TECHNISCHE UNIVERSITÄT MÜNCHEN

TU MÜNCHEN

Fakultät für Informatik

LEARNING SEQUENCE REPRESENTATIONS

Justin Bayer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 22.06.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 16.09.2015 angenommen.

ABSTRACT

Recurrent Neural Networks (RNNs) are rich models for sequential data. Their capability of approximating any measurable sequence-to-sequence mapping with arbitrary accuracy and their Turing completeness make them appealing candidates for the processing of all kinds of sequential data.

This work contributes to learning representations of data with Neural Networks (NNs), and RNNs in particular, in three ways.

First, we will show how NNs can be augmented with additional calculations to allow the propagation of not only points, but random variables summarised by their expectation and variance through an NN. This generalises Fast Dropout (FD), which we show to be an outstanding regularisation method for RNNs. It further allows us to obtain approximations of the marginal likelihood and the predictive distribution of NNs, which we will use to implement Variational Bayes (VB) and related methods for the estimation of parameters.

Second, we will introduce the framework of *sequence reduction*. It consists of using RNNs in conjunction with pooling operators to reduce sequences of arbitrary length to fixed-length points, enabling further analysis.

Third, we will leverage advances in Variational Inference (VI) to learn latent state representations of sequences. These are obtained by stochastic Recurrent Networks (STORNs), where a standard RNN is augmented with stochastic units, making it able to represent arbitrarily complex distributions. The model is trained by means of Stochastic Gradient Variational Bayes (SGVB), making it probabilistic and paving the way for applications such as denoising, missing value imputation, synthesis and more.

We evaluate the proposed methods on a wide range of experiments, showing their effectiveness. In several cases, we obtain results which advance the state of the art.

ZUSAMMENFASSUNG

Rekurrente neuronale Netze sind mächtige Modelle für sequenzielle Daten. Da jede messbare Sequenz zu Sequenz Abbildung mit beliebiger Genauigkeit repräsentiert werden kann und rekurrente Netze

Turing-vollständig sind, sind sie beliebte Kandidaten für viele Arten von statistischer Sequenzverarbeitung.

Der Beitrag dieser Arbeit sind drei Methoden, neue Repräsentationen von Daten mittels neuronaler Netze, insbesondere rekurrenter neuronaler Netze, zu lernen.

Zuerst zeigen wir, wie neuronale Netze erweitert werden können damit nicht nur Punkte, sondern Zufallsvariablen (dargestellt durch ihren Erwartungswert und ihre Varianz) durch sie hindurchpropagiert werden können. Dies erweitert die „Fast Dropout"-Methode, die–wie wir zeigen werden–rekurrent Netze sehr gut regularisiert. Es ermöglicht es uns auch, die Randverteilungen und Vorhersagedichten von neuronalen Netzen anzunähern. Wir werden dies benutzen um die Gewichte von neuronalen Netzen mittels „variational Bayes" zu schätzen.

Danach stellen wir *Sequenz-Reduktion* vor, ein Rahmenwerk um Sequenzen beliebiger Länge auf Punkte mit fester Länge abzubilden. Dazu werden rekurrente Netze und „Pooling"-Operatoren eingesetzt.

Zuletzt nutzen wir Entdeckungen aus dem Gebiet der „variational inference" aus um latente Sequenzen zu lernen. Dies wird durch stochastische rekurrente Netze erreicht, welche gewöhnliche rekurrente Netze sind die mit stochastischen Einheiten erweitert werden. Dieses Modell wird mittels „Stochastic Gradient Variational Bayes" trainiert und ist von daher voll probabilistisch. Dies ermöglicht fortgeschrittene Anwendungen wie Entrauschen, Inferenz unbeobachteter Größen und Synthese.

Wir werten die vorgeschlagenen Methoden in einer Reihe von Experimenten aus um ihre Effektivität zu untersuchen. In mehreren Fällen bringen wir den „Stand der Technik" damit vorran.

Einer immer begabter als du
Du liest
Er lernt
Du lernst
Er forscht
Du forschst
Er findet:
Einer immer noch begabter.

— Robert Gernhardt, „Immer“, 2. Strophe.

PREFACE

The research that I conducted in the recent years is hardly what I imagined it to be like in the beginning of this journey. The quickly moving field of *machine learning* has seen several "micro-breakthroughs" since I started working at the lab of *Biomimetic robotics and machine learning*[1]. This often made me pursue directions which were inconceivable only a few months before. Subsequently, this work is to a certain degree eclectic: its chapters are mostly connected by the underlying method of Neural Networks (NNs).

Throughout the thesis, the parts which have been adopted from previous publications have been marked with a note on the margin like this one.

In chapter 1, I will introduce machine learning and neural networks in general. This serves to line out the basics to understand the work as well as to introduce notation and the terms used.

In chapter 2, I will present a framework to treat uncertainty in the activations of neural networks. This chapter also incorporates previous work from

- Justin Bayer, Christian Osendorfer, Sebastian Urban, et al. Training neural networks with implicit variance. In *Proceedings of the 20th International Conference on Neural Information Processing , ICONIP-2013*, 2013,
- Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. In *Proceedings of the International Conference on Learning Representations*, 2014,
- Justin Bayer, Maximilian Karl, Daniela Korhammer, and Patrick van der Smagt. Fast adaptive weight noise. *arXiv preprint arXiv:1507.05331*, 2015.

In chapter 3, I will show how RNNs can be used to reduce sequences to points. This chapter incorporates previous work from

- J. Bayer, C. Osendorfer, and P. van der Smagt. Learning sequence neighbourhood metrics. In *ICANN 2012 - 22nd International Conference on Artificial Neural Networks*, pages 531–538. Springer, 2012.

In chapter 4, I will apply VI to find latent states of sequential data. This chapter incorporates work from

- Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

1 http:\brml.org

During my work, I have co-authored several further publications which have not been incorporated into this thesis.

- Jörn Vogel, Justin Bayer, and Patrick van der Smagt. Continuous robot control using surface electromyography of atrophic muscles. In *International Conference on Intelligent Robots and Systems*, 2013
- Sebastian Urban, Justin Bayer, Christian Osendorfer, Goran Westling, Benoni B Edin, and Patrick van der Smagt. Computing grip force and torque from finger nail images using gaussian processes. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4034–4039. IEEE, 2013
- Christian Osendorfer, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. Convolutional neural networks learn compact local image descriptors. In *Neural Information Processing*, pages 624–630. Springer, 2013a
- Christian Osendorfer, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. Unsupervised feature learning for low-level local image descriptors. *arXiv preprint arXiv:1301.2840*, 2013b
- Saahil Ognawala and Justin Bayer. Regularizing recurrent networks-on injected noise and norm-based methods. *arXiv preprint arXiv:1410.5684*, 2014
- Nutan Chen, Sebastian Urban, Christian Osendorfer, Justin Bayer, and Patrick van der Smagt. Estimating finger grip force from an image of the hand using convolutional neural networks and gaussian processes. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3137–3142. IEEE, 2014
- Dominic Lakatos, Daniel Rüschen, Justin Bayer, Jörn Vogel, and Patrick van der Smagt. Identification of human limb stiffness in 5 dof and estimation via emg. In *Experimental Robotics*, pages 89–99. Springer, 2013

CONTENTS

LIST OF ACRONYMS

BPTT	Backpropagation through Time
biRNN	Bidirectional Recurrent Neural Network
DLGM	Deep Latent Gaussian Model
FAWN	Fast Adaptive Weight Noise
FAWN-ROPD	FAWN for Regularised Direct Optimisation of the Predictive Distribution
FAWN-VB	FAWN for Variational Bayes
FD	Fast Dropout
GP	Gaussian process
IVN	Implicit Variance Network
LSTM	Long Short-Term Memory
MAP	Maximum a Posteriori
MDL	Minimum Description Length
ML	Maximum Likelihood
MSE	Mean Squared Error
NCA	Neigbourhood Component Analysis
NLL	negative log-likelihood
NN	Neural Network
RIM	Regularized Information Maximization
RNN	Recurrent Neural Network
SGVB	Stochastic Gradient Variational Bayes
sRNN	simple Recurrent Neural Network
STORN	stochastic Recurrent Network
VAE	Variational Auto Encoder
VB	Variational Bayes
VI	Variational Inference
VP	Variance Propagation

1 NEURAL NETWORKS

One of the first instructions for the creation of an artificial human-like being date back to the sixteenth century, when Paracelsus (1537) gave the recipe for the creation of a homunculus. In the twentieth century, man greatly lowered his ambitions. McCulloch and Pitts (1943) proposed a mechanism to model the smallest *computational* unit of the human, a neuron. In the meantime, Hebb (1952) proposed a hypothesis on how learning in the human brain might work. One of the first practical "learning machines"[1]was then built by Rosenblatt (1958), being able to assign labels to observations–at least if this assignment is easy. That it is impossible to work for hard problems was then shown by Minsky and Papert (1969), leading to a period of reduced interest in neural networks in the western world, while it went on in the USSR (Ivakhnenko and Lapa, 1965).

Interest increased in the 80's, partially due to an article in Nature magazine (Rumelhart et al., 1986), but declined shortly after due to the apparent superiority of convex models such as support vector machines from the mid-90's to the mid-2000's. Ironically, putting the word "neural" in the title of a paper submitted to the "Neural information processing systems" conference decreased the odds of acceptance.

The neural network community has overcome these so-called "winters" of research, where the field had not been able to fulfil the expectations. Today, they are the methods of choice for a wide range of tasks.

This thesis aims to contribute to the field of neural networks. In this introduction, we will recap the most important ideas, notations and algorithms. We will do so to introduce a common notation, but also to express the author's own views on certain concepts which were developed over the years that it took to create this work.

The text assumes that the reader is familiar with basic probability and information theory, analysis, linear algebra, computer science and machine learning concepts. A good introductory text book that will help to rule out any mis- and non-understandings has been written by Murphy (2012).

1 At least the first not involving computation performed by humans. Otherwise, see the works of Legendre (1805); Gauss (1809, 1821).

1.1 INTRODUCTION

In this work, the biological motivation of neural networks will be mostly ignored. Instead, we want to highlight the perspective on neural networks as that of differentiable function approximators. It is a well known and important result that neural networks are universal in the sense that any function can be approximated to arbitrary accuracy, as shown by Kolmogorov (1963); Hecht-Nielsen (1989); Hornik et al. (1989).

The exact function a neural network implements is determined by its architecture and a set of adaptable parameters $\theta \in \Theta$. It is then applied to some input $\mathbf{x}$ to produce some output $\mathbf{y}$. More formally,

$$f : \Theta \times \mathcal{X} \to \mathcal{Y} \tag{1}$$

where the domain consists of the parameter space Θ, the input space $\mathcal{X}$ and the co-domain of the output space $\mathcal{Y}$.

The architecture (or functional form) is typically designed by an expert, while the parameters θ are algorithmically found. A discussion of the former is deferred until Sections 1.2 and 1.3. For performing the latter, the field of numerical optimisation (Wright and Nocedal (1999) provides a good textbook) offers a fitting formal framework. Given we can formulate the desired behaviour of a function as a scalar function (possibly with the help of additional targets $\mathbf{z} \in \mathcal{Z}$)

$$l : \mathcal{Y} \times \mathcal{Z} \mapsto \mathbb{R}$$

we can use the composition of f and l to arrive at a complete loss function $\mathcal{L} = l \circ f$ with

$$\mathcal{L} : \Theta \times \mathcal{X} \times \mathcal{Z} \to \mathbb{R}.$$

If both f and l are differentiable, gradient-based optimisation can be used to find good parameters θ after an application of the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial \theta}.$$

This makes neural networks a versatile tool, since we can define the desired characteristics of a function in terms of a computable quality criterion. In Section 1.4 we will discuss a few methods to obtain probabilistically inspired loss functions.

In machine learning we are mostly interested in approximating a process from which we have sampled a data set.

Two typical tasks are that of *supervised learning* and *unsupervised learning*. In supervised learning we are given a data set of the form

$$\mathcal{D}_{\text{train}} = \{({}^{i}\mathbf{x}, {}^{i}\mathbf{z})\}_{i=1}^{N}$$

where ${}^i\mathbf{x} \in \mathcal{X}$ and ${}^i\mathbf{z} \in \mathcal{Z}$. The goal is then to obtain a "machine" that is able to find those $\mathbf{z} \in \mathcal{Z}$ to which a given $\mathbf{x} \in \mathcal{X}$ relates. Often, this relation will be a function of the form $f : \mathcal{X} \to \mathcal{Z}$. Notable exceptions are formed by inverses of surjective functions; a popular example is that of inverse kinematics of redundant robots (cf. (Bishop et al., 2006, p. 272 ff.)).

In unsupervised learning, the data set in question is of the form

$$\mathcal{D} = \{{}^i\mathbf{x}\}_{i=1}^N$$

where ${}^i\mathbf{x} \in \mathcal{X}$. The tasks belonging to this category are more versatile: clustering, density estimation or feature extraction. Typical applications include pre-processing for supervised tasks, data analysis, denoising, imputation of missing values and more. In the remainder of the section, we will consider the training samples to be of the form $\mathbf{x}$, even though the discussion holds for the supervised case of $(\mathbf{x}, \mathbf{z})$.

EXAMPLE: LINEAR MODEL AND SUM OF SQUARES

A widely used loss measure for regression is that of the sum of squares or Mean Squared Error (MSE). In the case of two dimensional inputs and one dimensional outputs we have $\mathcal{X} = \mathbb{R}^2, \mathcal{Y} = \mathcal{Z} = \mathbb{R}$. Given we are using a linear model

$$\mathbf{y} = \mathbf{x}^T\mathbf{w} + b,$$

the set of parameters will consist of a single vector and scalar

$$\theta = \{\mathbf{w}, b\}.$$

The MSE can be written as

$$l(\mathbf{x}, \mathbf{z}) = ||\mathbf{z} - \mathbf{y}||_2^2,$$

where $|| \cdot ||_2$ denotes the Frobenius norm of a vector. Given a data set $\mathcal{D} = \{({}^i\mathbf{x}, {}^i\mathbf{z})\}_{i=1}^N$ the loss for the whole data set is then

$$\hat{\mathcal{L}}_{\text{linear}} = \sum_{i=1}^N ||{}^i\mathbf{x}^T\mathbf{w} + b - {}^i\mathbf{z}||_2^2.$$

All that is left is to find the derivatives of the loss with respect to the parameters, $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ and $\frac{\partial \mathcal{L}}{\partial b}$, and subsequently use an off-the-shelf optimisation algorithm. Note that this example has a pure illustratory purpose. More efficient algorithms for the least squares problem exist (Björck, 1996). We will cover the motivation of the MSE in Section 1.4.1.

1.1.1 *Generalisation: Learning is not just Optimisation*

In machine learning, we are interested in the loss on future data; the loss on the training set serves as a proxy for that.

A common simplification among the mathematically trained approaching machine learning for the first time is to reduce machine learning to optimisation. This is only true to some extent. In machine learning we are given a finite sample from the set of all possible data points. From this restricted set, hence called the "training set", we need to *generalise*: We not only need to have good performance on the training set but also on future, unseen data.

More formally, let the training samples be distributed according to the *unknown data generating distribution*, $\mathbf{x} \sim p_{\mathcal{D}}$. While optimisation is concerned with the minimisation of the training loss

$$\hat{\mathcal{L}} = \sum_{i=1}^{N} \mathcal{L}(\theta, {}^{i}\mathbf{x}),$$

learning constitutes to the minimisation of the generalisation loss

$$\tilde{\mathcal{L}} = \mathrm{E}\left[\mathcal{L}(\theta, \mathbf{x})\right]_{\mathbf{x} \sim p_{\mathcal{D}}}.$$

This shows that excellent performance (resulting from a good optimisation procedure) on the training set is not necessarily an indicator of good generalisation loss. In practice, one often runs into the problem of *overfitting*. This happens when the model becomes sensitive to phenomena only within the training set: it mistakes noise as a regularity.

A related (and not necessarily mutually exclusive) problem is that of *underfitting*, where the model is not able to capture the correlations of the data.

It is not clear when exactly during optimisation overfitting and underfitting occur, since both can only be tested by evaluation of the loss on a set of held-out samples in comparison to the training loss $\hat{\mathcal{L}}$. This test will, however, merely reflect the overall effect of over-/underfitting. It is quite possible that both occur to different degrees in input and/or parameter space. Yet, underfitting is generally dealt with by choosing a more "powerful" model f or add additional input coordinates, so called "features".

Overfitting can be limited to some extent by *regularisation*. The idea is that we can use a loss $\mathcal{J}$ instead of $\hat{\mathcal{L}}$ which is believed to be a better "proxy" for $\tilde{\mathcal{L}}$. Learning is then done by the optimisation of $\mathcal{J} = \hat{\mathcal{L}} + \mathcal{R}$. A common example is that of *weight decay*, where the Frobenius norm of the parameters is penalised. We will pay more attention to that in Section 1.4.

EXAMPLE: RIDGE REGRESSION
In ridge regression, a linear model is used where the parameters are penalised according to their squares. This is often called "weight decay" in the context of neural networks. Continuing with the previous example, the resulting regulariser is as follows:

$$\mathcal{R}_{\text{wd}}(\theta) = \lambda ||\mathbf{w}||_2^2.$$

where $\lambda \in \mathbb{R}$ constitutes a trade off between regularisation and training loss. Thus, the overall regularised loss is

$$\begin{aligned}\mathcal{J}_{\text{ridge}} &= \hat{\mathcal{L}}_{\text{linear}} + \mathcal{R}_{\text{wd}} \\ &= \sum_{i=1}^{N} ||{}^{i}\mathbf{x}^T\mathbf{w} + b - {}^{i}\mathbf{z}||_2^2 + \lambda ||\mathbf{w}||_2^2.\end{aligned}$$

This regularisation method will be justified in Section 1.4.1.

In the following, we will review feed-forward and recurrent neural networks as two of the most common neural computation architectures, and thus choices for f.

1.2 FEED FORWARD NEURAL NETWORKS

In this work we will stick to the most common architecture of feed forward neural networks at the time of writing. We are especially concerned with that of a chain of K layers, each implementing a part of the computation.

1.2.1 *Forward propagation: Producing the Output of a Neural Network*

Each of these layer functions is implemented as an affine transformation of its input and a nonlinear transfer function, which is applied element-wise:

We consider the case of feed-forward neural networks that are a stack of affine transformations followed by a non-linear transfer function applied element-wise.

$$\begin{aligned}\mathbf{a}^{(k)} &= \mathbf{y}^{(k-1)}\mathbf{W}^{(k)} + \mathbf{b}^{(k)} \\ \mathbf{y}^{(k)} &= \sigma^{(k)}(\mathbf{a}^{(k)}),\end{aligned}$$

where we set $\mathbf{y}^{(0)} = \mathbf{x}$. Furthermore, $\mathbf{W} \in \mathbb{R}^{\kappa^{(k)} \times \omega^{(k)}}$, $\mathbf{b} \in \mathbb{R}^{\omega^{(k)}}$ and consequently $\mathbf{x} \in \mathbb{R}^{\kappa^{(k)}}$. The transfer function follows $\sigma^{(k)} : \mathbb{R}^{\omega^{(k)}} \to \mathbb{R}^{\omega^{(k)}}$, but often it will just be a scalar function $\sigma^{(k)} : \mathbb{R} \to \mathbb{R}$. In that case, by writing $\sigma^{(k)}(\mathbf{a})$ we mean $[\sigma^{(k)}(a_0), \sigma^{(k)}(a_1), \ldots, \sigma^{(k)}(a_{\omega^{(k)}})]$. We call $\mathbf{a}$ the pre-synpatic and $\mathbf{y}$ the post-synaptic activations. The process is show in Figure 1.

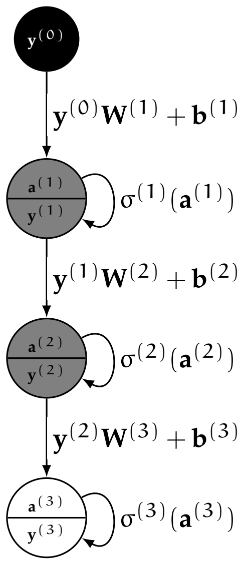

$\mathbf{y}^{(0)}$
$\mathbf{y}^{(0)}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$
$\mathbf{a}^{(1)}$
$\mathbf{y}^{(1)}$
$\sigma^{(1)}(\mathbf{a}^{(1)})$
$\mathbf{y}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$
$\mathbf{a}^{(2)}$
$\mathbf{y}^{(2)}$
$\sigma^{(2)}(\mathbf{a}^{(2)})$
$\mathbf{y}^{(2)}\mathbf{W}^{(3)} + \mathbf{b}^{(3)}$
$\mathbf{a}^{(3)}$
$\mathbf{y}^{(3)}$
$\sigma^{(3)}(\mathbf{a}^{(3)})$

Figure 1: Illustration of a feed forward network with the notation introduced in this section. The four layers of nodes are called the input layer (black), hidden layers (gray) and output layer (white). The hidden layer and output layer nodes are divided horizontally into *pre-synaptic* activations **a** and *post-synaptic* activations **y**. Between each of the layers, the affine transformation defined by **W** and **b** is applied.

We will also collect the parameters of a single layer into a common set: $\theta^{(k)} = \{\mathbf{W}^{(k)}, \mathbf{b}^{(k)}\}$. Note that $\theta = \theta^{(K)} \cup \theta^{(K-1)} \cup \ldots \cup \theta^{(1)}$. Often we will omit the layer indices to relax the notation.

Producing an output from a given neural network f is then done by performing a loop over all layers in the architecture, passing the results of the *incoming* layer into the *outgoing* one.

1.2.2 *Back-propagation: Calculating the Gradients*

The true origins of Back-propagation go back further than most research papers recognise.

The work of Rumelhart et al. (1986) is arguably the most cited article when it comes to perform gradient-based learning of the weights of neural networks. This lets numerous efforts by other researchers come short however, as techniques similar to back-propagation had been explored theoretically and practically before that. E.g. Bryson and Denham (1962); Dreyfus (1973); Kelley (1960); Dreyfus (1962) exploit steepest descent techniques in the context of the calculus of variations (Euler, 1774) with applications to trajectory optimisation. Apparently, Werbos (1981) was the first to explicitly employ back-propagation-like calculations for NNs, extending the work of his thesis (Werbos, 1974). He was apparently not aware of the previous work of Linnainmaa (1970), which included FORTRAN code. We refer the interested reader to the survey article by Schmidhuber (2015) for a rigorous investigation.

Since we are interested in the minimisation of the loss function $\mathcal{L}$, we need to find its derivatives.

Back-propagation can be seen as the combination of the chain rule and dynamic programming.

In practice, automatic differentiation with tools such as Theano (Bergstra et al., 2010) is advisable, as the manual differentiation and its implementation can be quite error prone.

As $\mathcal{L} = l \circ f$, we already concluded that $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial f}\frac{\partial f}{\partial \theta}$. Finding the first factor, $\frac{\partial \mathcal{L}}{\partial f}$ is specific to the loss function and is depending on the task at hand. We will only consider the second factor, which is the gradient of our model.

Back-propagation is a combination of dynamic programming (Bellman, 1956) and the chain rule (L'Hôpital, 1696; Leibniz, 1676). The saving of computational overhead becomes apparent when considering the complete unrolled derivative for a specific parameter set $\theta^{(k)}$:

$$\begin{aligned}\frac{\partial f}{\partial \theta^{(k)}} &= \frac{\partial f}{\partial \mathbf{y}^{(K)}}\frac{\partial \mathbf{y}^{(K)}}{\partial \mathbf{y}^{(k)}}\frac{\partial \mathbf{y}^{(k)}}{\partial \theta^{(k)}}\\ &= \frac{\partial f}{\partial \mathbf{y}^{(K)}}\frac{\partial \mathbf{y}^{(K)}}{\partial \mathbf{y}^{(K-1)}}\frac{\partial \mathbf{y}^{(K-1)}}{\partial \mathbf{y}^{(K-2)}}\cdots\frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{y}^{(k)}}\frac{\partial \mathbf{y}^{(k)}}{\partial \theta^{(k)}}\\ &= \frac{\partial f}{\partial \mathbf{y}^{(K)}}\prod_{j=k+1}^{K}\frac{\partial \mathbf{y}^{(j)}}{\partial \mathbf{y}^{(j-1)}}\frac{\partial \mathbf{y}^{(k)}}{\partial \theta^{(k)}}\end{aligned}$$

We can see that to compute the gradients with respect to $\theta^{(k)}$, we can reuse a lot of computation from the gradients of $\theta^{(j)}, j > k$. That is because

$$\frac{\partial \mathbf{y}^{(K)}}{\partial \mathbf{y}^{(k)}} = \frac{\partial \mathbf{y}^{(K)}}{\partial \mathbf{y}^{(k+1)}}\frac{\partial \mathbf{y}^{(k+1)}}{\partial \mathbf{y}^{(k)}},$$

which lets us incrementally compute the gradients by moving backwards through the network–hence the term "back-propagation".

Another opportunity for saving computations comes from reusing calculation from the forward pass. Let us inspect the derivative of the output of some layer $\mathbf{y}^{(k)}$ with respect to its weight matrix $\mathbf{W}^{(k)}$. We omit the super indices here.

$$\begin{aligned}\frac{\partial \mathbf{y}}{\partial \mathbf{W}} &= \frac{\partial \mathbf{y}}{\partial \mathbf{a}}\frac{\partial \mathbf{a}}{\partial \mathbf{W}}\\ &= \frac{\partial \sigma(\mathbf{a})}{\partial \mathbf{a}}\frac{\partial \mathbf{a}}{\partial \mathbf{W}}\\ &= \frac{\partial \sigma(\mathbf{xW}+\mathbf{b})}{\partial \mathbf{xW}+\mathbf{b}}\frac{\partial \mathbf{xW}+\mathbf{b}}{\partial \mathbf{W}}\\ &= \sigma'(\mathbf{xW}+\mathbf{b})\mathbf{x}^T.\end{aligned}$$

Calculating the derivative of a weight vector just requires the dot product of a local error signal with the input to the layer. Often, this local error signal is depicted as

$$\delta^{(k)} := \frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(K)}} \frac{\partial \mathbf{y}^{(K)}}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{a}^{(k)}}.$$

It follows that $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(k)}} = \delta^{(k)}(\mathbf{x}^{(k)})^T$.

We will omit a discussion for the bias parameter $\mathbf{b}^{(k)}$; a straightforward derivation can be performed by absorbing it into the weight matrix $\mathbf{W}$ and extending the input $\mathbf{x}$ with a corresponding column always set to 1.

1.3 RECURRENT NEURAL NETWORKS

RNNs give the ability of universal computation to neural networks, in contrast to that of universal approximation for feed-forward neural networks.

In the context of recurrent networks, we will encounter data sets where the individual samples are not part of a vector space, but instead from the set of non-zero length sequences over that space, denoted by $(\mathbb{R}^K)^+$. A single sample then consists of a sequence $\mathbf{x}_{1:T}$ of length T. To denote a single time step t, we write x_t. A single component at a single time step is denoted as $x_{t,d}$ with $d \in \mathbb{R}^K$.

