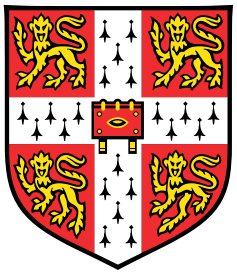
Biological applications, visualizations, and extensions of the long short-term memory network



Jos van der Westhuizen

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Christ's College

December 2018

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains no more than 64,782 words including appendices, bibliography, footnotes, tables and equations and has fewer than 36 figures.

Jos van der Westhuizen
December 2018

Acknowledgements

I would like to thank the members of the breakfast club – Sammy Mahdi, Kelly Marchisio, Alex Johnston, Jonathan Lynn, and Stratos Markou. Our interesting dinner conversations at Upperhall provided those essential daily social breaks. I especially thank Stratos for all our fun missions, coding projects, cinema nights, and coffee walks – one of the people I learned the most from during my time in Cambridge.

Members of the SigProc group cannot be thanked enough. Any mathematical problem could be solved between Mahed Abroshan and Marina Riabiz, and any programming problem between Rich Wareham and Adam Creig. Thank you to Kuan Hsieh for your frequent interesting questions and comments, James Li for our math brainstorms, Oliver Bonner for your proficient use of the English language, Jacob Vorstrup for your interesting conversations and mathematical prowess, Amarjot Singh for your radical ideas, and Ehsan Asadi for your unconditional everyday friendliness. I thoroughly enjoyed the conversations with everyone over coffee and cake, and the general cheerful spirit in the office.

A big special thanks to Fergal Cotter without whom the PhD would most likely have become too lonely to complete. Fergal provided lots of computer-, writing-, and mathematical-help and has an inspirational capability of thoroughly understanding new concepts. We started our machine learning PhDs together and he was the person I could talk to about the inevitable struggles of completing the doctorate. Thanks for letting me run experiments on "your" GPUs for most of the time. Our coding nights were lots of fun, and although we had some difficulties with Vim, your Ubuntu and technical skills helped speed up progress a lot, thanks! Without you, completing my thesis would have been much more difficult.

My collaborators at CBAS are thanked for their enthusiasm for machine learning. I thank my academic collaborators, David Janz for your impeccable programming skills, Brooks Paige and Matt J. Kusner for showing me what a good paper looks like, José Miguel Hernández-Lobato for your frequent inspiring ideas and guidance. This thesis required some medical knowledge that I was fortunate to learn from my brilliant medical collaborators: Cristine Sortica, Peter Smielewski, Vadir Baktash, and Steve Foulkes – I cannot thank you enough. I thank my proofreaders: Tris Edwards, Fergal Cotter, Jacob Vorstrup, Cordelia

O'Connell. Also my entrepreneurial colleagues, Emil Hewage, Joned Sarwar, and Aleksi Tukiainen for helping me satisfy my startup-hunger during the PhD.

Thanks to my friends back in South Africa, who started asking when I finish my PhD after my first year abroad. Thanks to my loving mother, inspiring father, and smarter younger brother for their support, diligent proofreading, and funny Skype sessions.

Most important of all, I thank one of the friendliest people in academia, my supervisor Joan Lasenby. Joan would always approach you with an infectious smile. She is supportive, brilliantly intelligent, compassionate, relaxed, and an excellent writing critic. I probably owe most of the joy in my PhD research to her perfect supervision approach. Academia (possibly the world) should have more people like Joan.

This research was funded by the Skye Cambridge Trust. I thank Christ's College and the Cambridge engineering department for generously sponsoring me to attend conferences related to the work in this thesis.

Abstract

Sequences are ubiquitous in the domain of biology. One of the current best machine learning techniques for analysing sequences is the long short-term memory (LSTM) network. Owing to significant barriers to adoption in biology, focussed efforts are required to realize the use of LSTMs in practice. Thus, the aim of this work is to improve the state of LSTMs for biology, and we focus on biological tasks pertaining to physiological signals, peripheral neural signals, and molecules. This goal drives the three subplots in this thesis: biological applications, visualizations, and extensions.

We start by demonstrating the utility of LSTMs for biological applications. On two new physiological-signal datasets, LSTMs were found to outperform hidden Markov models. LSTM-based models, implemented by other researchers, also constituted the majority of the best performing approaches on publicly available medical datasets. However, even if these models achieve the best performance on such datasets, their adoption will be limited if they fail to indicate when they are likely mistaken. Thus, we demonstrate on medical data that it is straightforward to use LSTMs in a Bayesian framework via dropout, providing model predictions with corresponding uncertainty estimates.

Another dataset used to show the utility of LSTMs is a novel collection of peripheral neural signals. Manual labelling of this dataset is prohibitively expensive, and as a remedy, we propose a sequence-to-sequence model regularized by Wasserstein adversarial networks. The results indicate that the proposed model is able to infer which actions a subject performed based on its peripheral neural signals with reasonable accuracy.

As these LSTMs achieve state-of-the-art performance on many biological datasets, one of the main concerns for their practical adoption is their interpretability. We explore various visualization techniques for LSTMs applied to continuous-valued medical time series and find that learning a mask to optimally delete information in the input provides useful interpretations. Furthermore, we find that the input features looked for by the LSTM align well with medical theory.

For many applications, extensions of the LSTM can provide enhanced suitability. One such application is drug discovery – another important aspect of biology. Deep learning can aid drug discovery by means of generative models, but they often produce invalid molecules

due to their complex discrete structures. As a solution, we propose a version of active learning that leverages the sequential nature of the LSTM along with its Bayesian capabilities. This approach enables efficient learning of the grammar that governs the generation of discrete-valued sequences such as molecules. Efficiency is achieved by reducing the search space from one over sequences to one over the set of possible elements at each time step – a much smaller space.

Having demonstrated the suitability of LSTMs for biological applications, we seek a hardware efficient implementation. Given the success of the gated recurrent unit (GRU), which has two gates, a natural question is whether any of the LSTM gates are redundant. Research has shown that the forget gate is one of the most important gates in the LSTM. Hence, we propose a forget-gate-only version of the LSTM – the JANET – which outperforms both the LSTM and some of the best contemporary models on benchmark datasets, while also reducing computational cost.

Publications

A list of publications relevant to the work in this thesis:

Chapter 3

Jos van der Westhuizen and Joan Lasenby (2016). A review of machine learning applied to medical time series. *Technical Report*. CUEDF-INFENG, TR-702, Cambridge.

Jos van der Westhuizen and Joan Lasenby (2016). Modelling Physiological Time Series with Sequential Machine Learning. Oral at *MEIbioeng*. Oxford, United Kingdom.

Jos van der Westhuizen and Joan Lasenby (2016). Combining Sequential Deep Learning and Variational Bayes for Semi-Supervised Inference. In *Neural Information Processing Systems: Bayesian Deep Learning Workshop*. Barcelona, Spain.

Chapter 4

Jos van der Westhuizen, Tris Edwards, Raphael Schmetterling, Robert Tinn, Oliver Armitage, Joan Lasenby, Emil Hewage (2017). Using adversarial autoencoders to infer actions from the peripheral nervous system. In *Neural Information Processing Systems: Learning with Limited Data Workshop*. Long Beach, United States.

Chapter 5

Jos van der Westhuizen and Joan Lasenby (2018). Visualization techniques for LSTMs applied to electrocardiograms. In *International Conference on Machine Learning: Workshop on Human Interpretability in Machine Learning.* Stockholm, Sweden.

Chapter 6

David Janz, Jos van der Westhuizen, José Miguel Hernández-Lobato (2017). Actively Learning What Makes a Discrete Sequence Valid. In *International Conference on Machine Learning: Principled Approaches to Deep Learning Workshop*. Sydney, Australia.[1]

David Janz, Jos van der Westhuizen, Brooks Paige, Matt Kusner, José Miguel Hernández-Lobato (2018). Learning a generative model for validity in complex discrete structures. In *International Conference on Learning Representations*. Vancouver, Canada.

[1] Authors contributed equally

Chapter 7

Jos van der Westhuizen and Joan Lasenby (2018). The unreasonable effectiveness of the forget gate. *arXiv:1804.04849[cs,stat]*.

Table of contents

List of figures

List of tables

Nomenclature

Numbers and Arrays

a	A scalar
$\mathbf{a}$	A vector
$\mathbf{A}$	A matrix
a_i	Element i of vector $\mathbf{a}$, with index starting at 1
$\mathbf{a}_{-i}$	All elements of vector $\mathbf{a}$ except for element i
$\mathbf{a}_t$	A single vector at time step t of the sequence of vectors $\mathbf{a}_{1:T}$
$\mathbf{x}^{(n)}$	The n-th example (input) from a dataset. We omit this superscript whenever it is clear that we are referring to terms associated with a single data point.
$\mathbf{e}^{(k)}$	Standard basis vector $[0,\dots,0,1,0,\dots,0]$ with a 1 at position k

Functions

$c[\cdot]$	Concatenation function
$\odot$	Element-wise product (Hadamard product)
$\mathbb{I}(cond)$	Indicator function which takes value 1 if $cond$ is true and value 0 otherwise.
$\mathrm{KL}(p \,\|\|\, q)$	Kullback-Leibler divergence between distributions p and q
$\log x$	Natural logarithm of x
$\|\mathbf{x}\|_p$	The L_p norm: $\|\mathbf{x}\|_p = (\sum_i \|x_i\|^p)^{\frac{1}{p}}$
$\|\mathcal{X}\|$	The cardinality of the set $\mathcal{X}$
$\sigma(x)$	The sigmoid (logistic) function: $\frac{1}{1+\exp(-x)}$

Chapter 1

Introduction

Sequences exist everywhere; the winning numbers in the lottery, the moves made in chess, the letters in a word, the words in a sentence, the questions in Alan Turing's "imitation game" (Turing, 1950), the pixels in an image, the atoms in a molecule, and the pulses of your heartbeat. Biology, in particular, is a domain laden with sequences where data range from DNA sequences to the medical notes made by a clinician.

In many cases, such biological sequences are best analysed by machine learning techniques. In recent years there has been a surge of unprecedented advances in the field of machine learning (Krizhevsky et al., 2012; Lanchantin et al., 2017; Sutskever et al., 2014; Zhou and Troyanskaya, 2015). This progress is mainly due to a collection of new techniques, referred to as deep learning, that have demonstrated gains over existing best-in-class machine learning algorithms across several fields. To highlight a few success stories, deep learning surpasses human performance in classifying images (He et al., 2015), recognizing speech (Saon et al., 2017; Xiong et al., 2016), playing Go (one of the most challenging classic games for artificial intelligence) (Silver et al., 2017), and deep learning outperforms cardiologists at classifying heartbeat arrhythmias (Rajkomar et al., 2018).

Simple machine learning algorithms can solve many machine learning problems if they are provided with the right set of features extracted from the data. However, for many tasks, it is difficult to determine what the *right* set of features is. One solution to this problem is to use machine learning to discover not only the mapping from features to output but also the features themselves. Deep learning techniques are especially suited to leveraging large amounts of data for finding these representations (features) and often provide novel insights. In a famous example, a neural network was shown to discover that pedestrians, faces, and cats were important components of online videos, without being instructed to look for them (Le, 2013). Naturally, researchers would like to know whether deep learning could solve the

challenges in biology by identifying the "cats" hidden in the data – the patterns unknown to researchers – and suggest ways to act on them.

This ability to learn effective representations promises that deep learning will equal or surpass the predictive performance of traditional methods that require hand-engineered features (Ching et al., 2018; Farabet et al., 2013; Lipton et al., 2015a). Although this increased performance is promised, it remains unrealized for many applications, as we show later. Thus, a continuing theme in this thesis is the search for a competitive implementation that does automatic feature extraction. An added bonus of this ability is a model capable of adapting to new tasks with minimal human intervention. In turn, it could ease the preprocessing of datasets, which is widely held to consume 80% of the effort in a machine learning approach (Rajkomar et al., 2018). Moreover, manually designing features for a complex task requires a great deal of human time and effort and it can take decades for an entire community of researchers to find the optimal design (Goodfellow et al., 2016).

When analysing sequences, techniques modified to better handle sequential data yield better results than those treating the input as static data (Längkvist et al., 2014). A deep learning technique tailored for sequence analysis is the *long short-term memory* (LSTM) network. The characteristic that makes LSTMs rise above their machine learning counterparts is their ability to learn and carry out complicated transformations of the entries in long sequences (Graves and Jaitly, 2014). Being a deep learning technique, LSTMs automatically learn effective representations and have been shown to outperform other models on raw data for speech recognition, minimizing the need for feature engineering (Lipton et al., 2015b).

LSTMs have yielded state-of-the-art results for many problems, such as image captioning (Vinyals et al., 2015), natural language processing (Sutskever et al., 2014), and handwriting recognition (Graves et al., 2009). Owing to ever-growing datasets, LSTMs have also become more applicable in biology and medicine (Ching et al., 2018). Although their outstanding performance in many fields has led to widespread adoption, the field of biology often poses greater barriers to practical implementation and therefore requires focussed efforts.

Because LSTMs are difficult to realize in biology, the aim of this thesis is to improve the current state of the LSTM for biological tasks. This goal drives the three subplots of this thesis. First, is to demonstrate the efficacy of LSTMs in **biological applications** – something we focus on during the earlier parts of this work and a theme persisting throughout the thesis. The inability of models to indicate when they are uncertain can limit their adoption or cause serious accidents, thus we also demonstrate this capability of LSTMs. Second, for many biological tasks, adopting LSTMs requires an understanding of the model decision-making process (e.g., the risk-averse nature of clinical decision means interpretability is paramount) and permitting such interpretations could lead to novel insights of the underlying

biological processes. We pursue this interpretability via **visualizations** of salient input features. Third, for many biological applications, **extensions** (modifications) of the LSTM are often desired for enhanced suitability. In other words, tractability, improved accuracy, and hardware efficiency are sought. Lastly, biology subsumes many categories; here we group physiological signals, peripheral neural signals, and molecules (for drug discovery) under biological data.

1.1 Thesis outline

This thesis is structured as follows. In chapter 2 we provide the required preliminaries on machine learning and LSTMs for this thesis. The body of the thesis is then divided into three main categories.

Biological applications In chapter 3 we apply LSTMs to medical datasets and show that they perform better than hidden Markov models. Here we also show that LSTMs can easily be used in a Bayesian framework to provide uncertainty measures of their predictions. A different flavour of medical applications is provided in chapter 4. Here we infer the actions of a specimen from their peripheral neural signals, using a sequence-to-sequence model that is regularized by Wasserstein adversarial networks.

Visualizations In chapter 5 we explore multiple visualization techniques for LSTMs in order to find the best. On an electrocardiogram dataset, we also determine the alignment between the found salient input features and medical theory.

Extensions In chapter 6 we demonstrate that LSTMs enable a type of active learning that allows efficient learning of the grammar of discrete-valued sequences. We demonstrate this approach on Python 3 expressions and molecules and show that it bolsters generative models. In chapter 7 we propose a forget-gate-only version of the LSTM, the JANET, which provides both computational and accuracy gains.

Finally, chapter 8 provides conclusions and recommendations for future work.

Chapter 2

Background

To make this thesis relatively self-contained, this chapter provides the necessary background on machine learning and long short-term memory (LSTM) networks.

2.1 The objective

Machine learning is useful whenever we want a computer to perform a task so intricate that it cannot be programmed by conventional means. Most of the machine learning tasks explored in this thesis are of the supervised learning form. To describe supervised learning, let p_{data} be a data generating distribution over $\mathcal{X} \times \mathcal{Y}$, where inputs $\mathbf{x} \in \mathcal{X}$ have corresponding (possibly noisy) outputs $\mathbf{y} \in \mathcal{Y}$. The goal of supervised learning is to use a **training set** consisting of N independent and identically distributed samples[1] $\{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{N} \sim p_{\text{data}}$, in order to find a function $f : \mathcal{X} \to \mathcal{Y}$ that eventually minimizes the error for the unseen data points in the **test set**. This optimal function is given by

$$f^* = \underset{f}{\operatorname{argmin}} \, \mathbb{E}_{\mathbf{x},\mathbf{y} \sim p_{\text{data}}}[L(f(\mathbf{x}), \mathbf{y})], \tag{2.1}$$

where $L(f(\mathbf{x}), \mathbf{y})$ measures the error incurred when predicting $\mathbf{y}$ as $f(\mathbf{x})$. In this work, parametric models resemble different functions via different sets of model parameters, θ. These models define a distribution $p_{\text{model}}(\mathbf{y}|\mathbf{x}, \theta)$, which estimates the true distribution $p_{\text{data}}(\mathbf{y}|\mathbf{x})$.

[1] We omit the superscript whenever it is clear that we are referring to terms associated with a single data point.

To find the parameters that provide the most accurate model, we can maximise the likelihood estimate of the parameters

$$\begin{aligned}\theta^* &= \underset{\theta}{\operatorname{argmax}} \prod_{n=1}^{N} p_{\text{model}}(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \theta) \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log\Big(p_{\text{model}}(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \theta)\Big) \\ &= \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\mathbf{y}|\mathbf{x}, \theta)], \end{aligned} \tag{2.2}$$

where $\hat{p}_{\text{data}}$ is the empirical distribution defined by the training data. We can change this into a minimization problem by taking the negative of the expectation. The objective function[2], which measures how well a machine learning model performs a task, then becomes

$$J(\theta) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\mathbf{y}|\mathbf{x}, \theta)]. \tag{2.3}$$

Unless otherwise stated, models in this thesis use this objective function, which is known as the negative log-likelihood and is the cross-entropy between the empirical distribution and the distribution defined by the model. Owing to the distinction between supervised and unsupervised learning being informal (Goodfellow et al., 2016, §5.8), we detail the task and the objective function when they deviate from the explanations above. Before proceeding to explain models, we detail how the training data can be used to optimize the model.

2.1.1 Gradient-based optimization

Here, optimization refers to the task of minimizing the objective function $J(\theta)$ by altering θ. The derivative $\frac{\mathrm{d}h(x)}{\mathrm{d}x}$ provides the slope of the function $h(x)$ at the point x, i.e., how to scale a small change in x to get the corresponding change in the function output. The **gradient** generalizes the notion of derivative to the case where the derivative is with respect to a vector: the gradient of h is the vector containing all of the partial derivatives, denoted $\nabla_{\mathbf{x}} h(\mathbf{x})$. For the function h the gradient points directly uphill, and the negative of the gradient points directly downhill. Thus, a minimum in $J(\theta)$ can be found by moving in the direction of the negative gradient. This is known as gradient descent which proposes new parameters

$$\theta' = \theta - \eta \nabla_\theta J(\theta), \tag{2.4}$$

[2]Also known as the error function, loss function, or cost function.

where the scalar η is the size of the iterative steps taken in the direction of the negative gradient, known as the **learning rate**, and θ' indicates updated parameters at each iteration. Minima of the objective function are located where the gradients are zero, and when the full solution is intractable, gradient descent is useful for iteratively approaching these points.

Gradient descent computes $\nabla_\theta J(\theta)$ on the entire training dataset, which can become prohibitively expensive for a large number of data points. Fortunately, stochastic gradient descent provides an efficient alternative by approximating the gradient

$$\nabla_\theta J(\theta) \approx \mathbf{g} = \frac{1}{K}\sum_{k=1}^{K} \nabla_\theta L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \theta), \tag{2.5}$$

on a randomly chosen K data points in the training set, known as a **minibatch**. Hence, new parameters are proposed by

$$\theta' = \theta - \eta \mathbf{g}. \tag{2.6}$$

We refer to an **epoch** as the number of iterations required for stochastic gradient descent to see all the data points in the training dataset.

Many variations of stochastic gradient descent have been proposed to improve convergence rates. These include the momentum (Polyak, 1964), Nesterov momentum (Sutskever et al., 2013), AdaGrad (Duchi et al., 2011), YellowFin (Zhang et al., 2017), and AMSGrad (Reddi et al., 2018) variations. Many of these methods effectively adapt the learning rate during training because the learning rate has a significant impact on the optimizer performance and is, therefore, difficult to set (Goodfellow et al., 2016, §8.5). Two such optimizers with per-parameter adaptive learning rates are used in this thesis, namely *RMSProp* (Hinton et al., 2012) and *Adam* (Kingma and Ba, 2015).

RMSProp accumulates an exponentially decaying average of the history of squared gradients to scale the learning rate. Using the approximate gradient $\mathbf{g}$, the parameter updates are given by

$$\begin{aligned} \mathbf{r}' &= \rho \mathbf{r} + (1-\rho)\mathbf{g} \odot \mathbf{g} \\ \theta' &= \theta - \frac{\eta}{\sqrt{\mathbf{r}' + \varepsilon}} \odot \mathbf{g}, \end{aligned} \tag{2.7}$$

where $\odot$ is the element-wise (Hadamard) product, $\frac{\eta}{\sqrt{\mathbf{r}+\varepsilon}}$ is applied element-wise, $\varepsilon = 1 \times 10^{-10}$ is used for numerical stability, and primed variables denote the updated values at each iteration. Our implementations use a decay rate $\rho = 0.9$.

Adam, derived from the phrase "adaptive moments", is the optimization technique we use most frequently. Adam could be seen as a variation of RMSProp combined with momentum

(computed by $\mathbf{s}$ in the equation below). Additionally, Adam includes bias corrections for the first-order and the second-order moments to account for their initialization. Using the approximate gradient $\mathbf{g}$, the parameter updates are given by

$$\begin{aligned}
\mathbf{s}' &= \beta_1 \mathbf{s} + (1-\beta_1)\mathbf{g} \\
\mathbf{r}' &= \beta_2 \mathbf{r} + (1-\beta_2)\mathbf{g} \odot \mathbf{g} \\
\hat{\mathbf{s}} &= \frac{\mathbf{s}'}{1-\beta_1^t} \\
\hat{\mathbf{r}} &= \frac{\mathbf{r}'}{1-\beta_2^t} \\
\theta' &= \theta - \eta \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}+\varepsilon}} \odot \mathbf{g},
\end{aligned} \tag{2.8}$$

where β_* is raised to the power of the current iteration step, t. All divisions in eq. 2.8 proceed element-wise, and we set $\beta_1 = 0.9$ and $\beta_2 = 0.999$ in our implementations.

