.

One systematic problem of simple ESNs is that they are not invariant against distinct time warping of the input trajectories. In [31], time warping invariant echo state networks have been realized by using leaky integrator neurons and the authors even point out that this approach might be used for handwriting recognition. Intuitively, this seems to be the most promising extension, to significantly improve ESN's classification performance, as it might achieve invariance against different writing speeds.

Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer, August 2006. [cited at p. 1, 2, 3, 4, 7, 8, 9, 11, 12, 14, 18, 24, 26, 31, 58]

[2] S. M. Kay, *Fundamentals of Statistical Signal Processing - Estimation Theory*, vol. 1 of *Prentice Hall Signal Processing Series*, Prentice Hall, 1993. [cited at p. 1, 3, 7, 8, 17, 18]

[3] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 3, pp. 328–339, March 1989. [cited at p. 1]

[4] J. Frankel, K. Richmond, S. King, and P. Taylor, "An Automatic Speech Recognition System using Neural Networks and Linear Dynamic Models to Recover and Model Articulatory Traces," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP00)*, 2000, pp. 254–257. [cited at p. 1]

[5] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1996. [cited at p. 1, 2, 3]

[6] C. Cox, K. Mathia, J. Edwards, and R. Akita, "Modern Adaptive Control with Neural Networks," in *International Conference on Neural Network Information Processing*, pp. 1–5. [cited at p. 2]

[7] M. Bodén, "A guide to Recurrent Neural Networks and Back Propagation," *The DALLAS project. Report from the NUTEK-supported project AIS-8, SICS. Holst: Application of data analysis with learning systems*, 2001. [cited at p. 2]

[8] K. Doya, "Bifurcations in the learning of recurrent neural networks," in *Proceedings of IEEE International Symposium on Circuits and Systems*, May 1992, vol. 6, pp. 2777–2780. [cited at p. 2, 15]

[9] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," GMD Report 148, GMD - German National Research Institute for Computer Science, 2001. [cited at p. 2, 15, 17, 18, 36]

[10] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: a new framework for neural computation based on perturbations.," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, November 2002. [cited at p. 2, 15]

[11] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, August 2009. [cited at p. 2, 3, 15, 17, 18, 20]

[12] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Advances in Neural Information Processing Systems*, S. Becker, S. Thrun, and K. Obermayer, Eds., 2003, vol. 15, pp. 593–600. [cited at p. 2]

[13] H. Paugam-Moisy, R. Martinez, and S. Bengio, "Delay learning and polychronization for reservoir computing," *Neurocomputing*, vol. 71, no. 7-9, pp. 1143–1158, March 2008. [cited at p. 2]

[14] H. Jaeger, "Short term memory in echo state networks," GMD-Report 152, GMD - German National Research Institute for Computer Science, 2002. [cited at p. 2, 17, 18, 19]

[15] G. Holzmann, "Echo State Networks with Filter Neurons and a Delay&Sum Readout with Applications in Audio Signal Processing," M.S. thesis, Graz University of Technology, 2008. [cited at p. 2, 18, 19]

[16] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm Environments Using the SAGE Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977. [cited at p. 3, 26]

[17] D. J. C. MacKay, *Information theory, inference, and learning algorithms*, Cambridge University Press, September 2003. [cited at p. 3, 7, 10, 11]

[18] M. J. Beal, *Variational Algorithms for Approximate Bayesian Inference*, Ph.D. thesis, Gatsby Computational Neuroscience Unit, University College London, 2003. [cited at p. 4, 9, 10, 11, 12, 25]

[19] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The state of the art in online handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 8, pp. 787–808, 1990. [cited at p. 4]

[20] R. Niels and L. Vuurpijl, "Using Dynamic Time Warping for Intuitive Handwriting Recognition," in *Proceedings of the 12th Conference of the International Graphonomics Society (IGS2005)*, Salerno, Italy, June 2005, pp. 217–221. [cited at p. 4]

[21] M. Bashir and J. Kempf, "Reduced Dynamic Time Warping for Handwriting Recognition Based on Multi- dimensional Time Series of a Novel Pen Device," *International Journal of Computer Science*, vol. 4, no. 3, pp. 173–179, 2009. [cited at p. 4]

[22] J. Makhoul, T. Starner, R. Schwartz, and G. Chou, "On-line cursive handwriting recognition using hidden markov models and statistical grammars," in *HLT '94: Proceedings of the workshop on Human Language Technology*, Morristown, NJ, USA, 1994, pp. 432–436, Association for Computational Linguistics. [cited at p. 4]

[23] M. Kherallah, L. Haddad, A. M. Alimi, and A. Mitiche, "On-line handwritten digit recognition based on trajectory and velocity modeling," *Pattern Recognition Letters*, vol. 29, no. 5, pp. 580–594, 2008. [cited at p. 4]

[24] I.S.I. Abuhaiba, "Offline Signature Verification Using Graph Matching," *Turk J Elec Engin*, vol. 15, no. 1, pp. 89–104, 2007. [cited at p. 4]

[25] M.N. Abdi and M. Khemakhem, "Off-Line Text-Independent Arabic Writer Identification using Contour-Based Features," *International Journal of Signal and Image Processing*, vol. 1, pp. 4–11, 2010. [cited at p. 4]

[26] A. Namboodiri and S. Gupta, "Text independent writer identification from online handwriting," in *Proceedings of 10th International Workshop on Frontiers in Handwriting Recognition*, La Baule, France, October 2006, pp. 131–147. [cited at p. 4]