1.3.1 *Forward Propagation*

A Recurrent Neural Network (RNN) can be perceived as a temporal extension of a standard neural network. The hidden layers can then "forward" information in time, which is typically done by letting the pre-synaptic activation at time step t for layer k depend on the layer at the same level at the previous time step, and the previous layer at the same time step:

$$\mathbf{a}_t^{(k)} = f^{(k)}(\mathbf{y}_{t-1}^{(k)}, \mathbf{y}_t^{(k-1)}),$$

while the calculation of the post-synaptic is similar to that of feed-forward networks:

$$\mathbf{y}_t^{(k)} = \sigma^{(k)}(\mathbf{a}_t^{(k)}).$$

The most common form of an RNN is that of a so called simple Recurrent Neural Network (sRNN) (Elman, 1990), where

$$\mathbf{a}_t^{(k)} = \mathbf{y}_t^{(k-1)}\mathbf{W}_{\text{in}}^{(k)} + \mathbf{y}_{t-1}^{(k)}\mathbf{W}_{\text{rec}}^{(k)} + \mathbf{b}^{(k)}, \tag{2}$$

with the last layer not using recurrence:

$$\mathbf{a}_t^{(K)} = \mathbf{y}_t^{(K-1)}\mathbf{W}_{\text{in}}^{(K)}\mathbf{b}^{(K)}.$$

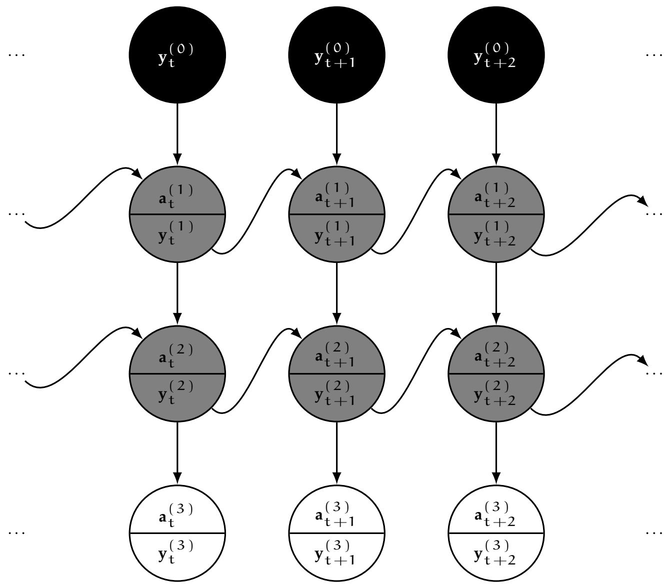


Figure 2: Illustration of an RNN. Compare Figure 1. We have two hidden layers, which have been unfolded in time from left to right. We do not show the operators along the arrows to keep the diagram uncluttered.

The set of parameters for a specific layer is then $\theta^{(k)} = \{\mathbf{W}_{\text{rec}}^{(k)}, \mathbf{W}_{\text{in}}^{(k)}, \mathbf{b}^{(k)}\}$.

The calculations are illustrated in Figure 2.

It should be noted that even this very simple form is universal. It has been shown that a Turing machine can be implemented (Siegelmann and Sontag, 1991) and any measurable sequence to sequence mapping can be approximated to arbitrary accuracy under mild conditions (Hammer, 2000).

1.3.2 *Back-propagation through Time*

As with feed-forward networks, the training algorithm of choice is generally gradient-based optimisation in conjunction with Backpropagation through Time (BPTT) Werbos (1990), although a wide range of alternatives is available (cf. (Pearlmutter, 1989; Williams and Zipser, 1989)). Notable and successful exceptions exist, however. The echo state approach (Jaeger and Haas, 2004) skips learning the recurrent connections of RNNs altogether, making use of parameter adaption only for the hidden to output weight matrix $\mathbf{W}_{\text{out}}$; in the case of a squared error, this can be performed extremely efficient via least-squares. Schmidhuber et al. (2007) take this one step further: the calculation of gradients is side stepped by using evolution strategies for optimisation, while relying on least-squares for $\mathbf{W}_{\text{out}}$ as well.

For gradient-based learning, we require the derivatives of the loss with respect to the individual parameters, i.e.

$$\frac{\partial \mathcal{L}}{\partial \theta^{(k)}}.$$

This is distinct from feed-forward neural networks since the same parameter will play a role at different time steps, since it is "reused" or "shared". We will revisit the concept of $\delta^{(k)}$ here, but will need to add the time index:

$$\delta_t^{(k)} := \frac{\partial \mathcal{L}}{\partial \mathbf{a}_t^{(k)}}.$$

In the case of a network with only a single hidden layer sRNN, the derivatives for the recurrent weight matrix are given by

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{\text{rec}}^{(k)}} &= \sum_{t=1}^{T} \delta_t^{(k)} \frac{\partial \mathbf{a}_t^{(k)}}{\partial \mathbf{W}_{\text{rec}}^{(k)}} \\ &= \sum_{t=1}^{T} \delta_t^{(k)} \sigma'^{(k)}(\mathbf{a}_t^{(k)})(\mathbf{y}_{t-1}^{(k)})^T,\end{aligned}$$

while the derivative for the feed forward connection is equivalent to that of the feed-forward network:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{\text{in}}^{(k)}} &= \sum_{t=1}^{T} \delta_t^{(k)} \frac{\partial \mathbf{a}_t^{(k)}}{\partial \mathbf{W}_{\text{in}}^{(k)}}, \\ &= \sum_{t=1}^{T} \delta_t^{(k)} \sigma'^{(k)}(\mathbf{a}_t^{(k)})(\mathbf{y}_t^{(k-1)})^T.\end{aligned}$$

All that remains is to find an expression for $\delta_t^{(k)}$. Similar to feed forward networks, it turns out that this quantity can be calculated efficiently via dynamic programming and the chain rule:

$$\begin{aligned}\delta_t^{(k)} &:= \frac{\partial \mathcal{L}}{\partial \mathbf{a}_t^{(k)}} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_t^{(k)}} \frac{\partial \mathbf{y}_t^{(k)}}{\partial \mathbf{a}_t^{(k)}} \\ &= \left[\frac{\partial \mathcal{L}}{\partial \mathbf{y}_t^{(k+1)}} \frac{\partial \mathbf{y}_t^{(k+1)}}{\partial \mathbf{a}_t^{(k+1)}} \frac{\partial \mathbf{a}_t^{(k+1)}}{\partial \mathbf{y}_t^{(k)}} + \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{t+1}^{(k)}} \frac{\partial \mathbf{y}_{t+1}^{(k)}}{\partial \mathbf{a}_{t+1}^{(k)}} \frac{\partial \mathbf{a}_{t+1}^{(k)}}{\partial \mathbf{y}_t^{(k)}} \right] \frac{\partial \mathbf{y}_t^{(k)}}{\partial \mathbf{a}_t^{(k)}} \\ &= \left[\delta_t^{(k+1)} \frac{\partial \mathbf{a}_t^{(k+1)}}{\partial \mathbf{y}_t^{(k)}} + \delta_{t+1}^{(k)} \frac{\partial \mathbf{a}_{t+1}^{(k)}}{\partial \mathbf{y}_t^{(k)}} \right] \frac{\partial \mathbf{y}_t^{(k)}}{\partial \mathbf{a}_t^{(k)}}\end{aligned}$$

where $\delta_{T+1}^{(k)} = 0$.

1.3.3 *The Vanishing and Exploding Gradient Problems*

The vanishing and exploding gradient problems are fundamental to the training of deep hierarchies of non-linear layers with numerical optimisation. It was first identified by Hochreiter (1991) (and later by Bengio et al. (1994) leading to a joint paper (Hochreiter et al., 2001)). Its theoretical basis was refined by Pascanu et al. (2012).

RNNs are hard to train, but thankfully the research community has identified the vanishing and exploding gradient problems as the major issues.

The key point is that when an error signal travels through many non-linear transformations, its gradient will either blow up exponentially fast or vanish exponentially fast–hence confronting the practitioner with learning procedures that inhibit unstable behaviour (as updates of the parameters are calculated from the gradients of the model) or make no progress at all on the training objective.

Here, we will review both problems in line with Pascanu et al. (2012), focusing on RNNs.

We have shown in Section 1.3.2 that the error terms δ_t of an RNN can be calculated in a recursive manner. Here we will inspect the terms from a different perspective to illustrate the problems of the vanishing and exploding gradients. Consider only the flow of error through the recurrent connections, i.e. assume that there is no error except at the last time step:

$$\begin{aligned}\delta_t &= \delta_{t+1}\frac{\partial \mathbf{a}_{t+1}}{\partial \mathbf{y}_t}\frac{\partial \mathbf{y}_t}{\partial \mathbf{a}_t}\\ &= \delta_{t+1}\frac{\partial \mathbf{a}_{t+1}}{\partial \mathbf{a}_t}\\ &= \delta_T\frac{\partial \mathbf{a}_T}{\partial \mathbf{a}_{T-1}}\frac{\partial \mathbf{a}_{T-1}}{\partial \mathbf{a}_{T-2}}\cdots\frac{\partial \mathbf{a}_{t+1}}{\partial \mathbf{a}_t}\\ &= \delta_T\prod_{k=t+1}^{T}\frac{\partial \mathbf{a}_k}{\partial \mathbf{a}_{k-1}}.\end{aligned} \tag{3}$$

At this point we already note that the gradients contain the product of $T-t$ Jacobians. The exact form of each is as follows:

$$\frac{\partial \mathbf{a}_t}{\partial \mathbf{a}_{t-1}} = \operatorname{diag}(\sigma'(\mathbf{a}_{t-1}))\mathbf{W}_{\text{rec}}^{T}.$$

Putting this back into Equation (3), we arrive at

$$\delta_t = \delta_T\prod_{k=t+1}^{T}\operatorname{diag}(\sigma'(\mathbf{a}_{k-1}))\mathbf{W}_{\text{rec}}^{T}.$$

In the special case of a linear transfer function, it can be seen that this reduces to the power of the recurrent weight matrix:

$$\begin{aligned}\delta_T\prod_{k=t+1}^{T}\operatorname{diag}(\operatorname{id}'(\mathbf{a}_{k-1}))\mathbf{W}_{\text{rec}}^{T} &= \delta_T\prod_{k=t+1}^{T}\mathbf{W}_{\text{rec}}^{T}\\ &= \delta_T{\mathbf{W}_{\text{rec}}^{T}}^{K}\end{aligned}$$

Conditions when the training of RNNs fails have been established based on the Eigenvalues of the recurrent weight matrix and the derivatives of the transfer functions used.

Pascanu et al. (2012) note that the norm of δ_t i) *might explode*, if the spectral radius of the recurrent weight matrix exceeds 1, i.e. $|\rho(\mathbf{W}_{\text{rec}})| > 1$; ii) *will vanish* if it is below 1: $|\rho(\mathbf{W}_{\text{rec}})| < 1$. Note that this is a necessary condition in the first and a sufficient condition in the latter case.

Let $\sigma'(\xi) < \psi \forall \xi$. For more general transfer functions, Pascanu et al. (2012) show that the necessary condition for the gradients to explode is that $|\rho(\mathbf{W}_{\text{rec}})| > 1/\psi$ and the sufficient condition for the gradients to vanish is that $|\rho(\mathbf{W}_{\text{rec}})| < 1/\psi$.

Quite a few heuristics have been proposed to alleviate the vanishing and exploding gradient problems for sRNNs.

These insights directly lead to a practical algorithm for the initialisation of the recurrent weight matrix (Sutskever et al., 2013). If it is assured that the spectral radius of the recurrent weight matrix is "reasonably" near $1/\psi$, the gradients will neither vanish nor explode in the beginning of training. Given an already initialised recurrent weight matrix $\tilde{\mathbf{W}}_{\text{rec}}$, we can perform an Eigendecomposition to obtain $\tilde{\mathbf{W}}_{\text{rec}} = U\Lambda U^T$, where Λ is a diagonal matrix with the Eigenvalues of $\tilde{\mathbf{W}}_{\text{rec}}$ on the diagonal. Given the spectral radius ρ (which is the absolute value of the largest Eigenvalue) we can recompose our final recurrent weight matrix as

$$\mathbf{W}_{\text{rec}} = U^T(\Lambda \frac{1}{c\psi\rho})U.$$

It is recommended by Sutskever et al. (2013) to use $c = 1.1$, as it leads to slightly richer dynamics and better learning performance.

Since this only affects the beginning of learning, Pascanu et al. (2012) advocate the re-normalisation of the gradient. Whenever the length of the gradient of the loss $||\frac{\partial \mathcal{L}}{\partial \theta}||_2$ exceeds a certain threshold τ, the gradient is projected to the length of τ.

The vanishing and exploding gradient problems manifest as "plains" and "cliffs" in the error landscape.

Pascanu et al. (2012) also explore the vanishing and exploding gradients from a geometric perspective. In a pathological example, it is possible to construct an error landscape which consists of cliffs of exploding gradients and plains of vanishing gradient. A simple recurrent model without any input of the following form is used:

$$x_t = \sigma(wx_{t-1} + b),$$

with $x_t, w, b \in \mathbb{R}$ and $h_0 = 0$. The objective is the distance from some target value at the fiftieth time step:

$$\mathcal{L} = (x_{50} - 0.7)^2.$$

The error landscape is visualised in Figure 3. It is apparent that even this simple problem leads to flat and steep regions which make learning difficult.

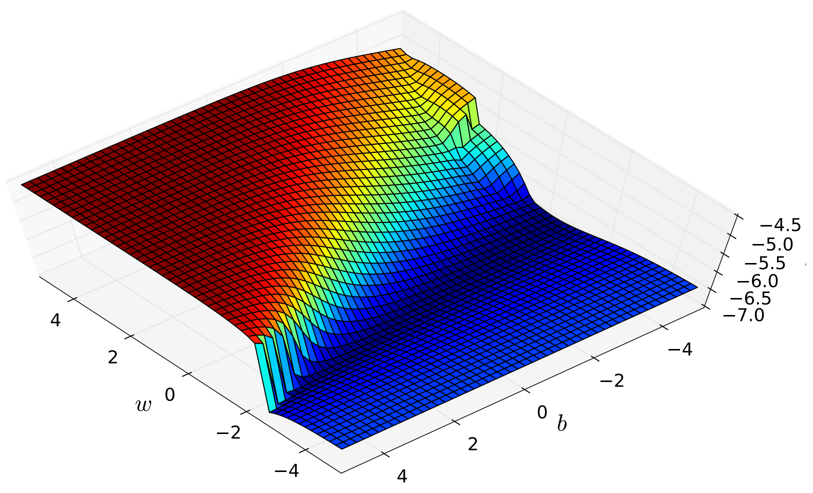


Figure 3: Illustration of vanishing gradient plains and exploding gradient cliffs in a simple recurrent model. The global minimum is contained in a ridge with a cliff of exploding gradients to one side and a smooth, concave ascent on the other. At both sides are plains of vanishing gradients. Note that taking a gradient step with a fixed step size from "within" the cliff would lead to an extreme jump in parameter space, catapulting the network into the flat region on the other side of the minimum. The loss is shown on a logarithmic scale on the z-axis.

1.3.4 *Long Short-Term Memory*

LSTM is a very effective mean of dealing with the learning of RNNs by making use of a special network topology.

The problem of the vanishing gradient rendered learning of recurrent networks largely impractical on tasks with long term dependencies. Before the discovery by Martens and Sutskever (2011) that Hessian-free optimisation is capable of dealing with these challenges to some extent as well, LSTM (Hochreiter and Schmidhuber, 1997) was the method of choice for dealing with time lags spanning hundreds or thousands of steps, for which it was designed.

LSTM nevertheless is still the method underpinning most successful applications of recurrent networks on real world problems, despite of recent successes to train "LSTM-free" RNNs on problems with long term dependencies (Sutskever et al., 2013; Martens and Sutskever, 2011). It constitutes the state of the art in speech recognition, handwriting recognition and statistical machine translation (Graves et al., 2008, 2013; Sutskever et al., 2014).

LSTM is typically referred to as a special "cell". We will employ the notion of treating LSTM as a special transfer function, which maintains a state over time [2].

2 Strictly speaking, it is not a transfer *function* anymore then.

We define $\phi(\chi, \nu) = \nu\sigma(\chi)$ with σ being the sigmoid function $(1 + e^{-\xi})^{-1}$ ranging from 0 to 1 as a *gating function*. Further, let the *states* of the cell be a sequence $(s_1, s_2, \dots, s_T), s_t \in \mathbb{R}^\gamma$.

LSTM can be perceived as a differentiable hardware register with "load", "save" and "reset" operations.

A nice metaphor for these gating function is that of a differentiable `if` statement, where c is the condition and ν is the value. If χ is true $(\chi \gg 0)$, ν will be passed on; if it is false $(\chi \ll 0)$, it will not. This resemblance has also been exploited to implement differentiable push-down automatons (Das et al., 1992), control of memory (Schmidhuber, 1992) and Turing machines more recently (Graves et al., 2014).

Given that the size of the hidden layer is γ, LSTM and the pre-synaptic activation to the layer is denoted by $\mathbf{a} \in \mathbb{R}^{4\gamma}$ with its post-synaptic $\mathbf{y} \in \mathbb{R}^\gamma$.

$$
\begin{aligned}
[\xi_t \; \iota_t \; \nu_t \; o_t] &= a_t \\
s_t &= \underbrace{\phi(\iota_t, \xi_t)}_{\text{input gate}} + \underbrace{\phi(\nu_t, s_{t-1})}_{\text{forget gate}} \\
y_t &= \sigma(\underbrace{\phi(o_t, s_t)}_{\text{output gate}})
\end{aligned}
$$

The computation is illustrated in Figure 4.

We note that all the operations are differentiable, which is why gradient-based learning can be employed. A part of the derivatives is particularly interesting: that of the states s_t. Within the cell, the gradient of a state with respect to its immediate predecessor, $\frac{\partial s_t}{\partial s_{t-1}}$, is the only part where gradients flow through time. The exact form is

$$
\begin{aligned}
\frac{\partial s_t}{\partial s_{t-1}} &= \frac{\partial \phi(\nu_t, s_{t-1})}{\partial s_{t-1}} \\
&= \frac{\partial \sigma(\nu_t) s_{t-1}}{\partial s_{t-1}} \\
&= s_{t-1} \underbrace{\frac{\partial \sigma(\nu_t)}{\partial s_{t-1}}}_{=0} + \underbrace{\frac{\partial s_{t-1}}{\partial s_{t-1}}}_{=1} \sigma(\nu_t) \\
&= \sigma(\nu_t),
\end{aligned}
$$

and thus with $t' > t$

$$
\frac{\partial s_{t'}}{\partial s_t} = \prod_{k=1}^{t'-t} \sigma(\nu_{t+k}).
$$

While this quantity can certainly approach zero, there is no intrinsic factor in it (i.e. some weight) which will drive the derivative to zero for very large time lags, i.e. $t' \gg t$. This part is also safe from exploding, since it is bounded by 1 as each of the factors is due to the activation function σ. Nevertheless, the whole gradients of the network might explode due to the other parts of the total derivative $\frac{\partial \mathcal{L}}{\partial \theta}$.

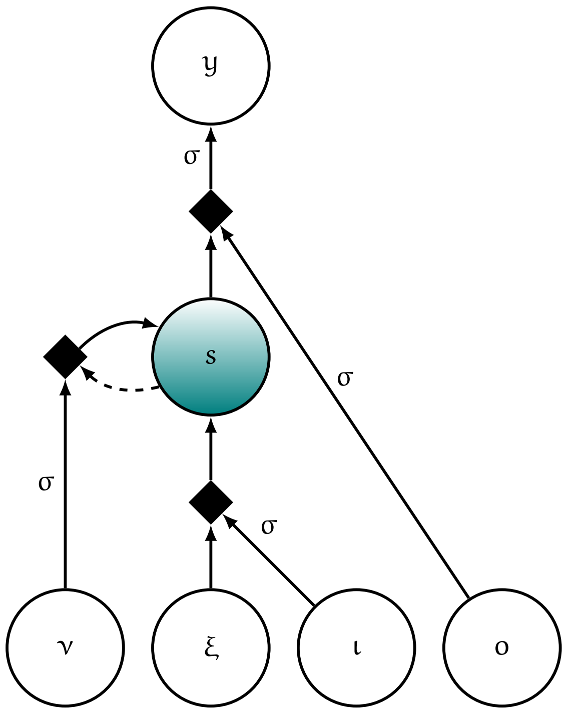


Figure 4: Illustration of the LSTM cell. The computation flows from the bottom to the top. All edges depict connections, with the dashed one being recurrent, i.e. delayed by one time step. The diamonds indicate multiplicative interactions. Edges labelled with σ indicate that a transfer function is applied during feeding that connection forward.

It should be noted that the development of LSTM was incremental. The first version did only include the input and output gate. The second version, which is the one we present here, introduced the forget gate (Gers et al., 2000). Additional "peephole" connections were added by Gers et al. (2003). Interestingly, additional variations were introduced many years later by Bahdanau et al. (2014). The insight that many different architectures work well inspired evolving alternative memory cells (Bayer et al., 2009). However, a recent study (Greff et al., 2015) shows that the version presented here is not performing significantly worse than these arguably more complex extensions.

1.3.5 *Bidirectional Recurrent Neural Networks*

RNNs can be extended to take future information into account by making use of biRNNs.

If we consider a supervised task such as transcribing an audio recording with a sequence of phonemes, the property of RNNs to only depend on the past might prove problematic: information from the future might lead to drastic improvements in performance. Schuster (1999) proposed an architectural extensions, the so-called Bidirectional Recurrent Neural Networks (biRNNs). The method has proven so effective in the past, that it is part of many successful RNN applications, such as the work by Graves et al. (2013).

The idea is to use one RNN to scan the input from the beginning to the end, and use another one to scan it from the end to the beginning. This is reflected by an additional set of parameters ${\leftarrow}\theta = \{{\leftarrow}\mathbf{W}_{\text{in}}^{(k)}, {\leftarrow}\mathbf{W}_{\text{rec}}^{(k)}, {\leftarrow}\mathbf{b}^{(k)}\}_{i=1}^{K}$, which is used to compute the reverse layer activations:

$$
\begin{aligned}
{\leftarrow}\mathbf{a}_t^{(k)} &= \dot{\mathbf{y}}_t^{(k-1)} {\leftarrow}\mathbf{W}_{\text{in}}^{(k)} + {\leftarrow}\mathbf{y}_{t+1}^{(k)} {\leftarrow}\mathbf{W}_{\text{rec}}^{(k)} + {\leftarrow}\mathbf{b}^{(k)}, \\
{\leftarrow}\mathbf{y}_t^{(k)} &= \sigma^{(k)}({\leftarrow}\mathbf{a}_t^{(k)}),
\end{aligned}
$$

which are then combined with the forward layer activations:

$$
\dot{\mathbf{y}}_t = {\leftarrow}\mathbf{y}_t + \mathbf{y}_t.
$$

Hence, the information of the two directions is joined at each layer.

1.4 LOSS FUNCTIONS

A data set can be summarised trivially by its *empirical distribution*:

There is a multitude of training methods available for neural networks.

$$p_{\text{emp}}(x) \propto \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(x = {}^{i}x),$$

where

$$\mathbb{I}(a) = 1 \text{ iff } a \text{ else } 0$$

is the *indicator function*. It puts all its probability mass exactly at the locations of the points from the data set. While this distribution is efficient to calculate simple statistics such as the mean or the variance, more complicated schemes of inference are hard. E.g. we cannot answer complex queries about the true distribution underlying the data, such as the probability of a previously unobserved event occurring.

To move from the empirical distribution to a distribution which is practical, we will often need to chose an *inductive bias*, which is an axiomatic way of picking a model or, as presented in this section, an estimator.

A well known inductive bias is that of Occam's razor: "Non sunt multiplicanda entia sine necessitate."[3], which states that the simplest plausible solution should be preferred. Of course, "simple" and "plausible" have to be well defined and translated into formulas.

In this section, we will summarise a few principled approaches to infer hidden variables via differentiable loss functions. These can be either unobserved variables or the parameters of models at hand.

These will stem from the *frequentist* as well as from the *Bayesian* approach. In the case of the former (Section 1.4.1), the randomness is part of the data and point estimates are found. In the latter case (Sections 1.4.2 and 1.4.3), the uncertainty is reflected in the variables and we will arrive at probability distributions over them.

Other inductive biases, such as maximum margin are not covered here (see e.g. the standard textbook by Vapnik (2000).)

1.4.1 *Maximum Likelihood and Maximum a Posteriori*

Given a data set $\mathcal{D}_{\text{train}}$ and a model p parameterised by θ, ML finds the parameters that make the observed data most likely:

ML and MAP are the most simple and thus most popular choices among practitioners.

$$\text{argmax}_{\theta} p(\mathcal{D}_{\text{train}}|\theta).$$

3 The author's own translation is: "Verkompliziere Dinge nicht ohne Notwendigkeit."

Often, we will look at several independent training samples, which lets the probability factorise:

$$p(\mathcal{D}_{\text{train}}|\theta) = \prod_{i=1}^{N} p({}^{i}x).$$

It is then convenient to instead minimise the negative log-likelihood of the data:

$$\mathcal{L}_{\text{ml}}(\theta) = -\sum_{i=1}^{N} \log p({}^{i}x|\theta),$$

as the product turns into a sum of independent terms. This method is termed ML.

The exact choice of the *likelihood function* p is depending on the data at hand, and usually set by design. In the continuous domain, likelihood functions are typically used to represent measurement errors. Often, the assumption of a Gaussian measurement error is valid, leading to a Gaussian likelihood function. Fat-tailed distributions such as the Laplace or Student's t are typically more robust towards outliers. If the data is strictly positive, a Gamma distribution is adequate. A skewed normal can prove sensible if the measurement errors are not symmetric. For discrete data, the Bernoulli, Categorical and Multinoulli as well as combinations thereof are typically chosen. For an overview we refer the reader to Murphy (2012).

A drawback of ML is that it is very likely to overfit: given a model of sufficient expressive capabilities, it will converge to the empirical distribution. In fact, maximum likelihood learning equates to minimising the KL-divergence between the model and the empirical distribution (Barber, 2012).

A partial solution is to resort to *maximum a posteriori*: a prior belief is formulated on the parameters in the form of a density function over the parameter space:

$$\text{argmax}_{\theta} p(\mathcal{D}_{\text{train}}|\theta)p(\theta)$$

which, after taking the log, results in an additional regularisation term:

$$\begin{aligned}\mathcal{L}_{\text{map}}(\theta) &= -\log p(\theta) - \sum_{i=1}^{N} \log p({}^{i}x|\theta) \\ &= \mathcal{L}_{\text{ml}} + \mathcal{R}_{\text{map}}.\end{aligned}$$

Note that the regularisation term is not part of the sum. Therefore, in the limit of infinite data $N \to \infty$, maximum a posteriori will converge to the maximum likelihood solution. Nevertheless, it can be shown that the choice of prior and parameterisation are intervened, rendering the method not invariant to reparameterisation. Further, it does not supply a measure of uncertainty. Also, the mode of a distribution is not necessarily a representative summary of a distribution. For further details, see the textbook by Murphy (2012).

1.4.2 *Bayesian Learning*

Bayesian methods overcome several problems of ML and MAP, at the cost of intractability and the resulting need for approximations.