For artificial neural networks, the gradients required for optimization are efficiently computed via the back-propagation algorithm (Rumelhart et al., 1986). Effectively, back-propagation allows the information from the objective function, $J(\theta)$, to flow backwards through the network in order to compute the gradients. These gradients are then used for optimization until the objective function is small enough for our needs and the learning problem is considered solved. Prior to all of this though, we have to specify the model to be optimized. Thus, to explain LSTMs, we start with standard feedforward neural networks.

2.2 Feedforward neural networks

Feedforward neural networks, also known as multilayer perceptrons, were first proposed in 1943 (McCulloch and Pitts, 1943) as a model for how the human brain processes information. They are the most basic form of artificial neural networks and comprising layers of artificial neurons, termed units (figure 2.1), they are capable of modelling functions $f : \mathcal{X} \to \mathcal{Y}$. Formally, a neural network with L hidden layers is parametrized by $L+1$ weight matrices $\mathbf{W}^{(0)}, ..., \mathbf{W}^{(L)}$ and $L+1$ vectors biases $\mathbf{b}^{(0)}, ..., \mathbf{b}^{(L)}$. To classify an input $\mathbf{x}$ the neural network computes the probability vector $\hat{\mathbf{y}}$ as follows:

$$\begin{aligned}
\mathbf{h}^{(0)} &= \mathbf{x} \\
\mathbf{h}^{(l+1)} &= g(\mathbf{W}^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}), \quad l = \{0, \ldots, L-1\} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)}),
\end{aligned} \tag{2.9}$$

where g is an element-wise nonlinear function, such as the sigmoid $g(\mathbf{x}) = \sigma(\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{x})}$. The **softmax function** is defined as

$$\text{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{C} \exp(x_j)}, \tag{2.10}$$

where C is the number of elements in $\mathbf{x}$ (i.e., number of classes). With the probability vector $\hat{\mathbf{y}}$, our cross-entropy objective function becomes

$$J(\theta) = -\sum_{n=1}^{N} \sum_{j=1}^{C} y_j^{(n)} \log(\hat{y}_j^{(n)}) \tag{2.11}$$

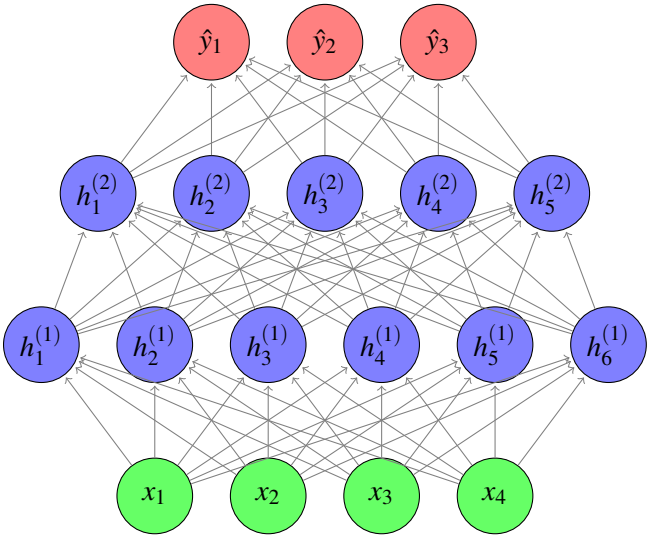


Fig. 2.1 Example of a feedforward neural network with 2 hidden layers (blue nodes). The input is depicted by the green nodes and the output by the red nodes.

Comment: Deep neural networks loosely refer to artificial neural networks with many layers. This field of deep learning is large and instead of listing the plethora of developments, here we list some of the inspiring reviews that are applicable to the work in this thesis. To start, a recent, thorough crowd-sourced review on deep learning applied to biological data is provided by Ching et al. (2018). For machine learning in healthcare, we refer the reader to the reviews by Clifton et al. (2015) and Miotto et al. (2017). Lastly, a review of recurrent neural networks is provided by Lipton et al. (2015a) and a review on deep learning for drug discovery by Baskin et al. (2016).

In neural network parlance, all the weight matrices and biases constitute the **parameters**, θ, that fully specify the function computed by the network. We use the term **weights** to exclusively refer to the weight matrices without the biases. The behaviour of neural network algorithms can be controlled using several settings, referred to as the **hyperparameters**. Hyperparameters are not modified by the model itself and include, for example, the number of units in each layer and the number of layers.

The feedforward neural network described above takes a vector as input. In the case where the inputs are multivariate sequences, this fully connected structure could result in an unwieldy model. Because we know the basic structure of the input – a sequence – we can account for the structure by sharing a set of parameters over the steps of the sequence. This more efficient solution is known as the recurrent neural network.

2.3 Recurrent neural networks

Much like the popular convolutional neural network (LeCun, 1986) that shares parameters over a grid of values, recurrent neural networks (RNNs) (Jordan, 1986) share parameters across different time steps of a sequence, which makes it possible to generalize to examples of different sequence length (Goodfellow et al., 2016, §10)(Murphy, 2012, §17.2). In other words, the transformations of the input at each time step[3] use the same parameters. For data believed to exhibit sequential correlations, RNNs are usually a better alternative to sequence-position-independent classifiers and sequential models that treat each time step differently. For example, if we wanted to make a prediction based on a binary sequence of whether it rained or not on a particular day, when treating each day independently, the only information gleaned from the data would be the frequency of rainy days, but we know in practice that weather often exhibits trends that may last for several days (Bishop, 2006, §13.1).

One way of interpreting an RNN is as a feedforward neural network unfolded in time (figure 2.2). Thus, to control the input-to-hidden information flow, a weight matrix $\mathbf{W}$ is used again. The additional hidden-to-hidden connections are mediated by the weight matrix $\mathbf{U}$. Given an input sequence $\mathbf{x}_{1:T}$ of length T, a single layer RNN calculates a probability vector

[3] With "time step" we refer to an entry along the sequential axis of a sequence, regardless of whether the sequence is temporal.

output $\hat{\mathbf{y}}$ as follows:

$$\begin{aligned}
\mathbf{h}_0 &= \mathbf{0} \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}_h), \quad t = 1, \ldots, T \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{V}\mathbf{h}_T + \mathbf{b}_y),
\end{aligned} \tag{2.12}$$

where g is an element-wise nonlinearity, usually the tanh function, $\mathbf{V}$ is the hidden-to-output weight matrix, and $\mathbf{b}_*$ are the bias vectors[4]. Having a model that maps the inputs to outputs, we can compute the gradients with back-propagation and optimize the RNN as described in section 2.1.1. In this case, back-propagation applied to the unrolled computational graph of the RNN (figure 2.2) is known as back-propagation through time.

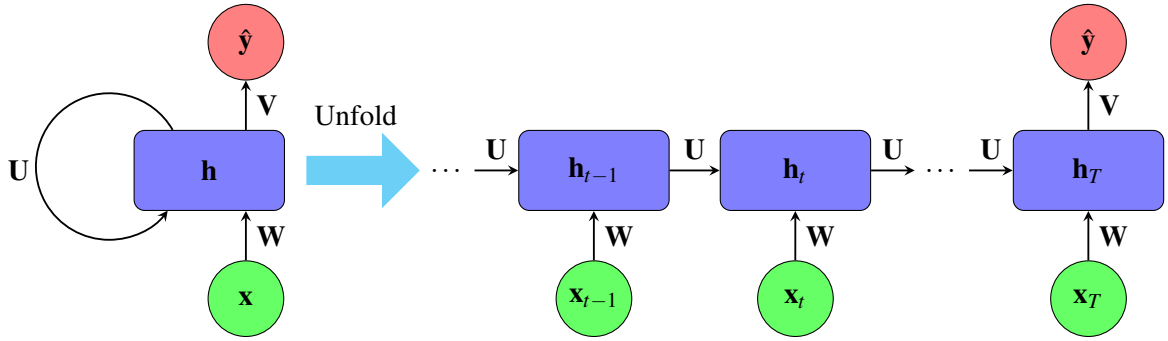


Fig. 2.2 Example of an unfolded recurrent neural network. The inputs are shown for each time step t, these are usually vectors, such as the input in figure 2.1. The hidden layer (blue rectangle) is a vector at each time step, similar to the blue nodes in figure 2.1. Input-to-hidden weight matrices are denoted by $\mathbf{W}$ and hidden-to-hidden weight matrices by $\mathbf{U}$. In this example, we have a single output vector at the final time step T of the sequence.

Owing to their iterative nature, RNNs are difficult to train, especially on problems with long-range temporal dependencies (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997; Sutskever, 2013). A small change to an iterative process can compound and result in very large effects many iterations later. This sensitivity to changes in the input effectively makes the objective function of the RNN discontinuous.

This difficulty in training can be shown through the choice of the activation function g. In modern neural networks, the default recommendation is to use the rectified linear unit (ReLU) activation function for hidden units (Glorot et al., 2011; Jarrett et al., 2009; Nair and Hinton, 2010). This ReLU nonlinearity is only lower bounded, which makes its use in RNNs difficult. To show why the activation function g used in the RNN has to be bounded from

[4]Note that this formulation is one of many possible RNN constructions. It is also possible, for example, to produce an output vector $\hat{\mathbf{y}}_t$ at each time step.

above and below, we follow Cho (2015) and consider the case when g is a linear function, $g(a) = a$, which is unbounded. In this case, assuming that $\mathbf{x}_{t'}$ is the only nonzero entry in the input sequence, the hidden activation vector after τ input elements is given by

$$\begin{aligned}\mathbf{h}_\tau &= \mathbf{U}(\mathbf{U}(\mathbf{U}(\mathbf{U}(...) + \mathbf{W}\mathbf{x}_{\tau-3}) + \mathbf{W}\mathbf{x}_{\tau-2}) + \mathbf{W}\mathbf{x}_{\tau-1}) + \mathbf{W}\mathbf{x}_\tau \\ &= \left(\prod_{t=1}^{\tau-t'} \mathbf{U}\right) \mathbf{W}\mathbf{x}_{t'}, \end{aligned} \tag{2.13}$$

where we omit the bias term for brevity. If $\mathbf{U}$ admits an eigendecomposition of the form

$$\mathbf{U} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}, \tag{2.14}$$

where Λ is the diagonal matrix of eigenvalues, and the corresponding eigenvectors constitute the columns of matrix $\mathbf{Q}$, then

$$\mathbf{h}_\tau = \mathbf{Q}\Lambda^{\tau-t'}\mathbf{Q}^{-1}\mathbf{W}\mathbf{x}_{t'}. \tag{2.15}$$

If the largest eigenvalue $\lambda_{\max} = \max \Lambda$ is greater than 1, the norm of $\mathbf{h}_\tau$ will explode, i.e., $\|\mathbf{h}_\tau\| \to \infty$. The rate of growth is exponential with respect to the sequence length, thus, even if the sequence is relatively short, $\|\mathbf{h}_\tau\|$ will grow quickly if $\lambda_{\max}$ is reasonably larger than 1. Therefore, we bound the hidden activations from above and below and in the case of the $\tanh : \mathbb{R} \to [-1, 1]$, we have

$$\|\mathbf{h}_\tau\| \leq \dim(\mathbf{h}_\tau). \tag{2.16}$$

Although the tanh prevents the hidden activations from exploding, complications remain for the gradients of the RNN.

2.3.1 Gradient problems

The basic problem is that gradients propagated over many time steps will tend to either grow or decay (Bengio et al., 1994; Pascanu et al., 2013). To show how gradient problems easily arise in RNNs, we again assume that $\mathbf{x}_{t'}$ is the only nonzero entry in the input sequence, and the matrix $\mathbf{U}$ is full rank. We denote the pre-activation vectors as

$$\mathbf{s} = \mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}_h. \tag{2.17}$$

The derivative of our objective function J with respect to the non-zero input is given by

$$\frac{\partial J}{\partial \mathbf{x}_{t'}} = \frac{\partial J}{\partial \mathbf{h}_T} \left(\prod_{t=t'+1}^{T} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right) \frac{\partial \mathbf{h}_{t'}}{\partial \mathbf{x}_{t'}}. \tag{2.18}$$

When the number of time steps $T - t'$ is large, the propagation of gradients greatly depends on the product term. We can re-write the term inside the product as

$$\begin{aligned} \frac{\partial \mathbf{h}_t}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{h}_{t-1}} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{s}} \mathbf{U} \\ &= \begin{bmatrix} g'(s_1) & 0 & \dots & 0 \\ 0 & g'(s_2) & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & g'(s_d) \end{bmatrix} \mathbf{U} \end{aligned} \tag{2.19}$$

where g is the chosen nonlinearity (eq. 2.12), and d is the number of hidden units. The derivatives of the commonly used upper and lower bounded nonlinearities are given by

$$\begin{aligned} \sigma'(x) &= \sigma(x)(1 - \sigma(x)), & 0 < \sigma'(x) \leq 0.25 \\ \tanh'(x) &= 1 - \tanh^2(x), & 0 < \tanh'(x) \leq 1. \end{aligned} \tag{2.20}$$

To show how this can lead to vanishing gradients, we denote this upper bound of $g'(x)$ with α and we assume that the maximum singular value $\lambda_{\max}$ of $\mathbf{U}$ is bounded as $\lambda_{\max} < \frac{1}{\alpha}$. With the spectral norm denoted by $\|\cdot\|_s$, we know that for any matrices $\mathbf{A}$ and $\mathbf{B}$

$$\|\mathbf{AB}\|_s \leq \|\mathbf{A}\|_s \|\mathbf{B}\|_s, \tag{2.21}$$

and

$$\|\mathbf{A}\|_s = \lambda_{\max}. \tag{2.22}$$

Thus the spectral norm of the gradient (eq. 2.19) is bounded as

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right\|_s \leq \left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{s}} \right\|_s \|\mathbf{U}\|_s \leq \alpha \lambda_{\max} < 1, \ \forall t. \tag{2.23}$$

Let $\gamma \in \mathbb{R}$ be such that $\forall t$, $\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right\|_s \leq \gamma < 1$; the existence of γ is given by eq. 2.23. By induction over t, we can show that

$$\left\| \prod_{t=t'+1}^{T} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right\|_s \leq \gamma^{T-t'}. \tag{2.24}$$

Thus, because $\gamma < 1$, long-term contributions to the gradient, for which $T - t'$ is large, go to 0 exponentially fast – known as the **vanishing gradient** problem. Conversely, if we assume that the largest singular value $\lambda_{\max}$ of $\mathbf{U}$ is larger than $\frac{1}{\alpha}$, the upper bound of the long-term contributions to the gradient will tend to ∞ exponentially fast. The **exploding gradient** problem occurs when the long-term contributions to the gradients tend to ∞ (Pascanu et al., 2013). Fortunately, the exploding gradient problem can easily be addressed by clipping the gradients.

2.3.2 Gradient clipping

By inspecting the gradients of the objective function with respect to the parameters, we can detect whether a gradient has exploded. Once detected, we can simply clip the gradient. This clipping is performed by scaling the gradients to have a smaller norm, which guarantees that each clipped update step is in the same direction as the original gradient (Pascanu et al., 2013). In other words, if the gradient norm $\|\nabla_\theta J\|$ is larger than some predefined threshold $\gamma > 0$, we scale the gradients to have a norm equal to γ. Otherwise, we leave it as is

$$\tilde{\nabla} = \begin{cases} \gamma \frac{\nabla}{\|\nabla\|} & \text{if } \|\nabla\| > \gamma \\ \nabla & \text{otherwise} \end{cases}, \tag{2.25}$$

where we used the shorthand notation ∇ for $\nabla_\theta J$. The rescaled gradient $\tilde{\nabla}$ is then used to update the parameters during optimization (section 2.1.1). Whereas the exploding gradient can easily be clipped, mitigating the vanishing gradient is more difficult. One solution is to have skip connections that enable unimpeded propagation of gradients through time. Such skip connections are enabled by the gating mechanisms of the LSTM, described next.

2.4 Long short-term memory

The long short-term memory (LSTM) network (Gers et al., 2000; Hochreiter and Schmidhuber, 1997) extends the recurrent neural network with a memory unit and three gating units that control the flow of information to and from the memory unit. Formally, a single layer

LSTM calculates hidden layer activations at each time step $t = \{1, ..., T\}$ as follows:

$$\begin{aligned}
\mathbf{c}_0 &= \mathbf{h}_0 = \mathbf{0} \\
\mathbf{i}_t &= \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t + \mathbf{b}_o) \\
\mathbf{f}_t &= \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t + \mathbf{b}_f) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{W}_c \mathbf{x}_t + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned} \tag{2.26}$$

These operations are visualized in figure 2.3. Information flow is controlled by the input gate **i**, output gate **o**, and the forget gate **f**. These gating units are implemented by multiplication, thus it's natural to restrict their domain to $[0, 1]$, motivating the use of the sigmoid nonlinearity.

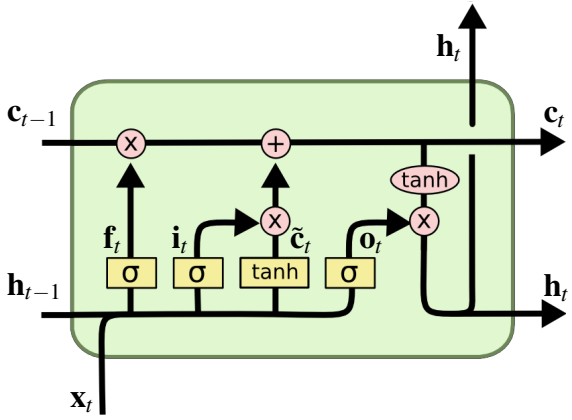


Fig. 2.3 The operations performed by the LSTM unit during a single time step. The unit receives the previous memory value $\mathbf{c}_{t-1}$, the previous hidden activation $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$. The input $\mathbf{i}_t$, output $\mathbf{o}_t$, and forget $\mathbf{f}_t$ gates modulate the information flow through the unit. The current memory value $\mathbf{c}_t$ is passed on to the next time step, along with the current hidden activation $\mathbf{h}_t$, which can also be used for output at the current step (upward $\mathbf{h}_t$). Pink nodes denote element-wise operations. The yellow rectangles take the vectors $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ as input, multiply these vectors by the appropriate weight matrices, sum the resulting vectors, and finally apply the depicted element-wise nonlinearity. Biases are omitted for brevity. (Adapted from Olah (2015).)

These gating units and the recurrent self-connection of the memory unit mitigate the gradient problems. When the forget gate is open $\mathbf{f}_t = \mathbf{1}$ and the input gate closed $\mathbf{i}_t = \mathbf{0}$ for all t, the gradients can flow through the memory unit without alteration for an indefinite amount of time, thus overcoming the vanishing gradient problem. In practice, the gates do not necessarily isolate the memory unit; instead, they address the vanishing gradient

problems in some situations, as demonstrated by the easier long-term dependency learning of LSTMs compared with RNNs (Bengio et al., 1994; Graves, 2012, 2013; Hochreiter and Schmidhuber, 1997; Sutskever et al., 2014).

Information can be propagated unaltered through the memory units if the forget gate is open and the input gate is closed. The state of the gates, however, are calculated and changed at each time step. Effectively, the LSTM has to learn when to create these skip connections through time. If learning is started from bad initial conditions, few of the gates may learn to create long-term skip connections because the short-term connections provide easier rewards. Therefore, careful parameter initialization is paramount for successful implementation of LSTMs.

2.4.1 Parameter initialization

LSTMs do not completely solve the gradient problems. To further alleviate gradient problems, careful parameter initialization is required. Glorot and Bengio (2010a) showed that the initialization of deep neural networks is crucial. To maintain information flow, variances of the forward activations and back-propagated gradients are required to remain roughly the same as they move up and down the network. This information flow is kept when the weights are initialized with the Xavier initializer (Glorot and Bengio, 2010a)

$$\mathbf{W}^{(l)} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}}, \frac{\sqrt{6}}{\sqrt{n_l + n_{l+1}}}\right], \tag{2.27}$$

where n_l is the size of layer l and $\mathcal{U}$ denotes the uniform distribution. The Xavier initializer applies to only the weights of the network. On the other hand, initialization of the biases in the LSTM was found to be important by Jozefowicz et al. (2015). They found that initializing all the biases to zero except for the forget gate bias $\mathbf{b}_f$, which is initialized to one, produces the best performing LSTM. Larger initial forget gate biases lead to forget gate values closer to one, which allows information to be kept over more time steps at the start of training.

During the completion of this thesis, multiple experiments were done for the initialization of LSTM parameters. For example, we attempted *simple parameter guessing* (similar to that in Hochreiter and Schmidhuber (1997)), whereby the best randomly selected parameters over K iterations are used for the initial parameters of training. Over $K = 10,000$ iterations, the various Gaussian and normal distributions sampled from did not yield any initializations as good as the initialization scheme mentioned above. Moreover, proceeding with such non-Xavier initialized weights would often result in LSTMs that are unable to learn the task.

Thus, our results confirm the importance of careful LSTM parameter initialization. Given the intricacies of the LSTM, our standard LSTM setup is provided in the following section.