[27] G. X. Tan, C. Viard-Gaudin, and A. C. Kot, "Automatic writer identification framework for online handwritten documents using character prototypes," *Pattern Recognition*, vol. 42, no. 12, pp. 3313–3323, Dec. 2009. [cited at p. 4]

[28] M.D. Skowronski and J.G. Harris, "Minimum mean squared error time series classification using an echo state network prediction model," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Kos, Greece, 2006, pp. 3153–3156. [cited at p. 5]

[29] A. Frank and A. Asuncion, "UCI machine learning repository," http://archive.ics.uci.edu/ml, 2010, [Online; accessed 8-July-2010]. [cited at p. 5, 38]

[30] H. Jaeger, "Echo state networks," http://www.scholarpedia.org/article/Echo_state_network, 2007, [Online; accessed 20-June-2010]. [cited at p. 16, 17, 60]

[31] M. Lukoševičius, D. Popovici, H. Jaeger, and U. Siewert, "Time Warping Invariant Echo State Networks," Tech. Rep. 2, Jacobs University Bremen, 2006. [cited at p. 19, 40, 41, 50]

[32] U. Siewert and W. Wustlich, "Echo-State Networks with Band-Pass Neurons: Towards Generic Time-Scale-Independent Reservoir Structures," Internal status report, PLANET Intelligent Systems GmbH, 2007. [cited at p. 19]

[33] C. Zechner and D. Shutin, "Bayesian Learning of Echo State Networks with Tunable Filters and Delay & Sum Readouts," in *Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing*, Dallas, Texas, 2010. [cited at p. 19, 21, 26, 27, 31]

[34] D. Shutin and C. Zechner, "Regularized Variational Bayesian Learning of Echo State Networks with Delay & Sum Readout," Unpublished: Work in Progress. [cited at p. 21, 26, 27, 31, 37, 38]

[35] J.A. Fessler and A.O. Hero, "Space-Alternating Generalized Expectation-Maximization Algorithm," *IEEE Transactions on Signal Processing*, vol. 42, no. 10, pp. 2664–2677, 1994. [cited at p. 26, 27, 35]

[36] B.H. Fleury, M. Tschudin, R. Heddergott, D. Dahlhaus, and K. Ingeman Pedersen, "Channel parameter estimation in mobile radio environments using the SAGE algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 3, pp. 434–450, March 1999. [cited at p. 26]

[37] D. Shutin, B. Fleury, S. Kulkarni, and V. Poor, "Variational Bayesian Space-Alternating Inference on Bayesian Networks," Unpublished: Work in Progress. [cited at p. 26, 27]

[38] "www.themathworks.com," . [cited at p. 33]

[39] B. H. Williams, "UCI character trajectories," http://archive.ics.uci.edu/ml/datasets/Character+Trajectories, 2010, [Online; accessed 27-April-2010]. [cited at p. 39]

[40] F. Prat, M. J. Castro, D. Llorens, A. Marzal, and J. M. Vilar, "UCI uji pen characters (version 2) data set," http://archive.ics.uci.edu/ml/datasets/UJI+Pen+Characters+%28Version+2%29, 2010, [Online; accessed 3-May-2010]. [cited at p. 39]

[41] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," http://matrixcookbook.com/, February 2008. [cited at p. 58]

Appendices

Appendix A

Appendix A

A.1 Graphical Notation

Figure A.1 explains the graphical notation used in this thesis.

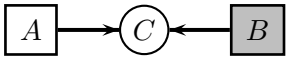


Figure A.1: Graphical notation.

Generally, network nodes denote quantities from the statistical model, whereas circles (node C) denote random quantities and squares represent deterministic parameters (nodes A and B). Furthermore, the background color gray indicates that the corresponding node is known (node B), while unknown (i.e. hidden) variables are colored white. Arcs denote dependencies between nodes.

A.2 Useful Identities

For a real-valued random vector $\boldsymbol{w}$ with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$,

$$\mathbb{E}\left[\boldsymbol{w}^T \boldsymbol{A} \boldsymbol{w}\right] = Tr\left[\boldsymbol{A}\boldsymbol{\Sigma}\right] + \boldsymbol{\mu}\boldsymbol{A}\boldsymbol{\mu}. \tag{A.1}$$

Furthermore:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} Tr\left[\boldsymbol{A}\boldsymbol{\Sigma}\right] = \boldsymbol{A}^T \tag{A.2}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \ln |\boldsymbol{\Sigma}| = \boldsymbol{\Sigma}^{-1} \tag{A.3}$$

$$\ln |\mathrm{diag}\,\{a_0, \ldots, a_{L-1}\}\,| = \sum_{l=0}^{L-1} \ln a_l \tag{A.4}$$

For further informations, the reader should consult [41] or [1].

List of Symbols and Abbreviations

Abbreviation	Description	Definition
EM	Expectation-Maximization	page 2
ESN	Echo State Network	page 2
ESP	Echo State Property	page 17
FFNN	Feed Forward Neural Network	page 2
LSM	Liquid State Machine	page 2
MAP	Maximum A Posteriori	page 2
NN	Neural Network	page 1
RNN	Recurrent Neural Network	page 2
RNN	Space Alternating Generalized EM	page 26
VB	Variational Bayes(-ian)	page 4
VBSAGE	Variational Bayesian SAGE	page 26

List of Figures

List of Tables

List of Algorithms