A common criticism of the frequentist approach resulting in maximum likelihood and maximum a posteriori is that information about the uncertainty in the estimates of the parameters are not part of the model. The Bayesian approach thus constitutes in finding a distribution over the parameters by means of Bayes' theorem:

$$p(\theta|\mathcal{D}_{\text{train}}) = \frac{p(\mathcal{D}_{\text{train}}|\theta)p(\theta)}{p(\mathcal{D}_{\text{train}})}. \tag{4}$$

A related concept is that of the marginal likelihood:

$$p(\mathbf{x}) = \int_\theta p(\mathbf{x}|\theta)p(\theta)d\theta \tag{5}$$

This quantity is not a probability (as it is not normalised), but comes close to a loss function in Bayesian learning.

Predictions of some unknown quantity $\mathbf{z}$, are then performed by marginalising out the posterior belief in the parameters:

$$p(\mathbf{z}|\mathcal{D}_{\text{train}}, \mathbf{x}) = \int_\theta p(\mathbf{z}|\mathbf{x}, \theta)p(\theta|\mathcal{D}_{\text{train}})d\theta,$$

which is commonly referred to as the *predictive distribution*. Note that this quantity is normalised, as it is a convex combination of distributions.

Due to the multiplication of two densities in Eq. (4) and the subsequent re-normalisation, Bayesian learning can be solved exactly in closed form only in the most trivial cases. The two most popular approximation schemes are sampling based approaches on the one hand and variational Bayes on the other. While the former is asymptotically correct, it is computationally demanding as it requires sampling from a Markov chain; it will not be covered here and the interested reader is referred to the work of Neal (1993). The latter is more efficient but only provides an “incorrect” solution in form of an upper bound on the negative log-likelihood. We will pay special attention to it in the next section.

1.4.3 *Variational Inference or Minimum Description Length*

By means of VI, Bayesian learning can be approximated via optimisation.

We have seen that in Bayesian learning the posterior distribution over the hidden variables given the observations is proportional to the product of the prior and the likelihood:

$$p(\theta|\mathcal{D}_{\text{train}}) \propto \underbrace{p(\mathcal{D}_{\text{train}}|\theta)}_{\text{likelihood}}\underbrace{p(\theta)}_{\text{prior}}.$$

To obtain a proper distribution, we need to re-normalise it, which is done by dividing by the evidence $\int_\theta p(\mathcal{D}_{\text{train}}|\theta)p(\theta) = p(\mathcal{D}_{\text{train}})$.

If the prior is *conjugate* to the likelihood, calculation of the posterior is straightforward (Murphy, 2012). This is restricted to priors and likelihoods from the exponential family however and thus not applicable in the general case.

Instead, we can introduce an approximation of $q(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{x})$. It turns out that an upper bound on the negative log likelihood can be obtained (which we will show in Section 1.4.3.1):

$$\begin{aligned} -\log p(\mathbf{x}) &\leqslant -\mathbb{E}\left[\log p(\mathbf{x}|\mathbf{z})\right]_q + \mathbb{KL}[q(\mathbf{z})||p(\mathbf{z})] \\ &=: \mathcal{L}_{\text{mdl}} \end{aligned} \tag{6}$$

This loss function has several interpretations. In physics, it is typically referred to as the *variational free energy*. It is also a central component of the expectation maximisation algorithm (Dempster et al., 1977).

A perspective related to compression is that of the "bits-back" argument by Hinton and Van Camp (1993). For that, note that each probability distribution p induces an optimal code, of which the length of the data is $-\log p(x)$ measured in *nats*[4] (Grünwald, 2007).

Consider a *sender* and a *receiver*, where the former aims to send the data $\mathbf{x}$ to the latter. Imagine that the sender has obtained a distribution $q(\theta)$ over the parameters of the model to compress the data. It now sends it to the receiver by making use of $p(\theta)$ as a code. If the receiver draws the model from p, the sender will have to send a correction to make it a sample from q instead. This is given by the KL-divergence of p from q measured in nats–the second term on the RHS of Eq. 6.

After having received the correct model, the receiver will make errors if he tries to recover $\mathbf{x}$ from the model in expectation. These errors are exactly the first term on the RHS, measured in nats as well. Therefore, the variational upper bound directly minimises the number of nats needed to compress the data.

This interpretation is also known as two-part Minimum Description Length (MDL) (Grünwald, 2007).

4 The equivalent of bits, but with a base of e instead of 2.

1.4.3.1 *Derivation of the Variational Bound*

Consider the differential Kullback-Leibler divergence of a distribution p from a distribution q:

$$\mathbb{KL}[q(z)||p(z)] = \int_z q(z) \log \frac{q(z)}{p(z)} dz, \tag{7}$$

which measures the amount of excess nats needed to encode samples from q with the optimal code for p instead of q.

We will now apply Bayes theorem to the RHS of Eq. (7) yielding

$$\begin{aligned} \mathbb{KL}[q(\mathbf{z})||p(\mathbf{z})] &= \int_\mathbf{z} q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z})} d\mathbf{z} \\ &= \int_\mathbf{z} q(\mathbf{z}) \log \frac{q(\mathbf{z})p(\mathbf{x}|\mathbf{z})}{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})} d\mathbf{z} \\ &= \mathbb{KL}[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] + \int_\mathbf{z} q(\mathbf{z}) \log p(\mathbf{x}|\mathbf{z}) d\mathbf{z} \\ &\quad - \log p(\mathbf{x}) \end{aligned}$$

where we have made use of the fact that $p(\mathbf{x})$ is independent of $\mathbf{z}$ to remove the sum. Rearranging yields that the negative marginal log-likelihood can be decomposed and bounded as

$$\begin{aligned} -\log p(\mathbf{x}) &= -\mathrm{E}\left[\log p(\mathbf{x}|\mathbf{z})\right]_q + \mathbb{KL}[q(\mathbf{z})||p(\mathbf{z})] \\ &\qquad - \mathbb{KL}[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] \\ \Rightarrow -\log p(\mathbf{x}) &\leqslant -\mathrm{E}\left[\log p(\mathbf{x}|\mathbf{z})\right]_q + \mathbb{KL}[q(\mathbf{z})||p(\mathbf{z})] \\ &= \mathcal{L}_{\text{mdl}}, \end{aligned} \tag{8}$$

where we have exploited that $\mathbb{KL}[q||p] \geqslant 0 \; \forall q, p$.

The variational lower bound is often substantially easier to optimise than the marginal likelihood if the prior $p(\mathbf{z})$ and the variational approximation $q(\mathbf{z})$ are of an adequate form. More specifically, i) the KL divergence between the two and ii) the expected data likelihood (the first term of the RHS of Eq. (8)) need to be tractable. In the case of the latter, this can be approximated via Monte Carlo if we can efficiently sample from $q(\mathbf{z})$.

1.4.3.2 *Stochastic Gradient Variational Bayes*

The text in this section has appeared in parts previously in Bayer and Osendorfer (2014).

SGVB was introduced independently by Rezende et al. (2014) and Kingma and Welling (2013). Along, Deep Latent Gaussian Models (DLGMs) and Variational Auto Encoders (VAEs) where proposed– essentially deep neural networks with stochastic units trained to represent a data set in an unsupervised fashion.

The methods belong to a growing body of work where so-called *recognition* or *inference* models $q(\mathbf{x}|\mathbf{z})$ are used as a variational approximation of the posterior over the latent variables given the data (see e.g.

(Hinton et al., 1995; Mnih and Gregor, 2014; Bornschein and Bengio, 2014)):

$$\begin{aligned}
-\log p(\mathbf{x}) &= -\log \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \\
&= -\log \int_{\mathbf{z}} \frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})dz \\
&\leqslant \mathbb{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] - \mathbb{E}_{z\sim q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] \\
&=: \mathcal{L}_{\text{sgvb}}.
\end{aligned}$$

Here, we have made use of Jensen's inequality to obtain the variational upper bound.

Interestingly, we are "allowed to look at the data", as q depends on $\mathbf{x}$. However, specifying q (by encoding it with the prior) needs to cost less (as measured by the KL-divergence) than making an error during the reconstruction (as measured by the expected reconstruction loss).

We call q the *recognition model* since it allows for fast approximate inference of the latent variables $\mathbf{z}$ given the observed variables $\mathbf{x}$. In fact, the learning procedure will try to make it close to the posterior as it is a variational approximation of $p(\mathbf{z}|\mathbf{x})$, which is the inverse of the *generating model*[5] $p(\mathbf{z}|\mathbf{x})$ that cannot be found in general.

This can be seen by noting that the variational upper bound can be written as

$$\mathcal{L}_{\text{mdl}} = -\log p(\mathbf{x}) + \mathbb{KL}[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})].$$

Since the first term of the RHS is independent of q, the learning algorithm is free to make q closer to the posterior.

Both the recognition and the generating model can be chosen arbitrarily in their computational form with the possibility to represent probability distributions as outputs and stochastic training being the only requirements. In order to minimise the upper bound of the negative log-likelihood $\mathcal{L}_{\text{sgvb}}$ with numerical means, it is convenient to chose parametric models. In that case we write $p(\mathbf{x}|\mathbf{z}, \theta^g)$ and $q(\mathbf{z}|\mathbf{x}, \theta^r)$ to make the dependency on the respective parameter sets explicit. Learning good parameters can then be done by performing stochastic optimisation of $\mathcal{L}$ with respect to both θ^r and θ^g, where the expectation term is approximated by single draws from q in each training step.

Designing a model is then done by the following steps: (1) Choice of a prior $p(\mathbf{z})$ over the latent variables. (2) Choice of a recognition model $q(\mathbf{z}|\mathbf{x}, \theta^r)$. The Kullback-Leibler divergence between the prior and the recognition model has to be tractable and efficient to compute. (3) Choice of a generating model $p(\mathbf{x}|\mathbf{z}, \theta^g)$, which is often given by the type of data under investigation.

5 We use the non-standard term "generating model" for $p(\mathbf{x}|\mathbf{z})$ to distinguish it more clearly from the generative model $p(\mathbf{x})$.

VARIATIONAL AUTO ENCODERS ARE UNIVERSAL An important question is that of the representation capabilities of such a model. It turns out that if the distribution $p(x|z)$ is a universal *function* approximator, so is the overall model.

An argument for the one-dimensional case is as follows. Assume random variables x and z with respective distribution functions F_x and F_z. According to the inverse transform technique theorem (Grimmett and Stirzaker, 1992), $u = F_x(x)$ will be uniformly distributed over the interval $[0, 1]$ and so will be $u' = F_z(z)$. Equating gives $F_z(z) = F_x(x) \Rightarrow F_x^{-1}(F_z(z)) = x$. Therefore setting $p(x|z) := \delta(x = F(z))$ with $F = F_x^{-1} \circ F_z$ makes $p(x) = \int_z p(x|z)p(z)dz$.

VAEs can represent any probability distribution of which the cumulative density function and its inverse can be represented by a neural network.

An extension to the multidimensional case can be done by applying the above to the individual factors of a cascade decomposition and requiring x and z to be of the same dimensionality. The burden is then on the learning algorithm to find a good approximation for F.

2 APPROXIMATE PROPAGATION OF VARIANCE

2.1 INTRODUCTION

Standard neural networks cannot deal with uncertainties of their inputs and weights in an efficient way.

Neural networks are versatile tools to implement all kinds of point to point mappings. In Equation (1) we have presented them as functions with real valued domains and co-domains. There are many contexts in which we want to know what a neural network puts out if either the inputs or its parameters are not fixed, but uncertain: each of its possible values is associated with a likelihood. More formally, we are interested in the distribution of the outputs of a neural net given its distributions over its inputs and parameters:

$$\int_{\theta}\int_{\mathbf{x}} f(\theta, \mathbf{x}) p(\theta, \mathbf{x}) d\mathbf{x} d\theta.$$

One practical application is if we want to investigate the effect of uncertainties in the inputs on uncertainty of the outputs. Another is Bayesian learning, where we are frequently interested in the evidence or marginal likelihood of the data,

$$p(\mathcal{D}_{\text{train}}) = \int_{\theta} p(\mathcal{D}_{\text{train}}|\theta) p(\theta) d\theta.$$

In the supervised case where $\mathcal{D}_{\text{train}} = \{({}^{i}\mathbf{x}, {}^{i}\mathbf{z})\}$, the predictive distribution after obtaining the posterior $p(\theta|\mathcal{D}_{\text{train}})$ lets us infer the unknown value $\mathbf{z}$ of a given input $\mathbf{x}$:

$$p(\mathbf{z}|\mathbf{x}) = \int_{\theta} p(\mathbf{z}|\mathbf{x}, \theta) p(\theta|\mathcal{D}_{\text{train}}) d\theta. \tag{9}$$

A straightforward but expensive approach is to resort to sampling:

$$p(\mathcal{D}_{\text{train}}) \approx \frac{1}{S}\sum_{s=1}^{S} p(\mathcal{D}_{\text{train}}|{}^{(s)}\theta), \quad {}^{(s)}\theta \sim p(\theta)$$

as long as sampling from $p(\theta)$ is efficient. In practice, this approach has the downside of being computationally very demanding.

Originally used to find a fast approximation of dropout, Wang and Manning (2013) exploit the structure of neural networks to employ a Gaussian approximation of the pre-synaptic activations of hidden units justified by the central limit theorem. This enables approximate propagation of the first two moments through the network under dropout noise.

We will show to to propagate the expectation and variance of inputs and weights of a neural network forward and backward. This will be exploited in various applications.

In this chapter, we will generalise this to various other forms of stochasticity—we will cover all types that fast dropout out like calculations are able to approximate (Sections 2.3 and 2.4). We will refer to this more general technique as *Variance Propagation* (VP). Additionally, we will show that this form of deterministic noise injection makes a good regulariser for RNNs, which we will verify experimentally (Section 2.5). We will then use the insight that the propagation of variance essentially performs a marginalisation of different noise processes within the network to introduce a novel way to learn predictive distributions (Section 2.6.1), VB (Section 2.6.2.1) and present a novel non-Bayesian scheme to directly find predictive distributions (Section 2.6.2.2) with uncertain weights. We will experimentally verify the methods proposed in Sections 2.6.3 and 2.6.4. The methods place themselves favourably among the alternative methods to train neural networks with predictive distributions.

2.2 RELATED WORK

The idea to treat weights in a neural network in a stochastic way, i.e. impose a distribution on them, goes back at least to Buntine and Weigend (1991). Albeit dated, MacKay (1995) is a survey article on probabilistically motivated approaches to neural networks, containing many concepts and ideas from the literature. Employing sampling based techniques, Graves (2011) develops a practical algorithm based on VI, which has been pushed forward by Kingma et al. (2015); Blundell et al. (2015). More recently, Hernández-Lobato and Adams (2015) have developed a method to treat units in neural networks in terms of their first two moments; they apply an assumed density filtering method to find a Gaussian approximation of the true posterior. Most relevant to this section are the results from Wang and Manning (2013)—in fact, their work served as a starting point for this chapter.

The capability of using uncertain inputs for prediction has been added to Gaussian Processes (GPs) as well by Girard et al. (2003).

2.3 PROPAGATION OF VARIANCE FOR A LINEAR MODEL

The text in this section has appeared in parts previously in Bayer et al. (2015).

We will first show how variances can be propagated for a simple linear model.

Consider a linear model of the form

$$a = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} + \tilde{b}, \tag{10}$$

$$y = f(a). \tag{11}$$

Here, a is the *pre-synaptic* activation, which indicates that we have not applied the transfer function f yet to yield the *post-synaptic* activation y.

Assume that the different quantities, namely $\tilde{\mathbf{x}}, \tilde{\mathbf{w}}$ and $\tilde{\mathfrak{b}}$ are given in terms of their expectation and variance. We will now show that under further assumptions it is possible to calculate the expectation and variance of $\mathbf{y}$.

This part is based on the works of Wang and Manning (2013), where it was shown for the case of $\tilde{\mathbf{x}} = \mathbf{x} \odot \boldsymbol{\zeta}$ where $\zeta_i \sim \mathcal{B}(d)$ follows a Bernoulli distribution with rate d and $\odot$ represents element-wise multiplication. Here, $\tilde{\mathbf{x}}$ is the input to the model corrupted by "dropout" noise.

2.3.1 *Addition and Multiplication of Expectation and Variance*

First, we will revisit a few elementary facts of probability (Grimmett and Stirzaker, 1992).

Given that two random variables A and B are independent, the expectation/variance of their sum is the sum of their expectations/variances, i.e.

We establish addition and multiplication of mean/variance pairs first.

$$\mathrm{E}[A+B] = \mathrm{E}[A] + \mathrm{E}[B], \tag{12}$$
$$\mathrm{V}[A+B] = \mathrm{V}[A] + \mathrm{V}[B]. \tag{13}$$

This is similar in case of the expectation of a product

$$\mathrm{E}[AB] = \mathrm{E}[A]\,\mathrm{E}[B], \tag{14}$$

but is more involved for the variance of a product:

$$\mathrm{V}[AB] = \mathrm{E}[A]^2\,\mathrm{V}[B] + \mathrm{V}[A]\,\mathrm{E}[B]^2 + \mathrm{V}[A]\,\mathrm{V}[B]. \tag{15}$$

2.3.2 *Distribution of the Pre-Synaptic Activation*

If $\tilde{\mathbf{w}}, \tilde{\mathbf{x}}$ and $\tilde{\mathfrak{b}}$ are independent of each other, have sufficiently many components and finite mean and variance, the central limit theorem (Grimmett and Stirzaker, 1992) applies. This makes a distributed approximately according to a Gaussian, i.e. $a \sim \mathcal{N}(\mathrm{E}[a], \mathrm{V}[a])$. More specifically, consider a distribution $q(\tilde{\theta})$ over the parameters of the model with $\tilde{\theta} = \{\tilde{\mathbf{w}}, \tilde{\mathfrak{b}}\}$. Further, consider a distribution $c(\tilde{\mathbf{x}}|\mathbf{x})$ that "distorts" the input of the network. Then

Thanks to the central limit theorem, the pre-synaptic activation of a neural network with corrupted quantities is often Gaussian.

$$\begin{aligned} p(a|\mathbf{x}) &= \int_{\tilde{\theta}} \int_{\tilde{\mathbf{x}}} q(\tilde{\theta}) p(a|\tilde{\mathbf{x}}, \tilde{\theta}) c(\tilde{\mathbf{x}}|\mathbf{x}) d\tilde{\mathbf{x}} d\tilde{\theta} \\ &\approx \mathcal{N}(\mathrm{E}[a], \mathrm{V}[a]). \end{aligned}$$

If we neglect the noise process on the data, we obtain an approximation of the predictive distribution (cf. Eq. (9)):

$$\begin{aligned} p(a|\mathbf{x}) &= \int_{\tilde{\theta}} q(\tilde{\theta})p(a|\mathbf{x},\tilde{\theta})d\tilde{\theta} \\ &\approx \mathcal{N}(\mathrm{E}[a], \mathrm{V}[a]). \end{aligned} \tag{16}$$

All that is left to determine is then the expectation and variance of a. Since both are sums and/or products of quantities with known expectation and variance, the calculations are given by

$$\mathrm{E}[a] = \mathrm{E}[\tilde{\mathbf{w}}]^T \mathrm{E}[\tilde{\mathbf{x}}] + \mathrm{E}\left[\tilde{b}\right], \tag{17}$$

$$\begin{aligned} \mathrm{V}[a] &= \mathrm{V}\left[\tilde{b}\right] + \sum_i \mathrm{V}[\tilde{w}_i\tilde{x}_i] \\ &= \mathrm{V}\left[\tilde{b}\right] + \sum_i \mathrm{V}[\tilde{w}_i]\,\mathrm{E}[\tilde{x}_i]^2 + \mathrm{V}[\tilde{x}_i]\,\mathrm{E}[\tilde{w}_i]^2 + \mathrm{V}[\tilde{x}_i]\,\mathrm{V}[\tilde{w}_i] \\ &= \mathrm{V}\left[\tilde{b}\right] + \sum_i \mathrm{V}[\tilde{w}_i]\,\mathrm{E}[\tilde{x}_i]^2 + \sum_i \mathrm{V}[\tilde{x}_i]\,\mathrm{E}[\tilde{w}_i]^2 \\ &\quad + \sum_i \mathrm{V}[\tilde{x}_i]\,\mathrm{V}[\tilde{w}_i] \\ &= \mathrm{V}\left[\tilde{b}\right] + \mathrm{V}[\tilde{\mathbf{w}}]^T \mathrm{E}[\tilde{\mathbf{x}}]^2 + \mathrm{V}[\tilde{\mathbf{x}}]^T \mathrm{E}[\tilde{\mathbf{w}}]^2 + \mathrm{V}[\tilde{\mathbf{x}}]^T \mathrm{V}[\tilde{\mathbf{w}}], \end{aligned} \tag{18}$$

where we have assumed once again that all components of $\tilde{\mathbf{w}}, \tilde{\mathbf{x}}$ and $\tilde{b}$ are independent.

2.3.3 *Distribution of the Post-synaptic activation*

For several special cases, we can find the post-synaptic mean and variance efficiently.

If a is normal distributed, the expectation and variance of the post-synaptic activation $y = f(a)$ can be obtained by solving the following integrals:

$$\mathrm{E}[y] = \int f(x)\mathcal{N}(x|\mathrm{E}[a], \mathrm{V}[a])dx, \tag{19}$$

$$\mathrm{V}[y] = \int (f(x) - \mathrm{E}[y])^2\mathcal{N}(x|\mathrm{E}[a], \mathrm{V}[a])dx. \tag{20}$$

Note that this does not imply Gaussianity of y. While not in general tractable, the fact that the integral is one dimensional allows for a wide range of approximations. The most straightforward is the use of a table. Monte Carlo integration or the unscented transform (Julier and Uhlmann, 1997) are further options. For the case of the rectifier transfer function and the logistic sigmoid, a closed form and a very good approximation are available. We present them here for the sake of completeness, but refer the interested reader to the corresponding paper by Wang and Manning (2013) for a derivation.

In case of the rectifier $f(a) = \max(a, 0) = y$ we have:

$$\begin{aligned}
r =& \frac{\mathrm{E}[a]}{\sqrt{\mathrm{V}[a]}}, \\
\mathrm{E}[y] =& \Phi(r)\mathrm{E}[a] + \phi(r)\sqrt{\mathrm{V}[a]}, \\
\mathrm{V}[y] =& \mathrm{E}[a]\sqrt{\mathrm{V}[a]}\phi(r) + (\mathrm{E}[a]^2 + \mathrm{V}[a])\Phi(r) - \mathrm{E}[a]^2
\end{aligned}$$

where $\Phi(\xi)$ and $\phi(\xi)$ are the cumulative distribution function and probability density function of the standard normal, respectively.

Let the scaled sigmoid be as follows:

$$\sigma(m, v) = \sigma\Big(\frac{m}{\sqrt{1 + \pi v/8}}\Big),$$

with $\sigma(\xi) = (1 + \exp(-\xi))^{-1}$. Then we have

$$\begin{aligned}
\mathrm{E}[y] \approx& \sigma(\mathrm{E}[a], \mathrm{V}[a]), \\
\mathrm{V}[y] \approx& \sigma(a(\mathrm{E}[a] - b), a^2\mathrm{V}[a]) - \mathrm{E}[a]^2,
\end{aligned}$$

with $a = 4 - 2\sqrt{2}, b = -\log\sqrt{2} - 1$.

2.3.4 *Noise Processes*

We introduce noise processes to abstract away different kinds of uncertainty in the network.

We now consider that the quantities $\tilde{\mathbf{w}}, \tilde{\mathbf{x}}, \tilde{b}$ are corrupted versions of the true underlying quantities $\mathbf{w}, \mathbf{x}, b$. We will focus on $\tilde{\mathbf{x}}$ first, but the discussion is equivalent for $\tilde{\mathbf{w}}$ and $\tilde{b}$. We define a noise process to be a probability distribution over possible corruptions given a clean input, i.e. $c(\tilde{\mathbf{x}}|\mathbf{x})$. If we can obtain $\mathrm{E}[\tilde{\mathbf{x}}]$ and $\mathrm{V}[\tilde{\mathbf{x}}]$ given $\mathrm{E}[\mathbf{x}], \mathrm{V}[\mathbf{x}]$ and c, we can integrate c seamlessly into the calculations.

Hence we already gave the respective rules above, two obvious choices are additive and multiplicative noise. Given a vector of independent noise variables $\boldsymbol{\epsilon}$ with known expectation and covariance, let $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$, then

$$\begin{aligned}
\mathrm{E}[\tilde{\mathbf{x}}] &= \mathrm{E}[\mathbf{x}] + \mathrm{E}[\boldsymbol{\epsilon}], \\
\mathrm{V}[\tilde{\mathbf{x}}] &= \mathrm{V}[\mathbf{x}] + \mathrm{V}[\boldsymbol{\epsilon}].
\end{aligned}$$

Analogously if $\tilde{\mathbf{x}} = \mathbf{x} \odot \boldsymbol{\zeta}$,

$$\begin{aligned}
\mathrm{E}[\tilde{\mathbf{x}}] &= \mathrm{E}[\mathbf{x}] \odot \mathrm{E}[\boldsymbol{\zeta}], \\
\mathrm{V}[\tilde{\mathbf{x}}] &= \mathrm{E}[\mathbf{x}]^2 \odot \mathrm{V}[\boldsymbol{\zeta}] + \mathrm{V}[\mathbf{x}] \odot \mathrm{E}[\boldsymbol{\zeta}]^2 + \mathrm{V}[\mathbf{x}] \odot \mathrm{V}[\boldsymbol{\zeta}],
\end{aligned}$$

where $\odot$ denotes element-wise multiplication.

Depending on the exact natures of $\boldsymbol{\epsilon}$ and $\boldsymbol{\zeta}$, several noise injecting regularisers can be approximated. Wang and Manning (2013) approximate Dropout (Hinton et al., 2012). We will show later how to approximate DropConnect Wan et al. (2013) and Gaussian weight noise (Graves, 2013).

2.3.4.1 *Equivalence of Noise Models under the Approximation*

Gaussian additive and multiplicative noise on inputs, weights and bias generalises all possible noise processes when using Variance Propagation (VP).

It should be noted that two different noise processes c and c′ can give rise to the same approximations. This is the case if $\mathrm{E}[\tilde{\mathbf{x}}]_c = \mathrm{E}[\tilde{\mathbf{x}}]_{c'}\, \forall \mathbf{x}$ and $\mathrm{V}[\tilde{\mathbf{x}}]_c = \mathrm{V}[\tilde{\mathbf{x}}]_{c'}\, \forall \mathbf{x}$, i.e. the noise processes lead to the same expectation and variance.

If we restrict ourselves to additive and multiplicative noise processes on either the inputs or the weights, one model is as follows:

$$a = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} + \tilde{b},$$

where

$$\tilde{\mathbf{w}} = \boldsymbol{\zeta}_{\mathbf{w}} \odot \mathbf{w} + \boldsymbol{\epsilon}_{\mathbf{w}},$$
$$\tilde{b} = \zeta_b \odot b + \epsilon_b$$

and

$$\tilde{\mathbf{x}} = \boldsymbol{\zeta}_{\mathbf{x}} \odot \mathbf{x} + \boldsymbol{\epsilon}_{\mathbf{x}}.$$

with each $\boldsymbol{\zeta}_{\mathbf{w}}, \boldsymbol{\epsilon}_{\mathbf{w}}, \zeta_{\mathbf{x}}, \boldsymbol{\epsilon}_{\mathbf{x}}$ being distributed according to a Gaussian distribution.