2.4.2 Default LSTM setup

Unless otherwise stated, the LSTMs referred to in this thesis were set up as follows. We implement the standard LSTM (eq. 2.26) using Tensorflow (Abadi et al., 2015) and Python 3. All the biases are initialized to zero, except for the forget gate, which is initialized to one. All weight matrices are initialized using the Xavier uniform distribution (section 2.4.1). We clip the gradients to have a maximum gradient norm $\gamma = 5$ (eq. 2.25). Lastly, the predicted class probabilities $\hat{\mathbf{y}}$ are obtained by means of a softmax function applied to the hidden activations $\mathbf{h}_T$ at the final time step T, formally

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{V}\mathbf{h}_T + \mathbf{b}_y), \tag{2.28}$$

where $\mathbf{V}$ is the hidden-to-output weight matrix and $\mathbf{b}_y$ the output biases.

2.5 Gated recurrent unit

The LSTM is a gated recurrent neural network (RNN). Another popular gated RNN was recently proposed by Cho et al. (2014) and is called the gated recurrent unit (GRU) network (see figure 2.4). The main difference between the LSTM and GRU is that the GRU uses a single gating unit (update gate $\mathbf{z}$) to control the forget and input factors (Chung et al., 2014). Also having a reset gate $\mathbf{r}$, GRUs calculate the hidden activations at each time step $t = \{1, ..., T\}$ as

$$\begin{aligned}
\mathbf{r}_t &= \sigma(\mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{W}_r\mathbf{x}_t + \mathbf{b}_r) \\
\mathbf{z}_t &= \sigma(\mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{W}_z\mathbf{x}_t + \mathbf{b}_z) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{U}_h(\mathbf{r} \odot \mathbf{h}_{t-1}) + \mathbf{W}_h\mathbf{x}_t + \mathbf{b}_h) \\
\mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t.
\end{aligned} \tag{2.29}$$

Several studies on architectural variations of the LSTM and GRU have found no variant that would definitely outperform the rest on a wide a range of tasks (Goodfellow et al., 2016; Greff et al., 2015; Jozefowicz et al., 2015). Jozefowicz et al. (2015) found that initializing the forget gate bias to one makes the LSTM as strong as the best of the explored variants. Thus we chose to proceed with the LSTM.

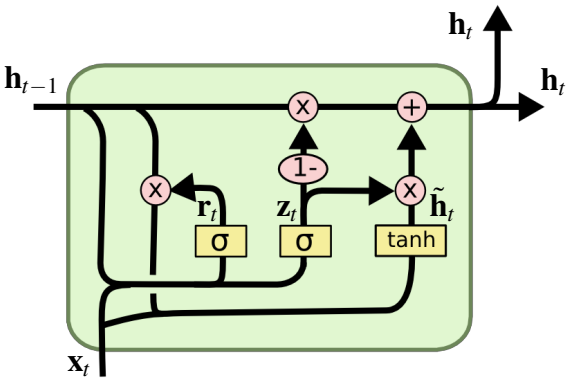


Fig. 2.4 The operations performed by the GRU during a single time step. The unit receives the previous hidden activations $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$. The update $\mathbf{z}_t$ and reset $\mathbf{r}_t$ gates modulate the information flow through the unit. The current hidden activation vector $\mathbf{h}_t$ is then passed on to the next time step and can also be used for output at the current step (upward $\mathbf{h}_t$). Pink nodes denote element-wise operations. The yellow rectangles take two vectors as input, multiply these vectors by the appropriate weight matrices, sum the resulting vectors, and finally apply the depicted element-wise nonlinearity. Biases are omitted for brevity. (Adapted from Olah (2015).)

Another reason for choosing the LSTM is that the additional gate intuitively should make the LSTM more flexible than the GRU, and therefore the LSTM should have a larger representational capacity. Towards the end of this thesis (chapter 7), we find that having enticing conditions for skip connections is more important.

2.6 Overfitting

The main aim of machine learning is to perform well on unseen data points, not just those that the model was trained on (Goodfellow et al., 2016, §5.2). A model achieves good **generalization** if it performs well on previously unobserved data points. Conversely, a model that does not generalize well, suffers from **overfitting** because the gap between the training error and the test error is large. We can control the overfitting tendency of a model by altering its capacity. Informally, a model's capacity is its ability to fit a wide variety of functions. Models with a high capacity can overfit, whereas models with a low capacity can struggle to learn any function that reasonably explains the data. Deep learning models usually have a large capacity and are therefore prone to overfitting. Training on more data results in better generalization, but it's often infeasible. In the following section, we describe a technique that limits the capacity of a model in order to improve generalization.

2.6.1 Regularization

Regularization is any modification to the learning algorithm intended to reduce overfitting. There are various techniques for regularizing artificial neural networks; a descriptive list is provided by Goodfellow et al. (2016, §7). Here we describe the three regularization techniques that are frequently used in this thesis.

First, is the popular **weight decay** regularization technique, which encourages the model to have smaller weights. The weight decayed objective function is given by

$$J(\theta) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\mathbf{y}|\mathbf{x},\theta)] + \lambda \|\omega\|_2^2, \tag{2.30}$$

where $\|\cdot\|_2^2$ denotes the squared L_2 norm, ω are all the weights in the set of parameters θ, λ is a value chosen ahead of time that specifies our preference for smaller weights, and the first term is the original cross-entropy from eq. 2.3. Weight decay is typically only applied to the weights of a network and not the biases. Because each bias value controls only a single variable, little variance is induced by leaving them unregularized, and regularizing them could lead to underfitting (Goodfellow et al., 2016, §7.1).

Dropout, the second of our regularization techniques, regularizes a neural network by setting a random selection of units to zero during an iteration (single minibatch) of training (Srivastava et al., 2014). More formally, for the input and each hidden layer, a corresponding binary vector $\mathbf{u}^{(l)}$, with the same size as the layer, is sampled at each training iteration. The elements of vector $\mathbf{u}^{(l)}$ take value 0 with probability $p^{(l)}$ – the dropout probability. For each layer l (excluding the output layer), the units, $\mathbf{v}$, are dropped by element-wise multiplication, $\mathbf{v}^{(l)} = \mathbf{v}^{(l)} \odot \mathbf{u}^{(l)}$, and training proceeds as per usual. At test time dropout is not used and the outgoing weights of each unit are multiplied by the **keep probability** $1 - p^{(l)}$ to approximately average the predictions of all the thinned neural networks.

Owing to the LSTM being more intricate than the standard feedforward neural network, different approaches to dropout for LSTMs have been proposed; the same dropout mask at each time step (Gal and Ghahramani, 2016b), dropping only the non-recurrent connections (Pham et al., 2014; Zaremba et al., 2014), and dropout without memory loss (only the memory unit update $\tilde{\mathbf{c}}$ is dropped) (Semeniuta et al., 2016). In our implementations, unless otherwise stated, dropout is applied to the hidden activations $\mathbf{h}_t$ that are fed into new hidden layers or softmax outputs. The hidden activations between time steps are maintained.

The third regularization technique, which is used in all of our LSTM implementations, is early stopping. To explain early stopping, we first introduce the validation set. Whereas the held-out test set is not used during training, the **validation set** is used during training, not for updating the parameters but rather for estimating the model's generalization error. When

training deep learning models with sufficient representational capacity to overfit the task, the training set error decreases steadily over time and at some point, the validation set error begins to increase. At this point, the parameters are considered to achieve the best generalization before overfitting ensues. Thus, to prevent overfitting, we train for a pre-determined number of epochs (enough to see a steady rise in validation set error) and return the parameters of the model at the best validation set error over all of the epochs. This procedure is known as **early stopping** and acts as a regularizer by restricting the optimization to explore a small volume of the parameter space in the neighbourhood of the initial parameter values. Therefore, it is similar to weight decay, but it has the advantage of automatically determining the correct amount of regularization (Goodfellow et al., 2016, §7.8).

Having described the necessary basics of machine learning and LSTMs, we proceed to present our biological applications, visualizations, and extensions of LSTMs. We start with medical applications of LSTMs in the following chapter.

Chapter 3

Medical applications

At first pass, patients receive correct diagnoses and treatments less than 50% of the time (McGlynn et al., 2003). Clinicians typically determine the course of treatment given the current health status of the patient as well as their estimate of the outcome of possible future treatments. The effect of treatments for a given patient is non-deterministic (uncertain) and predicting the effect of a series of treatments over time compounds the uncertainty (Bennett and Hauser, 2013). There is a growing body of evidence that complex medical treatment decisions are better handled with the aid of modelling, compared to intuition alone (Meehl, 1986; Schaefer et al., 2005). It has been argued that machine learning holds substantial promise for the future of medicine and for our ability to tailor care to the particular physiology of the patient (Abston et al., 1997; Clifton et al., 2015; Cooper et al., 1997; Lapuerta et al., 1995; Mani et al., 1999; Morik et al., 2000; Ohmann et al., 1996; Sboner and Aliferis, 2005; Svátek et al., 2003; Vairavan et al., 2012).

In this chapter, we demonstrate some of the benefits that LSTMs hold for medical problems. We begin with a brief overview of the medical machine learning literature.

3.1 Overview of medical machine learning literature

There are myriad studies applying machine learning to medical data. Literature relevant to our work includes any machine learning applications to electronic health records or physiological measurements, such as vital signs recorded in intensive care units. Before delving into the literature we have to cover some preliminaries.

In a binary classification task, there are positive (one) and negative (zero) events. **Sensitivity** is defined as the percentage of correctly identified positive events, for example, the

percentage of patients with sepsis correctly classified as having the condition.

$$\text{sensitivity} = \frac{\text{true positive events}}{\text{true positive events} + \text{false negative events}} \tag{3.1}$$

Specificity is defined as the percentage of correctly classified negative events, for example, the percentage of healthy patients who are correctly classified as not having sepsis.

$$\text{specificity} = \frac{\text{true negative events}}{\text{true negative events} + \text{false positive events}} \tag{3.2}$$

Clearly, when developing a classifier for medical diagnostics, having a higher sensitivity takes precedence over specificity. However, when perfect sensitivity is combined with no specificity, false alarms become a hospital nuisance – only 23% of bedside alarms in the ICU are clinically relevant (Imhoff and Fried, 2009).

Classification accuracy provides a measure of performance for a single threshold of sensitivity and specificity. In medicine, where the sensitivity-specificity trade-off is important, we seek a measure that explores the entire range of the trade-off. The receiver operating characteristic (ROC) curve enables this by plotting the sensitivity against (1 - specificity) as the discrimination threshold is varied. The area under the curve (AUC) can then be used to measure the overall performance of the classifier, with a perfect model having an AUC = 1 and randomly choosing classes results in an AUC = 0.5. For multi-class classification problems, the average per-class AUC is used.

In table 3.1 we summarize a selection of the notable studies related to our work. Rows in ascending order of publication date show the best results obtained by each study. For a more detailed review of the literature, we refer the reader to Van Der Westhuizen (2016). From the table, it is evident that applying neural networks to medical data is no new venture. It was successfully attempted by Tu and Guerriere (1993). This was before neural networks were proven to be adept at learning the characteristics of large real-life datasets (Krizhevsky et al., 2012). On the other hand, recurrent neural networks have only recently received attention, which is most likely due to the advent of LSTMs. It should be noted, however, that the LSTM studies listed in table 3.1 are applications to low-resolution electronic health records. LSTMs, therefore, hold a great untapped potential for high-resolution data from physiological monitoring.

The machine learning techniques mentioned in table 3.1 include Gaussian processes, hidden Markov models, support vector machines, recurrent neural networks, convolutional neural networks (CNNs), feedforward neural networks, logistic regression, naive Bayes classifiers, and decision trees. It is difficult to compare the different methods because they

Table 3.1 Machine learning techniques applied to medical data over the years

Study	Description	Best results
Tu and Guerriere (1993)	Predict patient length of stay in ICU given 15 variables measured after cardiac surgery.	0.7 AUC with feedforward neural networks
Tsien et al. (2000)	Detect artefacts in 4 physiological signals in the NICU	0.94 AUC with decision trees
Clermont et al. (2001)	Predict mortality given the worst values of 16 variables in the first 24 hours of ICU	0.84 AUC with logistic regression
Shen et al. (2002)	Biometric identification with single-lead ECG signals	0.8 Accuracy with feedforward neural networks
Tong et al. (2002)	Predict ventilation duration from 6 variables in the NICU	0.93 accuracy with feedforward neural networks
Ramon et al. (2007)	Multiple prediction tasks including mortality prediction from manual daily observations in ICU	0.88 AUC with naive Bayes
Saria et al. (2010)	Predict morbidity based on the first 3 hours after admission to the NICU	0.92 AUC with logistic regression
Kim et al. (2011)	Predict patient mortality from 15 variables with values selected in the first 24 hours of ICU admission	0.98 AUC with decision trees
Guiza et al. (2013)	Predict patient intracranial pressure (ICP) 30 min in advance based on ICP and mean arterial pressure measured per minute	0.87 AUC with Gaussian processes
Ongenae et al. (2013)	Predict the need for dialysis in ICU given creatinine and diuresis measurements	0.87 AUC with naive Bayes
Gultepe et al. (2014)	Predict patient lactate levels from 7 EHR variables	0.9 AUC with hidden Markov models
Zhai et al. (2014)	Predict the need for paediatric ICU transfer within the first 24h of admission given EHRs	0.91 AUC with logistic regression
Mani et al. (2014)	Detect sepsis in the NICU given EHRs	0.65 AUC with decision trees
Stanculescu et al. (2014)	Predict sepsis from 4 physiological signals in the NICU	0.8 AUC with auto-regressive hidden Markov models
Ghassemi et al. (2015)	Predict patient mortality from EHRs	0.81 AUC with Gaussian processes
Lipton et al. (2015b)	Assign medical conditions to paediatric ICU patients given hourly measurements in EHRs	0.86 AUC with LSTMs
Churpek et al. (2016)	Predict mortality from EHRs and 8 variables measured every hour	0.8 AUC with random forests
Jagannatha and Yu (2016)	Classify EHR notes into 1 of 9 classes	0.8 F1 score with GRUs
Harutyunyan et al. (2017)	Multitask prediction (length of stay, phenotype, mortality) from the MIMIC-III critical care databse (Johnson et al., 2016)	0.86 AUC with multitask LSTMs
Rajpurkar et al. (2017)	Detect heartbeat arrhythmias from single-lead ECG signals	CNNs (0.81 F1 score) outperform cardiologists
Rajkomar et al. (2018)	Predict mortality from EHRs 24 hours after admission	0.95 AUC with ensemble of LSTMs and feedforward neural networks

AUC - area under the receiver operating characteristic curve
ECG - electrocardiogram
EHR - electronic health record
F1 score - see eq. 3.9
ICU - intensive care unit
NICU - neonatal intensive care unit

are applied to different datasets. Perhaps the most successful application of machine learning to medical data is the recent study by Rajpurkar et al. (2017). The study demonstrated that a large CNN with batch normalization (see section 3.3.2), and residual connections can outperform cardiologists at detecting heartbeat arrhythmias. One of the largest single-lead electrocardiogram (ECG) datasets was collected and this is one of the few machine learning studies to quantitatively determine the accuracy of human experts.

One of the most important facets of deep learning applications is the dataset size. The number of patients used in studies applying machine learning to medical data has ranged from 7 (Greene et al., 2007) to 270,000 (Churpek et al., 2016). The number of patients does not solely determine the size of the dataset but provides a good measure of how generalizable a model would be to new patients. Another measure of dataset size is the number of data points per patient, which could become rather large when dealing with high-resolution time series datasets.

A natural question one would consider is whether LSTMs are better than standard sequence modelling techniques for analysing biological data. In light of this, we compare the classification performance of LSTMs to a well-known and well-studied model, the hidden Markov model (HMM). HMMs have successfully been applied to medical data (Gultepe et al., 2014; Pimentel et al., 2015; Stanculescu et al., 2014; Williams et al., 2006), but have, to the best of our knowledge, yet to be compared to LSTMs on the same medical dataset.

3.2 Comparing LSTMs to hidden Markov models

3.2.1 Introduction to hidden Markov models

A hidden Markov model (HMM) is a structured probabilistic model where the observations, $\mathbf{x}$, are assumed to be generated by some latent (unobserved) process. On the latent space we define a discrete-state, discrete-time Markov chain with hidden states $z_t \in \{1,...,K\} \forall t$. An observation model $p(\mathbf{x}_t|z_t)$ determines how the observations relate to states, resulting in a joint distribution with the form

$$p(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}) = p(\mathbf{z}_{1:T}) p(\mathbf{x}_{1:T}|\mathbf{z}_{1:T}) = \left[p(z_1) \prod_{t=2}^{T} p(z_t|z_{t-1}) \right] \left[\prod_{t=1}^{T} p(\mathbf{x}_t|z_t) \right]. \tag{3.3}$$

When the observations $\mathbf{x}_t$ are continuous, as with most medical signals, a conditional Gaussian is commonly used for the observation model, known as the emission distribution

$$p(\mathbf{x}_t|z_t = j, \mu_j, \Sigma_j) = \mathcal{N}(\mathbf{x}_t|\mu_j, \Sigma_j). \tag{3.4}$$

It is also possible to have an M-component Gaussian mixture model as the emission distribution

$$p(\mathbf{x}_t|z_t = j, \boldsymbol{\mu}_j, \Sigma_j, \mathbf{c}_j) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_{jm}, \Sigma_{jm}), \tag{3.5}$$

where c_{jm} denotes the mixture weights.

Learning HMMs is done by estimating the parameters $\theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\phi}\}$, where $\boldsymbol{\phi}$ are the parameters of the class-conditional densities $p(\mathbf{x}_t|z_t = j)$, $A(i,j) = p(z_t = j|z_{t-1} = i)$ is the transition matrix, and $\boldsymbol{\pi}(i) = p(z_1 = i)$ is the initial state distribution. Expectation maximization is used for learning, which, when applied to HMMs is known as the Baum-Welch algorithm (see Murphy (2012, pg. 620) for details). Once the model is trained we can we can predict the probability of the next state $p(z_t = j)$ given a sequence prefix $\mathbf{x}_{1:t-1}$

$$p(z_t = j|\mathbf{x}_{1:t-1}) = \sum_i p(z_t = j|z_{t-1} = i)p(z_{t-1} = i|\mathbf{x}_{1:t-1}). \tag{3.6}$$

Using Bayes rule we can absorb the observed data at time step t

$$\begin{aligned} p(z_t = j|\mathbf{x}_{1:t}) &= p(z_t = j|\mathbf{x}_t, \mathbf{x}_{1:t-1}) \\ &= \frac{p(\mathbf{x}_t|z_t = j)p(z_t = j|\mathbf{x}_{1:t-1})}{p(\mathbf{x}_t|\mathbf{x}_{1:t-1})}. \end{aligned} \tag{3.7}$$

Together, these equations enable computing the probability of a sequence $\mathbf{x}_{1:T}$ given the model parameters θ. To avoid numerical underflow in practice, we work in the log domain

$$\log p(\mathbf{x}_{1:T}|\theta) = \sum_{t=1}^{T} \log p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = \sum_{t=1}^{T} \log \sum_j p(\mathbf{x}_t|z_t = j)p(z_t = j|\mathbf{x}_{1:t-1}). \tag{3.8}$$

In order to classify a sequence as belonging to a specific class, an HMM is trained for each of the classes (hyperparameters are kept the same for each HMM). The class of a new data point $\mathbf{x}'_{1:T}$ is then determined by $\text{argmax}_{\theta_c} \log(p(\mathbf{x}'_{1:T}|\theta_c))$, where θ_c denotes the parameters for a model from class c (Chiappa and Bengio, 2004). We implemented HMMs with the *hmmlearn* package `https://github.com/hmmlearn/hmmlearn` in Python. Given this high-level understanding of HMMs, we can now compare them to LSTMs on medical datasets.

3.2.2 Traumatic brain injury patient dataset

In a collaborative effort with the Department of Clinical Neurosciences at Addenbrookes hospital, we were fortunate to be given an anonymized dataset of patients from the traumatic

Table 3.2 Details of the traumatic brain injury dataset

Number of patients	101 (22 females)
Age range	15 to 76
Classes (number of patients)	GOS-1 (43) ǀ GOS-5 (58)
Measurement resolution	0.2 Hz
Segment lengths	720 time steps (1 hour)
Number of data points	12,473
Average recording duration [min,max]	123h 21m [3h,472h]
Number of inputs	18

brain injury (TBI) unit. The dataset was collected with the *intensive care monitoring plus* (ICM+) software package (Smielewski, 2011).

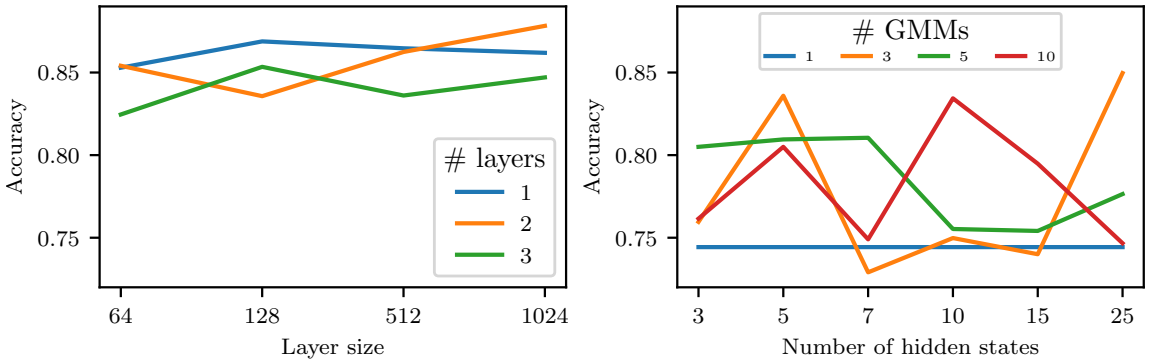
Details of the TBI dataset are provided in table 3.2. We have chosen a single data point[1] to span one hour (720 time steps). One hour is long enough to capture physiological changes of the patient and a length of 720 time steps does not require too much memory from sequence models. Patients were labelled according to the Glasgow Outcome Scale (GOS) (Jennett and Bond, 1975), which assigns a number between 1 (death) and 5 (healthy) to patients based on their health status 6 months after admission to the traumatic brain injury unit. This dataset only contains patients who were assigned a GOS score of 1 or 5, thus reducing the problem to a binary classification task between the two possible extremes. A comprehensive account of the variables measured is provided in Appendix A.