This model is general under the above restrictions. Note that all noise processes that are either multiplicative or additive form an equivalence class, where two noise processes are equivalent if and only if they have the same mean and variance. Since a Gaussian can represent any mean and variance (in contrast to a Bernoulli, of which the variance is upper bounded by 0.25), Gaussian noise models are sufficient for variance propagation.

2.3.4.2 *Soundness of the Approximation*

Even if we end up with a bad approximation, the resulting model will still be useful if the generalisation loss is low.

It was verified experimentally by Wang and Manning (2013) that the central limit theorem holds for deep neural networks in certain cases. This is however not possible in general and might fail in cases where inputs are low dimensional or sparse. A completely different question is whether this is of any importance. Considering that we are only interested in a function approximator, the exact interpretations of different quantities in the network are unimportant. Loosely speaking, we do not care whether our model constitutes a good approximation of a corresponding real model, as long as the model works well enough for the task at hand, as indicated by an estimate of the generalisation error.

2.4 VARIANCE PROPAGATION FOR DEEP AND RECURRENT NETWORKS

We have described how to obtain the output expectation and variance of a linear model with an output activation function given the expectations and variances of its inputs in Section 2.3. Deep and recurrent networks can be constructed by stacking many of these on top of each other.

The text in this section has appeared in parts previously in Bayer et al. (2015).

Deep and recurrent networks using variance propagation can be implemented via stacks of affine transformations and non-linear transfer functions.

2.4.1 *Deep Multilayer Perceptrons*

Given a multilayer perceptron of the form

$$\mathbf{a}^{(k)} = \mathbf{y}^{(k-1)}\mathbf{W}^{(k)} + \mathbf{b}^{(k)}$$
$$\mathbf{y}^{(k)} = \sigma^{(k)}(\mathbf{a}),$$

with K layers, input $\mathbf{x} := \mathbf{a}^{(0)}$ and output $\mathbf{y} := \mathbf{a}^{(K)}$ we can apply the rules from the previous section to obtain a modified procedure to propagate activations through the network. We have summarised this in Alg. 1, where we have generalised Eqs. (10,11 on page 10) to that of many linear models operating side by side. This boils down to replacing the weight vector $\mathbf{w}$ with a weight matrix $\mathbf{W}$ and the scalar bias b with a bias vector $\mathbf{b}$:

$$\mathbf{a} = \mathbf{x}\mathbf{W} + \mathbf{b},$$
$$\mathbf{y} = f(\mathbf{a}).$$

The transfer function f is now applied element-wise. We want to stress the fact that propagating $\mathbf{a}$ through the transfer function by integrating over each of its components a_i separately will introduce the assumption that all elements of $\mathbf{a}$ are statistically independent, which is certainly not completely justified.

Algorithm 1 Variance Propagation Neural Network Forward Pass

```
$E[\mathbf{y}^{(0)}] \leftarrow E[\mathbf{x}]$
$V[\mathbf{y}^{(0)}] \leftarrow V[\mathbf{x}]$
for $k = 1..K$ do
    $E[\mathbf{a}^{(k)}] \leftarrow E[\tilde{\mathbf{y}}^{(k-1)}] E[\tilde{\mathbf{W}}^{(k)}] + E[\tilde{\mathbf{b}}]$ (Eq. (17))
    $V[\mathbf{a}^{(k)}] \leftarrow$
        $V[\tilde{\mathbf{b}}] + E[\tilde{\mathbf{y}}^{(k-1)}]^2 \odot V[\tilde{\mathbf{W}}^{(k)}] +$
        $V[\tilde{\mathbf{y}}^{(k-1)}] \odot E[\tilde{\mathbf{W}}^{(k)}]^2 + V[\tilde{\mathbf{y}}^{(k-1)}] \odot V[\tilde{\mathbf{W}}^{(k)}]$ (Eq. (18))
    $E[\mathbf{y}^{(k)}] \leftarrow$ using implementation of Eq. (19)
    $V[\mathbf{y}^{(k)}] \leftarrow$ using implementation of Eq. (20)
end for
```

It should be noted that all operations are differentiable and thus gradient-based optimisation can be employed. However, the equations are rather complex and we do not provide them here. Use of an automatic differentiation tool is advisable (e.g. Bergstra et al. (2010)).

2.4.2 *Recurrent Networks*

Showing how to apply variance propagation to RNNs makes another intermediary step necessary. Inspecting Eq. (2), we see that the pre-synaptic activation is the sum of two terms, one stemming from the previous time step and one stemming from the incoming layer. Let the former, which we call the layer-wise contribution, be denoted by $\hat{\mathbf{a}}_t^{(k)}$. The latter, which we call the recurrent contribution, will be written as $\vec{\mathbf{a}}_t^{(k)}$.

We can apply the rules from the Section 2.3 to obtain the expectations and variances for both $\hat{\mathbf{a}}$ and $\vec{\mathbf{a}}$. If we assume both to be independent, their expectations will sum up according to Eqs. (12, 13), i.e.

$$\mathrm{E}\left[\mathbf{a}_t^{(k)}\right] = \mathrm{E}\left[\hat{\mathbf{a}}_t^{(k)}\right] + \mathrm{E}\left[\vec{\mathbf{a}}_t^{(k)}\right],$$
$$\mathrm{V}\left[\mathbf{a}_t^{(k)}\right] = \mathrm{V}\left[\hat{\mathbf{a}}_t^{(k)}\right] + \mathrm{V}\left[\vec{\mathbf{a}}_t^{(k)}\right].$$

The resulting computations are summarised in Alg. 2.

2.4.2.1 *Long Short-Term Memory*

We introduced LSTM as a transfer function in Section 1.3.4. We did so by making use of a gating function $\phi(\chi, \nu) = \sigma(\chi)\nu$, where σ denotes the logistic sigmoid function $\sigma(\xi) = (1 + \exp(-\xi))^{-1}$. Furthermore we split the pre-synaptic activation $\mathbf{a}$ into four equally sized activations ξ, ι, ν and o. Given that these stem from a linear combination of the post-synaptic activations of the incoming layer, we can assume each of these to be Gaussian distributed with expectations and variances calculated just as for a linear model.

Implementation of LSTM is practical even though the pre-synaptic Gaussian assumption is not justified.

Subsequently, the application of approximations for σ to each of these quantities is justified. With the rules of obtaining the expectation and variance of a multiplication given the expectations and variances of the respective factors (Eq. 14, 15) we can obtain the expectation and variance after the gating function. Thus, applying variance propagation to the sequence of states $\mathbf{s}_{1:T}$ is possible.

Nevertheless, we will have to leave the safe harbour of validity for each y_t, since there is no reason to assume that $\phi(o_t, s_t)$ will be Gaussian. Consequently, any closed-form approximation of y_t will not be justified. As argued in Section 2.3.4.2, this does not imply that

Algorithm 2 Variance Propagation Recurrent Network Forward Pass

```
$\mathrm{E}\left[\mathbf{y}_{1:T}^{(0)}\right] \leftarrow \mathrm{E}\left[\mathbf{x}_{1:T}\right]$
$\mathrm{V}\left[\mathbf{y}_{1:T}^{(0)}\right] \leftarrow \mathrm{V}\left[\mathbf{x}_{1:T}\right]$
**for** $k = 1..K$ **do**
    **for** $t = 1..T$ **do**
        $\mathrm{E}\left[\hat{\mathbf{a}}_t^{(k)}\right] \leftarrow \mathrm{E}\left[\tilde{\mathbf{y}}_t^{(k-1)}\right] \mathrm{E}\left[\tilde{\mathbf{W}}^{(k)}\right] + \mathrm{E}\left[\tilde{\mathbf{b}}\right]$ (Eq. (17))
        $\mathrm{V}\left[\hat{\mathbf{a}}^{(k)}\right] \leftarrow$
          $\mathrm{V}\left[\tilde{\mathbf{b}}\right] + \mathrm{E}\left[\tilde{\mathbf{y}}_t^{(k-1)}\right]^2 \odot \mathrm{V}\left[\tilde{\mathbf{W}}^{(k)}\right] +$
          $\mathrm{V}\left[\tilde{\mathbf{y}}_t^{(k-1)}\right] \odot \mathrm{E}\left[\tilde{\mathbf{W}}^{(k)}\right]^2 + \mathrm{V}\left[\tilde{\mathbf{y}}_t^{(k-1)}\right] \odot \mathrm{V}\left[\tilde{\mathbf{W}}^{(k)}\right]$
        $\mathrm{E}\left[\vec{\mathbf{a}}_t^{(k)}\right] \leftarrow \mathrm{E}\left[\tilde{\mathbf{y}}_{t-1}^{(k)}\right] \mathrm{E}\left[\tilde{\mathbf{W}_{\mathrm{rec}}}^{(k)}\right] + \mathrm{E}\left[\tilde{\mathbf{b}}\right]$ (Eq. (17))
        $\mathrm{V}\left[\vec{\mathbf{a}}^{(k)}\right] \leftarrow$
          $\mathrm{V}\left[\tilde{\mathbf{b}}\right] + \mathrm{E}\left[\tilde{\mathbf{y}}_t^{(k-1)}\right]^2 \odot \mathrm{V}\left[\tilde{\mathbf{W}_{\mathrm{rec}}}^{(k)}\right] +$
          $\mathrm{V}\left[\tilde{\mathbf{y}}_t^{(k-1)}\right] \odot \mathrm{E}\left[\tilde{\mathbf{W}_{\mathrm{rec}}}^{(k)}\right]^2 + \mathrm{V}\left[\tilde{\mathbf{y}}_t^{(k-1)}\right] \odot \mathrm{V}\left[\tilde{\mathbf{W}_{\mathrm{rec}}}^{(k)}\right]$
        $\mathrm{E}\left[\mathbf{a}_t^{(k)}\right] \leftarrow \mathrm{E}\left[\hat{\mathbf{a}}_t^{(k)}\right] + \mathrm{E}\left[\vec{\mathbf{a}}_t^{(k)}\right]$
        $\mathrm{V}\left[\mathbf{a}_t^{(k)}\right] \leftarrow \mathrm{V}\left[\hat{\mathbf{a}}_t^{(k)}\right] + \mathrm{V}\left[\vec{\mathbf{a}}_t^{(k)}\right]$
        $\mathrm{E}\left[\mathbf{y}_t^{(k)}\right] \leftarrow$ using implementation of Eq. (19)
        $\mathrm{V}\left[\mathbf{y}_t^{(k)}\right] \leftarrow$ using implementation of Eq. (20)
    **end for**
**end for**
```

applying fast dropout to LSTM will not result in an effective regulariser; conversely, we will show experimentally that it can improve over LSTM.

2.5 FAST DROPOUT

The text in this section has appeared in parts previously in Bayer et al. (2014).

As briefly mentioned before, dropout constitutes replacing the activations of individual units of a neural network with zeros with a certain chance, the *dropout rate* d. Here, we shall review FD as introduced by Wang and Manning (2013) and subsequently analyse it derivatives. In the case of a hidden layer we write $\tilde{\mathbf{x}} = \boldsymbol{\zeta} \odot \mathbf{x}$, and $\zeta_i \sim \mathcal{B}(1-d)$. Subsequently for the expectation we get

$$\begin{aligned} \mathrm{E}[\tilde{\mathbf{x}}] &= \mathrm{E}[\boldsymbol{\zeta}] \odot \mathrm{E}[\mathbf{x}] \\ &= (1-d)\mathrm{E}[\mathbf{x}]. \end{aligned}$$

For the variance

$$\begin{aligned} \mathrm{V}[\tilde{\mathbf{x}}] &= \mathrm{V}[\boldsymbol{\zeta}] \odot \mathrm{E}[\mathbf{x}]^2 + \mathrm{V}[\mathbf{x}] \odot \mathrm{E}[\boldsymbol{\zeta}]^2 + \mathrm{V}[\mathbf{x}] \odot \mathrm{V}[\boldsymbol{\zeta}] \\ &= d(1-d)\mathrm{E}[\mathbf{x}]^2 + \mathrm{V}[\mathbf{x}]\,(1-d)^2 + \mathrm{V}[\mathbf{x}]\,(1-d)d. \end{aligned}$$

Note that we have used several operators sloppily: we applied scalar operators on vectors. We let this denote the element-wise application.

2.5.1 *Derivatives for Fast Dropout*

Consider a loss $\mathcal{J} = \hat{\mathcal{L}} + \mathcal{R}$, where $\hat{\mathcal{L}}$ is the empirical loss of the model on the training set $\hat{\mathcal{L}} = \mathcal{L}(\mathcal{D}_{\text{train}}; \theta)$ under the current parameters. While it seems difficult to bring the objective function of fast dropout $\mathcal{J}_{\text{fd}}(\mathcal{D}; \theta)$ into the form of $\mathcal{L} + \mathcal{R}$, it is possible with the derivatives of each layer. For this, we perform back-propagation-like calculations. We will do so for both the case of pre-synaptic sampling and where closed form solutions for the post-synaptic moments are available.

2.5.1.1 *Closed Form*

The derivatives of the fast dropout loss with respect to one of the weights are

$$\frac{\partial \mathcal{J}}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \mathrm{E}[a]} \frac{\partial \mathrm{E}[a]}{\partial w_i} + \frac{\partial \mathcal{J}}{\partial \mathrm{V}[a]} \frac{\partial \mathrm{V}[a]}{\partial w_i}.$$

We know that $\mathrm{E}[a] = (d\mathbf{x})^T \mathbf{w}$ and thus

$$\frac{\partial \mathrm{E}[a]}{\partial w_i} = x_i d.$$

This can be recognised as the standard back-propagation term. We will thus define

$$\frac{\partial \mathcal{L}^a}{\partial w_i} := \frac{\partial \mathcal{J}}{\partial \mathrm{E}[a]} \frac{\partial \mathrm{E}[a]}{\partial w_i},$$

and subsequently refer to it as the local derivative of a. The second term can be analysed similarly. We apply the chain rule once more which yields

$$\frac{\partial \mathcal{J}}{\partial \mathrm{V}[a]} \frac{\partial \mathrm{V}[a]}{\partial w_i} = \underbrace{\frac{\partial \mathcal{J}}{\partial \mathrm{V}[a]}}_{:=\dot{\delta}^a} \frac{\partial \mathrm{V}[a]}{\partial w_i^2} \frac{\partial w_i^2}{\partial w_i}.$$

for which any further simplification of $\dot{\delta}^a$ depends on the exact form of $\mathcal{J}$. The remaining two factors can be written down explicitly, i.e.

$$\frac{\partial \mathrm{V}[a]}{\partial w_i^2} = d(1-d)\mathrm{E}[x_i]^2 + d\mathrm{V}[x_i],$$
$$\frac{\partial w_i^2}{\partial w_i} = 2w_i.$$

FD leads to two separate terms for the derivative.

Setting

$$\begin{aligned}\eta_i^a &:= |\dot{\delta}^a| \frac{\partial \mathrm{V}[a]}{\partial w_i^2} \\ &= |\dot{\delta}^a| d\big[(1-d)\mathrm{E}[x_i]^2 + \mathrm{V}[x_i]\big] \\ &> 0\end{aligned}$$

we conclude that

$$\begin{aligned}\frac{\partial \mathcal{J}}{\partial \mathrm{V}[a]} \frac{\partial \mathrm{V}[a]}{\partial w_i} &= 2\,\mathrm{sgn}(\dot{\delta}^a)\eta^a w_i \\ &=: \frac{\partial \mathcal{R}^a_{\text{closed}}}{\partial w_i}.\end{aligned}$$

This lets us arrive at

$$\frac{\partial \mathcal{J}}{\partial w_i} = \frac{\partial \mathcal{L}^a}{\partial w_i} + \frac{\partial \mathcal{R}^a_{\text{closed}}}{\partial w_i},$$

and offers an interpretation of fast dropout as an additive regularisation term.

FD lets weights grow in the face of errors, decays them when the network is right and has no effect near minima of the training loss.

We will now discuss three different scenarios for $\dot{\delta}_a$, depending on the sign.

- $\dot{\delta}^a = 0$ Since the error signal is zero the variance of the unit is optimal for the loss. The fast dropout term vanishes; note that this happens at overall optima of the training loss.
- $\dot{\delta}^a < 0$ The unit should increase its variance. The exact interpretation of this depends on the loss, but in many cases this is related to the expectation of the unit being quite erroneous and leads to an increase of scatter of the output. The fast dropout term encourages a quadratic growth of the weights.

- $\dot{\delta}^a > 0$ The unit should decrease its variance. As before, this depends on the exact loss function but will mostly be related to the expectation of the unit being quite right which makes a reduction of scatter desirable. The fast dropout term encourages a quadratic shrinkage of the weights.

This behaviour can be illustrated for output units by numerically inspecting the values and gradients of the pre-synaptic moments given a loss. For that we consider a single unit $y = f(a)$ and a loss $\ell(y, z)$ measuring the divergence of its output to a target value z. The pre-synaptic variance $\mathrm{V}[a]$ can enter the loss not at all or in one of two ways, respected by either the loss (see (Bayer et al., 2013)) or the transfer function. Three examples for this are as follows.

1. Squared loss on the mean, i.e.

$$\ell(y, z) = (\mathrm{E}[y] - t)^2,$$

with $y = a$.

2. Gaussian log-likelihood on the moments, i.e.

$$\ell(y, z) \propto (\mathrm{E}[y] - t)^2 / 2\mathrm{V}[y] + \log \sqrt{2\pi \mathrm{V}[y]},$$

with $y = a$.

3. Negative Bernoulli cross entropy, i.e.

$$\ell(y, z) = z \log \mathrm{E}[y] + (1 - z)\log(1 - \mathrm{E}[y]),$$

with $y = (1 + \exp(-a))^{-1}$.

We visualise the pre-synaptic mean and variance, their gradients and their respective loss values in Figure 5. For the latter two cases, erroneous units first increase the variance, then move towards the correct mean and subsequently reduce the variance.

2.5.1.2 *Sampling*

One other way of solving the integrals in Eqs. (19,20) is via sampling of the pre-synaptic activation, i.e. $\hat{a} \sim \mathcal{N}(\mathrm{E}[a], \mathrm{V}[a])a$. It follows that

$$\frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \hat{a}}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} \left[\frac{\partial \hat{a}}{\partial \mathrm{E}[a]} \frac{\partial \mathrm{E}[a]}{\partial w_i} + \frac{\partial \hat{a}}{\partial \mathrm{V}[a]} \frac{\partial \mathrm{V}[a]}{\partial w_i} \right].$$

As before,

$$\frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \hat{a}}{\partial \mathrm{E}[a]} \frac{\partial \mathrm{E}[a]}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} x_i d$$

is the standard back-propagation formula with dropout variables.

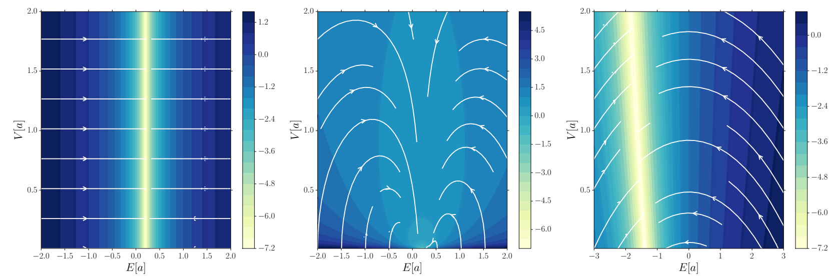


Figure 5: Visualisations of the behaviour of $\dot{\delta}^a$ for a single unit. The axes correspond to the pre-synaptic mean $E[a]$ and variance $V[a]$ feeding into a unit $y = f(a)$. A loss measuring the divergence from the target value 0.2 is then applied and indices the colour on a logarithmic scale. The gradients of the loss are shown as a vector field plot. Squared error is shown on the left, Gaussian log-likelihood in the middle and Bernoulli log-likelihood on the right. For the first two plots, the optimum is in the middle, for the last it is a slightly to the left.

We rewrite the variance as

$$\frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \mathcal{J}}{\partial V[a]} \frac{\partial V[a]}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \hat{a}}{\partial \sqrt{V[a]}} \frac{\partial \sqrt{V[a]}}{\partial V[a]} \frac{\partial V[a]}{\partial w_i^2} \frac{\partial w_i^2}{\partial w_i}$$

which, making use of results from earlier in the section, is equivalent to

$$\frac{\partial \mathcal{J}}{\partial \hat{a}} \frac{\partial \mathcal{J}}{\partial V[a]} \frac{\partial V[a]}{\partial w_i} = \frac{\partial \mathcal{J}}{\partial \hat{a}} \sqrt{d(1-d)E[x_i]^2 + dV[x_i]} s. \tag{21}$$

Since s is Gaussian and all other factors of Eq. (21) are deterministic, the derivative will be a zero-centred Gaussian random variable. Its scale does not depend on the current weight value and is determined by the post-synaptic expectations and variances of the incoming units, the dropout rate and the error signal.

Therefore, also in this case, we can write

$$\frac{\partial \mathcal{J}}{\partial w_i} = \frac{\partial \mathcal{L}^a}{\partial w_i} + \frac{\partial \mathcal{R}^a_{\text{sample}}}{\partial w_i},$$

where $\partial \mathcal{R}^a_{\text{sample}} / \partial w_i$ is defined as in Eq. (21) and essentially an adaptive noise term.

2.5.1.3 *Fast Dropout at Optima of $\hat{\mathcal{L}}$*

In the closed-form as well as in the sampling case the regularisation term vanishes at optima of the training loss. Consequently, no global attractor is formed, making the method theoretically interesting for RNNs: despite of not biasing towards certain solutions, the optimisation is certainly influenced.

2.5.1.4 *Relationship to Weight Decay*

The derivative of weight decay is given as

$$\partial \mathcal{R}_{\text{wd}} / \partial w_i = 2\lambda w_i.$$

We can thus interpret $\mathcal{R}_{\text{closed}}$ as a weight decay term with parameter wise scaling. This scaling depends on the current activations and can even be negative; in the latter case, it will be less of a weight *decay* and more of a weight *growth*.

2.5.2 *Experiments*

Performance of FD was evaluated on the widely-used piano rolls benchmarks, once to compare sRNN with the literature and once to compare sRNN with LSTM.

We conducted two sets of experiments to assess the performance of FD. Once in the context of sRNNs and once in the context of LSTM. For both, distinct experimental runs were performed, as we aimed to achieve close to state-of-the-art performance on these data sets in the first case. In the latter, we were interested in a comparison between LSTM and sRNN, which is why we considered a more open search space.

In both cases, we also resort to several "tricks" to improve the learning. Since the hidden-to-hidden connections and the hidden-to-output connections in an RNN can make use of hidden units in quite distinct ways, we found it beneficial to separate the dropout rates. Specifically, a hidden unit may have a different probability to be dropped out when feeding into the hidden layer at the next time step than when feeding into the output layer. Taking this one step further, we also consider networks in which we completely neglect fast dropout for the hidden-to-output connections; an ordinary forward pass is used instead. Note that this is not the same as setting the dropout rate to zero, since the variance of the incoming units is completely neglected. Whether this is done is treated as another hyper-parameter for the experiment.

Further, we initialised the recurrent weight matrices as described in Section 1.3.3. The input-to-hidden connections $\mathbf{W}^{(0)}$ were initialised with a different standard deviation than the rest of the parameters. Additionally, we set all but a few parameters to zero (Sutskever et al., 2013). For optimisation, we used either Adadelta (Zeiler, 2012) or rmsprop with Nesterov momentum (Sutskever et al., 2013). The decay rate, step rate, offset and momentum were part of the hyper parameter optimisation. All variables in this process were selected via random search to optimise performance on the validation set.

2.5.2.1 *sRNN*

To assess the performance of FD for sRNNs, we conducted experiments on the piano rolls benchmarks popularised by Boulanger-Lewandowski et al. (2012, 2013); Pascanu et al. (2013). It consists of four distinct data sets (“JSBChroales”, “Nottingham”, “Piano-Midi.de” and “Muse”) containing piano notes from different musical backgrounds. This is done by representing a single piece in midi format as a sequence of binary, 88-dimensional vectors. A complete data set is thus written as $\mathcal{D} = \{{}^{i}\mathbf{x}_{1:T}\}_{i=1}^{N}, x_t \in \{0, 1\}^D$ with $D = 88$.

The task is to find a probabilistic model via the cascade decomposition (Barber, 2012), i.e. given a single piece $\mathbf{x}_{1:T}$ we want to model its probability via

$$p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(x_t|\mathbf{x}_{1:t-1}),$$

where $\mathbf{x}_{1:0} := \emptyset$. Here, each $p(x_t|\mathbf{x}_{1:t-1})$ is identified with the output of an RNN at time step t which is parameterised by θ, see e.g. the work of Graves (2013). This is done by using the logistic sigmoid $\sigma(\xi) = (1 + \exp(-\xi))^{-1}$ for the transfer function of the output units and let it model the sufficient statistic of a Bernoulli variable, i.e. the rate of success.

Subsequently, maximising the log-likelihood can be done via minimisation of the negative log-likelihood:

$$\begin{aligned}\hat{\mathcal{L}}(\theta) = \sum_{t,d} & x_{t,d} \log p(x_{t,d}|\mathbf{x}_{1:t-1}) \\ & + (1 - x_{t,d})(1 - \log p(x_{t,d}|\mathbf{x}_{1:t-1})),\end{aligned}$$

where the sum runs over $t = 1 \ldots T$ and $d = 1 \ldots D$ and a single training sequence $\mathbf{x}$ is considered.

We performed randomised search over the hyper parameters (Bergstra and Bengio, 2012), showing the possible choices in Table 1 and the best performing ones in Table 2. The results of the experiments are shown in Table 3.

Table 1: Hyper parameters choices for FD-RNNs in the musical data experiments from Section 2.5.2.1.

Hyper parameter	Choices
#hidden layers	1
#hidden units	$200, 400, 600$
Transfer function	tanh
p(“dropout input”)	$0.0, 0.1, 0.2$
p(“dropout hidden to hidden”)	$0.0, 0.1, 0.2, 0.3, 0.4, 0.5$
p(“dropout hidden to output”)	$0.0, 0.2, 0.5$
Use fast dropout for final layer	yes, no
Step rate	$0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00001$
Momentum	$0.0, 0.9, 0.95, 0.99, 0.995$
Decay	$0.8, 0.9$
W	$\mathcal{N}(0, \sigma^2), \sigma^2 \in \{0.1, 0.01, 0.001, 0.0001\}$
$\mathbf{W}_{\text{in}}$	$\mathcal{N}(0, \sigma^2), \sigma^2 \in \{0.1, 0.01, 0.001, 0.0001\}$
b_h	$\mathbf{0}$
b_y	$-\mathbf{0.8}$
$\rho(\mathbf{W}_{\text{rec}})$	$1.0, 1.05, 1.1, 1.2$
ν	$15, 25, 35, 50$ or no

Table 2: Hyper parameters used for FD-RNNs in the musical data experiments from Section 2.5.2.1.