The data were randomly split according to patients with a train:validation:test dataset ratio of 70:10:20. The train and validation datasets were combined when training the HMMs. The LSTMs were trained using the *Adam* optimizer (Kingma and Ba, 2015) with a learning rate of 0.001, a dropout value of 0.1 and a minibatch size of 100.

In figure 3.1 we juxtapose the accuracies obtained by the LSTM and the HMM for various hyperparameter settings. From the results, the hyperparameter selection seems haphazard and exemplifies the importance of hyperparameter exploration. In the deep learning literature, it is widely held that more nonlinear activations (more layers) result in better performing models (Goodfellow et al., 2016). However, deeper LSTMs could also lead to gradient problems, which would explain the lower accuracies of the 3-layer LSTM. In general, the graphs show that the LSTM outperforms the HMM on this dataset. The best LSTM achieved an accuracy of 88% and the best HMM an accuracy of 85%. An additional limitation of HMMs, not shown here, is their computational cost, which became prohibitively expensive with more than 10 states for the CPU-implementation that we used. An HMM with more

[1] data point refers to the entire sequence $x_{1:T}$

hidden states could have a better representational capacity, but becomes computationally too expensive . An HMM with K states on sequences of length T has a computational cost of $\mathcal{O}(K^2T)$, scaling quadratically with the number of hidden states.



(a) LSTM **(b)** HMM

Fig. 3.1 Comparing model accuracies on the traumatic brain injury dataset. (a) Accuracies for the LSTM with the number of hidden layers indicated in the legend and the number of hidden units on the x-axis. (b) Accuracies for the hidden Markov model (HMM) with the number of Gaussian mixture models (GMMs) used for the emission distribution indicated in the legend, and the number of hidden states shown on the x-axis.

These results on the traumatic brain injury dataset demonstrate that LSTMs are able to generalize better than HMMs over different patients. Although this dataset has a relatively high resolution compared to the studies in table 3.1, most physiological signals are recorded at much higher frequencies, which we explore next.

3.2.3 The influence of resolution on accuracy

Given the growing capabilities in computation, we enter a paradigm where we could apply machine learning to physiological signals at high resolutions. Deep learning techniques have seen recent success on high-resolution audio waveforms (Van Den Oord et al., 2016). For physiological signals, higher resolution usually means more information, but unfortunately, it also means more noise.

Owing to hidden Markov models (HMMs) having a finite discrete hidden state, they are expected to have less representational capacity compared to LSTMs with continuous hidden representations. Therefore, LSTMs should be the obvious solution for the machine learning requirements in the high-resolution paradigm. In this section, we evaluate the effects of resolution on classification performance of HMMs and LSTMs.

Another anonymized dataset we were fortunate to have received from the collaboration described in section 3.2.2, is from the intensive care unit at Addenbrookes hospital. The phys-

iological signals in this dataset were provided at the original high measurement resolution. Details of the dataset are provided in table 3.3.

Table 3.3 Details of the intensive care unit dataset

Number of patients	4 (all males)
Age range	23 to 46
Classes (number of patients)	healthy (2) | death (2)
Measurement resolution	200 Hz or 240 Hz
Average recording duration [min,max]	21h 6m [6h, 40h]
Recorded signals	ECG, intracranial pressure, arterial blood pressure

The data were prepared by decimating the multivariate signals to different resolutions (40, 10, 1, and 0.1 Hz). This was achieved by means of the decimate function in the Scipy package (Jones et al., 2001) in Python. An eighth-order type 1 Chebyshev low-pass filter was applied to the signal before it was decimated. As recommended by the creators of the software package, when the downsampling factor was larger than 13, we decimated the signal multiple times to get to the desired resolution.

After decimation, the signals were segmented into shorter subsequences with lengths 100, 200, 500, or 1000. A single subsequence is referred to as a data point and has a corresponding label. These input-output pairs were then randomly shuffled and split (according to data points) to have a ratio of 70:10:20 for the train, validation, and test datasets. Unfortunately, medical datasets often have unbalanced classes – in this case, roughly 70% of the data was from the healthy patients and the remainder from the dying patients.

When the classes in a classification task are unbalanced, the accuracy could be a misleading metric. The F1 score is a commonly used measure of binary classification accuracy

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{3.9}$$

where recall is the same as in eq. 3.1

$$\text{recall} = \frac{\text{true positive events}}{\text{true positive events} + \text{false negative events}} = \text{sensitivity}, \tag{3.10}$$

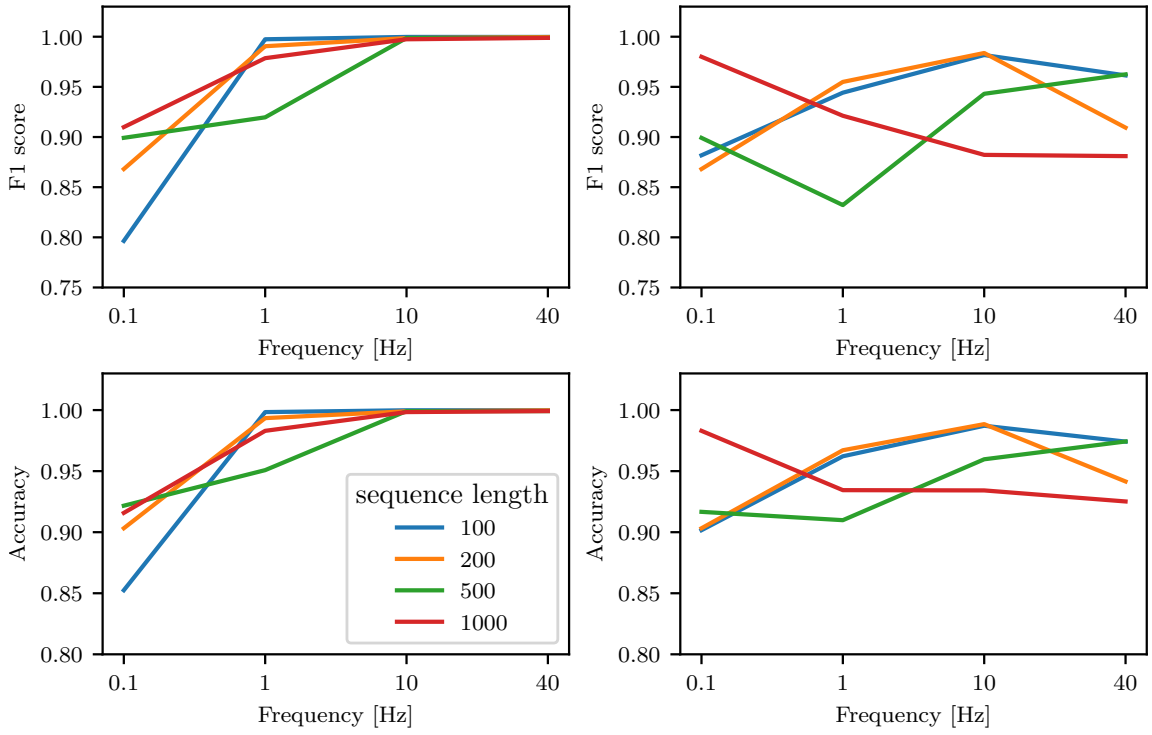
and precision is defined as

$$\text{precision} = \frac{\text{true positive events}}{\text{true positive events} + \text{false positive events}}. \tag{3.11}$$

Taking the average of the F1 score over all classes then provides a more informative measure of classifier performance than accuracy, especially when the classes are unbalanced.

We trained an LSTM using Adam with a learning rate of 0.001 and a minibatch size of 100. Because the dataset is small we regularized the model appropriately. A single hidden layer of 64 units was used with a dropout value of 0.3 and a weight decay factor of 0.01. For comparison, we use a hidden Markov model with 10 hidden states and an emission distribution of 5 Gaussian mixture models – based on manual hyperparameter exploration. For the HMM the train and validation datasets were combined to form the training data.

The F1 score and accuracy for both models are shown in figure 3.2. The results indicate that both models achieve decent classification performance on this dataset. LSTMs show a strong positive correlation between accuracy and resolution, whereas the HMM accuracy appears to be uncorrelated with frequency. Although we regularized the LSTM to minimize the effect of dataset size, there may still be traces of influence in the results. Therefore, we cannot conclude that higher resolution leads to better accuracy for LSTMs, but increasing the resolution seems harmless for LSTMs, which is not necessarily true for hidden Markov models. Larger high-resolution datasets could permit experiments that clarify this correlation for LSTMs.



(a) LSTM **(b)** HMM

Fig. 3.2 LSTM (a) and HMM (b) results on the intensive care unit dataset. The x-axis of each plot indicates the resolution of the data used to train and evaluate the models.

Because this is a small dataset, we used different subsequences from the same patient in both the test and train datasets. Therefore, these results are not an indication of how well these models can predict the outcome of unseen patients. Instead, the results serve as a comparison between the two models on high-resolution physiological signals.

In both our comparisons of LSTMs to HMMs, LSTMs were found to yield a better performance. HMMs provide an elegant methodology, but they suffer from a fundamental drawback: the structure of the HMM is often a poor model of the true process producing the data. The HMM has a finite number of hidden states, and in combination with the Markov property (the current state z_t only depends on the previous z_{t-1}), this could greatly inhibit the model's capacity (Dietterich, 2002). Some solutions to these problems have been proposed such as the infinite HMM (Beal et al., 2001), auto-regressive HMMs (Berchtold, 1999), input-output HMMs (Bengio and Frasconi, 1996; Chiappa and Bengio, 2004), and various other extensions (Bishop, 2006, §13.2.6). Most of the structural improvements provided by these solutions are also enabled by the flexible state representation provided by LSTMs. Nevertheless, LSTMs and HMMs are not the only models for analysing medical signals, on the contrary, there exist multiple other approaches. In the following section, we take a look at how LSTMs compare to contemporary techniques.

3.3 Evaluating LSTMs on the Physionet 2017 challenge

In the previous section, we answered the question of how LSTMs compare to a standard sequence model when applied to medical time series. Another question one might consider is how LSTMs compare to the latest approaches to medical signal analysis.

In most cases where medical data is analysed with machine learning techniques, the datasets are not available to the public, and as a result, comparing new methods to other studies is nearly impossible. The machine learning field has solved this problem with public online benchmark datasets such as ImageNet (Deng et al., 2009) and Penn treebank (Marcus et al., 1993). The recent advent of deep learning success has largely been attributed to this public benchmark approach (He et al., 2015). To our rescue comes the Physionet/Computation in Cardiology challenge (Clifford et al., 2015) with labelled medical time series datasets of high-resolution.

3.3.1 The Physionet 2017 challenge

The Physionet/Computation in Cardiology challenge is a public machine learning competition hosted annually with a specific task. This provides a great opportunity for researchers to compare their approaches to solving problems in cardiology. Each year the challenge provides a publicly downloadable dataset for participants to train on. Participants then submit their models to run on the server hosted by the challenge organisers. On the server, the model is tested on a held-out test dataset, and the participants are ranked according to their scores achieved on the test dataset.

To compare the LSTM to the flurry of other approaches possible, we partook in the Physionet 2017 challenge (Clifford et al., 2017). The 2017 challenge was to classify single-lead electrocardiograms (ECGs) into one of four classes. Details of the dataset are provided in table 3.4.

Table 3.4 Details of the Physionet 2017 dataset

Number of ECG recordings	12,186 (8,528 available for training)
Classes (number of patients)	normal (5,076) atrial fibrillation (758) other (2415) noisy (279)
Resolution	300 Hz
Recording duration [min,max]	[9s,61s]
Number of inputs	1

In this challenge, we endeavoured to find the best possible structure of an LSTM for the provided dataset. The following sections introduce contemporary techniques for improving neural networks that have not yet been introduced in this thesis.

3.3.2 Batch normalization

Standardizing or whitening input data has long been known to improve convergence rates of gradient-based optimization methods (LeCun et al., 2012). Ioffe and Szegedy (2015) go one step further to show that normalizing the pre-activations in each layer of a neural network over the minibatch of inputs greatly improves deep neural network convergence rates.

For a pre-activation $\mathbf{x} \in \mathbb{R}^d$, with d the size of the layer, the batch normalization transformation is

$$\mathrm{BN}(\mathbf{x};\lambda,\beta) = \lambda \odot \frac{\mathbf{x} - \widehat{\mathbb{E}}[\mathbf{x}]}{\sqrt{\widehat{\mathrm{Var}}[\mathbf{x}] + \varepsilon}} + \beta, \tag{3.12}$$

where $\varepsilon \in \mathbb{R}$ provides numerical stability. To maintain the representational capacity of neural networks, learnable parameters $\lambda \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ are introduced to determine the mean and standard deviation of the normalized activation. The division should be understood to proceed element-wise. The dataset statistics $\mathbb{E}[\mathbf{x}]$ and $\mathrm{Var}[\mathbf{x}]$ are estimated by the sample mean $\widehat{\mathbb{E}}[\mathbf{x}]$ and sample variance $\widehat{\mathrm{Var}}[\mathbf{x}]$ of the current minibatch. At test time, the statistics are typically estimated based on the entire training set.

A feedforward neural network has layer activations

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{3.13}$$

where ϕ is a nonlinear activation function applied element-wise. With batch normalization this becomes

$$\mathbf{y} = \phi(BN(\mathbf{W}\mathbf{x})), \tag{3.14}$$

where, the bias term is made redundant by the β parameter of batch normalization. Cooijmans et al. (2016) extended this to the LSTM such that

$$\begin{aligned}
\begin{pmatrix} \mathbf{i} \\ \mathbf{o} \\ \mathbf{f} \\ \mathbf{g} \end{pmatrix} &= \sigma\Big(\mathrm{BN}(\mathbf{U}\mathbf{h}_{t-1};\lambda_h,\beta_h) + \mathrm{BN}(\mathbf{W}\mathbf{x}_t;\lambda_x,\beta_x) + \mathbf{b}\Big) \\
\mathbf{c}_t &= \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tanh(\mathbf{g}) \\
\mathbf{h}_t &= \mathbf{o} \odot \tanh\Big(\mathrm{BN}(\mathbf{c}_t;\lambda_c,\beta_c)\Big),
\end{aligned} \tag{3.15}$$

where $\mathbf{U} \in \mathbb{R}^{4d\times d}$ is the concatenation of all the gate hidden weight matrices $c[\mathbf{U}_i,\mathbf{U}_o,\mathbf{U}_f,\mathbf{U}_g]$ and $\mathbf{U}_* \in \mathbb{R}^{d\times d}$. Similarly $\mathbf{W} \in \mathbb{R}^{4d\times s}$ is the concatenation of all the gate input weight matrices $c[\mathbf{W}_i,\mathbf{W}_o,\mathbf{W}_f,\mathbf{W}_g]$ and $\mathbf{W}_* \in \mathbb{R}^{d\times s}$. In practice, $\beta_h = \beta_x = 0$ to allow the existing parameter $\mathbf{b}$ to account for both biases.

In addition to decreasing the convergence time of models, batch normalization also acts as a regularizer. You can often use batch normalization instead of dropout because it adds both additive and multiplicative noise to the hidden units at training time (Goodfellow et al., 2016, §17.2).

3.3.3 Truncated back-propagation through time

The training of recurrent neural networks is difficult because the gradients tend to vanish or explode as a result of the multiplications through time. Sutskever (2013) proposed a variant of back-propagation where every k_1 steps the back-propagation algorithm is run for k_2 steps. This allows the number of steps k_2 the gradient has to be passed through to be small enough to prevent gradient problems. Long-term information is still retained by having the hidden states developing forward in the sequence as per usual. The technique is known as truncated back-propagation through time (TBPTT). LSTMs already mitigate the vanishing and exploding gradient problem to some extent, however, but the TBPTT technique results in multiple parameter updates for a single pass through a sequence. In turn, this could allow LSTMs to have quicker convergence rates.

3.3.4 Dark knowledge

When training neural networks for classification, we usually use "hard" one-hot encoded labels (e.g., $\mathbf{y} = [0,0,1,0]$). This teaches the model to assign labels with high probability, which could be detrimental if the labels are incorrect, or if there is some overlap between classes. Hinton et al. (2015) proposed a technique to distil knowledge from a **teacher** network to a **student** network by means of "soft" labels. Generally, neural networks produce class probabilities $p(y_i)$ by using a softmax function on the final layer logits $\mathbf{z}$

$$p(y_i) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)} \tag{3.16}$$

where the temperature $\tau = 1$. Higher values of τ result in "softer" probability distributions over the classes. The proposed technique entails training a teacher network with a high temperature. A student network is then trained with the same temperature, but the targets are the softmax outputs produced by the teacher network for each data point. After the student network has been trained it uses a temperature of 1. **Dark knowledge** was coined to refer to the "hidden" knowledge inside the teacher network.

This technique not only allows models to generalize better, it also allows smaller (distilled) networks to learn from larger (cumbersome) networks and yield similar performances. In our case this is particularly beneficial because we have an abundance of computational resources (multiple GPUs) during training time, but are severely limited at testing time[2]. This allowed us to train large models, and distil the knowledge to networks that would fit within

[2]Online submissions were evaluated with a single CPU core, 2GB of RAM, and a maximum of 2×10^{11} CPU instructions per data point.

the computational constraints of the Physionet 2017 challenge. We implemented a slight variation of dark knowledge, where we replace "hard" labels with "soft" labels if the "soft" labels produced by the teacher network were correct. $\tau = 5$ was used in all experiments.

3.3.5 Results

For all the benefits of the Physionet challenge, evaluating a model on the challenge server is arduous, and a maximum of five model submissions can be made per month. Hence, we only submitted models that showed promising results on our own randomly selected validation dataset. Before training each model variation a random 10% of the training data points were separated to create the validation dataset.

Generalization proved to be the biggest challenge on this dataset, which was most likely a result of the relatively small dataset for the objective. Models achieved high F1 scores on the training, demonstrating that learning from this data is possible. Deep learning techniques have been known to suffer from overfitting if there is limited data (Goodfellow et al., 2016). As the reader will notice, most of the model tuning was aimed at improving model regularization.

A common approach to increase the model generalization is data augmentation (Goodfellow et al., 2016). This process increases the size of the training data by transformed versions of the original data. These transformations do not affect the relevant properties of the data, for example, arbitrary rotations are commonly used for image datasets. For electrocardiogram (ECG) signals, this is slightly trickier. One technique is to add Gaussian noise to the ECG signals, which are naturally quite noisy. Another data augmentation approach is to start the sequence segmentation from different locations. This creates additional data points that have been shifted in time for the model to be able to "see" data points starting and ending with different conditions.

Zhang et al. (2017) argued that modern optimizers, such as Adam (Kingma and Ba, 2015) and *RMSProp* (Hinton et al., 2012) could exacerbate overfitting due to their per-parameter adaptive learning rates. Experimentation with standard stochastic gradient descent and Nesterov accelerated momentum (Bengio et al., 2013) on our models did not improve model generalization, and the Adam optimizer yielded better accuracies.

In table 3.5 we summarize the empirical design choices. Bold options depict the best settings. Experimental details are provided in Appendix B. For all of the models, we decayed the learning rate by a factor of 10 when the training loss decreased by less than 10^{-3} over 5 epochs.

Of the design choices listed in table 3.5, the best performing models on the validation dataset were submitted for evaluation on the Physionet Challenge server. The F1 scores (eq. 3.9) achieved on our validation set and the Physionet test set are listed in table 3.6. Details of

Table 3.5 LSTM design choices

Feature	Options	Comment
Segmentation length	500, **1000**, 1500	Longer sequences were found better for detecting atrial fibrillation and a sequence length of 1000 provided the best balance for all the classes.
Number of units	64, 128, 256, 300, **512**	Wider models provided sufficient capacity whilst generalizing well. 512 was the largest layer that could fit on the test machine.
Number of layers	**1**, 2, 3	Deeper models did better on the noise class but had a slightly worse performance overall.
Dropout	0.5, 0.2, **0.1**, 0.05, 0.0	No dropout made training harder and so did high dropout percentages.
Weight decay	1e-3, 1e-4, 1e-6, **0**	No weight decay yielded the best results. Weight decay would often lead to the production of invalid loss values.
Batch normalization	on, **off**	This strongly regularized the model and worsened the overall score. Owing to the calculation of minibatch statistics at each time step, this proved too computationally expensive during training.
TBPTT	$k_1 = k_2 =$ 250, 500, **1000**	As expected, this improved convergence speed slightly, but marginally reduced accuracy.
Dark knowledge	same size teacher, **larger teacher**	This technique always improved results and made training of the student network easier. Best results were achieved with a 1x1024 hidden layer teacher network and a 1x128 student network.
Gaussian data augmentation	$\sigma^2 =$ 0.05, **0.1**, 1.0	Augmenting the dataset with Gaussian noise added copies of the data improved regularization of the model, and marginally improved the scores.
Shifted data augmentation	on, **off**	This exacerbated overfitting. This is a good sign, indicating that the LSTM automatically learns to be agnostic to the initial conditions of the sequence.

each model are provided in the first column. Validation dataset scores are the best values obtained over 300 training epochs and the same model was used for the test dataset. All the models reported here used Adam as the optimizer with a learning rate of 0.001. The subsequence length was 1000 and dropout was set to 0.1.

Table 3.6 Physionet 2017 challenge results

Model description	Valid F1	Test F1
1x512	0.664	0.710
2x128, batch normalized	0.674	0.713
1x512, soft labels from 512 teacher	0.690	0.730
1x300, soft labels from 512 teacher	0.682	0.737
1x512, soft labels from 1024 teacher, Gaussian data augmentation	0.694	0.743

The results demonstrate how beneficial dark knowledge was for this challenge. A good validation set F1 score did not always translate into a good test F1 score. The difficulty with overfitting becomes clear when we compare the validation set scores to the test set scores. Usually, the validation score is expected to be higher than the test score because the test dataset is larger. In this case, there was a large difference between the training and validation scores, and the test scores ended up being slightly higher than the validation scores.

The Physionet 2017 challenge winner had an F1 score of 83. The top models included an LSTM combined with a gradient boosted tree (Teijeiro et al., 2017), an Adaboost cascaded binary classifier (Datta et al., 2017), random forests (Zabihi et al., 2017), and a combination of convolutional neural networks, LSTMs, and gradient boosted trees (Hong et al., 2017). All of the mentioned approaches used medical expert selected features as input to the models. Clever feature extraction can allow models to generalize better when the dataset is small. Moreover, the best performing deep learning models were combined with "shallow" models, such as gradient boosted trees, that are good for small data scenarios (Chen and Guestrin, 2016).

Hand-crafted features bolster the classification performance of deep learning techniques, but they sacrifice the strength of deep learning models as feature constructors. With enough data, deep learning may provide a meaningful and more data-driven approach to patient monitoring. This could identify new shared mechanisms that would otherwise be obscured by ad hoc historical definitions of disease. Thus, by re-evaluating data without our assumptions, deep learning could reveal novel medical insights (Ching et al., 2018).

The promise of LSTMs is demonstrated by their ubiquity among the top performing approaches for the Physionet 2017 challenge. It is clear that we are still at a stage where the combination of LSTMs, shallow models, and feature extraction outperforms an LSTM-only

approach for medical problems such as this challenge. It could be that the groundbreaking solution for LSTMs, analogous to the one found for CNN's (Krizhevsky et al., 2012), has not yet been engineered, meaning that standard machine learning approaches such as random forests, still prevail. However, LSTMs still have many benefits to be exploited for biological time series. We explore one such benefit in the next section.

3.4 Obtaining uncertainty measures for LSTMs in medicine

Recurrent neural networks (RNNs) have shown promising results on temporal medical data (Choi et al., 2016a; Harutyunyan et al., 2017; Jagannatha and Yu, 2016; Lipton et al., 2015b). However, when classifying or predicting with RNNs, classical approaches don't provide us with a measure of uncertainty. In most real-life scenarios, knowing what a machine learning model doesn't know is of utmost importance, especially when the model provides guidance to medical practitioners. Medical diagnosis models might consistently classify with high confidence, even in cases where they should flag difficult examples for human intervention. Such incorrect diagnoses could prevent adoption of these models in practice.

Comment: One may wonder whether the probabilities produced by the softmax function, often used during multi-class classification, already provide us with an estimate of the uncertainty. The softmax function only provides us with a measure of the uncertainty there is among the different possible classes and not the uncertainty among different input data points (Gal and Ghahramani, 2016a; Kendall et al., 2015). Say, for example, we train a model to classify between images of cats and dogs. When we show this model an image of an apple, the model might assign a higher probability to the dog class, but we would like to know the uncertainty the model has in classifying the image of the apple compared to the uncertainty in classifying an image of a cat or dog.

Bayesian probability theory offers a mathematically grounded technique to reason about model uncertainty (Gal and Ghahramani, 2016a). Earlier studies have explored the benefits of Bayesian techniques in medicine (Ghassemi et al., 2015; Guiza Grandas et al., 2006; Kononenko, 2001; Mani et al., 2014; Meyfroidt et al., 2009). However, these proposals do not harness the representative power exhibited by deep learning (Ongenae et al., 2013). Fortunately, some successful efforts have been able to frame neural networks in a Bayesian manner (MacKay, 1992b; Neal, 1995), but these techniques are often accompanied by a prohibitive computational cost. Recently Gal and Ghahramani (2016a) proposed a technique that allows standard neural networks to produce Bayesian uncertainties.

To start explaining how standard neural networks can be made Bayesian, we have to start with dropout. Recall that dropout is a regularization technique commonly used in neural networks to prevent overfitting and co-adaption of features (Srivastava et al., 2014). This is achieved by randomly setting the activations of units in the neural network to zero (dropping) with some specified probability p. The standard approach is to rescale the weights at test time by the **keep probability** $1-p$, known as **weight averaging**.

Rather conveniently, dropout can be used as approximate Bayesian inference over the weights of a network. This is achieved by sampling from the network with random units removed (dropped) at test time. These are empirical samples from the approximate posterior over the weights. Monte Carlo estimation can be used to find the predictive mean, giving rise to the name **Monte Carlo dropout**. In this setting, the network does not require any additional parameters and a Bernoulli approximating distribution is imposed over the weights. Monte Carlo dropout has successfully been implemented for recurrent neural networks (Gal and Ghahramani, 2016b; Sennrich et al., 2016), but the benefits have yet to be demonstrated in the medical domain. Next, we describe in a little more detail how Monte Carlo dropout can be used to form Bayesian neural networks.

3.4.1 Bayesian neural networks

First, some preliminaries. Given training inputs $\mathcal{X} = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$ and their corresponding outputs $\mathcal{Y} = \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(N)}\}$, a neural network, parametrized by ω, can be formulated as $\mathbf{y} = f^{\omega}(\mathbf{x})$. This allows us to define a likelihood distribution $p(\mathbf{y}|\mathbf{x}, \omega)$, which for classification tasks is the softmax likelihood

$$p(y = d|\mathbf{x}, \omega) = \frac{\exp(f_d^{\omega}(\mathbf{x}))}{\sum_{d'} \exp(f_{d'}^{\omega}(\mathbf{x}))} \tag{3.17}$$

Given this formulation, we closely follow Gal and Ghahramani (2016a) to explain neural networks in a Bayesian framework. We start with Bayes' theorem

$$p(\omega|\mathcal{Y}, \mathcal{X}) = \frac{p(\mathcal{Y}|\mathcal{X}, \omega)p(\omega)}{p(\mathcal{Y}|\mathcal{X})}, \tag{3.18}$$

which provides us with the posterior over the parameters $p(\omega|\mathcal{Y}, \mathcal{X})$. To classify a new input and output pair $(\mathbf{x}', \mathbf{y}')$, we integrate out the parameters

$$p(\mathbf{y}'|\mathbf{x}', \mathcal{X}, \mathbf{Y}) = \int p(\mathbf{y}'|\mathbf{x}', \omega)p(\omega|\mathcal{Y}, \mathcal{X})\,\mathrm{d}\omega. \tag{3.19}$$

The true posterior $p(\omega|\mathcal{X},\mathcal{Y})$ is usually hard to evaluate because it requires taking expectations with respect to complicated distributions. Instead, we define an approximating variational distribution $q_\theta(\omega)$, parametrized by θ, with a structure that is easy to evaluate. At test time we can now use Monte Carlo integration over the approximate posterior to get an estimate of the prediction

$$p(\mathbf{y}'|\mathbf{x}',\mathcal{X},\mathcal{Y}) \approx \frac{1}{k}\sum_{i=1}^{k} p(\mathbf{y}'|\mathbf{x}',\widehat{\omega}^{(i)}) \xrightarrow[k\to\infty]{} \int p(\mathbf{y}'|\mathbf{x}',\omega)q_\theta(\omega)\,\mathrm{d}\omega \tag{3.20}$$

where $\widehat{\omega}^{(i)} \sim q_\theta(\omega)$, and k is the number of Monte Carlo samples. For a neural network with a specific $q_\theta(\omega)$ eq. 3.20 can be realized by applying dropout during test time and computing the average of the k stochastic forward passes (Monte Carlo dropout). Thus, dropout provides a means of obtaining an approximate posterior predictive. Note the difference between this and weight averaging mentioned at the start of section 3.4.

For this approach to work, the approximate posterior $q_\theta(\omega)$ should be as close as possible to the true posterior $p(\omega|\mathcal{X},\mathcal{Y})$. The Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) provides a measure of the similarity between distributions which we can minimize to have $q_\theta(\omega)$ accurately approximate the true posterior

$$\mathrm{KL}(q_\theta(\omega)||p(\omega|\mathcal{Y},\mathcal{X})) = \int q_\theta(\omega)\log\frac{q_\theta(\omega)}{p(\omega|\mathcal{Y},\mathcal{X})}\,\mathrm{d}\omega. \tag{3.21}$$

Unfortunately evaluating $p(\omega|\mathcal{Y},\mathcal{X})$ is hard since it requires evaluating the intractable normalization constant $p(\mathcal{Y}|\mathcal{X})$. However, the unnormalized distribution $\tilde{p}(\omega) := p(\omega,\mathcal{Y}|\mathcal{X}) = p(\omega|\mathcal{Y},\mathcal{X})p(\mathcal{Y}|\mathcal{X})$ is tractable to compute. We can minimize the KL divergence between $q_\theta(\omega)$ and $\tilde{p}(\omega)$ which leads to the following set of equations

$$\begin{aligned} J(\theta) &= \int q_\theta(\omega)\log\frac{q_\theta(\omega)}{\tilde{p}(\omega)}\,\mathrm{d}\omega \\ &= \int q_\theta(\omega)\log\frac{q_\theta(\omega)}{p(\mathcal{Y}|\mathcal{X},\omega)p(\omega)}\,\mathrm{d}\omega \\ &= \int q_\theta(\omega)\log\frac{q_\theta(\omega)}{p(\omega)}\,\mathrm{d}\omega - \int q_\theta(\omega)\log p(\mathcal{Y}|\mathcal{X},\omega)\,\mathrm{d}\omega \\ &= \mathrm{KL}(q_\theta(\omega)||p(\omega)) + \mathbb{E}_{q_\theta(\omega)}[-\log p(\mathcal{Y}|\mathcal{X},\omega)]. \end{aligned} \tag{3.22}$$

Since the KL divergence is always non-negative, the objective $J(\theta)$ is an upper bound on the negative log-likelihood (second term), which we would like to minimize. Thus, by minimizing the upper bound we perform what is known as **variational inference** (Murphy,

2012, §21.2), with the objective

$$J_{VI}(\theta) = \mathrm{KL}(q_\theta(\omega)||p(\omega)) + \mathbb{E}_q[-\log p(\mathcal{Y}|\mathcal{X},\omega)] \geq -\log p(\mathcal{Y}|\mathcal{X}). \tag{3.23}$$

We can approximate the negative log-likelihood term as follows

$$\begin{aligned}\mathbb{E}_{q_\theta(\omega)}[-\log p(\mathcal{Y}|\mathcal{X},\omega)] &= -\sum_{n=1}^{N} \int \log p(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\omega) q_\theta(\omega)\,\mathrm{d}\omega \\ &\approx -\sum_{n=1}^{N} \log p(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\widehat{\omega}^{(n)}),\end{aligned} \tag{3.24}$$

where the integral over $q_\theta(\omega)$ is approximated with a single sample $\widehat{\omega}^{(n)} \sim q_\theta(\omega)$, which is an unbiased estimator (Gal and Ghahramani, 2016a). We are left with a Monte Carlo estimate of the variational objective

$$J_{MC}(\theta) = -\sum_{n=1}^{N} \left(\log p(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\widehat{\omega}^{(n)})\right) + \mathrm{KL}(q_\theta(\omega)||p(\omega)) \tag{3.25}$$

where $\mathbb{E}_{X,Y,\omega}[J_{MC}(\theta)] = J_{VI}(\theta)$.

Neural networks are usually trained by minimizing the negative log-likelihood and the L^2-norm of the weights (weight decay)

$$J_{drop}(\omega) = -\frac{1}{N}\sum_{n=1}^{N} \left(\log p(\mathbf{y}^{(n)}|\mathbf{x}^{(n)},\omega)\right) + \lambda \left\|\omega\right\|_2^2, \tag{3.26}$$

where λ is a value chosen ahead of time and controls the strength of our preference for smaller weights. With dropout, a single sample of the weights $\widehat{\omega}^{(n)} \sim q_\theta(\omega)$ is used in the summand, as in eq. 3.24. Here the approximate posterior is given by

$$q_\theta(\omega) := \int q_\theta(\omega|\varepsilon) p(\varepsilon)\,\mathrm{d}\varepsilon, \tag{3.27}$$

with

$$\begin{aligned} q_\theta(\omega|\varepsilon) &= \delta(\omega - g(\theta,\varepsilon)) \\ g(\theta,\varepsilon) &= \{\mathrm{diag}(\varepsilon^{(l)})\mathbf{M}^{(l)}, \mathbf{b}^{(l)}\}_{l=0}^{L}, \end{aligned} \tag{3.28}$$

where $\theta = \{\mathbf{M}^{(l)}, \mathbf{b}^{(l)}\}_{l=0}^{L}$ are the deterministic parameters for a neural network with L hidden layers and $\varepsilon^{(l)}$ is a vector of binary random variables which take value 0 with probability $p^{(l)}$ – the dropout probability. Thus, $p(\varepsilon^{(l)})$ is a product of Bernoulli distributions with

probabilities $1-p^{(l)}$, from which a realization would be a vector of zeros and ones. Note that the first terms of eq. 3.26 and eq. 3.25 are the same apart from a scaling factor N. By approximating $q_{\theta_r^{(l)}}(\omega_r^{(l)}|\varepsilon_r^{(l)})$ for each row r of the weight matrix on layer l as a narrow Gaussian, Gal (2016, §3.2.2) showed that the gradients of the objectives $J_{MC}(\theta)$ and $J_{drop}(\theta)$ are related as[3]

$$\frac{1}{N}\frac{\partial}{\partial\theta}J_{MC}(\theta)=\frac{\partial}{\partial\theta}J_{drop}(\theta) \tag{3.29}$$

if

$$\frac{\partial}{\partial\theta}\mathrm{KL}\left(q_\theta(\omega)||p(\omega)\right)=\frac{\partial}{\partial\theta}N\lambda\left\|\omega\right\|_2^2 \tag{3.30}$$

and eq. 3.30 holds if we set the model prior to $p(\omega)=\mathcal{N}(0,\mathbf{I}/\ell^2)$, where

$$\ell^2=\frac{2N\lambda}{1-p} \tag{3.31}$$

with p as the dropout probability.

We can now proceed with the standard neural network (eq. 3.26) to obtain model uncertainties. At test time, probability vectors from k stochastic forward passes are collected. Classifications are determined by averaging the probabilities over k as in eq. 3.20. Although the Monte Carlo dropout approach is computationally more expensive than the standard approach, it is highly parallelizable due to the samples being independent. Predictive entropy (Gal, 2016, §3.3.1) can then be used to obtain a measure of uncertainty

$$\begin{aligned}\mathbb{H}[y'|\mathbf{x}',\mathcal{X},\mathcal{Y}] &:= -\sum_c\left(\frac{1}{k}\sum_{i=1}^k p(y'=c|\mathbf{x}',\widehat{\omega}^{(i)})\right)\log\left(\frac{1}{k}\sum_{i=1}^k p(y'=c|\mathbf{x}',\widehat{\omega}^{(i)})\right)\\ &\xrightarrow[k\to\infty]{} -\sum_c\left(\int p(y'=c|\mathbf{x}',\omega)q^*(\omega)\,\mathrm{d}\omega\right)\log\left(\int p(y'=c|\mathbf{x}',\omega)q^*(\omega)\,\mathrm{d}\omega\right)\\ &\approx -\sum_c p(y'=c|\mathbf{x}',\mathcal{X},\mathcal{Y})\log p(y'=c|\mathbf{x}',\mathcal{X},\mathcal{Y}),\end{aligned} \tag{3.32}$$

where $q^*(\omega)$ is the optimum of eq. 3.25 and $\widehat{\omega}^{(i)}\sim q^*(\omega)$. Classes are represented by c. The predictive entropy will reach its minimum value (zero) when one class has probability 1. Next, we describe the Bayesian neural network extended to LSTMs.

[3]The dropout objective has been reparametrized.

3.4.2 Bayesian LSTMs

In this section we closely follow Gal and Ghahramani (2016b) to formulate Bayesian LSTMs. We start with the definition of an LSTM (similar to eq. 2.3)

$$\begin{aligned}\mathbf{i},\mathbf{o},\mathbf{f} &= \sigma(\mathbf{U}_{i,o,f}\mathbf{h}_{t-1} + \mathbf{W}_{i,o,f}\mathbf{x}_t + \mathbf{b}_{i,o,f})\\ \mathbf{c}_t &= \mathbf{f}\odot\mathbf{c}_{t-1} + \mathbf{i}\odot\tanh(\mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t + \mathbf{b}_c)\\ \mathbf{h}_t &= \mathbf{o}\odot\tanh(\mathbf{c}_t).\end{aligned} \tag{3.33}$$

We can re-write the operation as a function of f_h[4]:

$$\begin{aligned}\mathbf{h}_t &= f_h(\mathbf{x}_t,\mathbf{c}_{t-1},\mathbf{h}_{t-1})\\ &= \sigma(\mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t)\odot\tanh\Big(\sigma(\mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t)\odot\mathbf{c}_{t-1}\\ &\quad + \sigma(\mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t)\odot\tanh(\mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t)\Big),\end{aligned} \tag{3.34}$$

where $\mathbf{c}_{t-1}$ is the vector of hidden unit memories from the previous time step and is determined by a recursive function on $\mathbf{h}_{t-2}$. The output can be defined as $f_y(\mathbf{h}_T) = \mathbf{W}_y\mathbf{h}_T$. The LSTM could be framed as a probabilistic model by regarding the weights, $\omega = \{\mathbf{W}_*, \mathbf{U}_*\}$ to be random variables (following Gaussian prior distributions). From here we write the functions as f_h^ω and f_y^ω to denote the dependence on ω. With the described framework, log-likelihood for a single input and output pair $(\mathbf{x},\mathbf{y})$ becomes

$$\begin{aligned}-\int\log p(\mathbf{y}|f_y^\omega(\mathbf{h}_T))q(\omega)\,\mathrm{d}\omega &= -\int\log p\Big(\mathbf{y}|f_y^\omega(f_h^\omega(\mathbf{x}_T,\mathbf{c}_{T-1},\mathbf{h}_{T-1}))\Big)q(\omega)\,\mathrm{d}\omega\\ &= -\int\log p\Big(\mathbf{y}|f_y^\omega(f_h^\omega(\mathbf{x}_T,\mathbf{c}_{T-1},f_h^\omega(...f_h^\omega(\mathbf{x}_1,\mathbf{c}_0,\mathbf{h}_0)...)))\Big)q(\omega)\,\mathrm{d}\omega,\end{aligned} \tag{3.35}$$

with $\mathbf{h}_0 = \mathbf{c}_0 = \mathbf{0}$. As in eq. 3.24 we approximate the integral with a single sample $\widehat{\omega}^{(i)} \sim q(\omega)$ for the approximate variational minimization objective to become

$$\begin{aligned}J(q) \approx -\sum_{n=1}^{N}\log p\big(\mathbf{y}^{(n)}|f_y^{\widehat{\omega}^{(n)}}(f_h^{\widehat{\omega}^{(n)}}(\mathbf{x}_T^{(n)},\mathbf{c}_{T-1},f_h^{\widehat{\omega}^{(n)}}(...f_h^{\widehat{\omega}^{(n)}}(\mathbf{x}_1^{(n)},\mathbf{c}_0,\mathbf{h}_0)...)))\big)\\ + \mathrm{KL}(q(\omega)||p(\omega)).\end{aligned} \tag{3.36}$$

[4] We omit biases for brevity

According to eq. 3.29 this objective is the same as our standard LSTM objective scaled by the number of data points N

$$J_{LSTM}(\omega) = -\frac{1}{N}\sum_{n=1}^{N} \log p\big(\mathbf{y}^{(n)} | f_y^{\widehat{\omega}^{(n)}}(f_h^{\widehat{\omega}^{(n)}}(\mathbf{x}_T^{(n)}, \mathbf{c}_{T-1}, f_h^{\widehat{\omega}^{(n)}}(...f_h^{\widehat{\omega}^{(n)}}(\mathbf{x}_1^{(n)}, \mathbf{c}_0, \mathbf{h}_0)...))))\big) + \lambda \|\omega\|_2^2. \tag{3.37}$$

Owing to the derivatives of the variational objective and the LSTM objective being the same (eq. 3.30), our LSTM has now been shown to be Bayesian. Evaluating the model output $f_y^{\widehat{\omega}}(\cdot)$ with sample $\widehat{\omega}^{(n)}$ corresponds to randomly zeroing (masking) columns in the weight matrices $\mathbf{W}_*$ and $\mathbf{U}_*$ during the forward pass – i.e., performing dropout.

Gal and Ghahramani (2016b) recommends sampling a single realization $\{\widehat{\mathbf{W}}_*^{(n)}, \widehat{\mathbf{U}}_*^{(n)}\}$ for each sequence $\mathbf{x}_{1:T}^{(n)}$ such that the same function $f_h^{\widehat{\omega}^{(n)}}$ is applied to each sequence entry $\mathbf{x}_t^{(n)}$. This means that the same dropout mask should be applied to the network units at each step and is referred to as **variational dropout**. However, standard practice for LSTMs, has been to apply different dropout masks at each step t with $\widehat{\omega}_1^{(n)}, ..., \widehat{\omega}_T^{(n)} \sim q(\omega)$. To distinguish between the two forms of dropout we refer to this standard approach as **naive dropout**.