Hyper parameter	Piano-midi.de	Nottingham	MuseData	JSBChorales
#hidden units	600	400	600	400
p(“dropout input”)	0.1	0.1	0.2	0.1
p(“dropout hidden to hidden”)	0.3	0.4	0.3	0.2
p(“dropout hidden to output”)	–	0.0	0.0	0.5
Use fast dropout for final layer	no	yes	yes	yes
Step rate	0.005	0.001	0.0005	0.001
Momentum	0.995	0.99	0.995	0.99
Decay	0.8	0.9	0.9	0.8
σ^2 for $\mathbf{W}_{\text{rec}}, \mathbf{W}_{\text{out}}$	0.1	0.001	0.1	0.0001
σ^2 for $\mathbf{W}_{\text{in}}$	0.0001	0.1	0.0001	0.01
$\rho(\mathbf{W}_{\text{rec}})$	1.2	1.2	1.2	1.2
ν	25	no	no	15

Table 3: Results of fast dropout on the midi data sets. All numbers are average negative log-likelihoods on the test set, where "FD" represents our work; "plain" and "RNN-NADE" results are due to Bengio et al. (2012) while "Deep RNN" shows the best results by Pascanu et al. (2013). Note that "RNN-NADE" and "Deep RNN" employ various extensions of the model structure of this work, i.e. structured outputs and various forms of depths. Our results are the best for RNNs model with a single hidden layer.

Data set	FD	plain	RNN-NADE	Deep RNN
Piano-midi.de	7.39	7.58	7.05	–
Nottingham	3.09	3.43	2.31	2.95
MuseData	6.75	6.99	5.60	6.59
JSBChorales	8.01	8.58	5.19	7.92

DISCUSSION Fast dropout improves over unregularised sRNNs in every case, therefore confirming our hypothesis that fast dropout constitutes a powerful regulariser for RNNs. Most notably, our results are superior to all other approaches based on sRNN. A downside of the approach is the number of additional hyper parameters, namely the layer specific dropout rates–they vary substantially over different data sets, as can be seen in Table 2.

2.5.2.2 *Fast Dropout for LSTM*

We conducted further experiments as in Section 2.5.2.1 to evaluate the application of fast dropout to RNNs using LSTM. We conducted 64 experiments for each simple RNNs, simple RNNs with fast dropout, LSTM-RNNs and FD-LSTM-RNNs using fast dropout. In contrast to the experiments in Section 2.5.2.1, where the search space for the hyper parameters was tailored to achieve state-of-the-art performance, we considered a more open search space for these experiments in order to not bias the evaluation too much. Especially, all experiments shared the same search space with the exception of the number of hidden units (LSTM based networks used half the units of their simple RNN counterparts). Also, the spectral radius for LSTM was not set, as it is only defined for square matrices. We also considered the smaller two data sets (JSBChorales and Piano-midi.de) only, as the complete search for hyper parameters is computationally demanding on the two bigger ones: each experiment takes on the order of two days on an i7 CPU to train until convergence.

The hyper parameter options and the best hyper parameter choices are shown in Tables 4 and 5 respectively. The results are shown in Table 6.

Table 4: Hyper parameters options for FD-LSTM-RNNs and FD-RNNs in the musical data experiments from Section 2.5.2.2. A "/" in a cell is indicating that the argument before the slash is used for FD-RNNs, while the one after it is used for FD-LSTM-RNNs. $l \dots u$ indicates that we sample uniformly between l and u.

Hyper parameter	Choices
#hidden layers	1
#hidden units	$200 \dots 600$ / $100 \dots 300$
Transfer function	tanh/LSTM
p("dropout input")	$0.01 \dots 0.5$
p("dropout hidden to hidden")	$0.01 \dots 0.5$
p("dropout hidden to output")	$0.01 \dots 0.5$
Use fast dropout for final layer	yes, no
Step rate	$0.0001 \dots 0.1$
Momentum	$0.0 \dots 0.999$
Decay	$0.1 \dots 0.99$
σ^2 for $\mathbf{W}_{\text{rec}}, \mathbf{W}_{\text{out}}$	$0.0000001 \dots 0.1$
σ^2 for $\mathbf{W}_{\text{in}}$	$0.0001 \dots 1.5$
b_h	0
b_y	-0.8
$\rho(\mathbf{W}_{\text{rec}})$	$0.8 \dots 4$ / –
ν	$14 \dots 200$

DISCUSSION In the experiments conducted FD-LSTM improves over plain LSTM in only one of the two cases. In the other case, FD-RNN remains better than both LSTM based approaches, which makes us believe that it is not a case where long-term dependencies are of crucial importance. Consequently, we believe that FD-LSTM and FD-RNN are approaches which should be evaluated both for any learning task.

Table 5: Hyper parameters choices of the best performing experimental runs of FD-LSTM-RNNs and FD-RNNs on the musical data from Section 2.5.2.2.

Hyper parameter	Piano-midi.de		JSBChorales	
	FD-RNN	FD-LSTM	FD-RNN	FD-LSTM
#hidden units	577	512	457	592
p("dropout input")	0.338	0.423	0.183	0.105
p("dropout hidden to hidden")	0.339	0.459	0.216	0.248
p("dropout hidden to output")	0.387	0.498	0.135	0.499
Step rate	0.022	0.092	0.043	0.085
Momentum	0.967	0.681	0.987	0.719
Decay	0.677	0.546	0.862	0.202
σ^2 for $\mathbf{W}_{\text{rec}}, \mathbf{W}_{\text{out}}$	0.089	0.012	0.041	0.031
σ^2 for $\mathbf{W}_{\text{in}}$	0.785	1.193	1.023	0.287
$\rho(\mathbf{W}_{\text{rec}})$	1.354		3.398	
ν	166	183	61	112

Table 6: Results of fast dropout with and without LSTM on the midi data sets. All numbers are average negative log-likelihoods on the test set.

Data set	sRNN	LSTM	FDRNN	FDLSTM
Piano-midi.de	7.89	7.50	7.44	7.57
JSBChorales	8.81	8.38	8.08	8.04

2.6 VARIANCE PROPAGATION FOR REGRESSION

The text in this section has appeared in parts previously in Bayer et al. (2013, 2015).

The techniques from this chapter can be used to find predictive distributions stemming from uncertain quantities in the network.

If one is interested in predictive distributions instead of point predictions for continuous quantities, possibly due to post-processing of the outputs or integration into a graphical model, neural networks can be used via identification of their outputs with the statistics of a desired output distribution. A prominent example is that of mixture density networks (Bishop, 1994), where the outputs of a neural network are used for the responsibilities, expectations and covariances of a mixture of Gaussians. In this section, we will demonstrate how variance propagation can be used naturally to implement Gaussian likelihood functions, given several sources of uncertainty–be it within the weights or the input and hidden activations.

First, we will use the noise injected into a neural network by means of dropout and approximated via fast dropout not only for regularisation. Instead, we will incorporate the resulting variance of the predictions into the objective function (Section 2.6.1). We call this approach *Implicit Variance Network* (IVN). We will then set out to treat the network's parameters as stochastic, i.e. distributed according to a Gaussian distribution. This will lead to i) a fast approximation of VB for neural nets (called *FAWN for Variational Bayes* (FAWN-VB), Section 2.6.2.1) and ii) to the direct optimisation of the predictive distribution, which we regularise with a heuristic inspired by VB (called *FAWN for Regularised Direct Optimisation of the Predictive Distribution* (FAWN-ROPD), Section 2.6.2.2).

During the development of these methods, we noted that both FAWN-VB and FAWN-ROPD rarely overfit. Consequently, we used the *training loss* for model selection. Additionally, as both do not require regularisation parameters (in contrast to IVN, where the dropout rates need to be set), we distinguish between the methods during our experimental evaluation.

We will compare IVN, FAWN-VB and FAWN-ROPD side by side in Section 2.6.3. In Section 2.6.4, we will only pay attention to the latter two, as the tuning of IVNs turned out to be too time sensitive due to the necessity of a validation loop.

2.6.1 *Implicit Variance: Modelling Heteroscedastic Variance*

Recall that for all the hidden units as well as for the output units, we do not have a point but a pair consisting of an expectation and a variance describing the activation. Thus, the output units themselves are already given as their expectation $\mathrm{E}[\mathbf{y}]$ and variance $\mathrm{V}[\mathbf{y}]$. Plugging these into the log-likelihood of a Gaussian, we arrive at the *implicit variance loss*. Let $\Sigma := \mathrm{diag}(\mathrm{V}[\mathbf{y}])$ and $\mathbf{r} := \mathbf{z} - \mathrm{E}[\mathbf{y}]$, then:

$$\mathcal{L}_{\mathrm{iv}} = \frac{1}{2}\mathbf{r}^{\mathrm{T}}\Sigma^{-1}\mathbf{r} + \log(2\pi)^{D/2}|\Sigma|^{1/2}.$$

The uncertainty due to dropout is used for uncertainty in the outputs.

It should be noted that the integration of the variance now has two purposes: i) it adds a new type of quantity to the representative capabilities of the model and ii) it is used for regularisation.

During the development of the model, we found it helpful to introduce a parametric scaling of the variance of each unit. This was necessary to allow the model to achieve very low variances in regions where this is helpful; otherwise the model would "cheat" by copying units and obtain a reduction in variance due to the (invalid) assumption of independence between those units. The parametric scaling is achieved by introducing a new layer wise parameter β with $\beta_{\min} < \beta \in \mathbb{R}$, which scales the variance:

$$\mathrm{E}\left[\mathbf{y}'\right] = \mathrm{E}[\mathbf{y}],$$
$$\mathrm{V}\left[\mathbf{y}'\right] = \beta\mathrm{V}[\mathbf{y}].$$

In the following steps of Algs. 1,2, $\mathbf{y}'$ is used instead of $\mathbf{y}$. [1] The parameter β is optimised jointly with the other parameters of the network θ while $\beta_{\min}$ is optimised as an additional hyper-parameter during model selection. This is mainly to guard the method against collapsing of the model into points, a common issue for mixtures of Gaussians (cf. (Bishop et al., 2006)).

2.6.2 *Fast Adaptive Weight Noise*

Adding noise to the parameters of a model is an efficient mean of regularisation, which can be approximated with VP as well.

Adaptive weight noise is a practical method to perform VB in neural networks (Graves, 2011). The method is based on the approach of Hinton and Van Camp (1993), who utilise the MDL principle (Rissanen, 1985; Grünwald, 2007) as an inductive bias (see Section 1.4.3).

As usual in the Bayesian setting, the parameters of the model under consideration are not found via point estimates, but represented as a

1 Note that we have omitted the layer and time indices.

distribution over the weight space. Here, each parameter θ_i will be represented by a Gaussian, i.e. $q(\theta_i) = \mathcal{N}(\mu_i, \sigma_i^2)$.

If we are given a likelihood function and we consider q as a variational approximation to the true posterior over the parameters having seen the data, the training criterion can be derived by means of VI:

$$\begin{aligned}
\mathcal{L}_{\text{mdl}} &:= -\sum_i \int_\theta q(\theta) \log p({}^i\mathbf{z}|{}^i\mathbf{x}, \theta) d\theta + \mathbb{KL}[q(\theta)||p(\theta)] \\
&= -\sum_i \mathrm{E}\left[\log p({}^i\mathbf{z}|{}^i\mathbf{x}, \theta)\right]_{\theta\sim q} + \mathbb{KL}[q(\theta)||p(\theta)] \qquad (22) \\
&\approx -\frac{1}{S}\sum_i \sum_{s=1}^{S} \log p({}^i\mathbf{z}|{}^i\mathbf{x}, \theta_s) + \mathbb{KL}[q(\theta)||p(\theta)||,]\ \theta_s \sim q(\theta) \qquad (23) \\
&=: \mathcal{L}_{\text{awn}},
\end{aligned}$$

where the outer sum goes over the training samples. The "trick" that Hinton and Van Camp (1993) introduce is that the prior $p(\theta)$ is not set or further specified by a hyper-prior but instead *learned as any other parameter in the model* and thus essentially set by data. The contribution of Graves (2011) was then to approximate the expectation in Eq. (22) by Monte Carlo sampling with Eq. (23).

Here we use the previously introduced techniques to find a closed form approximation to adaptive weight noise. Consider a single layer with $\theta = \{\mathbf{w}\}$, $y = \sigma(\mathbf{x}^T\mathbf{w})$, where we have no dropout variables and the weights are Gaussian distributed with $\mathbf{w} \sim \mathcal{N}(\mu_\mathbf{w}, \sigma_\mathbf{w}^2)$, with covariance diagonal and organised into a vector. Note that we have omitted the bias b for notational clarity. Again, we assume Gaussian density for $a = \mathbf{x}^T\mathbf{w}$. Using the rules from Section 2.3.1, we find that

$$\mathrm{E}[a] = \mathrm{E}[\mathbf{x}]^T \mu_\mathbf{w}, \qquad (24)$$

$$\mathrm{V}[a] = \mathrm{V}[\mathbf{x}]^T \mu_\mathbf{w}^2 + \mathrm{V}[\mathbf{x}]^T \sigma_\mathbf{w}^2 + (\mathrm{E}[\mathbf{x}]^2)^T \sigma_\mathbf{w}^2. \qquad (25)$$

A perspective that we have not used so far on the process is that this is a convolution of point predictions, each performed by a slightly different neural network with weights drawn from their respective distribution. Consider a neural network $f(\mathbf{x}, \theta)$ with $\theta = \{\theta_i\}$, where each θ_i is a Gaussian distributed random variable with mean μ_i and variance σ_i^2. Let the network represent a distribution $p(\mathbf{z}|\theta)$ for the random variable $\mathbf{y}$, which is the network's output. The output of the network with marginalised weights will be approximated as:

$$\int_\theta p(\mathbf{z}|\mathbf{x}, \theta) q(\theta) d\theta \approx \mathcal{N}(\mathrm{E}[\mathbf{y}], \mathrm{V}[\mathbf{y}]), \qquad (26)$$

where $q(\theta)$ depicts the joint over all weights and the moments of the Gaussian variable on the RHS are obtained as in Eqs. (24, 25).

2.6.2.1 *Fast Variational Inference for Gaussian Likelihoods*

We will now use variance propagation to obtain an approximation to the first term of $\mathcal{L}_{\text{mdl}}$ for the special case of a Gaussian likelihood.

Approximating VI with VP requires propagation of the variance through the log of the likelihood function. This is easy in the case of a Gaussian, since just a squaring operation remains.

Consider the first term of the RHS of Eq. (22) for the case that $\mathbf{z}$ is assumed to be a uni-variate Gaussian, i.e. we will only consider the problem for $\mathbf{z} \in \mathbb{R}$ and thus write z for the targets and y for the output of the network. Further, we leave out the dependency on θ for brevity. This generalises to the multi-dimensional case by considering them as independent and summing up the relevant terms. Then,

$$\begin{aligned}
&\mathbb{E}\left[\log p(z|y)\right] \\
&\quad =\mathbb{E}\left[\log \mathcal{N}(z|y,\sigma^2)\right] \\
&\quad =\mathbb{E}\left[\frac{-(z-y)^2}{2\sigma^2} - \log\sqrt{2\pi}\sigma\right] \\
&\quad =\frac{-\mathbb{E}\left[(z-y)^2\right]}{2\sigma^2} - \log\sqrt{2\pi}\sigma \\
&\quad =-\frac{\mathbb{V}\left[y\right]}{2\sigma^2} - \frac{(z-\mathbb{E}\left[y\right])^2}{2\sigma^2} - \log\sqrt{2\pi}\sigma \\
&\quad =\log\mathcal{N}(\sqrt{\mathbb{V}\left[y\right]}|0,\sigma^2) + \log\mathcal{N}(z|\mathbb{E}\left[y\right],\sigma^2) + \log\sqrt{2\pi}\sigma,
\end{aligned}$$

where we have made use of the identity $\mathbb{V}\left[y\right] = \mathbb{E}\left[y^2\right] - \mathbb{E}\left[y\right]^2$. The last line offers a partially probabilistic interpretation of this specific instance of variational inference. It puts a zero centred prior on the square root of the output's variance and on the error, sharing the same (prior) variance—which is itself encouraged to be big. The last term can be seen as a measure against the variance collapsing to zero, which would lead to large likelihoods on the training set.

We will call this method FAWN-VB in the experimental section.

2.6.2.2 *Regularised Optimisation of the Predictive Distribution*

A problem of the approach of Section 2.6.2.1 is that a derivation of the expectation and variance of each considered log-likelihood function is necessary. Since we now have an efficient approximation of the predictive distribution (cf. Eq. (9)), an obvious next step is to directly optimise it with respect to the parameter distributions $q(\theta)$. This will essentially lead to a maximum likelihood approach (see Section 1.4.1) and thus inherit its tendency to overfit the training data. Accounting for that is possible by a fully Bayesian treatment, which means to set $q(\theta) = p(\theta|\mathcal{D}_{\text{train}})$, or an adequate approximation obtained by Laplace's method or VI.

Here we shall follow a different route, which is to make use of a regulariser, namely the KL-divergence between the found $q(\theta)$ and a reference distribution $p(\theta)$. Note that in this case, $p(\theta)$ is *not* a prior per se, as we do not follow a Bayesian approach here.

A simple but effective regulariser for learning predictive distributions based on uncertain weights is the KL-divergence from a reference distribution.

$$\mathcal{L}_{\text{fawn}} := -\sum_i \log \int_\theta q(\theta) p({}^i\mathbf{z}|{}^i\mathbf{x}, \theta) d\theta + \mathbb{KL}[q(\theta)||p(\theta)],$$

where the sum runs over the training samples $\mathcal{D}_{\text{train}} = \{({}^i\mathbf{x}, {}^i\mathbf{z})\}_{i=1}^N$. We want to stress that no regularisation coefficient is required.

We will call this method FAWN-ROPD in the experimental section.

2.6.2.3 *Instances of FAWN used*

For all experiments using Fast Adaptive Weight Noise (FAWN), we chose a Gaussian likelihood where we assumed that

$$z_i = y_i + \epsilon_i, \epsilon_i \sim \mathcal{N}(0, \hat{\sigma}_i^2),$$

which resembles a Gaussian distributed measurement error with variance $\hat{\sigma}_i$ for output dimension i. We integrate the $\hat{\sigma}_i$ into the set of parameters and optimise it jointly with all other parameters.

We used a global uni-variate Gaussian for the prior and a Gaussian to represent each of the parameters:

$$p(\theta) = \prod_i \mathcal{N}(\theta_i | \tilde{\mu}, \tilde{\sigma}^2),$$
$$q(\theta) = \prod_i \mathcal{N}(\theta_i | \dot{\mu}_i, \dot{\sigma}_i^2).$$

The KL-divergence is then given by[2]:

$$\mathbb{KL}[q(\theta)||p(\theta)] = \sum_i \log \frac{\tilde{\sigma}}{\dot{\sigma}_i} + \frac{\dot{\sigma}_i^2 + (\dot{\mu}_i - \tilde{\mu})^2}{2\tilde{\sigma}^2} - \frac{1}{2}.$$

2.6.3 *Experiments with Model Selection*

FAWN has very little dependency on hyper parameters. Since it has very good and general regularisation reducing its tendency to overfit, the results are less dependent on the computational power of the model (as set by the number of layers and hidden units). This is not the case for Implicit Variance Network (IVN): as it is prone to overfitting, the hyper parameters (i.e. model topology and drop out rates), have a huge impact on the final performance. This section aims to show that FAWN can achieve reasonable performance if provided a

We conduct experiments comparing IVNs, FAWN-VB and FAWN-ROPD where the former is allowed to try out many different hyper parameter configurations. The latter methods do not need this.

2 Obtained with the help of the Q&A community "crossvalidated" at http://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians.

powerful enough model with a single training run, where we chose the parameter set with the lowest training error. IVN on the other hand needs a whole validation loop, but is expected to achieve better performance if we have a reliable estimator of the generalisation loss which allows us to tune the hyper parameters very accurately.

We evaluate IVNs and FAWN side by side on two different data sets, together with plain neural networks, fast dropout networks and density networks (a mixture density network with only a single component).

First we revisit a synthetic toy data set introduced by Le et al. (2005) to assess the quality of Gaussian processes to model heteroscedastic data. We will then move to a robotic inverse dynamics data set, the popular SARCOS data.

For IVN, the hyper parameters were optimised using random search as advocated by Bergstra and Bengio (2012) for IVNs. We optimised for the number of hidden layers (either 1, 2 or 3), the transfer function (sigmoid or rectifier), the parameters of rmsprop (step rate, decay rate, momentum) and the variance of the input as well as the variance offset β_{min}. Further hyper parameter ranges are specified in the respective sections.

In the case of FAWNs, we did without model selection–instead we used a fixed topology, given in the respective section. For optimisation, we used Adam (Kingma and Ba, 2014) with a step rate of $\alpha = 0.001$ and training was done until the convergence of the *training loss*.

2.6.3.1 *Toy data set*

The toy data set allows comparison of in a heteroscedastic regression problem.

The data set is governed by the equations

$$\begin{aligned} \mu_i &= 2\left(\exp\left(-30(x_i - \frac{1}{4})^2\right) + \sin(\pi x_i^2)\right), \\ \sigma_i^2 &= \exp(\sin(2\pi x_i)), \end{aligned}$$

which specify Gaussian distributions $p(z_i|x_i) = \mathcal{N}(\mu_i, \sigma_i^2)$. To generate a data set we sample $\{x_i\}$ points uniformly from the input range $[-0.1, 1]$, and then sample $\{z_i\}$ accordingly. To compare plain neural networks (NN), density networks (DN), networks trained with fast dropout (FD) and implicit variance networks (IVN), we constructed a setting which is far from tailored towards neural networks: very few data.

The data set contained only 50 points for training, 10 for validation and 50 additional points to assess the performance as a test set. We trained the IVNs for 5000 epochs with batch size either 10, 25, 50 and 10, 25, 50 or 100 hidden units. For FAWN, we used 100 hidden units in a single hidden layer.

The results are summarised in Table 7 on page 52.

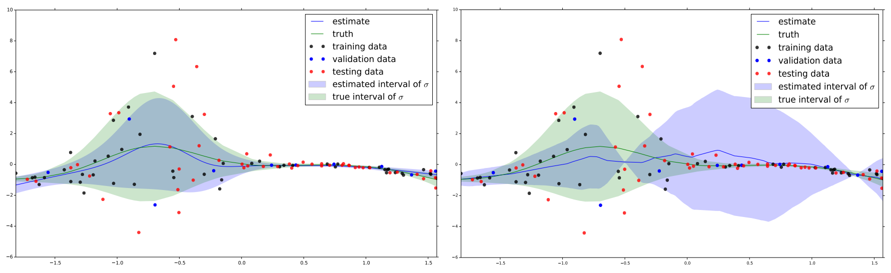


Figure 6: Predictive distributions of IVNs and DNs on the toy benchmark.

discussion Notably, all methods except ours perform as bad as expected for a neural network model in this setting. While the density networks are en par with our method in terms of mean squared error, they overfit extremely with respect to their predictive distribution. Neural networks neither trained classically nor with fast dropout achieve good results; the variance of fast dropout seems to be meaningless, which is not surprising as it is not trained. FAWN for Variational Bayes (FAWN-VB) did not manage to cope with this data set–instead it consistently found minima where it used the variance of the likelihood function to explain the data. Therefore, it predicted a straight line and the empirical variance over the whole data. We hypothesise that the small data setting here is not a setting where VB is a good fit. FAWN for Regularised Direct Optimisation of the Predictive Distribution (FAWN-ROPD) achieved performance superior to IVN in terms of likelihood and comparable to it in terms of squared error, rendering it the best method on this data set.

2.6.3.2 *SARCOS*

We evaluated the models under consideration on a standard benchmark for learning robot inverse dynamics, the "Sarcos" data set. We trained IVNs for 500 epochs and picked the batch size from $\{64, 128, 256, 512\}$ and the number of hidden units from $\{50, 100, 200, 300\}$.

We compare IVNs, FAWN-VB and FAWN-ROPD on the robot inverse dynamics tasks SARCOS.

For FAWN, we used 100 hidden units in three hidden layers each.

The results are summarised in Table 7 on page 52.

discussion We want to stress several observations. For one, IVNs seem to be the best choice if one is interested in good performance of both MSE and NLL. Secondly, plain neural networks perform surprisingly well in our experiments. While both Gaussian processes and LWPR models have different advantages compared to neural networks (model uncertainty and efficient incremental online learning, respectively) our experiments show that both are outperformed in terms of predictive quality. FAWN-VB and FAWN-ROPD both feature model uncertainty and are superior to GPs in these experiments.

Table 7: Results on the heteroscedastic toy benchmark introduced by Le et al. (2005) and the Sarcos data set for regression methods introduced in this chapter. NN refers to a simple neural network, DN to density network, FD to a neural network trained with fast dropout. Results for Gaussian processes (GP) and Locally-weighted projection regression (LWPR) taken from Rasmussen (2006). We report the mean squared error (MSE) and negative log-likelihood (NLL). Best results shown in bold.

	Toy		**Sarcos**	
Method	MSE	NLL	MSE	NLL
NN	4.2395	2.2694	**0.0047**	-1.1893
DN	**3.8706**	9.7303	0.0096	-1.2532
FD	4.3491	43486.7	0.0065	1.2667
GP	–	–	0.011	2.25
LWPR	–	–	0.040	–
IVN	3.8985	1.6187	0.0079	**-1.3606**
FAWN-VB	4.2893	3.0230	0.0088	-0.9081
FAWN-ROPD	3.9488	**1.1538**	0.0123	-1.2196

2.6.4 *Experiments without Model Selection*

We compare FAWN-VB and FAWN-ROPD with results from the literature. Due to the size of the experiments IVNs, which require a validation loop, were not considered.

The experiments in this section serve a different purpose than those in the previous section. Here, the hyper parameters are fixed over a wide range experiments (with the exception of slightly larger hidden layers in one case). Further, the data sets are split into folds and the generalisation error is estimated on a left out fold, in contrast to a single train/test split.

These tasks are typically not where neural networks excel and practitioners resort to GPs instead, which is why we compare with them.

All experiments were performed using a similar protocol to the one used by Hernández-Lobato and Adams (2015): we used single layer networks with 50 hidden units using the rectifier transfer function, with the exception of the "Year" data set where 100 hidden units were used and the official train/test split was used. We report the negative-log likelihood of the data with means and standard deviations coming from ten different random splits into 90% training and 10% testing data. The parameters of neural networks using FAWN were drawn from a zero centred Gaussian with standard deviation 0.2.

Training was performed using Adam (Kingma and Ba, 2014) with a step rate of $\alpha = 0.001$ until convergence of the *training loss*. No separate validation set was used. Gradients were estimated using 128 samples in a single mini batch.

The results for GPs were obtained using a the sum of a linear and a squared exponential kernel using automatic relevance determination. We do not provide results for data set with more than 1′500 data points, since training times where magnitudes higher for GPs in those cases. Three random restarts were performed. We used GPy (The GPy authors, 2012–2014) for the experiments.

The results are shown in Table 8 on page 54.

Table 8: Results for FAWN. Results for probabilistic back-propagation (PBP) and adaptive weight noise (VB) taken from Hernández-Lobato and Adams (2015). Results for GPs obtained via GPy (The GPy authors, 2012–2014). Best results shown in bold.