We evaluated the efficacy of both naive and variational dropout on various datasets. When using naive dropout the LSTM minimization objective can be written as

$$J_{LSTM}(\omega) = -\frac{1}{N}\sum_{n=1}^{N} \log p\big(\mathbf{y}^{(n)} | f_y^{\widehat{\mathbf{W}}_y^{(n)}}(f_h^{\widehat{\omega}_T^{(n)}}(\mathbf{x}_T^{(n)}, \mathbf{c}_{T-1}, f_h^{\widehat{\omega}_{T-1}^{(n)}}(...f_h^{\widehat{\omega}_1^{(n)}}(\mathbf{x}_1^{(n)}, \mathbf{c}_0, \mathbf{h}_0)...))))\big) + \lambda \|\omega\|_2^2, \tag{3.38}$$

where T is the sequence length and $\widehat{\omega}_t^{(n)} = \{\widehat{\mathbf{W}}_{i,o,f}^{(n),t}, \widehat{\mathbf{U}}_{i,o,f}^{(n),t}\}$ are the realizations of the weights at each time step t as a result of dropout. A comparison of these approaches is provided in section 3.4.4, but first, we introduce three new datasets.

3.4.3 Datasets

Three new datasets were used for experiments with the Bayesian LSTM.

MNIST

A popular machine learning benchmark dataset is the MNIST handwritten digit dataset (LeCun, 1998). The dataset consists of 55,000:5,000:10,000 train:validation:test images, with pixel values $x \in [0,1]^{28x28}$ as provided by Tensorflow (Abadi et al., 2015). When

analysing images with LSTMs the pixels are processed in scanline order (Cooijmans et al., 2016). In our analysis, the MNIST dataset provided assurance that the model performed as intended.

MIT-BIH Arrhythmia

Another easy-to-interpret dataset is the MIT-BIH arrhythmia dataset (Goldberger et al., 2000; Moody and Mark, 2001). The dataset comprises 48 half-hour electrocardiogram (ECG) recordings from 47 patients on two channels. Channel 2 had greater variation in the leads and more noise than channel 1, thus only the recordings on channel 1 were used. These single-lead ECG signals were filtered using a bandpass finite impulse response filter, with high- and lowpass frequencies of 3 Hz and 45 Hz. Single heartbeats were then extracted from the longer, filtered signals by means of the Hamilton QRS detector (Hamilton, 2002). Both the filtering and heartbeat segmentation was performed using the *BioSPPy* package (Carreiras et al., 2015) in Python 3. Of the various heartbeat categories, we include only those from the four categories that are best represented over different patients in the dataset. The patients were randomly split to have heartbeats from 33 train-, 5 validation-, and 9 test-patients (a ratio of 70:10:20). An acceptable split was considered to have all classes in each set contain at least $0.9\gamma \times$ *smallest-class-size* data points, where γ is the split-fraction (0.7, 0.1, or 0.2). Further details of the dataset are provided in table 3.7.

Table 3.7 Details of the MIT-BIH arrhythmia dataset

Number of patients	47 (22 females)
Age range	23 to 89
Classes	normal beat right bundle branch block beat paced beat premature ventricular contraction
Measurement resolution	360 Hz
Chosen segment lengths	216 time steps (single beat)
Number of data points	89,670
Average recording duration	30m
Input signals	single-lead ECG signal

Physionet 2016 challenge

Section 3.3 introduced the Physionet/Computation in cardiology challenge. Here we introduce the phonocardiogram dataset of the Physionet 2016 challenge (Clifford et al., 2016; Liu

et al., 2016). This comprehensive dataset was recently collected, is multi-centre, and has been analysed in multiple studies. Details of the dataset are provided in table 3.8.

What makes this dataset interesting, are the long- and short-term dependencies, which make accurate classification difficult. Moreover, being able to accurately detect heartbeat arrhythmias from phonocardiograms is an extremely important problem to solve, especially for developing communities (Springer et al., 2016).

Among the top performing techniques for the 2016 challenge were convolutional neural networks, random forests, regularized neural networks, and an ensemble of support vector machines (Clifford et al., 2016). The winning team had a score of 0.860 and used expert feature extraction techniques with Adaboost and CNNs as the classifier (Potes et al., 2016). The runner-up had a score of 0.859. They also used feature extraction, but did not segment the signal and made use of an ensemble of 20 neural networks (Zabihi et al., 2016). This score, provided by the online evaluation, is based on sensitivity (eq. 3.1) and specificity (eq. 3.2) (Clifford et al., 2016).

Table 3.8 Details of the Physionet 2016 dataset

Number of heart sound recordings	4,430 (3,126 available for training)
Classes	normal abnormal
Measurement resolution	2 kHz
Analysed resolution	1 kHz
Chosen segment lengths	1000 time steps (1s)
Recording duration [min,max]	[7s,287s]
Number of inputs	1

The first step in our preprocessing of the phonocardiograms (PCGs) was to normalize each signal independently to have zero mean and unit variance, in order to minimize the changes required in weight values of the LSTM during training. The second step was to decimate each signal to 1 kHz. Owing to LSTMs being difficult to train on extremely long sequences (Neil et al., 2016), the third step was to segment the signals into subsequences with a length of at most 1,000 time steps. Because the test set is already provided by the challenge, we construct a validation set from a random 10% of the subsequences obtained from each recording, and use the remainder for training.

Although only two classes were provided in the training data, classifying a signal into a third class, noisy, was allowed and would result in less of a penalty on a participants score compared to an incorrect classification. Essentially this allows the model to indicate when it's too uncertain about the data point to be able to classify it, which is a built-in feature of our Bayesian LSTM!

The uncertainty threshold for new samples is determined by first collecting the probability vectors of k stochastic forward passes over the combined train and validation dataset. The first and second moments of the predictive entropy (eq. 3.32) are then calculated

$$\mu_{VI} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{H}_k[y^{(n)}|\mathbf{x}^{(n)}, \mathcal{X}, \mathcal{Y}]$$
$$\sigma_{VI}^2 = \frac{1}{N}\sum_{n=1}^{N} (\mathbb{H}_k[y^{(n)}|\mathbf{x}^{(n)}, \mathcal{X}, \mathcal{Y}] - \mu_{VI})^2, \tag{3.39}$$

where N indicates the number of data points and $\mathbb{H}_k[\cdot]$ denotes that the predictive entropy is computed over k stochastic forward passes. A new data point will yield an uncertain classification (noise class) when

$$\mathbb{H}_k[y'|\mathbf{x}', \mathcal{X}, \mathcal{Y}] > \mu_{VI} + 3\sqrt{\sigma_{VI}^2}. \tag{3.40}$$

3.4.4 Results

Before comparing the efficacy of Bayesian LSTMs to standard LSTMs for medical data, we analysed the naive and variational dropout approaches. We reiterate that for the variational approach each epoch represents a single sample (dropout mask) of the weights. For the naive approach, each epoch sees multiple different weight samples for the same input sequence. For this comparison two identical LSTM models were trained on the MNIST dataset (section 3.4.3) for 120 epochs, each with a single hidden layer of 256 units, a dropout of 0.1, and weight decay set to 10^{-7}. Adam was used for optimization with a learning rate of 0.001.

On MNIST, we found it impossible to successfully train a model with variational dropout applied to the hidden and input layers, whereas naive dropout could be trained successfully. The reason could be that MNIST sequences, as many of the sequences analysed here, have single inputs. With variational dropout applied to the input layer, the single input could be removed for all time steps, and the model would see a sequence of zeros as input. Thus, for the variational and naive approaches, we apply dropout to only the hidden activations that are propagated to succeeding layers or time steps. In this regime, the variational approach would still occasionally fail catastrophically. Dropping the same activations for all time steps could have detrimental effects; for MNIST, where there are long subsequences of zeros, dropping the activations responsible for long-range dependencies could lead to zero gradients. We found that using a smaller weight decay factor alleviates this issue to some extent. For the results reported here, we retrained the variational dropout LSTMs if they completely failed to learn.

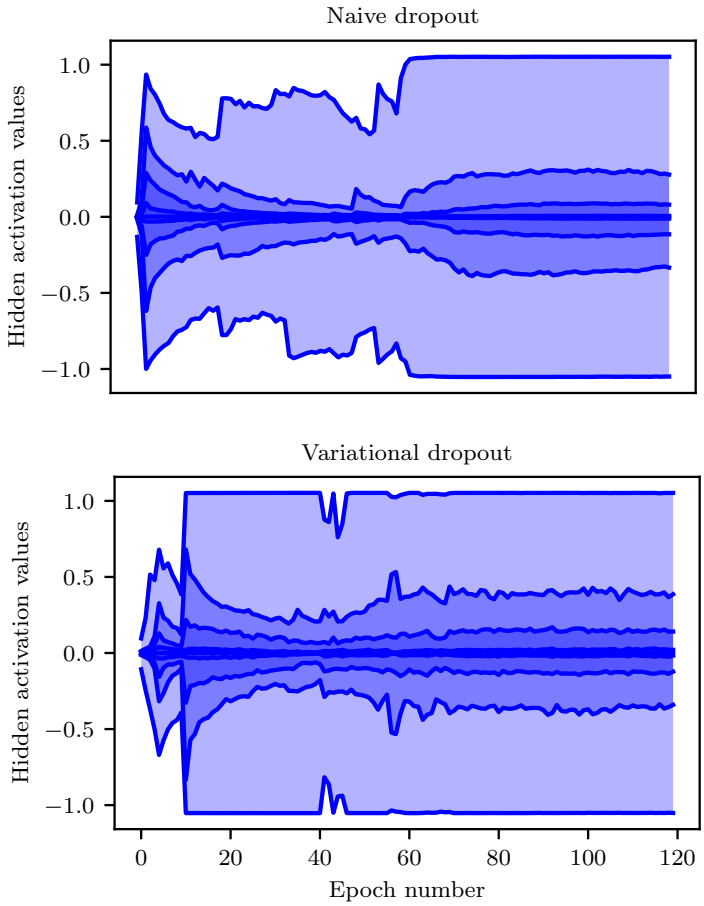


Fig. 3.3 Hidden activation distributions for the naive and variational dropout approaches. From top to bottom, the graph lines represent the maximum, 93^{rd}, 86^{th}, 69^{th}, 31^{st}, 14^{th}, 7^{th} percentiles, and the minimum. The output values exceed the (-1,1) range due to the Tensorflow implementation of dropout scaling the activations by $1/(\text{keep probability})$ during training (Abadi et al., 2015).

Hidden unit activations $\mathbf{h}_t$ on each layer can act as a proxy for the post-dropout sampled weights with each forward pass. By observing the activation distributions, we can learn about the effect of each dropout approach on the weight distributions. Before starting each epoch, we collected the hidden unit activations $\mathbf{h}^l_{1:T}$ on layer l for a single forward pass on a hand-picked input sequence. Note that for both naive and variational dropout the same arbitrary input sequence was used. In figure 3.3 the distributions of the activations are plotted over epochs for both dropout approaches. The similar activation distributions at the end of training explain why both approaches achieve similar accuracies (table 3.9).

We analysed multiple hidden activation distributions for different hyperparameters. The analysis showed that both approaches result in similar models, but their weight exploration differs. A common characteristic, as seen in figure 3.3, was that activations for the variational dropout approach would quickly span the entire space, whereas activations for naive dropout

would only reach the maximum and minimum values a few epochs into training. The variational approach would also reach a distribution close to the final distribution at a relatively early stage compared to the naive approach. This could explain why the variational approach would occasionally fail – it seems to be less forgiving than the naive approach.

We analysed the Bayesian LSTM on multiple datasets – the traumatic brain injury (TBI) (section 3.2.2), MNIST, MIT-BIH, and Physionet 2016 datasets (section 3.4.3). Results along with model hyperparameters are listed in table 3.9. All models were trained with Adam at a learning rate of 0.001 and with weigth decay set to 10^{-7}. By guidance of Gal and Ghahramani (2016a) and Kendall et al. (2015) we used 30 samples for Monte Carlo dropout in the Bayesian LSTM. We would like to emphasise that the models were identically trained, the only difference was the use of statistics from 30 stochastic forward passes during testing for the Bayesian LSTM. The variational LSTMs were trained using variational dropout.

Table 3.9 Model accuracies for the Bayesian LSTM comparison

Dataset	layers	dropout	LSTM	variational LSTM	Bayesian LSTM	variational Bayesian LSTM
MNIST	256	0.1	0.9885	0.9858	**0.9890**	0.9870
Physionet 2016 [a]	2x128	0.25	0.7780	–	**0.7980**	–
MIT-BIH	128	0.2	0.8740	0.8783	0.8798	**0.8820**
TBI	128	0.3	0.8619	–	**0.8723**	–

[a]Online score based on sensitivity and specificity (Clifford et al., 2016), not accuracy.

From the results, it is evident that Bayesian LSTMs yield accuracies at least as good as standard LSTMs. For the best model on the Physionet dataset, the sensitivity and specificity values obtained were 0.675 and 0.88 for the standard LSTM, and 0.707 and 0.889 for the Bayesian LSTM. Accuracies obtained for the MNIST dataset are similar to those found in Cooijmans et al. (2016) (0.989) and Zhang et al. (2016) (0.981).

We scored slightly lower than the best results of the official Physionet 2016 challenge. This is believed to be a result of the computational constraints enforced by the online evaluation. The virtual machine used for evaluation had a single CPU core and 2GB of RAM. Additionally, the model is allowed only 2 x 10^{11} CPU instructions per phonocardiogram recording. By deep learning standards, this is a significant constraint – most current implementations use GPUs.

LSTMs easily cope with multivariate inputs, thus the univariate nature of phonocardiograms could be enriched by additional inputs, such as extracted temporal features. The best performing approaches for the Physionet 2016 challenge employed expert feature extraction

techniques (Clifford et al., 2016). However, as argued in chapter 1, using machine learning to discover relevant representations could lead to better performance than can be obtained from hand-designed features (Goodfellow et al., 2016; He et al., 2015), which is the general aim of this work.

An interesting result is that on our smallest dataset, the traumatic brain injury (TBI) dataset (12,473 data points), we see a larger increase in accuracy when using Bayesian LSTMs. This agrees with the work by Kendall and Gal (2017), which showed that smaller datasets gain more from Monte Carlo dropout than large datasets. With larger datasets, the uncertainty is explained away by the sheer number of data points. Thus, Monte Carlo dropout is particularly beneficial to medical datasets, which if labelled by experts, are typically small.

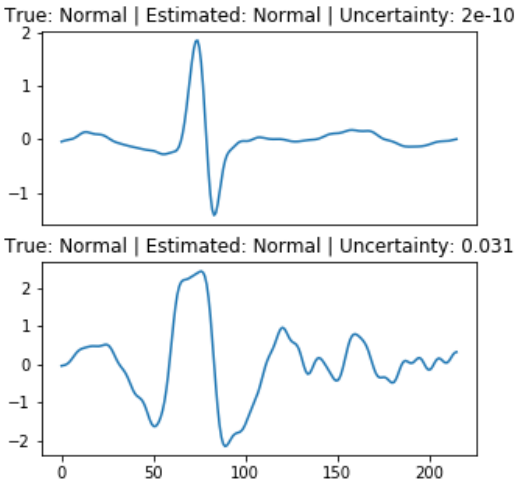
On some of the datasets, we also compared the accuracies of the variational and naive dropout approaches. The variational approach yielded a significantly better accuracy on the MIT-BIH dataset but yielded lower accuracies for the MNIST dataset. Whether variational dropout is better than naive dropout remains inconclusive, and it appears to be problem-dependent. In the rest of our work, unless otherwise specified, we used naive dropout, since this is the standard implementation for LSTMs (Abadi et al., 2015; Gal and Ghahramani, 2016b; Zaremba et al., 2014).

In addition to increased accuracy, the Bayesian approach also provides a measure of uncertainty. In figure 3.4 we juxtapose data points from the datasets analysed, for which the Bayesian LSTM (naive dropout) yielded the most confident and most uncertain classifications. From the figure it is clear that the model is uncertain about abnormal data points, such as an abnormally wide QRS complex (Kashani and Barold, 2005) in ECG signals (figure 3.4a), incomplete digits (figure 3.4b), and phonocardiograms with too low a resolution (figure 3.4c). We would like to draw attention to the benefit such uncertainties could provide in the medical domain, where medical practitioners could be queried for additional information when the model is too uncertain.

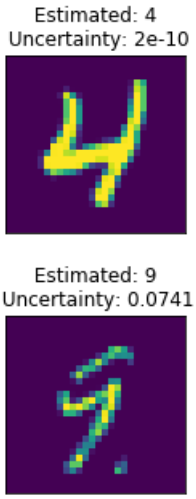
Comment: Recurrent neural networks are known to be extremely difficult to train. The countless experiments, as part of the work in this section, revealed a practical rule of thumb for applying dropout to LSTMs; the dropout value (1 - keep probability) should be smaller than 0.3. LSTMs were found to converge to poor optima and even overfit when using dropout values larger than 0.3.

In essence, this section showed that a simple modification to the way we glean information from deep learning models can provide us with a vital piece of information – classification uncertainty. This is especially beneficial to medical data, where knowing what the model doesn't know is important. We are aware that crude assumptions were made to frame dropout as a Bayesian approach, nevertheless, Monte Carlo dropout empirically provided useful

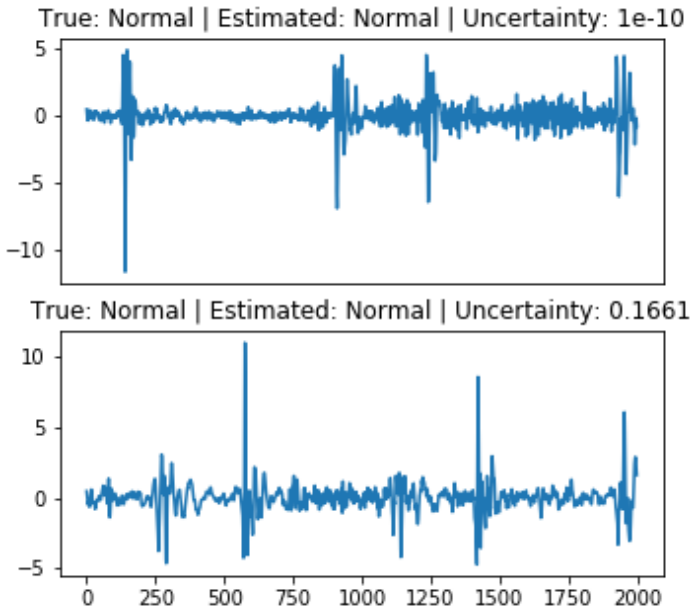
uncertainty measures. A more general assumption made up to now is that the subsequences of a signal all belong to the signal class. It is possible for subsequences of a signal to have a different meaning to that of the entire signal. In the next section, we describe how LSTMs could help us circumvent this assumption.



(a) MIT-BIH electrocardiogram



(b) MNIST



(c) Physionet 2016 phonocardiogram

Fig. 3.4 Examples of confident classifications (top row of each image) and uncertain classifications (bottom row) by the Bayesian LSTM. The medical signals displayed have been segmented.

3.5 Dealing with low-resolution labels

Routinely collected patient healthcare data is approaching the same size and complexity as that of genomic data (Rajkomar et al., 2018). Despite the abundance of data stemming from medical monitoring, much of the data is unlabelled. Labelling medical data is expensive because it requires experts that have been trained for multiple years (Längkvist et al., 2014). This dearth of true labels is perhaps among the biggest obstacles for machine learning analysis of medical data (Ching et al., 2018). In this section, we describe how LSTMs could provide a remedy.

Medical time series belong to a class of data that is prone to low granularity in the annotations (Längkvist et al., 2014), and these annotations are often incorrect due to human error. If labelled, these signals have a known final outcome, but the states and transitions of random subsequences within the signals are unknown. For example, a patient with an overall healthy outcome who survived cardiac arrest during their 24h stay in an intensive care unit would have had several periods in their signal indicative of death. As in Oresko et al. (2010) and Lugovaya (2005), work in previous chapters have attributed such phenomena to noise. However, there is value in being able to improve the granularity of signal labels by identifying segments that definitely belong to the assigned crude label, and those subsequences that don't. The first step to finding a label for a subsequence is to find a way of comparing subsequences to each other.

3.5.1 The autoencoder

To find representations of data that could be used for comparison, the deep learning toolbox provides the well-suited **autoencoder**. Autoencoders are neural network models that are trained to learn a representation $\mathbf{z}$ of the data $\mathbf{x}$ that can be used to reconstruct the data. They consist of an encoder, $\mathbf{z} = f^{enc}(\mathbf{x})$, and a decoder that produces a reconstruction $\widehat{\mathbf{x}} = f^{dec}(\mathbf{h})$ which are often modelled by feedforward neural networks (Goodfellow et al., 2016, chapter 14). The objective of this network is to minimize the squared difference between the original input and the reconstructions

$$J = \frac{1}{N}\sum_{n=1}^{N}\left\|\mathbf{x}^{(n)} - f^{dec}(f^{enc}(\mathbf{x}^{(n)}))\right\|_2^2, \tag{3.41}$$

with the number of data points denoted by N. This provides a variant of feature selection similar to that of principal component analysis (PCA) (Hinton and Salakhutdinov, 2006).

Using feedforward neural networks for the encoder and decoder places us back in a situation that is not suited for sequential data (section 1.2). Instead, we chose LSTMs for

the encoder and decoder functions. This is reminiscent of a sequence-to-sequence model (Sutskever et al., 2014), which we describe in more detail in section 4.2.