	N	D	VB	PBP	GP	FAWN-VB	FAWN-ROPD
Boston	506	13	2.903±0.071	**2.550±0.089**	2.6313±0.2885	3.0045±0.2734	**2.5588±0.1614**
Concrete	1030	8	3.391±0.017	3.136±0.021	**2.8934±0.09514**	3.1828±0.0772	3.1071±0.1342
Energy	768	8	2.391±0.029	1.982±0.027	**0.7110±1.4771**	1.7618±0.6548	1.3696±0.84190
Kin8Nm	8192	8	0.897±0.010	-0.964±0.007	-	-1.006±0.0274	**-1.2111±0.03268**
Naval	11′934	16	-3.734±0.116	-3.653±0.004	-	-6.7507±0.1183	**-6.8370±0.1305**
Power Plant	9568	4	2.890±0.010	**2.838±0.008**	-	2.8491±0.0421	**2.8190±0.0287**
Protein	45′730	9	2.992±0.006	2.974±0.002	-	2.9728±0.0221	**2.8824 ± 0.0682**
Wine	1599	11	0.980±0.013	0.966±0.014	-	0.9429±0.0370	**0.9082±0.0785**
Yacht	308	6	3.439±0.163	1.483±0.018	0.6147 ± 0.7562	1.4482±0.3925	**0.3360 ± 0.2709**
Year	515′345	90	3.622±N/A	3.603±N/A	-	3.8074 ± N/A	**3.4729±N/A**

DISCUSSION The results show that FAWN obtains competitive performance over a wide range of regression tasks. These tasks include ones with very little samples (order of a few hundred) as well as many samples (several thousands). We note that FAWN-VB improves over its sampling based counterpart VB in all but two cases and FAWN-ROPD is always better than the other neural network based approaches, and inferior to Gaussian processes only twice.

2.7 FURTHER APPROXIMATIONS

A few more regularisation schemes from the literature can be approximated using the framework of variance propagation. We present them here for for completeness, but did not verify them experimentally.

2.7.1 *Approximated DropConnect*

The noise process proposed by Wan et al. (2013) performs dropout of weights, i.e.

$$\tilde{\mathbf{w}} = \mathbf{w} \odot \zeta$$

with ζ being a mask of Bernoulli distributed binary variables. The derivation is symmetrical to the one of dropout, which is why we leave it out for brevity.

2.7.2 *Approximated Gaussian Weight Noise*

Gaussian weight noise is popular in the context of recurrent networks (see e.g. (Graves, 2013)), where regularisers such as weight decay do not work well (Pascanu et al., 2012). Here, the noise process is defined as $\tilde{\mathbf{w}} = \mathbf{w} + \boldsymbol{\epsilon}, \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ and similarly done for the bias $\tilde{b}$.

The variance σ^2 is typically set via during model selection. The overall model is then

$$a = \tilde{\mathbf{w}}^T \mathbf{x} + \tilde{b}.$$

The expectation of a random variable after application of the corruption operator is then simply the expectation of the input

$$\begin{aligned} \mathrm{E}[\tilde{\mathbf{w}}] &= \mathrm{E}[\mathbf{w}] + \mathrm{E}[\boldsymbol{\epsilon}] \\ &= \mathrm{E}[\mathbf{w}]. \end{aligned}$$

The variance on the other hand is increased:

$$\begin{aligned} \mathrm{V}[\tilde{\mathbf{w}}] &= \mathrm{V}[\mathbf{w}] + \mathrm{V}[\boldsymbol{\epsilon}] \\ &= \mathrm{V}[\mathbf{w}] + \sigma^2 \mathbf{I}. \end{aligned}$$

2.8 CONCLUSION

In this chapter, we have contributed a general method of propagating random variables up to second order through neural networks. Most importantly, this method does not need sampling operations, which i) are expensive and ii) lead to highly-variant gradient estimates, potentially slowing down learning.

This has enabled us to approach certain well-known models and learning methods from novel perspectives. We have shown theoretically that fast dropout (Wang and Manning, 2013) does not suffer from issues rendering classical regularisers detrimental to RNNs. Subsequently, we have experimentally verified its applicability with sRNNs and RNNs based on LSTM and improved upon the state-of-the-art in the conducted experiments. Further, we have applied these techniques to problems with continuous output variables. This included a deterministic method to perform variational inference for the learning problem of neural network weights, which distinguishes it from approaches based on sampling(Graves, 2011). Additionally, we have proposed to directly optimise the predictive distribution under an information theoretic constraint on the distribution of the parameters in a non-Bayesian framework, which yielded results better than previous NN based approaches and GPs.

2.9 FUTURE WORK

The work in this chapter forms a basis for a wide range of future research directions. Several other approaches to find approximations of the Bayesian posterior distributions are tempting, such as finding the expectations and variances of the parameters via Markov Chain-Monte Carlo (Neal, 1993) or better approximations of the predictive distributions yielded by a Laplace approximation (MacKay, 1992). Obviously, the proposed methods from Section 2.6 need to be extended to use with RNNs.

3 REDUCING SEQUENCES TO POINTS

3.1 INTRODUCTION

The text in this section has appeared in parts previously in Bayer et al. (2012).

Machine learning problems are traditionally formulated as the estimation of a function $f : \mathbb{R}^m \to \mathbb{R}^n$. In many areas, such as text, audio, video, robotics, bio informatics, signal processing etc., the input data is obtained in a sequential form. One reason for this might be a sensor which is queried with a certain frequency. There are many scenarios this is not due to any temporal nature of the data, e.g. text or an event list of credit card transactions (Bayer, 2002).

A manual mean to deal with this issue is that of manual feature extraction. In that case, domain experts devise *features* of the input data, that is numerical values which are believed to be informative for the task at hand. These are then fed into a subsequent off-the-shelf regressor or classifier. While this requires domain knowledge and is rather time consuming, very good results are usually achieved.

We advocate the use of RNN-based architectures to reduce sequential data to fixed-length representations.

This chapter investigates an alternative, automated route. Recurrent sequence reduction, a framework for finding fixed length representations of sequential data, is introduced. The combination of recurrent neural networks, pooling operators, finely tuned initialisations and various objective functions leads to efficient methods of embedding sequential data into a feature space.

These can be used twofold. Firstly, exploratory data analysis can be performed, e.g. by visually inspecting the features. Secondly, the resulting embeddings will live in a vectorial space in which standard methods are applicable: any discriminative algorithm tailored towards the static domain can thus be stacked on top. If it happens to be differentiable, the resulting system can be fine tuned further by making use of the chain rule and a gradient-based optimiser.

Results on a wide range of data sets task are presented. We also analyse the obtained embeddings visually.

3.2 RELATED WORK

The text in this section has appeared in parts previously in Bayer et al. (2012).

Only a few principled approaches exist for extracting fixed length features from sequential data. If a distance measure was given, similarity matrix-based techniques (Kruskal, 1964) were straightforward to use. Dynamic time warping (Berndt and Clifford, 1994) works well for smooth time series but does not capture any underlying dynamics. Unsupervised feature extraction from time series is widely used for denoising and dimensionality reduction. Classic approaches, such as the discrete Fourier transform or matrix decompositions like PCA (leading to singular spectrum analysis (Vautard et al., 1992)) can only be applied to finite-length time windows.

Another approach is to identify a model (such as a polynomial or a linear dynamical system) for each sample in the data set and use the estimated parameters as features (see the work done by Li and Prakash (2011)). This essentially ties the feature dimensionality to the number of parameters of the model making it a non-free parameter. Also, depending on the complexity of the task and the size of the data set, estimation of a single model for each sample is not feasible in reasonable time.

One idea to circumvent the latter is to train a global model for all samples and to use the sample specific derivatives of the objective with respect to the parameters as features for that sample. Fisher kernels (Jaakkola and Haussler, 1998) exploit this: the features for a sequence are the elements of the gradient of the log-likelihood of this sequence with respect to the model parameters. This choice can presumably be very bad: if the distribution represented by the trained model closely resembles the data distribution the gradients for all sequences in the data set will be nearly zero. A recent paper by van der Maaten (2011b) alleviates this problem by exploiting label information and employing ideas from metric learning. Obviously, this only works if class information is available. Further, the dimensionality of the embedding space becomes tied to that of the parameter space of the model as well.

In principle, any sequential clustering technique can be used as a feature extractor by treating the scores (e.g. the posterior likelihood in case of a generative model or the distances to a node of a self-organising map) as features.

Using recurrent architectures for learning distributed representations of sequences goes back at least to Pollack (1990). A related approach are recursive auto encoders (Socher et al., 2011). RNNs have previously been trained with unsupervised criterions in this regard by Klapper-Rybicka et al. (2001). More recently, Sutskever et al. (2014) proposed a supervised approach: an input sequence is directly mapped to an output sequence, where the hidden layer at the last time step of the input serves as an "information bottleneck". In a similar manner, Fabius et al. (2014) compress a sequence to a fixed length representation using VI as a guiding principle. Consequently, the resulting model is based on probabilistic principles, allowing a wide range of operations.

3.3 FRAMEWORK

Similar to Collobert et al. (2011) we reduce output sequences to a single vector with a *pooling operation*. A pooling operation is a function $\rho : X^+ \to X$ that reduces an undefined number of inputs to a single output of the same set, e.g. taking the sum or picking the maximum. If our pooling operation is differentiable as well, we can use it as a gateway to arbitrary objective functions that are defined on real vectors.

The text in this section has appeared in parts previously in Bayer et al. (2012).

Given an RNN f parameterised by θ, a data set $\mathcal{D} = \{\mathbf{x}_i\}$, a pooling operation ρ and an objective function $\mathcal{L}$ we proceed as follows.

The output sequence of an RNN is reduced to a fixed-length vector via a pooling operation. This architecture can be trained by gradient-based schemes.

1. Process input sequences $\mathcal{D}_{\text{train}} = \{{}^i\mathbf{x}_{1:T}\}_{i=1}^N$ to produce output sequences $\{{}^i\mathbf{y}_{1:T}^{(K)}\}$.
2. Use a pooling operator ρ to reduce the output sequences to a point via ${}^ie = \rho({}^i\mathbf{y}_{1:T}^{(K)})$.
3. Calculate the objective function $\mathcal{L}(\{{}^i\mathbf{e}\})$.

Since the whole calculation is differentiable, we can evaluate the derivative of the objective function with respect to the parameters of the RNN via

$$\frac{\partial \mathcal{L}}{\partial \rho}\frac{\partial \rho}{\partial f}\frac{\partial f}{\partial \theta}. \tag{27}$$

Subsequently, we can use gradient-based optimisation to find a set of parameters θ which leads to good embeddings $\{e_i\}$.

Note that this method has two appealing characteristics from a computational perspective: first, finding a descriptor for a new sequence has a complexity in the order of the length of that sequence. Furthermore, the memory requirements for that descriptor are invariant of the length of the sequence and can thus be tailored towards memory requirements. Indeed, millions of such descriptors can easily be held in main memory to allow fast similarity search.

We chose the criterions in a way that no metric in the sequence space is assumed, since this is what we essentially want to learn from the data. Four unsupervised and one supervised criterion are considered: We will first establish randomly initialised and untrained RNNs as a baseline. We will then train RNNs to do one-step prediction, maximise information with a regularisation penalty and use an objective that encourages lifetime and population sparsity in the found features. For the supervised case, we consider an objective function that encourages samples of the same class to be close in the embedding space.

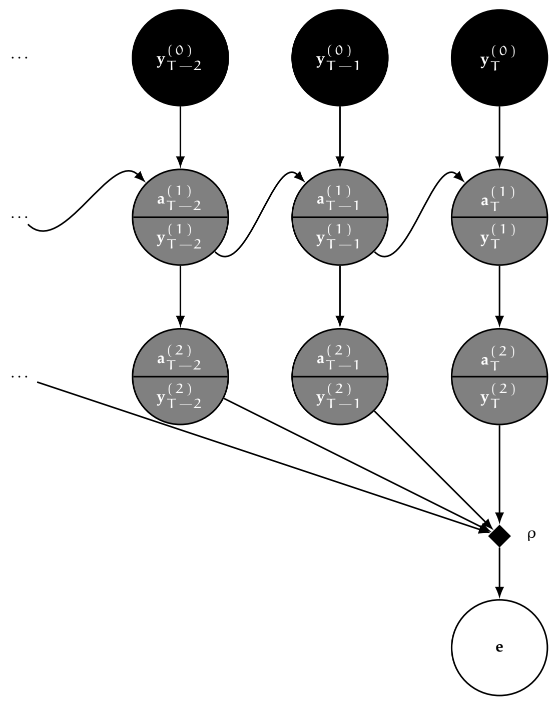


Figure 7: Illustration of the sequence reduction framework. Compare Figure 2. The pooling operator is shown as a black diamond. The sequence is scanned completely, after which internal representations (post-synaptic output in this case, but it can vary) are fed into the pooling operator shown as a black diamond. The pooling operator does not take the ordering of its inputs into account. The result is then the embedding **e**.

3.4 SUPERVISED SEQUENCE REDUCTION WITH NCA

The text in this section has appeared in parts previously in Bayer et al. (2012).

Goldberger et al. (2004) note that metrics and features are actually closely related: by measuring pairwise distances between the data points, the data can be embedded into a metric space. A Mahalanobis distance is learned by mapping the high-dimensional input space to a low-dimensional embedding space in which k-nearest neighbour classification performance is maximised. Since the hard assignment of k neighbours is relaxed to a soft probabilistic neighbouring scheme, the resulting objective function is differentiable with respect to the embedding.

Similar to Salakhutdinov and Hinton (2007), we use a different model than a linear map to represent the embedding function, that is RNNs.

3.4.1 *Neighbourhood Component Analysis*

The central assumption of Neigbourhood Component Analysis (NCA) (Goldberger et al., 2004; Salakhutdinov and Hinton, 2007) is that items of the same class lie near each other on a lower-dimensional manifold. To exploit this, we want to learn a function $f : \mathcal{X}^+ \rightarrow \mathcal{Y}$ from the sequence space $\mathcal{X}^+$ to a metric space $\mathcal{Y}$ that reflects this. Recall that in our case, the embedding function is $\rho(f(\mathbf{x}_{1:T};\theta))$. Given a set of sequences with an associated class label $\{{}^i\mathbf{x}_{1:T}, {}^iz\}$ mapped to a set of embeddings $\{{}^i\mathbf{e}\}$, we define the probability that a point a selects another point b as its neighbour based on Euclidean pairwise distances as

$$p_{ab} = \frac{\exp(-||{}^a\mathbf{e} - {}^b\mathbf{e}||_2^2)}{\sum_{z \neq a} \exp(-||{}^a\mathbf{e} - {}^z\mathbf{e}||_2^2)},$$

while the probability that a point selects itself as a neighbour is set to zero: $p_{aa} = 0$. The probability that a point i is assigned to class k depends on the classes of the points in its neighbourhood $p(z_i = k) = \sum_j p_{ij}\mathbb{I}(z_j = k)$, where $\mathbb{I}$ is the indicator function. The overall objective function is then the negative expected number of correctly classified points

NCA aims to project points of the same class close to each other.

$$\mathcal{L}_{\text{nca}} = -\sum_i \sum_j p_{ij}\mathbb{I}(c_i = c_j).$$

Although NCA has a computational complexity that is quadratic in the number of samples in the training set for training, using batches containing roughly 1000 samples made this negligible. We did not observe any decrease of test performance.

3.4.2 *Classifying Sequences*

We first train an RNN on our data set with the NCA objective function. Afterwards, all training sequences are propagated through the network and the pooling operator to obtain embeddings $\{{}^{i}\mathbf{e}\}$ for each of them. We then build a nearest neighbour classifier for which we use all embeddings of the training set. A new sequence $\tilde{\mathbf{x}}_{1:T}$ is classified by first forward propagating it through the RNN and obtaining an embedding $\tilde{\mathbf{e}}$. We then find the k-nearest neighbours and obtain the class by a majority vote.

3.4.3 *Experiments*

To show that our algorithm works as a classifier we present results on several data sets from the UCR Time Series archive (E. et al., 2006). We refer the reader to the corresponding web page for detailed descriptions of each data set. The data sets from UCR are restricted in the sense that all are uni-variate and of equal length. Since our method is well suited to high-dimensional sequences, we proceed to the well known TIDIGITS benchmark afterwards.

3.4.3.1 *UCR Time Series Data*

The hyper parameters for each experiment were determined by random search. We did 200 experiments for each data set, reporting the test error for those parameters which performed best on the training set. The hyper parameters were the number of hidden units, the used transfer function (sigmoid, tangent hyperbolic rectified linear units or LSTM cells), the optimisation algorithm (either RPROP or LBFGS), the pooling operator (either sum, max or mean), whether to centre and whiten each sequence or the whole data set and the size of the batch to perform gradient calculations on.

The training and test performances stated are the average probabilities that a point is correctly classified by the stochastic classifier used in the formulation of NCA. We also report the error for 1-nearest neighbour classification on the test set as 1NN with the training set as a data base to perform nearest neighbour queries on. 1NN-DWT corresponds to the best DWT classification results on the UCR page. If a certain data set from the UCR repository is not listed, performance was not satisfactory. We attribute this to small training set sizes in comparison with the number of classes with which our method seems to struggle. This is not at all surprising, as the number of parameters is sometimes exceeded by the number of training samples.

Table 9: Results of NCA with RNNs on several data sets from the UCR archive.

Data set	Train	Test	our 1NN	DWT 1NN
Wafers	0.984	0.987	0.987	0.995
Two Patterns	0.992	0.996	0.99725	0.9985
Swedish Leaf	0.797	0.772	0.848	0.843
OSU Leaf	0.684	0.457	0.579	0.616
Face (all)	0.938	0.833	0.647	0.808
Synthetic Control	0.999	0.962	0.96	0.983
ECG	0.999	0.846	0.88	0.88
Yoga	0.684	0.73	0.699	0.845

DISCUSSION While the NCA in conjunction with RNNs is able to achieve non-trivial performance, it is outperformed by simple nearest neighbour techniques. We want to stress however, that nearest neighbour has an unfair advantage here: the sequences are all of equal length and one-dimensional. Further, plain nearest-neighbour is not able to generalise to unknown dynamics. We will move to a data set in the next section which is a better battleground for RNN-based methods.

3.4.3.2 *TIDIGITS Data*

TIDIGITS (Leonard and Doddington) is a data set consisting of spoken digits by adult and child speakers. We restricted ourselves to the adult speakers. The audio was preprocessed with mel-frequency cepstrum coefficient analysis to yield a 13-dimensional vector at each time step.

During training we went along with the official split into a set of 2240 training and 2260 testing samples. 240 samples from the training set were used for validation. We trained the networks until convergence and report the test error with the parameters achieving the best validation error. We used 40 LSTM cells to get 30 dimensional embeddings. For comparison, we also trained LSTM-RNNs of similar size with the cross entropy error function for comparison. Since both methods yield discriminative models, we can report the the average probability that a point from the testing set is correctly classified, which was 97.9% for NCA and 92.6% for cross entropy.

DISCUSSION In direct comparison, NCA in combination with RNNs has a clear advantage over the classical loss function, binary cross-entropy. Our method also naturally yields embeddings which can be

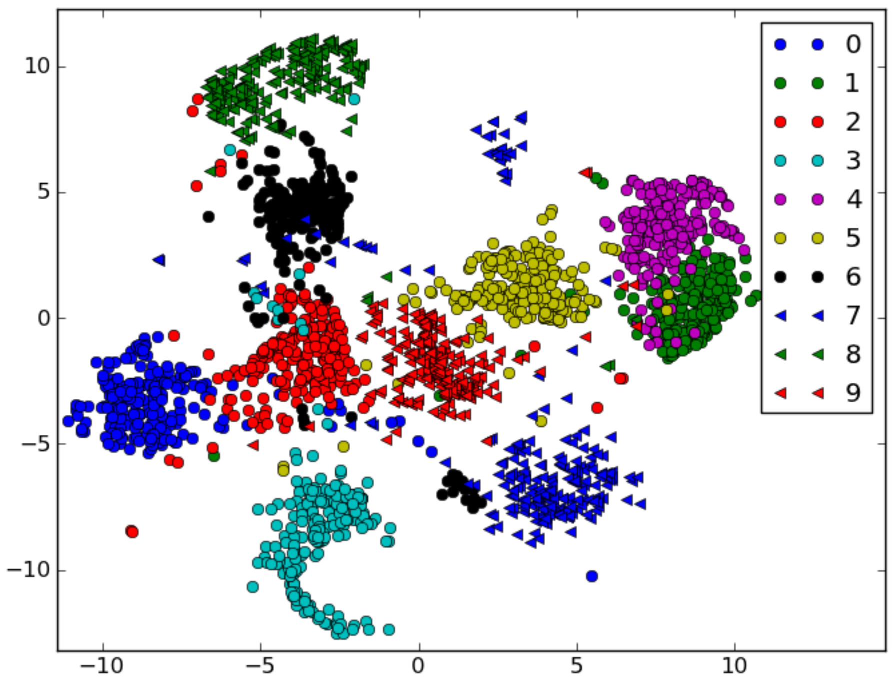


Figure 8: The output of our method after reducing the dimensionality of the embeddings with t-SNE (van der Maaten and Hinton, 2008) [1]. The data is arranged into mostly distinct clusters. Interestingly, the NCA objective also makes it possible for points of the same class to arrange in several clusters. This is not the case for objectives that try to separate the data with a functional form such as a hyperplane.

visually investigated. For a visualisation of the found embeddings and a small discussion, see Figure 8.

3.5 UNSUPERVISED SEQUENCE REDUCTION

Most of the data available is not labelled. At the time of writing, it is unproblematic to acquire huge repositories of sensory data, log files, user generated content or publicly available texts. Yet, the process of labelling data, (i.e. the expert-based connection of it with side information) requires either direct human intervention or an automated process based on a model. Consequently, we can expect that most of the data available will always be unlabelled. To distil this data into useful information, unsupervised methods are required which

1 t-Distributed Stochastic Neigbourhood embedding (van der Maaten and Hinton, 2008) is a method to embed high-dimensional spaces in 2D or 3D to be easily visualizable. This is done by assuming distances to be distributed according to a Student's t in the low and a Gaussian distribution in the high-dimensional space. Local structure is well preserved as long distances in the high-dimensional space are less precisely modelled than short ones.

attempt to find reoccurring patterns, cluster data into groups and find distributed and possibly sparse representations. This is the main topic of this section.

3.5.1 *Objective Functions*

We will revisit several objective functions from the literature which have so far been applied to static data only. Due to the flexibility of our framework, these are easily integrated.

3.5.1.1 *Randomly initialised RNN*

RNNs are initialised similar to echo state networks.

All RNNs were initialised randomly by sampling from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ and then setting the spectral radius as described in Section 1.3.3.

For finding a representation, we performed the pooling operation on the hidden states over time. Similar to Echo-State Networks (Jäger et al., 2003), we assume that the dynamics in the network will suffice to form descriptors carrying important information.

3.5.1.2 *One-step prediction RNN*

The RNNs are trained as a generative model of the data, predicting the next time step.

Predicting the next time step of a sequence is a way of using an RNN as a generative model, e.g. as done by Graves (2013). Essentially, the distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_{1:t})$ is estimated by an RNN the outputs of which can be interpreted as the sufficient statistics of p. Given that the outputs $\mathbf{a}_{1:T}^{(K)}$ are once more transformed by a non-linearity, e.g. a softmax or a sigmoid, multinomial or Bernoulli distributions can be modelled. In our case, we restrict ourselves to linear outputs (i.e. $f^{(K)}(\xi) = \xi \Rightarrow \mathbf{a}_t^{(K)} = \mathbf{y}_t^{(K)}$) which represent the expectation of a Gaussian distributed random variable $\mathcal{N}(\mathbf{y}_t^{(K)}, \sigma^2)$ for some fixed variance. Maximising the likelihood of observed data then results in minimising the negative log-likelihood which is equivalent to the sum of squares error. Let the dimensionality of each input at a given time step be κ. The loss is then

$$\mathcal{L}_{\text{osp}} = \sum_{i=1}^{N} \sum_{t=1}^{T-1} \sum_{c=1}^{\kappa} ({}^{i}\mathbf{y}_{t,c} - {}^{i}\mathbf{x}_{t+1,c})^2,$$

where we let ${}^{i}\mathbf{x}_{t,i}$ and ${}^{i}\mathbf{y}_{t,i}$ denote the c'th component at the t'th time step of sample i of the input or output respectively.

The reasoning behind this approach is that similar sequences will also have similar activities in the hidden activations, since these are ultimately responsible for modelling them. Henceforth, we pool each hidden unit along the time axis to find an embedding ${}^{i}\mathbf{e}$ for sequence ${}^{i}\mathbf{x}$.

3.5.1.3 *Regularised Information Maximisation*

The RNNs are trained to assign the data to one of a set of clusters.

Regularized Information Maximization (RIM) was originally introduced as a discriminative clustering technique by Gomes et al. (2010): the multinomial distribution over cluster labels given an input $p(z|\mathbf{x})$ is modelled with an affine map in its simplest case. The training objective is to maximise the mutual information between the input $\mathbf{x}$ and the cluster assignment z, $\mathbb{MI}[z, \mathbf{x}]$. All parameters but the constant offset of the output ("bias") are regularised with strength λ by a penalty term $\mathcal{R}$:

$$\mathcal{L}_{\text{rim}} = \mathbb{MI}[z, \mathbf{x}] - \lambda\mathcal{R}(\theta \setminus \{b^{(K)}\}).$$

It was empirically observed that different values for λ give rise to different numbers of clusters. Since the unregularised bias can draw a single component of $p(y|\mathbf{x})$ arbitrarily close to zero, only a subset of the possible outputs are actually used by the algorithm. Thus, the number of clusters need not be fixed beforehand (as with most clustering techniques such as K-Means) but is related to the continuous choice of λ.

Given a model $p(z|\mathbf{x}, \theta)$, we can calculate the quantities empirically by aggregating over the training set. First note that

$$\mathbb{MI}[z, \mathbf{x}] = \mathbb{H}[\hat{p}(z|\theta)] - \frac{1}{N}\sum_{i} \mathbb{H}[p(z|{}^{i}\mathbf{x}, \theta)],$$

where $\mathbb{H}[q]$ denotes the entropy of q. The probability of cluster z being selected marginalised over the input space can be approximated via $\hat{p}(z|\theta) \approx \frac{1}{N}\sum_i p(z|\mathbf{x}_i, \theta)$. For $\mathcal{R}$, we use an L_2 term as in the original article.

To obtain a model of $p(z|\mathbf{x})$, we apply a softmax activation function to the found embeddings $\{{}^{i}\mathbf{e}\}$. This essentially leads to our representation being related to a distribution over local representations, the corresponding cluster labels.

3.5.1.4 *Sparse Filtering RNN*

The RNNs are trained to find representations which are sparse over the data samples and the dimensions of the embedding.

Sparse filtering was introduced by Ngiam et al. (2011) as a mean of extracting features from image patches. The idea is to explicitly punish lifetime and population sparsity. For one, a specific feature should be turned off (which means that it is zero) for most samples. Conversely, a specific sample should only have a little amount of turned on features. To avoid degenerate solutions (e.g. by turning off all features), the features are forced into competition by a joint normalisation factor: The c′th component of the embedding is first normalised by its L2-norm across the whole data set:

$$^{i}\tilde{\mathbf{e}}_c = \frac{^{i}\mathbf{e}_c}{\sum_j {}^{j}\mathbf{e}_c^2},$$

and then inserted into the sparse filtering objective:[2]

$$\mathcal{L}_{\text{sf}} = \sum_{i=1}^{N} \left\| \frac{^{i}\tilde{e}}{\|^{i}\tilde{e}\|_2} \right\|_2 .$$

The reader is referred to the original paper for a more detailed motivation of the construction of the loss.