When considering the task of clustering subsequences of a coarsely labelled signal we notice that such a task would only be performed in retrospect. In other words, we know where the end of each subsequence will be, and could thus analyse the signal from front to back and from back to front to get a richer hidden representation $\mathbf{h}_{1:T}$. A natural model for such a problem is the bidirectional LSTM, which generates a hidden state that is the combination of two LSTMs, one analysing the sequence forwards, and one analysing the sequence in reversed order (Graves et al., 2013). The hidden state activations of both the forward and backward LSTMs are concatenated to construct a more informative latent state.

By denoting the LSTM unit function (eq. 3.33) as $\mathcal{H}$ we can write the bidirectional LSTM as

$$\begin{aligned}\overrightarrow{\mathbf{h}}_t &= \mathcal{H}(\mathbf{U}_{\overrightarrow{h}}\overrightarrow{\mathbf{h}}_{t-1} + \mathbf{W}_{\overrightarrow{h}}\mathbf{x}_t + b_{\overrightarrow{h}}) \\ \overleftarrow{\mathbf{h}}_t &= \mathcal{H}(\mathbf{U}_{\overleftarrow{h}}\overleftarrow{\mathbf{h}}_{t+1} + \mathbf{W}_{\overleftarrow{h}}\mathbf{x}_t + \mathbf{b}_{\overleftarrow{h}}).\end{aligned} \tag{3.42}$$

The encoder output is the concatenation of both forward and backward outputs $\mathbf{y}_t^{enc} = c[\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t]$ with $\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t \in \mathbb{R}^s$ and $\mathbf{y}_t^{enc} \in \mathbb{R}^{2s}$. The decoder takes as input $\mathbf{y}^{enc}$ and generates its own concatenated output $\mathbf{y}^{dec}$. The vector in each time step of the decoder output is then linearly mapped into the reconstructed output

$$\widehat{\mathbf{x}}_t = \mathbf{W}_r\mathbf{y}_t^{dec} + \mathbf{b}_r \tag{3.43}$$

In figure 3.5, we illustrate our autoencoder architecture. The input is encoded into a latent representation via the forward (green) and backward (blue) LSTM. The latent representation is then decoded by another bidirectional LSTM with a projection layer (yellow) that linearly maps the bidirectional LSTM activations to the original input dimensions.

The aim of this model is to generate a rich latent representation. We chose to have a latent representation that has a higher dimensionality than the input, which is referred to as an **overcomplete** autoencoder. Bengio et al. (2007) showed that overcomplete autoencoders can learn useful representations if they are properly constrained. In order to prevent the model from learning a trivial identity mapping, we regularise the model by adding noise ε to the input $\tilde{\mathbf{x}} = \mathbf{x} + \varepsilon$ and training the model to reconstruct the original input $\mathbf{x}$. This is known as a denoising autoencoder (Vincent et al., 2008). By carefully choosing the type of noise added to the input, the model could be made robust to specific unwanted features in the input

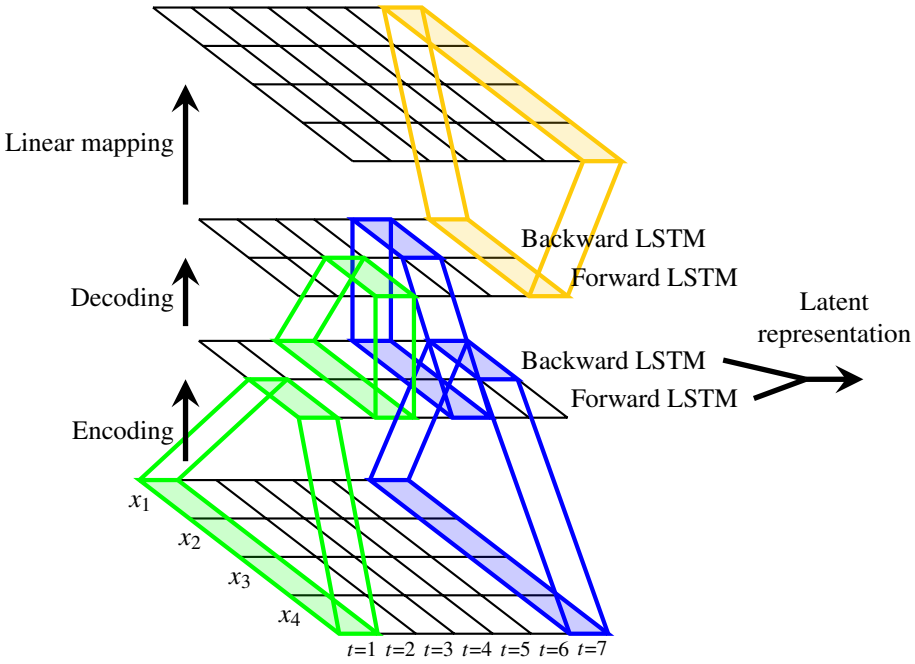


Fig. 3.5 Bidirectional autoencoder model implemented in this study (number of elements in each layer are arbitrary examples). Time steps are denoted by t for the multivariate time series $\mathbf{x}$. Hypothetical operations performed by the forward LSTM of the bidirectional LSTM are denoted by green lines, where operations of the backward LSTM are denoted by blue lines. An encoder generates the latent representation which is decoded by another bidirectional LSTM with a linear projection layer (yellow lines). The reconstructed inputs are depicted by the top grid in this diagram. The arrow indicates that the latent representation is used for clustering after an input has been encoded.

space. The objective of the denoising autoencoder is

$$J = \frac{1}{N}\sum_{n=1}^{N}\left\|\mathbf{x}^{(n)} - f^{dec}(f^{enc}(\tilde{\mathbf{x}}^{(n)}))\right\|_2^2, \quad \tilde{\mathbf{x}}^{(n)} = \mathbf{x}^{(n)} + \varepsilon, \tag{3.44}$$

where ε is some predefined noise.

Our model was chosen to have 15 hidden units, which results in a latent dimensionality of 30 outputs $\mathbf{h}_t$ per time step t. Adam with a learning rate of 0.001 was used for optimization with a dropout value of 0.2 for additional regularization. Training was stopped when the objective decreased by less than 10^{-4} for 15 consecutive epochs.

To further ensure that our model does not simply learn an identity mapping, we visually inspected the reconstructions to ensure that they are not perfect replicas of the original input. In figure 3.6 we show an example of the reconstruction for an arbitrary electrocardiogram signal. Note that the reconstructed signal is not an exact replica of the original signal, but rather a smoothed version of it.

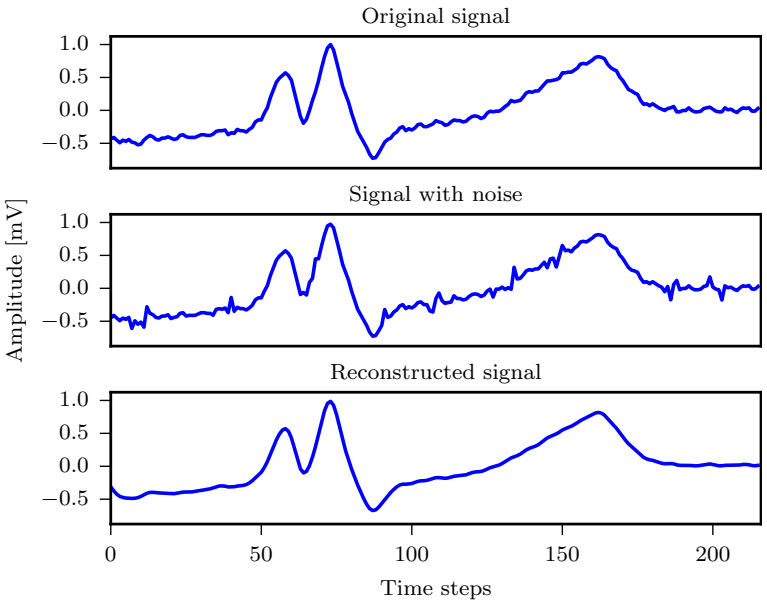


Fig. 3.6 Example reconstruction of an electrocardiogram signal using the autoencoder. Gaussian noise with a zero mean and 0.1 variance was added (middle plot) to a randomly selected 30% of the original signal (top plot). The reconstructed signal (bottom plot) is not a precise replication of the original signal, confirming that the autoencoder does not learn a trivial identity mapping.

After successfully training the autoencoder, we can generate good representations for every data point in our dataset. For visualization and clustering purposes we would then like to reduce the dimensionality of the latent representations, which is described next.

3.5.2 t-distributed stochastic neighbour embedding

The t-distributed stochastic neighbour embedding (t-sne) method developed by Van Der Maaten and Hinton (2008) maps high-dimensional data into a low-dimensional (typically 2D or 3D) manifold and groups similar samples. This technique has recently become one of the most popular dimensionality reduction tools for visualizations in deep learning. The dimensionality reduction is achieved by minimizing the divergence between a distribution that measures the pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding. Conventional methods for minimizing the divergence between these distributions scale quadratically with the number of data points N, which limits their applicability to a few thousand data points. Owing to our dataset having 106,484 data points, we remedy the computational problem with the Barnes-Hut approximation of t-sne (Van Der Maaten, 2014). This approximation was implemented in Python with the code supplied by (Van Der Maaten, 2014).

For t-sne, a perplexity hyperparameter has to be specified. This loosely guides the balance of attention between global and local aspects of the data. In a sense, the perplexity is a guess of the number of close neighbours each point has (Wattenberg et al., 2016). Typical values of perplexity are between 5 and 50. Empirically we found a perplexity value of 50 to provide the best results, which met expectations, provided a large number of data points and relatively small number of foreseen clusters.

A weakness of t-sne is that it is sensitive to data with high intrinsic dimensionality. However, this is mitigated by performing t-sne on the nonlinear representation produced by our autoencoder (Van Der Maaten and Hinton, 2008).

With the t-sne method, we decreased the dimensionality of the latent representation $\mathbf{z}$ to two dimensions. After obtaining the low-dimensional representation of the data points, we proceed to cluster points that are close to each other.

3.5.3 Clustering

The 2D representations were clustered by fitting a Gaussian mixture model (GMM) using variational inference (Bishop, 2006, §10.2). To infer the number of clusters automatically, a Dirichlet process was used to sample the random distribution over the parameters of the GMMs, resulting in what is known as a Dirichlet process mixture model (Blei and Jordan, 2006). The Dirichlet process was constructed using the stick-breaking process, for which the number of components almost always depends on the data (Murphy, 2012, §25.2.2.1). This is a good fit for our case as, *a priori*, we don't know the number of clusters in the dataset.

We used the *BayesianGaussianMixture* function of the Scikit-learn package in Python to implement this clustering technique. The upper bound on the number of clusters (components) in our implementation was 1,000, much larger than the expected number of clusters. The concentration hyperparameter was chosen as 1, where a lower value leads to fewer clusters and vice versa.

After the clusters were determined, the label of the cluster was determined by the mode of the labels of the data points within each cluster.

$$\tilde{y}_a = \underset{c}{\operatorname{argmax}} \sum_{n \in S_a} \mathbb{I}[y^{(n)} = c], \tag{3.45}$$

where $S_a \subset \{1,\ldots,N\}$ is the subset of indices for which the data points are in cluster a. Thus it is assumed that data points with a label different to that of their cluster were incorrectly labelled. These data points were then assigned new labels y' based on the labels of their clusters; $y'^{(n)} = \tilde{y}_a$ if $n \in S_a$, with $n = 1,\ldots,N$.

3.5.4 Results

The efficacy of the described labelling approach was evaluated on the MIT-BIH dataset of electrocardiogram (ECG) signals (section 3.4.3). Seeing that the Hamilton QRS detector (Hamilton, 2002) does not have perfect accuracy for heartbeat detection on this dataset (reported sensitivity of 99.74%), there are bound to be incorrectly labelled data points in our processed dataset. This easy-to-understand problem is exactly what our proposal aims to solve.

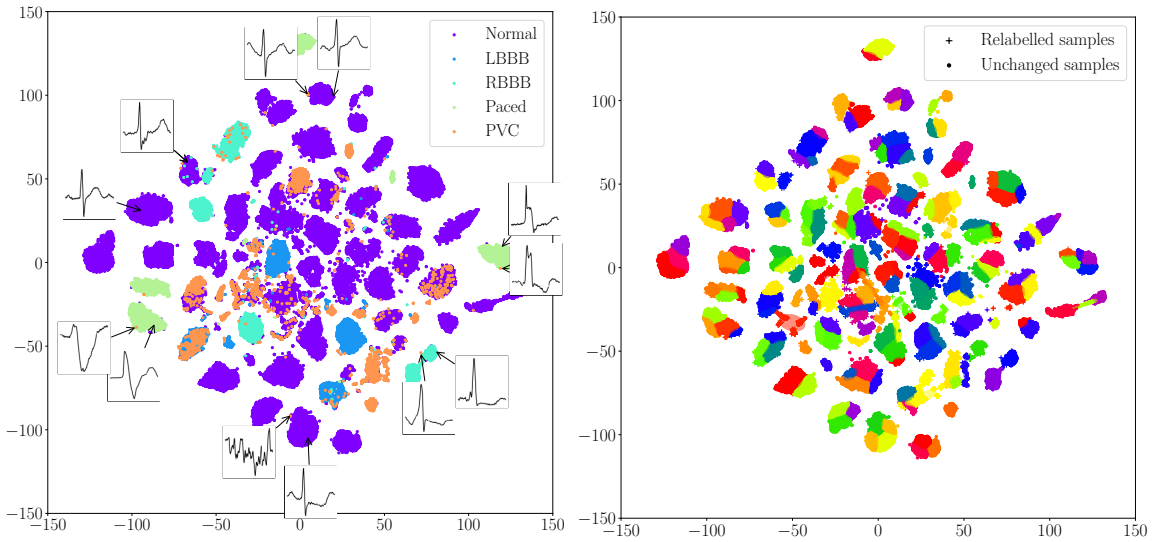
Owing to this being a clustering exercise, we slightly modify the MIT-BIH dataset setup from that described in section 3.4.3. To provide a wider variety of cluster options, we use the five most populated heartbeat categories: normal, left bundle branch block, right bundle branch block, paced, and premature ventricular contraction. Moreover, the heartbeats from all patients are mixed together to provide better conditions for stochastic gradient descent.

The bidirectional LSTM with 15 hidden units and trained as described in section 3.5.1, resulted in the latent representation having a dimensionality of 216×30. Empirically, we found that Gaussian noise $\mathcal{N}(\varepsilon; 0, 0.1)$ applied to a random 30% of the entries in each input sequence produced in the best results.

The 2D t-sne embedding after 10,000 iterations is illustrated in figure 3.7a. Different classes of heartbeats are well separated and with the ECG profiles shown on the plot, one can observe the differences between the characteristics of the clusters. The Gaussian mixture model (GMM) clusters fitted to the 2D embedding of the signals are shown in figure 3.7b. The different colours represent different clusters and in turn their labels $\tilde{y}_a$. If we compare the labels in figure 3.7a to the new labels (clusters) in figure 3.7b our clustering method finds clusters that are too small. However, having a greater number of small GMM clusters is better in this case because it enables relabelling to be more precise. In other words, if the t-sne method placed two clusters close to each other, the clustering method will still be able to distinguish the two, and thus, assign the best approximate new labels.

The t-sne method adapts its notion of distance to regional density variations in the dataset. Consequently, it naturally contracts sparse clusters and expands dense ones, evening out the cluster sizes. Thus the relative cluster sizes are not elucidated in the t-sne plots (Wattenberg et al., 2016). Moreover, the distance between clusters greatly depends on the chosen perplexity and is therefore not an informative metric. t-sne is a popular technique because it's incredibly flexible and can often find structure where other dimensionality reduction techniques cannot. Unfortunately, the flexibility also makes the output difficult to interpret.

After finding clusters in the latent representation and relabelling the entire dataset accordingly, we determined the effect of the new dataset on the accuracy of an LSTM classifier.



(a) t-sne plot of ECG-representations **(b)** GMM clusters

Fig. 3.7 Illustrations of the t-sne 2D embedded output (a) and the Gaussian mixture model (GMM) clustering (b). Hand-picked signals are displayed in (a), and the original classes are indicated by different colours. The five heartbeat categories are normal, left bundle branch block (LBBB), right bundle branch block (RBBB), paced, and premature ventricular contraction (PVC). Hand-picked signals depict the similarity of outliers to their cluster. In (b) each colour represents a different cluster with its own mode label.

Consisting of a single hidden layer of 128 units, the LSTM was trained using Adam, with a learning rate of 0.001, a minibatch size of 100, and a dropout value of 0.2. The 106,848 heartbeats were randomly split into a ration of 70:10:20 (train:validation:test), and the validation set was used for early stopping.

Averaged over five independent simulations, the LSTM achieved accuracies of 98.12% and 98.37% on the original and relabelled datasets, respectively[5]. The increase in accuracy could mean that our approach was able to correctly label the heartbeats incorrectly extracted by the Hamilton QRS detector. It is difficult, however, to determine whether the newly-assigned classes are clinically accurate; the assigned labels could merely be easier for the LSTM to analyse. Nevertheless, the increase in accuracy indicates that this approach is beneficial for LSTMs.

In sum, this chapter showed that LSTMs are better for analysing medical time series than hidden Markov models. This superiority is supported by the fact that many of the best performing approaches on publicly available medical-signal datasets include LSTMs

[5] Although the recordings from each patient contained multiple heartbeat types, these accuracies do not provide an indication of generalization over unseen patients because different heartbeats from the same patient were present in the training and test sets.

(Clifford et al., 2017). We also showed that it's straightforward to obtain uncertainty measures of the LSTM predictions, which is vital for medical applications. Lastly, we showed that LSTMs can be used in an autoencoder fashion to mitigate the labelling issues in medicine. Nevertheless, this approach leaves room for improvement. Instead of using an overcomplete autoencoder, we could leverage the built-in sequence compression of LSTMs to learn useful representations. Moreover, the approach described in this section involves multiple steps where hyperparameters need to be selected – an unnecessary complication. Hence, in the following section, we propose an improved approach that allows end-to-end learning of the entire process.

Chapter 4

A sequence-to-sequence model: Inferring actions from the peripheral nervous system

The peripheral nervous system is an abundant source of information on physiological processes conveyed between the central nervous system and other systems in the human body. Thanks to recent advances in neural interface technology, it is possible to access long-term recordings of such information from subjects in natural (non-laboratory) environments. In this setting, high-resolution data can be recorded for months or years at a time, presenting a new paradigm for the neuroprosthesis community and a rich novel type of dataset.

A new kind of problem is posed by these long-term peripheral nervous system datasets – the datasets are too large and too nuanced for manual human analysis, let alone manual labelling. In this chapter, we propose a bespoke sequence-to-sequence model which will enable label generation for such datasets. Being able to infer labels from peripheral neural signals could mitigate the issue of scant or expensive labels and as a result, could provide useful control signals for neurological human-machine interfaces. In turn, this holds tremendous potential benefits for amputees, as this would provide a way for them to seamlessly integrate with smart prosthetic devices.

4.1 The problem with limited labelled data

As alluded to in section 3.5, real-life time series often lack high resolution annotations, and consequently, assumptions have to be made about lower-level temporal aspects. Moreover, high definition labelling of big data is too expensive and the amount of unlabelled data is

still orders of magnitude larger than the amount of labelled data. Annotations can also be ambiguous or incorrect, and few real-world datasets have labels that are 100% accurate (Russakovsky et al., 2015).

Long-term recordings of peripheral nervous system signals have an abundance of data but are often accompanied by insufficient and sparse labels. The procedure, in general, is to have the monitoring device record neural signals continuously, and then to create time-stamped annotations at predetermined intervals. The problem is that the frequency of these annotations is orders of magnitude lower than that of the recorded signals. This procedure and problem generalise to many long-term biological recordings, for example, patients are continuously monitored in intensive care units and clinicians would typically provide hourly or daily time-stamped annotations of their status (Längkvist et al., 2014).

For peripheral nervous system signals, we can elucidate this problem by means of an arbitrary example. Suppose there is a left-leg (transfemoral) amputee patient living their normal life at home whilst having the neural signals in their leg recorded by a smart prosthetic device. In addition, the patient provides a video recording of themselves performing a set of everyday actions. A schematic of this procedure is shown in figure 4.1. The stick figure represents the video recording of the patient, and the blue line depicts the recorded neural signals. From subsequences in the video, we are able to derive crude labels y_i for the accompanying neural signal subsequence γ_a. Eventual control of the prosthetic device via neural signals requires inference of the label solely from a subsequence of the neural signals.

There are two problems that arise for this postulated situation. First, the labels are sparse – the number of video subsequences used to assign labels y_i is much lower than the number of neural signal subsequences γ_a. Second, even if we can assign a label to the neural signal subsequence, the label is a crude approximation of what the subsequence should resemble. In other words, all the high-resolution time-steps t in subsequence γ_a are given the same label, where they might have different meanings in reality.