In our case we chose to obtain the features by pooling over the outputs of the RNN: $^{i}\mathbf{e} = \rho(^{i}\mathbf{y}_{1:T}^{(K)})$.

3.5.2 *Experiments*

All experiments were conducted by performing a random search over the architecture and details of the training procedure, as advocated by Bergstra and Bengio (2012). One architecture was specified by the choice of pooling operation and the choice of transfer function in the hidden layer. The transfer functions were either from the tanh, softsign, rectifier or softplus. For pooling operations, we used max, sum, mean and stochastic. The batch sizes were sampled from the set $\{\frac{N}{8}, \frac{N}{4}, \frac{N}{2}, N\}$, where N is the size of the training set. The standard deviation for picking the initial weights was selected uniformly from $[0.005, 1]$, the weight of the sparsity penalty was drawn uniformly from the interval $[10^{-7}, 0.1]$. The number of hidden units and the embedding dimensionality were selected data set wise.

To achieve more efficient training, we prefixed all sequences with zeroes until they reached equal length; despite performing more computation, this allows the reduction of all the dot products involving $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{out}}$ over all time steps and samples in a batch to a single one each; this is particularly useful for implementations in interpreted languages such as Python and resulted in major speedups.

In all experiments, we used *rmsprop* with Nesterov momentum for optimisation. The step rate was set to 0.0001, the momentum to 0.9 and the decay to 0.9. We found these values to be fairly robust in all cases.

2 We deviate from the original formulation by using the L_2 norm instead of the theoretically more sound L_1 norm, since we found it to yield substantially better results. The authors of the original paper did so as well, according to their public code repository.

We conducted a fixed number of trials for each criterion on each data set allowing at most 15 passes over the training data. During training, we monitored the performance of the subsequent classification task by estimating the generalisation error via performing logistic regression either on validation data (if available) or stratified splits of the training data. The parameters which performed best were chosen for the final evaluation, in which we estimated another logistic regression model with the same hyper parameters on the whole training set. To make sure that a model trained on an objective does not really consist of a very good model found by random initialisation, we only include those trials where the classification performance improved during optimisation.[3]

3.5.2.1 *UCR Time Series*

The UCR time series archive (Keogh et al., 2006) provides a number of data sets to evaluate classification and clustering algorithms on. The data sets are all one dimensional and have a fixed length; methods such as dynamic time warping excel on these time series, while RNNs typically achieve less than state of the art results, even when trained discriminatively. Nevertheless, we chose to include the results since they give an intuition on how well the different criterions perform over a wide range of tasks. It is striking that no clear winner emerges, albeit it appears that the rectifier transfer function dominates the results. For all experiments, 32 hidden and 16 output units were used. Results are shown in Table 10.

3.5.2.2 *TIDIGITS*

We prepared the data in the same way as in Section 3.4.3.2. We chose to use 64 hidden and 32 output units. The results are summarised in Table 11.

3.5.2.3 *Characters*

The online handwritten character data set consists of 2858 time series of three variates: the x and y directions of the movement of a pen as well as its pressure against the table. Each sample represents one of 20 handwritten characters. While the data is not balanced, the variations are negligible. 64 hidden and 32 output units were used in all experiments. We selected a random stratified 20% of the data for testing and the rest for training. Results are summarised in Table 12.

3 Otherwise a good result from random initialisation would have been attributed to another method.

Data set	**Method**	**Error**	ρ	σ
Two Patterns	RI	0.352	stochastic	rectifier
	OSP	0.165	mean	rectifier
	SF	0.271	stochastic	softsign
	RIM	0.287	max	tanh
Beef	RI	0.6	mean	tanh
	OSP	0.63	mean	rectifier
	SF	0.13	stochastic	tanh
	RIM	0.26	stochastic	rectifier
CBF	RI	0.23	mean	rectifier
	OSP	0.29	mean	rectifier
	SF	0.14	mean	rectifier
	RIM	0.1	max	rectifier
Lighting 2	RI	0.38	mean	tanh
	OSP	0.36	mean	rectifier
	SF	0.31	stochastic	tanh
	RIM	0.34	mean	rectifier
Lighting 7	RI	0.53	mean	rectifier
	OSP	0.6	mean	rectifier
	SF	0.42	stochastic	rectifier
	RIM	0.45	max	rectifier
ECG	RI	0.29	mean	rectifier
	OSP	0.34	stochastic	rectifier
	SF	0.24	stochastic	tanh
	RIM	0.23	max	rectifier

Table 10: Test errors of our methods on a subset of the UCR data base. The method with the best test error is shown in bold face.

Method	**Error**	ρ	σ
SF	0.167	mean	rectifier
RI	0.0637	stochastic	rectifier
OSP	0.0699	mean	softplus
RIM	0.1434	mean	softplus
NCA (Section 3.4)	0.021	max	LSTM
CE (Section 3.4)	0.079	max	LSTM

Table 11: Test errors of our methods on TIDIGITS compared with two supervised approaches, RNNs with LSTM units trained on the neighbourhood component analysis and the cross entropy objectives.

Method	**Error**	ρ	σ
RI	0.014	max	rectifier
OSP	0.017	stochastic	rectifier
SF	0.033	mean	rectifier
RIM	0.025	sum	tanh
Fisher Kernels	0.0446		
Discriminative Fisher Kernels	0.0326		

Table 12: Test errors of our methods on the Characters data set. We also mention other results due to van der Maaten (2011a), following a slightly different protocol: instead of our fixed train/test split with cross validation performed on the train split only, cross validation on the whole data set is performed in their case. Yet, our embedding dimensionality is much greater, 16 compared to 5 and 10.

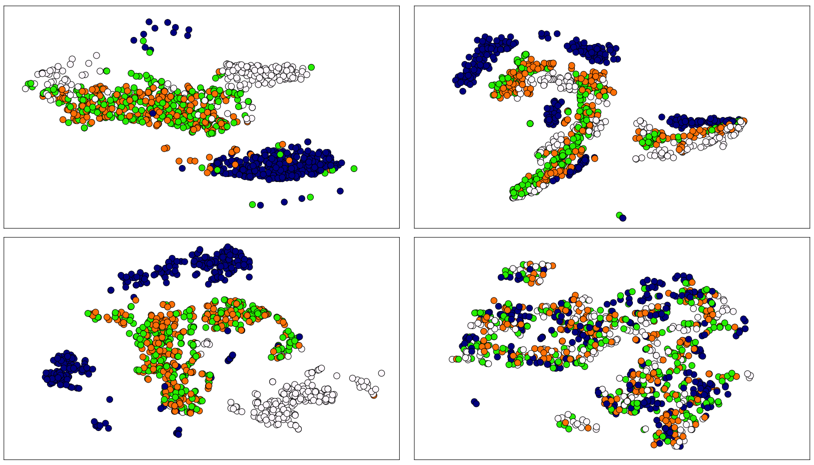
3.5.3 *Discussion*

Perhaps the most striking result of this work is that the randomly initialised networks perform unexpectedly good. Put even more drastically, for two data sets *TIDIGITS* and *Characters*, all the objectives we evaluated seem to actually hurt performance. Nevertheless, random initialisation fails on certain tasks on the UCR time series where other methods reach good performance. Consequently, no clear winner emerges from this zoo of methods; this is backed up by visualisations of the found embeddings, where different methods lead to different qualities of separation of the data. We show plots of the embeddings found on the TwoPatterns and CBF data sets from the UCR time series archive in Figure 9. We also show the features found by random initialisation in the TIDIGITS data set in Figure 10. While the performance is quite different, the t-SNE visualisations found by the other methods do not substantially differ.

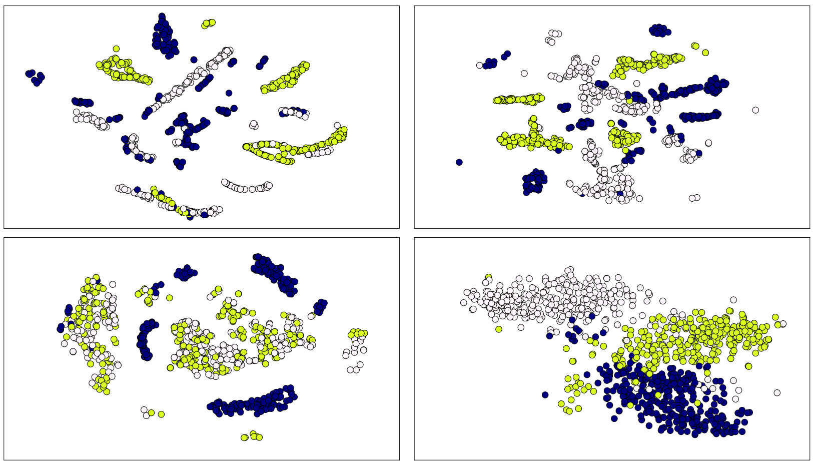
SEQUENTIAL JACOBIAN The sequential Jacobian is a mean of inspecting the sensitivity of certain outputs of an RNN with respect to its inputs (Graves et al., 2008). Given we want to find out what a specific dimension in our embedding space is responsible for, we can compute $\frac{\partial e_i}{\partial x_t}$ for all inputs of interest. In general, this will be a function of the parameters of the networks as well as the current input sequence.

We show several representative sequential Jacobians in Figure 11.

NEAREST NEIGHBOURS The notion of similarity is best qualitatively evaluated by inspecting examples which lie close together in



(a)



(b)

Figure 9: Plots of the embeddings found after reducing the dimensionality with t-SNE (van der Maaten and Hinton, 2008): RI-RNN, OSP-RNN, RIM-RNN and SF-RNN (clockwise) on the (a) TwoPatterns and (b) CBF data sets from the UCR time series archive. The plots illustrate that quite some variation is present within the different embeddings found and a sensitivity to different phenomena in the data seems apparent which diversifies the embeddings. The performance of the methods is also quite varying; in the case of RIM-RNN, practically no useful structure is found for the classification of TwoPatterns. With CBF however, it very nicely recovers a plausible clustering of the data.

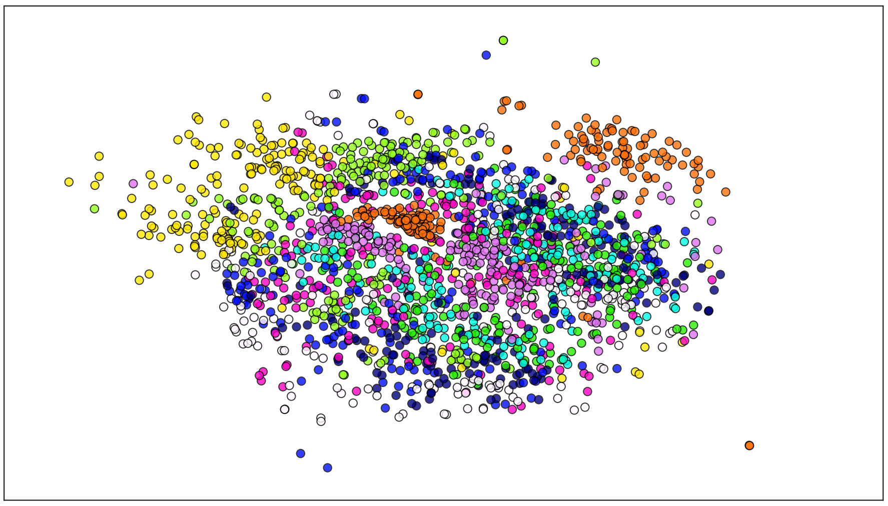


Figure 10: Visualisation of the embeddings found by a randomly initialised RNN on the TIDIGITS data set after reducing the dimensionality with t-SNE (van der Maaten and Hinton, 2008).

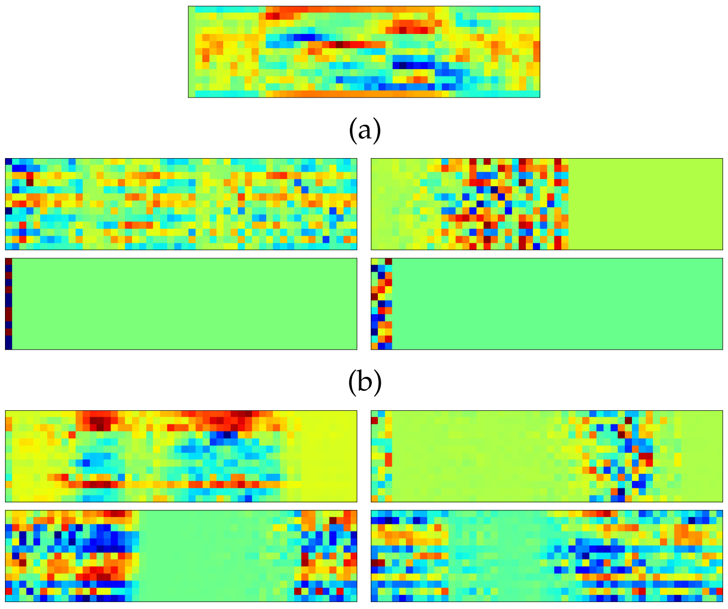


Figure 11: (a) A sample from the TIDIGITS data set. Sequential Jacobians of four features on that sample from (b) a randomly initialised RNN and (c) trained with one-step prediction. The former has two features with a very short attention span which is caused by the stochastic pooling function which allows sudden halts. Conversely, the latter is focusing its attention to distinct regions of the input. Thanks to the pooling we can see that both networks can distribute their attention over the whole input, thus reducing but not eliminating the problem of hard to learn long-term dependencies.

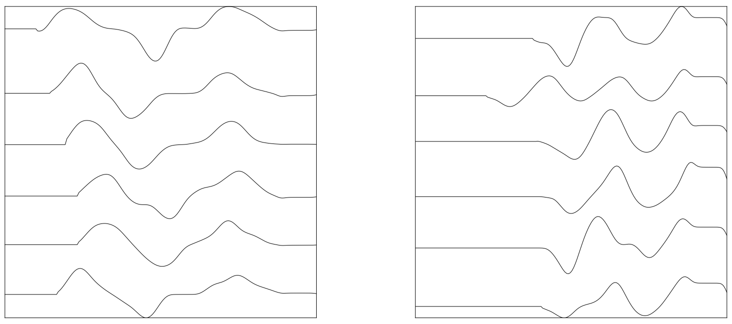


Figure 12: The first dimension of samples from the Characters data set. In both plots, the first row is a randomly selected sample; the remaining rows are its nearest neighbours. We randomly picked two samples and plotted one of the two spatial dimensions of the sample itself (top row) and those of its five nearest neighbours according to the Euclidean distance of the embeddings found by a Sparse Filtering RNN. Note that the similarity is preserved even though sequences are not perfectly correlated along time.

feature space. In Figure 12 we show one input dimension of close samples from the characters data set.

3.6 conclusion

This chapter has cast already existing ideas (Klapper-Rybicka et al., 2001) how to use objective functions tailored towards static data for sequences into a wider framework. For that we have adopted several objective functions and performed extensive empirical evaluations, covering qualitative and quantitative aspects. We have also explored the space of pooling operators, which lead to quite distinct learning performance depending on the task.

In the unsupervised case, we conclude that none of the proposed methods qualifies as superior for sequence reduction in an unsupervised setting. The experiments in Section 3.5 have shown that the success of a method can vary quite a bit over the range of data sets. In the presence of label information, i.e. in the supervised case, we have shown that reducing sequences to points can lead to much better classification performance than plain discriminative training of RNNs.

4 VARIATIONAL INFERENCE OF LATENT SEQUENCES

4.1 INTRODUCTION

One typical application for RNNs is to model probability distributions over sequences, i.e. $p(\mathbf{x}_{1:T})$. This is done by writing the distribution in cascade form,

The text in this section has appeared in parts previously in Bayer and Osendorfer (2014).

$$p(\mathbf{x}_{1:T}) = \prod_{t=0}^{T-1} p(x_{t+1}|\mathbf{x}_{1:t}),$$

where $\mathbf{x}_{1:0} = \emptyset$. Each $p(x_{t+1}|\mathbf{x}_{1:t})$ is then represented by the output of an RNN at a single time step, identifying each of its components with the statistics of the distribution. A simple example is that of a Bernoulli, i.e.

$$p(x_{t+1,c} = 1|\mathbf{x}_{1:t}) = \mathbf{y}_{t,c}^{(K)} \tag{28}$$

where $x_{t+1,c}$ corresponds to the c'th component of the $t+1$'th time step of $\mathbf{x}$ with $c = 1, \dots, \omega$ and $t = 1, \dots, T$. Each $\mathbf{y}_{t,c}(\mathbf{x}_{1:t})$ is the c'th output of some RNN at time step t, constrained to lie in the interval $(0, 1)$, e.g. by letting the output transfer function $f^{(K)}$ be the logistic sigmoid. Learning such an RNN then boils down to minimising the negative log-likelihood of the data with respect to the parameters of the network.

This framework gives practitioners a powerful tool to model rich probability distributions over sequences. A common simplification is a naïve Bayes assumption that the individual components factorise:

Practitioners typically assume the output variables to be independent; this is clearly wrong in many situations.

$$p(x_{t+1}|\mathbf{x}_{1:t}) = \prod_{c} p(x_{t+1,c}|\mathbf{x}_{1:t}).$$

While sufficient for many applications, reintroduction of dependency among the components of x_t leaves room for improvement. This is especially true for sequences over spaces which are high-dimensional and tightly coupled.

In this chapter, we propose to consider adding latent variables similar to Tang and Salakhutdinov (2013) to the network. Using SGVB (see the works of Rezende et al. (2014); Kingma and Welling (2013) and Section 1.4.3.2) as an estimator, we train RNNs to model high-dimensional sequences.

4.2 RELATED WORK

The approach taken by Graves (2013) is to use a mixture distribution for $p(x_t|\mathbf{x}_{1:t-1})$. Arguably powerful enough to model any dependency between the components of x_t, a drawback is that the number of parameters scales at least linearly with the number of chosen mixture components.

Models based on restricted Boltzmann machines and variations (Boulanger-Lewandowski et al., 2012, 2013; Sutskever et al., 2008) provide a solution to this as well, yet come with tighter restrictions on the assumptions that can be made. E.g. RBMs are restricted to model data using posteriors from the exponential family (Welling et al., 2004), make use of an intractable objective function and require costly MCMC steps for learning and sampling.

4.3 METHODS

The text in this section has appeared in parts previously in Bayer and Osendorfer (2014).

We propose to combine SGVB and RNNs by making use of an sRNN for both the recognition model $q(z_t|\mathbf{x}_{1:t-1})$ and the generating model $p(x_t|\mathbf{z}_{1:t})$.

4.3.1 *The Generating Model*

The generative model is an RNN passing over a sequence of standard normal samples.

More specifically, the generating model is an sRNN where the latent variables form additional inputs:

$$\mathbf{a}_t^{(1)} = \mathbf{x}_t \mathbf{W}_{\text{in}}^{(k)} + \mathbf{y}_{t-1}^{(1)} \mathbf{W}_{\text{rec}}^{(1)} + \mathbf{b}^{(1)} + z_t \mathbf{W}_{\text{in}}'^{g}, \tag{29}$$

which replaces Eq. (2) on page 8 for the first layer. We let the last layer $\mathbf{y}_t^{(K)}$ represent the necessary statistics to fully determine $p(x_{t+1}|\mathbf{x}_{1:t})$. For the remainder of the chapter, we let the hidden layers $\mathbf{y}_{1:T}^{(k)}, k = 1, \dots, K-1$ be denoted as a single sequence by $\mathbf{h}_{1:T}$.

Note that the model reduces to an sRNN as soon as we remove any latent variables, e.g. by setting $\mathbf{W}_{\text{in}}'^{g} = \mathbf{0}$. Hence, such a model generalises sRNNs.

The only quantities bearing uncertainty in the calculation of $\mathbf{h}_{1:T}$ are the latent variables $\mathbf{z}_{1:T}$, as $\mathbf{x}_{1:T}$ stems from the data set and for all t,

h_t is a deterministic function of $\mathbf{x}_{1:t}$ and $\mathbf{z}_{1:t}$. The resulting factorisation of the data likelihood of a single sequence $p(\mathbf{x}_{1:T})$ is then

$$\begin{aligned} p(\mathbf{x}_{1:T}) &= \prod_{t=0}^{T-1} p(x_{t+1}|\mathbf{x}_{1:t}) \\ &= \int_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}) \prod_{t=0}^{T-1} p(x_{t+1}|\mathbf{x}_{1:t}, \mathbf{z}_{1:t}, \cancel{\mathbf{z}_{t+1:T}}) d\mathbf{z}_{1:T} \\ &= \int_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}) \prod_{t=0}^{T-1} \int_{h_t} \big[p(x_{t+1}|\mathbf{x}_{1:t}, \mathbf{z}_{1:t}, h_t) \\ &\qquad\qquad p(h_t|\mathbf{x}_{1:t}, \mathbf{z}_{1:t}) \big] dh_t d\mathbf{z}_{1:T}, \end{aligned}$$

where we have made use of the fact that x_{t+1} is independent of $\mathbf{z}_{t+1:T}$. Since h_t is a deterministic function of $\mathbf{x}_{1:t}$ and $\mathbf{z}_{1:t}$, we note that $p(h_t|\mathbf{x}_{1:t}, \mathbf{z}_{1:t})$ follows a Dirac distribution with its mode given by $\mathbf{y}_t^{(K)}$.

Thus, the integral over the hidden states is replaced by a single point; we make the dependency of h_t on both $\mathbf{z}_{1:t}$ and $\mathbf{x}_{1:t}$ explicit.

$$p(\mathbf{x}_{1:T}) = \int_{\mathbf{z}_{1:T}} p(\mathbf{z}_{1:T}) \prod_{t=0}^{T-1} p(x_{t+1}|h_t(\mathbf{x}_{1:t}, \mathbf{z}_{1:t})) d\mathbf{z}_{1:T}. \qquad (30)$$

The corresponding graphical model is shown in Figure 13. Even though the determinism of h_t might seem restrictive at first, we will argue that it is not. First, note that the sequence of hidden states $\mathbf{h}_{1:T}$ is deterministic given $\mathbf{x}_{1:T}$ and $\mathbf{z}_{1:T}$ and consequently, $p(\mathbf{h}_{1:T}|\mathbf{x}_{1:T}, \mathbf{z}_{1:T})$ will follow a Dirac distribution. Marginalising out $\mathbf{z}_{1:T}$ will however lead to a universal approximator of probability distributions over sequences, analogously to the argument given in Section 1.4.3.2.

Priors independent over time are universal in the sense of representational power.

An additional consequence is that we can restrict ourselves to prior distributions over the latent variables that factorise over time steps, i.e. $p(\mathbf{z}_{1:T}) = \prod_t p(z_t)$. This is much easier to handle in practise, as calculating necessary quantities such as the KL-divergence can be done independently over all time steps and components of z_t.

Despite of this, the distribution over $\mathbf{h}_{1:T}$ will be a Markov chain and can exhibit stochastic behaviour, if necessary for modelling the data distribution.

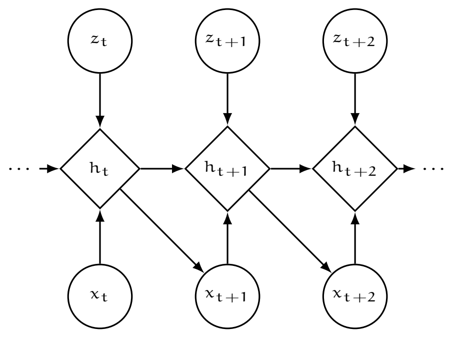


Figure 13: Graphical model corresponding to the factorisation given in Eq. (30). The hidden states h_t are shown as diamonds to stress that they are no source of stochasticity. Despite of this, marginalising out $\mathbf{z}_{1:T}$ makes $\mathbf{h}_{1:T}$ stochastic.

4.3.2 *Variational Inference for Latent State Sequences*

Applying VI to the latent variables leads to a training method for the model parameters.

The derivation of the training criterion is done by obtaining a variational upper bound on the negative log-likelihood via Jensen's inequality, where we use a variational approximation $q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}) \approx p(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})$. Let $q = q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T})$ be a shorthand, then:

$$\begin{aligned} &-\log p(\mathbf{x}_{1:T}) \\ =&-\log \int_{\mathbf{z}_{1:T}} \frac{q}{q} p(\mathbf{z}_{1:T}) \prod_{t=0}^{T-1} p(x_{t+1}|h_t(\mathbf{x}_{1:t}, \mathbf{z}_{1:t}))d\mathbf{z}_{1:T} \\ \leqslant& \mathrm{KL}(q|p(\mathbf{z}_{1:T})) - \mathbb{E}_{\mathbf{z}_{1:T}\sim q}[\sum_{t=0}^{T-1} \log p(x_t|h_{t-1}, \mathbf{z}_{1:t})] \\ :=& \mathcal{L}_{\mathrm{STORN}} \end{aligned} \tag{31}$$

In this work, we restrict ourselves to a standard Normal prior of the form

$$p(\mathbf{z}_{1:T}) = \prod_{t,i} \mathcal{N}(z_{t,i}|0,1),$$

where $z_{t,i}$ is the value of the i'th latent sequence at time step t.

The recognition model q will parameterise a single mean $\mu_{t,i}$ and variance $\sigma^2_{t,i}$ for each time step and latent sequence in this case. Both will be represented by the output of a recurrent net, which thus has 2ω outputs of which the first ω (representing the mean) will be unconstrained, while the second ω (representing the variance) need to be strictly positive. Given the output ${}^r\mathbf{y}_{1:T}$ of the recognition RNN ${}^r f$, we set

$$\begin{aligned} \mu_{t,i} &= {}^r y_{t,i}, \\ \sigma^2_{t,i} &= {}^r y^2_{t,i+\omega}. \end{aligned}$$

Note that the square ensures positiveness.

Going along with the reparameterisation trick of Kingma and Welling (2013), we will sample from a standard Normal at each time step, i.e. $\epsilon_{t,i} \sim \mathcal{N}(0,1)$ and use it to sample from q via $z_{t,i} = \mu_{t,i} + \sigma_{t,i}\epsilon_{t,i}$. Given the complete sample sequence $\mathbf{z}_{1:T}$ we calculate the two terms of Equation (31). The KL-divergence can be readily computed, while we need to pass $\mathbf{z}_{1:T}$ through the generating model ${}^g f$ which gives $-\log p(\mathbf{x}_{1:T}|\mathbf{z}_{1:T})$. The computational flow is illustrated in Figure 14.

4.3.3 *Comparison to RNNs*

STORNs generalise RNNs.

An important question is whether the proposed model offers any the-

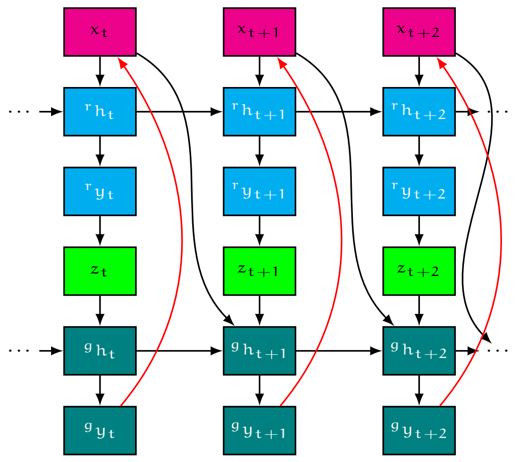


Figure 14: Diagram of the *computational* dependencies of STORNs. Each node of the graph corresponds to a vectorial quantity. The different types of nodes shown are data (magenta), the recognition model (cyan), samples (green) and the generating model (teal). Note that the outputs of the recognition model $^r y_t$ depict the statistics of $q(z_t|\mathbf{x}_{1:t})$, from which the sample z_t (green) is drawn. The output of the generating model, $^g y_t$ is used to represent $p(x_{t+1}|\mathbf{x}_{1:t})$. The red arrow expresses that this prediction is used to evaluate the loss, i.e. the negative log-likelihood.

oretical improvements over RNNs with no latent variables. The approximation capabilities (with respect to probability distributions) of RNNs result from the choice of likelihood function, i.e. the way the density of the observations at time step t is determined by the outputs of the network, y_t. See Eq. (28). We have argued in Section 4.1 that a naïve Bayes assumption reduces the approximation capabilities. One way to circumvent this is to use mixture distributions (Graves, 2013). The number of parameters of the latter scales poorly, though: linear in the number of modes, hidden units in the last layer and output dimensions.

Both approaches also share the drawback that the stochasticity entering the computation is not represented in the hidden layers: drawing a sample is determined by a random process invisible to the network.

STORN overcomes both of these issues. Introducing an additional mode merely requires an additional change of curvature in the approximation of F (compare Section 1.4.3.2). This can be obtained by additional hidden units, for which the number of parameters scales linearly in the number of hidden units in the incoming and outgoing layer. Further, the stochasticity in the network is stemming from z of which the hidden layer is a function.

4.4 EXPERIMENTS

For evaluation we trained the proposed model on a set of midi music, which was used previously (Bengio et al., 2012; Pascanu et al., 2013; Bayer et al., 2014; Boulanger-Lewandowski et al., 2012) to evaluate RNNs. We also investigated modelling human motion in the form of motion capture data (Boulanger-Lewandowski et al., 2012; Sutskever et al., 2008; Taylor et al., 2006). We employ RNNs with fast dropout (see Section 2.5) for both the recognition and the generating model. While we determine the dropout rates for the generating model via model selection on a validation set, we include them into the parameter set for the recognition model. In a manner similar to Bayer et al. (2013) (see Section 2.6.1), we exploit fast dropout's natural inclusion of variance as the variance for the recognition model, i.e. $\sigma^2_{t,k}$. We used Adadelta (Zeiler, 2012) enhanced with Nesterov momentum (Sutskever et al., 2013) for optimisation.

The text in this section has appeared in parts previously in Bayer and Osendorfer (2014).

4.4.1 *Polyphonic Music Generation*

All experiments were done by performing a random search (Bergstra and Bengio, 2012) over the hyper parameters, where 128 runs were performed for each data set. Both the recognition and the generating model used 300 hidden units with the logistic sigmoid as the transfer function. We report the estimated negative log-likelihood (obtained via the estimator proposed in (Rezende et al., 2014)) on the test set of the parameters which yielded the best bound on the validation set.

As expected, STORN improves over the models assuming a factorised output distribution (FD-RNN, sRNN, Deep RNN) in all cases. Still, RNN-NADE has a competitive edge throughout the experiments. The reasons for this remain unclear from the results alone, but the stochastic training and resulting noisy gradients are a viable hypothesis, since RNN-NADE does not suffer from those. Nevertheless, RNN-NADE requires an explicit factoristion of the output space $\mathcal{Z}$ and does not benefit from graphics processing units as much since this renders the computations less parallelisable. It is also tailored towards binary data–STORN works naturally with all conditional output distributions for which the likelihood function can be evaluated and differentiated efficiently. Consequently, we will move on to an experiment with continuous outputs in the next section.

The results are summarised in Table 13.

Table 13: Results on the midi data sets. All numbers are average negative log-likelihoods on the test set, where "FD-RNN" represents the work Section 2.5; "sRNN" and "RNN-NADE" results are due to Bengio et al. (2012) while "Deep RNN" shows the best results by Pascanu et al. (2013). The results of our work are shown as "STORN" and have been obtained by means of the importance sampler described by Rezende et al. (2014).

Data set	STORN	FD-RNN	sRNN	RNN-NADE	Deep RNN
Piano-midi.de	7.13	7.39	7.58	7.05	–
Nottingham	2.85	3.09	3.43	2.31	2.95
MuseData	6.16	6.75	6.99	5.60	6.59
JSBChorales	6.91	8.01	8.58	5.19	7.92

4.4.2 *Motion Capture Data*

The motion capture data set (Hsu et al., 2005; Taylor et al., 2006) is a sequence of kinematic quantities obtained from a human body during walking. It consists of 3128 time steps of 49 angular quantities each.

For motion capture data, we chose a Gaussian likelihood with a fixed standard deviation for the generating model. The recognition model was chosen to be a biRNN. While the standard deviation was fixed to 1 during training, we performed a binary search for a better value after training; the resulting estimate of the negative log-likelihood on the validation set was then used for model selection.

STORN achieves state-of-the-art results on the human motion modelling task, and is the first to feature an accurate estimate of the marginal likelihood.

DISCUSSION The estimated negative log-likelihood of the data was 15.99. Other models trained on this data set, namely the RNN-RBM, RTRBM and cRBM do not offer a tractable way of estimating the log-likelihood of the data, which is why there is no direct mean of comparison respecting the probabilistic nature of the models. In the case of the former two, the mean squared prediction error is reported instead, which is 20.1 and 16.2 respectively. Our method achieved an average MSE of 6.58, which is substantially better than previously reported results.

The results are summarised in Table 14.

The probabilistic nature of STORN missing value imputation and sampling.

For additional means of comparison, we performed approximate missing value imputation of motion capture data. We picked random sequences of length 60 and replaced all of the 49 channels from time steps 30 to 40 with standard normal noise. We then performed a *maximum a posteriori* point selection of the recognition model, i.e.

$$\operatorname*{argmax}_{\hat{\mathbf{z}}_{1:T}} q(\hat{\mathbf{z}}_{1:T}|\mathbf{x}_{1:T}),$$

Table 14: Results for RTRBM, RNN-RBM and STORN on the motion capture data. We report the *Mean Squared Error* (MSE) and *negative log-likelihood* (NLL). Lower is better.

	STORN	RNN-RBM	RT-RBM
MSE	6.58	20.1	16.2
NLL	15.99	–	–

from which we reconstructed the output via

$$\operatorname*{argmax}_{\hat{\mathbf{x}}_{30:40}} \log p(\mathbf{x}_{1:T}|\hat{\mathbf{z}}_{1:T}).$$

We then fed the reconstruction back into the model, repeating this iterative scheme ten times. The results of the imputations are shown in Figure 15.

To demonstrate the generative capabilities of the method, we drew 50 samples from the model after initialising it with a stimulus prefix. The stimulus had a length of 20, after which we ran the model in "generating mode" for another 80 time steps. This was done by feeding the mean of the model's output at time step t into the generating model at time step $t+1$. Additionally, we drew $\mathbf{z}_{20:80}$ from the prior. The results are visualised in Figure 16.

4.5 conclusion

The text in this section has appeared in parts previously in Bayer and Osendorfer (2014).

We have presented a model class of stochastic RNNs that can be trained with a recently proposed estimator, SGVB. The resulting model fulfils the expectation to greatly improve over the performance of sRNNs erroneously assuming a factorisation of the observed variables. An important take away message of this section is that the performance of RNNs can greatly benefit from more sophisticated methods that greatly improve the representative capabilities of the model.

While not shown here, STORNs can be readily extended to feature computationally more powerful architectures such as deep recurrent networks, LSTM or deep transition operators (Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013). This is mostly due to the fact that STORN generalises RNNs and therefore all its machinery can be transferred over. This is not the case for competitive methods such as the recurrent temporal RBM.

Further, STORNs can be applied in a multitude of ways, with complicated likelihood functions featuring discrete and continuous variables

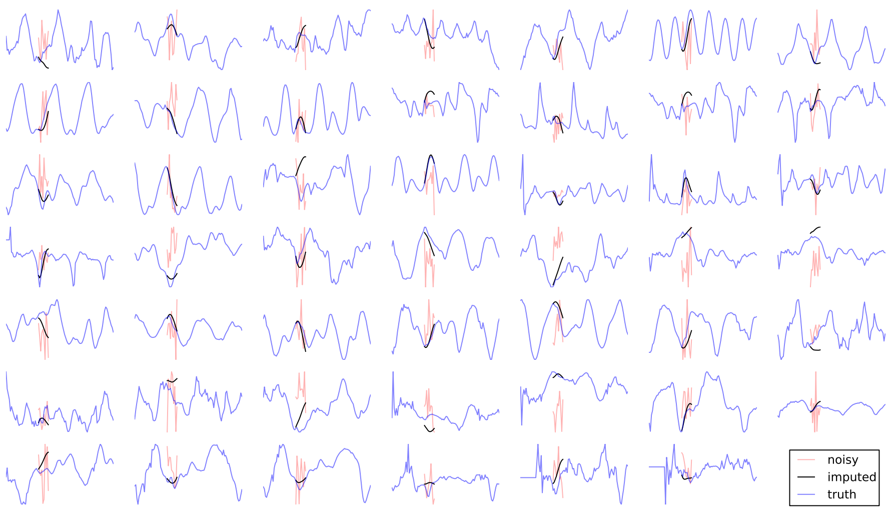


Figure 15: Illustration of missing value imputation on the motion capture data set. We show the first 48 of the 49 channels of a random sample, where time steps 30 to 40 were initialised with random noise. Subsequently, a maximum a posteriori point estimate of the latent variables was used to reconstruct the missing parts of the signals. Each of the plots corresponds to a different joint-angle of the human body during walking. STORN achieves reasonable reconstructions on most sensors.

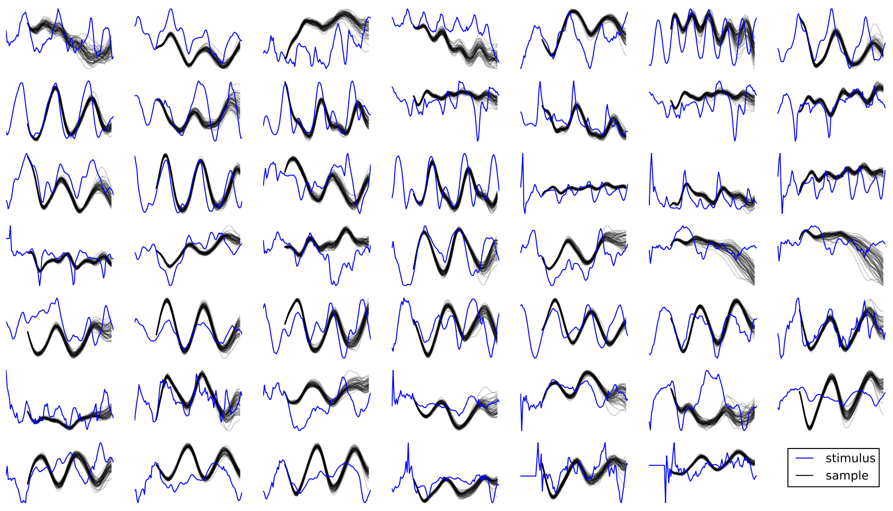


Figure 16: Samples from the model trained on motion capture data after providing a stimulus prefix sequence of 20 time steps. The uncertainty of the learned distribution is visible by the diversity of the samples; nevertheless, the distribution is rather uni-modal. The results clearly show that STORN can accurately predict the next few time steps, after which it mostly continues with a periodic movement and diverges only in a few cases.

of various distributions. It is therefore a good fit for a general model for sequences, especially as it inherits its generality from VAEs.

Still, an apparent weakness seems to be the stochasticity in the objective function, which is necessary for modelling highly complex distributions. STORN may very well benefit from advances in the area of unconstrained optimisation of stochastic loss functions.

5 CONCLUSION AND FUTURE WORK

Verteidige die Seele, das lustige Gebilde
Bis dahin: alle Energie auf die Reflektorschilde.

— Kettcar, "Ich danke der Academy".

This thesis has explored three quite different options of finding representations of sequences: i) extending the units of NNs from points to random variables summarised by their expectation and variance; ii) using RNNs to reduce sequences to fixed-length points; iii) finding probabilistic, latent sequences from data.

The first method, presented in chapter 2, has opened up numerous applications. We have first shown that in its original form, as proposed by Wang and Manning (2013), FD is a powerful regulariser for RNNs. We then used it to obtain predictive distributions with NNs, including the fast approximation of a Bayesian method. We also proposed a heuristic based on information theoretic quantities as an alternative to Bayesian learning. Future work includes finding a proper justification of that heuristic and the application of the proposed methods to RNNs.

The second contribution is the exploration of a wide framework, which appears promising in the supervised as well as in the unsupervised case. While the results obtained are not advancing the state of the art in terms of predictive performance, the general framework holds opportunities to find better components.

The third contribution has exploited very recent advances in VI. The results obtained for the prediction of human motion are among the best published. There is plenty of room for variations, such as the probabilistic assumptions as well as the architectures used.

BIBLIOGRAPHY

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. (Cited on page 15.)

David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012. (Cited on pages 18 and 39.)

J. Bayer, C. Osendorfer, and P. van der Smagt. Learning sequence neighbourhood metrics. In *ICANN 2012 - 22nd International Conference on Artificial Neural Networks*, pages 531–538. Springer, 2012. (Cited on pages 57, 59, and 61.)

Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014. (Cited on pages 21, 75, 76, 81, and 83.)

Justin Bayer, Daan Wierstra, Julian Togelius, and Jürgen Schmidhuber. Evolving memory cell structures for sequence learning. In *Artificial Neural Networks–ICANN 2009*, pages 755–764. Springer, 2009. (Cited on page 15.)

Justin Bayer, Christian Osendorfer, Sebastian Urban, et al. Training neural networks with implicit variance. In *Proceedings of the 20th International Conference on Neural Information Processing , ICONIP-2013*, 2013. (Cited on pages 36, 45, and 81.)

Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. In *Proceedings of the International Conference on Learning Representations*, 2014. (Cited on pages 34 and 81.)

Justin Bayer, Maximilian Karl, Daniela Korhammer, and Patrick van der Smagt. Fast adaptive weight noise. *arXiv preprint arXiv:1507.05331*, 2015. (Cited on pages 26, 31, and 45.)

Nikolaus D. Bayer. A comparison of fuzzy and neural technology in the context of fraud detection. Master's thesis, RWTH Aachen, 4 2002. (Cited on page 57.)

Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956. (Cited on page 7.)

Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. *arXiv preprint arXiv:1212.0901*, 2012. (Cited on pages 42, 81, and 82.)

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994. (Cited on page 11.)

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305, 2012. (Cited on pages 39, 50, 67, and 81.)

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. URL http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf. Oral Presentation. (Cited on pages 7 and 32.)

D Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370, 1994. (Cited on page 58.)

Christopher M Bishop. Mixture density networks. 1994. (Cited on page 45.)

Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006. (Cited on pages 3 and 46.)

Ake Björck. *Numerical methods for least squares problems*. Siam, 1996. (Cited on page 3.)

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015. (Cited on page 26.)

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*, 2014. (Cited on page 22.)

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012. (Cited on pages 39, 76, and 81.)

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. High-dimensional sequence transduction. In *ICASSP*, 2013. (Cited on pages 39 and 76.)

Arthur E Bryson and Walter F Denham. A steepest-ascent method for solving optimum programming problems. *Journal of Applied Mechanics*, 29(2):247–257, 1962. (Cited on page 6.)

Wray L Buntine and Andreas S Weigend. Bayesian back-propagation. *Complex systems*, 5(6):603–643, 1991. (Cited on page 26.)

Nutan Chen, Sebastian Urban, Christian Osendorfer, Justin Bayer, and Patrick van der Smagt. Estimating finger grip force from an image of the hand using convolutional neural networks and gaussian processes. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3137–3142. IEEE, 2014.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, (to appear), 2011. (Cited on page 59.)

Sreerupa Das, C Lee Giles, and Guo-Zheng Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, 1992. (Cited on page 14.)

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977. (Cited on page 20.)

Hubert L Dreyfus. The computational solution of optimal control problems with time lag. *Automatic Control, IEEE Transactions on*, 18 (4):383–385, 1973. (Cited on page 6.)

Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962. (Cited on page 6.)

Keogh E., X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR time series classification/clustering homepage. 2006. URL `http://www.cs.ucr.edu/~eamonn/time_series_data/`. (Cited on page 62.)

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2): 179–211, 1990. (Cited on page 8.)

Leonhard Euler. Methodus inveniendi lineas curvas maximi minimive proprietate gaudentes (appendix, de curvis elasticis). *Lausanne und Genf*, 1744, 1774. (Cited on page 6.)

Otto Fabius, Joost R van Amersfoort, and Diederik P Kingma. Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*, 2014. (Cited on page 58.)

C. F. Gauss. *Theoria combinationis observationum erroribus minimis obnoxiae (Theory of the combination of observations least subject to error)*. 1821. (Cited on page 1.)

Carl Friedrich Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. 1809. (Cited on page 1.)

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10): 2451–2471, 2000. (Cited on page 15.)

Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *The Journal of Machine Learning Research*, 3:115–143, 2003. (Cited on page 15.)

Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero-Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs? application to multiple-step ahead time series forecasting. 2003. (Cited on page 26.)

Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17*, pages 513–520. MIT Press, 2004. (Cited on page 61.)

Ryan Gomes, Andreas Krause, and Pietro Perona. Discriminative clustering by regularized information maximization. *Advances in Neural Information Processing Systems*, 23:775–783, 2010. (Cited on page 66.)

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011. (Cited on pages 26, 46, 47, and 56.)

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. (Cited on pages 29, 39, 55, 65, 76, and 80.)

Alex Graves, Santiago Fernández, Marcus Liwicki, Horst Bunke, and Jurgen Schmidhuber. Unconstrained online handwriting recognition with recurrent neural networks. *Advances in Neural Information Processing Systems*, 20:1–8, 2008. (Cited on pages 13 and 70.)

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *arXiv preprint arXiv:1303.5778*, 2013. (Cited on pages 13 and 16.)

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. (Cited on page 14.)

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015. (Cited on page 15.)

Geoffrey Grimmett and David Stirzaker. *Probability and random processes*, volume 2. Oxford Univ Press, 1992. (Cited on pages 23 and 27.)

Peter D Grünwald. *The minimum description length principle*. MIT press, 2007. (Cited on pages 20 and 46.)

Barbara Hammer. On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1):107–123, 2000. (Cited on page 9.)

Donald Olding Hebb. *The organisation of behaviour: a neuropsychological theory*. Wiley, 1952. (Cited on page 1.)

Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989. (Cited on page 2.)

José Miguel Hernández-Lobato and Ryan P Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. *arXiv preprint arXiv:1502.05336*, 2015. (Cited on pages 26, 52, and 54.)

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993. (Cited on pages 20, 46, and 47.)

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *SCIENCE-NEW YORK THEN WASHINGTON-*, pages 1158–1158, 1995. (Cited on page 22.)

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. (Cited on page 29.)

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut für Informatik, Technische Universität, München*, 1991. (Cited on page 11.)

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (Cited on pages 13 and 83.)

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001. (Cited on page 11.)

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. (Cited on page 2.)

Eugene Hsu, Kari Pulli, and Jovan Popović. Style translation for human motion. *ACM Transactions on Graphics (TOG)*, 24(3):1082–1089, 2005. (Cited on page 82.)

Alekseĭ GrigorÊ¹evich Ivakhnenko and Valentin Grigorévich Lapa. *Cybernetic Predicting Devices*. CCM Information Corporation, 1965. (Cited on page 1.)

Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *In Advances in Neural Information Processing Systems 11*, pages 487–493. MIT Press, 1998. (Cited on page 58.)

Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004. (Cited on page 9.)

Herbert Jäger et al. Adaptive nonlinear system identification with echo state networks. *networks*, 8:9, 2003. (Cited on page 65.)

Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics, 1997. (Cited on page 28.)

Henry J Kelley. Gradient theory of optimal flight paths. *ARS Journal*, 30(10):947–954, 1960. (Cited on page 6.)

Eammon Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR time series classification/clustering homepage. 2006. URL http://www.cs.ucr.edu/~eamonn/time_series_data/. (Cited on page 68.)

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on pages 50 and 52.)

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. (Cited on pages 21, 75, and 79.)

Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *arXiv preprint arXiv:1506.02557*, 2015. (Cited on page 26.)

Magdalena Klapper-Rybicka, Nicol N. Schraudolph, and Jürgen Schmidhuber. Unsupervised learning in lstm recurrent neural networks. In *In*, pages 684–691. Springer-Verlag, 2001. (Cited on pages 58 and 73.)

Andrei Nikolaevich Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Amer. Math. Soc. Transl*, 28: 55–59, 1963. (Cited on page 2.)

Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. (Cited on page 57.)

Dominic Lakatos, Daniel Rüschen, Justin Bayer, Jörn Vogel, and Patrick van der Smagt. Identification of human limb stiffness in 5 dof and estimation via emg. In *Experimental Robotics*, pages 89–99. Springer, 2013.

Quoc V Le, Alex J Smola, and Stéphane Canu. Heteroscedastic gaussian process regression. In *Proceedings of the 22nd international conference on Machine learning*, pages 489–496. ACM, 2005. (Cited on pages 50 and 52.)

Adrien Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des cometes*. F. Didot, 1805. (Cited on page 1.)

G. W. Leibniz. Memoir using the chain rule (cited in TMME 7:2&3 p 321-332, 2010). 1676. (Cited on page 7.)

RG Leonard and G Doddington. Tidigits (ldc93s10). (Cited on page 63.)

G. F. A. L'Hôpital. *Analyse des infiniment petits, pour l'intelligence des lignes courbes*. Paris: L'Imprimerie Royale, 1696. (Cited on page 7.)

Lei Li and B Aditya Prakash. Time series clustering: Complex is simpler! In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 185–192, 2011. (Cited on page 58.)

S Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis, Department of Computer Science, University of Helsinki*, 1970. (Cited on page 6.)

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992. (Cited on page 56.)

David JC MacKay. Probable networks and plausible predictions-a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995. (Cited on page 26.)

J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. *Proc. 28th Int. Conf. on Machine Learning*, 2011. (Cited on page 13.)

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. (Cited on page 1.)

Marvin Minsky and Seymour Papert. Perceptrons. 1969. (Cited on page 1.)

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014. (Cited on page 22.)

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. (Cited on pages 1, 18, 19, and 20.)

Radford M Neal. Probabilistic inference using markov chain monte carlo methods. 1993. (Cited on pages 19 and 56.)

Jiquan Ngiam, Pang Wei Koh, Zhenghao Chen, Sonia Bhaskar, and Andrew Y Ng. Sparse filtering. *Advances in Neural Information Processing Systems*, 24:1125–1133, 2011. (Cited on page 66.)

Saahil Ognawala and Justin Bayer. Regularizing recurrent networks-on injected noise and norm-based methods. *arXiv preprint arXiv:1410.5684*, 2014.

Christian Osendorfer, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. Convolutional neural networks learn compact local image descriptors. In *Neural Information Processing*, pages 624–630. Springer, 2013a.

Christian Osendorfer, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. Unsupervised feature learning for low-level local image descriptors. *arXiv preprint arXiv:1301.2840*, 2013b.

Theophratus Paracelsus. De natura rerum. 1537. (Cited on page 1.)

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. Technical report, Technical Report, 2012. (Cited on pages 11, 12, and 55.)

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013. (Cited on pages 39, 42, 81, 82, and 83.)

Barak A Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989. (Cited on page 9.)

Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990. (Cited on page 58.)

Carl Edward Rasmussen. *Gaussian processes for machine learning*. Citeseer, 2006. (Cited on page 52.)

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and variational inference in deep latent gaussian models. *arXiv preprint arXiv:1401.4082*, 2014. (Cited on pages 21, 75, 81, and 82.)

Jorma Rissanen. Minimum-description-length principle. *Encyclopedia of statistical sciences*, 1985. (Cited on page 46.)

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 (6):386, 1958. (Cited on page 1.)

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088): 533–536, 1986. (Cited on pages 1 and 6.)

Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. 2007. (Cited on page 61.)

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE]. (Cited on page 7.)

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4 (1):131–139, 1992. (Cited on page 14.)

Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez. Training recurrent networks by evolino. *Neural computation*, 19(3):757–779, 2007. (Cited on page 9.)

Mike Schuster. Better Generative Models for Sequential Data Problems: Bidirectional Recurrent Mixture Density Networks. In *Neural Information Processing Systems*, pages 589–595, 1999. (Cited on page 15.)

Hava T Siegelmann and Eduardo D Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991. (Cited on page 9.)

Richard Socher, Eric H Huang, Jeffrey Pennington, Andrew Y Ng, and Christopher D Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. *Advances in Neural Information Processing Systems*, 24:801–809, 2011. (Cited on page 58.)

I. Sutskever, G. Hinton, and G. Taylor. The recurrent temporal restricted boltzmann machine. *Advances in Neural Information Processing Systems*, 21, 2008. (Cited on pages 76 and 81.)

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. 2013. (Cited on pages 12, 13, 38, and 81.)

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014. (Cited on pages 13 and 58.)

Yichuan Tang and Ruslan Salakhutdinov. A new learning algorithm for stochastic feedforward neural nets. 2013. (Cited on page 75.)

Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. Modeling human motion using binary latent variables. In *Advances in neural information processing systems*, pages 1345–1352, 2006. (Cited on pages 81 and 82.)

The GPy authors. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, 2012–2014. (Cited on pages 53 and 54.)

Sebastian Urban, Justin Bayer, Christian Osendorfer, Goran Westling, Benoni B Edin, and Patrick van der Smagt. Computing grip force and torque from finger nail images using gaussian processes. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4034–4039. IEEE, 2013.

L. van der Maaten. Learning discriminative fisher kernels. 2011a. (Cited on page 70.)

L. van der Maaten and G. Hinton. Visualizing data using t-sne. *The Journal of Machine Learning Research*, 9(2579-2605):85, 2008. (Cited on pages 64, 71, and 72.)

Laurens van der Maaten. Learning discriminative fisher kernels. In *Proceedings of the 28th International Conference on Machine Learning*, 2011b. (Cited on page 58.)

Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2000. (Cited on page 17.)

Robert Vautard, Pascal Yiou, and Michael Ghil. Singular-spectrum analysis: A toolkit for short, noisy chaotic signals. *Physica D: Nonlinear Phenomena*, 58(1):95–126, 1992. (Cited on page 58.)

Jörn Vogel, Justin Bayer, and Patrick van der Smagt. Continuous robot control using surface electromyography of atrophic muscles. In *International Conference on Intelligent Robots and Systems*, 2013.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013. (Cited on pages 29 and 55.)

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126, 2013. (Cited on pages 25, 26, 27, 28, 29, 30, 34, 56, and 87.)

Max Welling, Michal Rosen-Zvi, and Geoffrey E Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems*, pages 1481–1488, 2004. (Cited on page 76.)

P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pages 762–770, 1981. (Cited on page 6.)

Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974. (Cited on page 6.)

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. (Cited on page 9.)

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. (Cited on page 9.)

SJ Wright and J Nocedal. *Numerical optimization*, volume 2. Springer New York, 1999. (Cited on page 2.)

Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. (Cited on pages 38 and 81.)