The combination of these problems warrants the use of unsupervised learning as a practical approach to learning useful representations of the signals. Deep learning has been revered for its power of extracting useful features from data (He et al., 2015; Krizhevsky et al., 2012). Employing such techniques could allow inference of the label for each subsequence γ_a. The LSTM provides a sound deep learning technique for the extraction of information from sequential data. For a sequence with any length, the LSTM provides a fixed-size representation – an important requirement for the generation of labels from sequences as we will see later. This property allows LSTMs to be implemented in a similar fashion to autoencoders, giving rise to the sequence-to-sequence model, which compress variable-length

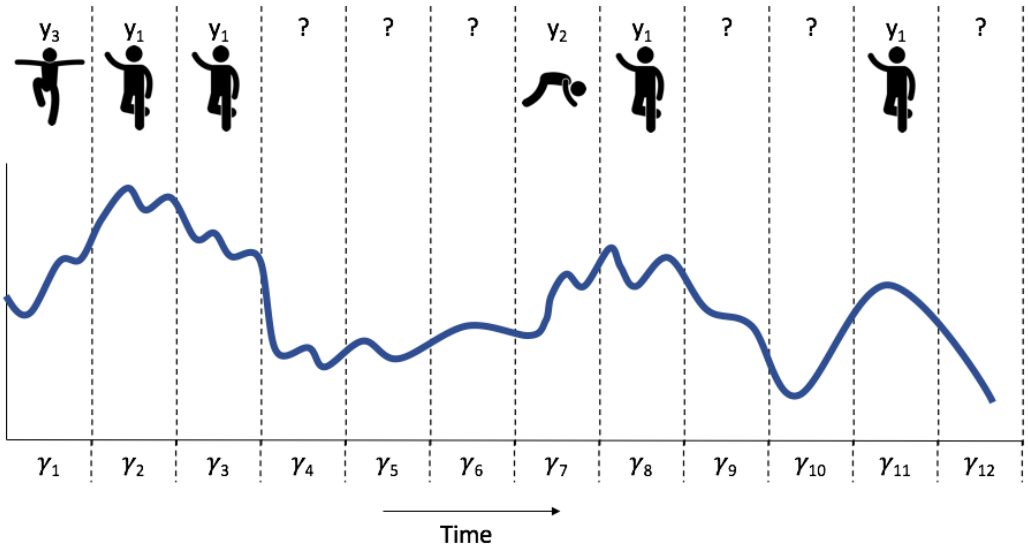


Fig. 4.1 Schematic af a hypothetical peripheral nervous system data collection setup. Recorded neural signals are depicted by the blue line, with subsequences indicated by γ_*. The stickman represents actions performed in a section of the video. Image reproduced with permission from Tris Edwards (Cambridge Bio-augmentation Systems).

sequences into information-rich fixed-size representations. In the next section, we elaborate on how this approach could remedy the problem of limited labelled data.

4.2 The sequence-to-sequence model

In most scenarios, it is safe to assume that some latent variable or set of latent variables is responsible for the generation of an observed sequence. Here we are concerned with the task of extracting these latent variables from the sequence for classification purposes. One way to obtain these latent variables would be to compress the sequence enough for only the most relevant latent features to remain. The **undercomplete** autoencoder, which has a latent space dimensionality smaller than the input dimension, is the deep learning model of choice for compressing data in this manner.

Recall from section 3.5.1 that the autoencoder model consists of an encoding function $f^{enc}(\mathbf{x})$ to generate a latent representation $\mathbf{z}$ which is then used to reconstruct the input by means of the decoding function $\widehat{\mathbf{x}} = f^{dec}(\mathbf{z})$. The objective function for this model is the

average squared error[1] of the reconstructions for the set of data points $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$

$$J = \frac{1}{N}\sum_{n=1}^{N}\left\|\mathbf{x}^{(n)} - f^{dec}(f^{enc}(\mathbf{x}^{(n)}))\right\|_2^2. \tag{4.1}$$

Returning to LSTMs, we can define the hidden unit operations with

$$\begin{aligned}\mathbf{i},\mathbf{o},\mathbf{f} &= \sigma(\mathbf{U}_{i,o,f}\mathbf{h}_{t-1} + \mathbf{W}_{i,o,f}\mathbf{x}_t + \mathbf{b}_{i,o,f})\\ \mathbf{c}_t &= \mathbf{f}\odot\mathbf{c}_{t-1} + \mathbf{i}\odot\tanh(\mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t + \mathbf{b}_c)\\ \mathbf{h}_t &= \mathbf{o}\odot\tanh(\mathbf{c}_t).\end{aligned} \tag{4.2}$$

Here the memory vector $\mathbf{c}_T$ and the activation vector $\mathbf{h}_T$, at the final time step T of the input sequence, is a latent representation of the entire input sequence, which can naturally lead to an autoencoder for sequences by having a decoder generate a reconstruction of the input $\widehat{\mathbf{x}}$ from $(\mathbf{c}_T, \mathbf{h}_T)$. This is known as the sequence-to-sequence model (Sutskever et al., 2014), where the latent representation is the final hidden state of the encoder LSTM, from which the decoder has to generate the reconstructions; $(\mathbf{c}_T^{enc}, \mathbf{h}_T^{enc}) = \mathbf{z} = (\mathbf{c}_1^{dec}, \mathbf{h}_1^{dec})$. The input at the first time step for the decoder $\mathbf{x}_1$ is a vector of zeros, which represents the "start" symbol for the decoder. A diagram of a typical sequence-to-sequence model is shown in figure 4.2.

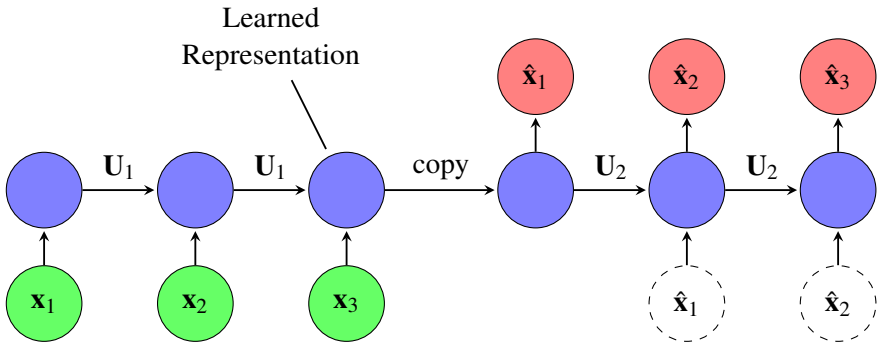


Fig. 4.2 Sequence-to-sequence model. Here $\mathbf{x}$ denotes the input, and the subscripts indicate different time steps. The hidden-to-hidden weights for the encoder and decoder are represented by $\mathbf{U}_1$ and $\mathbf{U}_2$ respectively. The learned representation in the encoder is used as the initial state for the decoder, from which the decoder attempts to reconstruct the inputs $\widehat{\mathbf{x}}$. The dashed nodes depict the optional conditional input that can be used in such a model.

We have described how to find fixed-size representations for sequences, but how do we ensure that labels can be obtained from the latent representation? In order to obtain labels from the latent representation we assume that along with the most relevant latent

[1] We will omit the sample number superscript (n) whenever it is clear that we are referring to terms associated with a single data point.

variable (class) generating the signals, other latent variables are present, which determine nuances (style) of the signals belonging to this class. These latent variables will have different constraints, which motivates the use of two latent representations in our sequence-to-sequence model.

Two latent representations are obtained from the encoder LSTM by splitting the memory vector $\mathbf{c}_T$. For the latent label representation, having the form of a probability vector is sensible. We denote the latent label representation as $\mathbf{y} = \mathbf{e}^{(k)}$, with $k \in \{1,\ldots,A\}$ and A the assumed number of different latent class variables we are interested in[2]. The latent style representation is denoted by $\mathbf{s} \in \mathbb{R}^B$, where the size of the memory vector $\mathbf{c}_T$ is $A+B$. To obtain a probability vector we use the softmax function, and both latent representations are then given by

$$\begin{aligned} \psi &= \mathbf{c}_{T,1:A} \\ y_a &= \frac{\exp(\psi_a)}{\sum_j \exp(\psi_j)} \\ \mathbf{s} &= \mathbf{c}_{T,A+1:A+1+B}, \end{aligned} \tag{4.3}$$

where $\mathbf{c}_{T,A+1:A+1+B}$ denotes all the elements from index $A+1$ to index $A+1+B$ of the vector $\mathbf{c}_T$. By denoting the concatenation of vectors with $c[\cdot]$, the latent representation of our sequence to sequence model is given by

$$\mathbf{z} = (c[\mathbf{y},\mathbf{s}], \mathbf{h}_T). \tag{4.4}$$

Since the primary goal of this approach is to assign informative labels to the signals, we place an additional constraint on the decoder. Instead of merely generating the reconstructed signal from the latent representation, we copy the encoded label vector $\mathbf{y}$ into the memory units of the decoder at each step. The memory vector update from eq. 4.2 becomes

$$\begin{aligned} \mathbf{g}_t &= \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \tanh(\mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{W}_c \mathbf{x}_t + \mathbf{b}_c) \\ \mathbf{c}_t^{dec} &= c[\mathbf{y}, \mathbf{g}_{t,A+1:A+1+B}]. \end{aligned} \tag{4.5}$$

This should in theory encourage the model to place more emphasis on the generation of an informative $\mathbf{y}$-representation.

Although the sequence-to-sequence model provides a sound approach for encoding sequences, it suffers from difficulty in training. During the initial stages of training, the gradients of the decoder output have to be back-propagated through the constrained latent

[2] $\mathbf{e}^{(k)}$ is a standard basis vector $[0,\ldots,0,1,0,\ldots,0]$ with a 1 at position k.

representation all the way back to the first steps of the encoder. By reversing the order of the reconstructions, however, the model is able to focus on short-term dependencies initially, before gradually looking at longer-term dependencies, which makes learning easier. Training a sequence-to-sequence model with reversed reconstruction targets was first proposed by Srivastava et al. (2015a). Early in our experimentation we found this to be beneficial and continued using the approach throughout our experiments, with the objective function now defined as

$$J = \frac{1}{N}\sum_{n=1}^{N}\left\|\mathbf{x}_{T:1}^{(n)} - f^{dec}(f^{enc}(\mathbf{x}_{1:T}^{(n)}))\right\|_2^2. \tag{4.6}$$

A diagram of this sequence-to-sequence model is shown in figure 4.3.

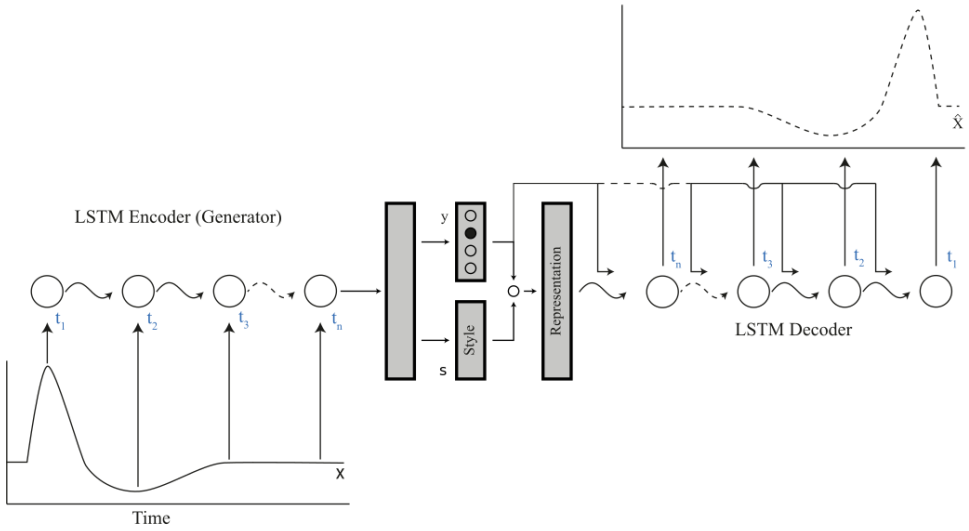


Fig. 4.3 Sequence-to-sequence model. Two latent representations are created by splitting the memory vector **c** of the encoding LSTM. The **y**-representation is created by means of a softmax function. The decoder attempts to reconstruct the input signal from the concatenation of the latent representations, and the **y**-representation is copied into the hidden state of the decoder at each time step.

Having the decoder reconstruct the input solely from the latent representation **z** requires the representation to contain sufficient information. A more informative latent representation is beneficial in our application, but it often raises complications during training. If the decoder reconstructions are unguided from the "start" symbol, the optimizer is more likely to converge to bad local minima. At these bad local minima, gradients would stop propagating information through the network, and obtaining any sensible reconstruction becomes impossible. Fortunately, we can guide the reconstructions by providing the correct input at each step, a process known as **teacher forcing**. Essentially, the decoder input becomes the encoder input prepended with a vector of zeros $\mathbf{x}_{1:T+1}^{dec} = c[\mathbf{0}, \mathbf{x}_{1:T}]$. This facilitates learning

by having the correct inputs for each time step of the decoder in lieu of the, possibly incorrect, decoder outputs from the previous time step. In our implementations, teacher forcing was randomly switched on with a probability of 0.5 for each minibatch iteration during training – providing a good trade-off between the stability of training and the information captured in the latent representation.

In this section, we have described the architecture of our sequence-to-sequence model which has two latent representations. Even though there are two latent representations, the probabilities in $\mathbf{y}$ could be close to a uniform distribution, rendering them uninteresting. In the sections that follow, we describe how to obtain latent representations with more desirable characteristics by means of regularization.

4.2.1 Standard model regularization

The aim of our approach is not to get near perfect reconstruction with the sequence-to-sequence model, but rather to obtain the best compression of the data. Thus, in addition to the existing constrained dimensionality of the latent space, we regularize both the $\mathbf{y}$ and $\mathbf{s}$-representations to yield the desired characteristics.

We would like to be able to assign a label to a signal based on our $\mathbf{y}$-representation. Therefore, we aimed to have the $\mathbf{y}$-representation be a standard basis vector, i.e., when $y_a = 1$ and $\mathbf{y}_{-a} = \mathbf{0}$ all the information of the class of the signal would be contained in a single element of $\mathbf{y}$. Provided with a specific one-hot vector representing class a, the decoder should then be able to generate a signal from this class.

One measure for the peakedness of a distribution is the entropy. Thus, a first approach to regularizing the $\mathbf{y}$-representation would be to minimize the squared error (eq. 4.6) along with the entropy

$$\mathbb{H}[\mathbf{y}] = -\sum_{a=1}^{A} y_a \log(y_a) \tag{4.7}$$

The $\mathbf{s}$-representation (eq. 4.3) theoretically conveys information about the "style" of the signal. However, if we constrain only the $\mathbf{y}$-representation, all of the important information could flow through the $\mathbf{s}$-representation, rendering the $\mathbf{y}$-representation uninteresting. A suitable regularizer for the $\mathbf{s}$-representation could be the L_2 norm, which penalizes vectors with larger magnitudes. This would limit the information flow through $\mathbf{s}$, and in turn encourage more information to flow through the $\mathbf{y}$-representation. The resulting objective function of the sequence-to-sequence model becomes

$$J = \frac{1}{N}\sum_{n=1}^{N}\Big(\Big\|\mathbf{x}_{T:1}^{(n)} - f^{dec}(f^{enc}(\mathbf{x}_{1:T}^{(n)}))\Big\|_2^2 - \lambda_1 \sum_{a=1}^{A}\big(y_a^{(n)}\log(y_a^{(n)})\big) + \lambda_2\Big\|\mathbf{s}^{(n)}\Big\|_2\Big), \tag{4.8}$$

where λ_1 and λ_2 weight the relative importance of the respective regularizers.

The regularization approaches described, provide a good first approach to obtaining the desired characteristics. However, the deep learning "toolbox" provides more suited approaches. The entropy term could exacerbate the problem of bad local minima, by "locking-in" the wrong class combinations during the early stages of training, and the L_2 norm could be placing too strict constraints on the values of each element in $\mathbf{s}$. In the following section, we describe a recently developed technique that provides a good solution to our problem.

4.2.2 Regularizing with generative adversarial networks

Instead of regularizing the latent representations by means of entropy and the L_2 norm, we could train additional neural networks that encourage the representations to have the desired characteristics. The recently developed generative adversarial network (GAN) (Goodfellow et al., 2014), introduced in this section, achieves a similar goal by means of a discriminative network.

The generative adversarial network creates a situation where the generator network must compete against an adversary. First, the generator network, parametrized by θ, produces a sample $\mathbf{z} = G_\theta(\mathbf{x})$ with $\mathbf{x}$ distributed according to p_x. The discriminator network then acts as the adversary in trying to distinguish between samples from a predefined **target distribution** p_τ, and samples drawn from the generator, distributed according to p_G. The discriminator emits a probability value given by $D_\omega(\mathbf{z})$, indicating the probability that $\mathbf{z}$ is a sample from the target distribution instead of a sample from the generator.

This can be formalized as a zero-sum game, in which a function $v(\theta, \omega)$ determines the payoff for the discriminator, and $-v(\theta, \omega)$ is the payoff for the generator. At training time, each "player" strives to maximize its own payoff, such that at convergence

$$G^* = \operatorname*{argmin}_G \max_D v(G, D). \tag{4.9}$$

The standard choice for v is

$$v(\theta, \omega) = \mathbb{E}_{\mathbf{z}\sim p_\tau} \log D_\omega(\mathbf{z}) + \mathbb{E}_{\mathbf{z}\sim p_G} \log(1 - D_\omega(\mathbf{z})). \tag{4.10}$$

However, standard generative adversarial networks (GANs) such as these are notoriously difficult to train (Gulrajani et al., 2017; Salimans et al., 2016). If the discriminator becomes too strong, gradients of the generator vanish. Moreover, the generator often suffers from **mode collapse**, where it learns to always produce the same outputs. Fortunately, Arjovsky

et al. (2017) proposed the Wasserstein generative adversarial network (WGAN) approach as a stable version of the GAN, which was found beneficial to our model. To have **stability** during training implies that the aforementioned problems would not occur in multiple independent runs of the algorithm.

Goodfellow et al. (2014) showed that during training, the discriminator maximizes eq. 4.10, which is a lower bound of $2\text{JS}(p_\tau, p_G) - 2\log 2$. Thus we are updating G against an objective that approximates the Jenson Shannon (JS) divergence. Let p_m be the mixture $(p_\tau + p_G)/2$, then the Jenson-Shannon divergence is given by

$$\text{JS}(p_\tau, p_G) = 0.5\text{KL}(p_\tau || p_m) + 0.5\text{KL}(p_G || p_m). \tag{4.11}$$

Arjovsky et al. (2017) showed that for the problems solved by GANs, the JS divergence often results in gradients that are zero or not smooth. They also prove that the Wasserstein (or earth-mover) distance will converge if the JS divergence converges. The Wasserstein distance is defined as

$$\text{W}(p_\tau, p_G) = \inf_{\gamma \in \prod(p_\tau, p_G)} \mathbb{E}_{(w_1, w_2)\sim\gamma}[\|w_1 - w_2\|], \tag{4.12}$$

where $\prod(p_\tau, p_G)$ denotes the set of all joint distributions $\gamma(w_1, w_2)$ whose marginals are respectively p_τ and p_G. Intuitively, $\gamma(w_1, w_2)$ indicates how much "mass" must be transported from w_1 to w_2 in order to transform the distributions p_τ into the distribution p_G. The Wasserstein distance then is the "cost" of the optimal transport plan. In contrast to the JS divergence, the Wasserstein distance has guarantees of continuity and differentiability, which in theory should make the training of generative adversarial networks more stable.

Although the Wasserstein distance would provide nicer properties when optimized, the infimum in eq. 4.12 is highly intractable. Fortunately, the Kantorovich-Rubinstein duality (Villani, 2009) tells us that

$$\text{W}(p_\tau, p_G) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{\mathbf{z}\sim p_\tau}[f(\mathbf{z})] - \mathbb{E}_{\mathbf{z}\sim p_G}[f(\mathbf{z})], \tag{4.13}$$

where the $K = 1$ and the supremum is over all the 1-Lipschitz functions $f : \mathcal{Z} \to \mathbb{R}$, with $\mathbf{z} \in \mathcal{Z}$. If we let K be any constant and we have a parameterized family of functions $\{f_\omega\}_{\omega\in\mathcal{W}}$ that are all K-Lipschitz then $\text{W}(p_\tau, p_G)$ can be calculated up to a multiplicative constant according to the following

$$v(\theta, \omega) = \text{W}(p_\tau, p_G) = \max_{\omega\in\mathcal{W}} \mathbb{E}_{\mathbf{z}\sim p_\tau}[f_\omega(\mathbf{z})] - \mathbb{E}_{\mathbf{x}\sim p_x}[f_\omega(G_\theta(\mathbf{x}))]. \tag{4.14}$$

Unlike the discriminator in standard GANs, the function f_ω does not output a probability. Therefore, in WGANs this function is referred to as the **critic** because it does not explicitly classify inputs, but rather serves as a helper for estimating the Wasserstein distance.

In our case, the critic function f is modelled by a neural network parametrized by ω. In order to ensure that f is K-Lipschitz, the parameters ω are constrained to lie in a compact space by clamping them to have a range $[-c, c]$ after each gradient update. If c is large, it takes longer to train the critic f to optimality because it takes longer for any parameters to reach their limit. Smaller values of c can increase the risk of vanishing gradients when the number of layers is large or if batch normalization (section 3.3.2) is not used – two typical characteristics of recurrent neural networks. In our experiments, the recommended value of 0.01 for c yielded good results.

For standard generative adversarial networks (GANs), the discriminator can quickly become too strong, resulting in vanished gradients for the generator. In contrast, because the Wasserstein distance is differentiable nearly everywhere, we can (and should) train f_ω to convergence before each generator update, to get as accurate an estimate of $W(p_\tau, p_G)$ as possible. The more accurate $W(p_\tau, p_G)$ is, the more accurate the gradient $\nabla_\theta W(p_\tau, p_G)$. In our implementation we train the critic three times for each update of the generator, striking a good balance between computational expense and optimal gradients.

Comment: In summary, to change a GAN into a WGAN the following changes are made:

- The discriminator (critic) outputs a real scalar value instead of a probability.
- The logarithms are removed from the objective function.
- After each gradient update the discriminator (critic) parameters are clipped to $[-c, c]$.

We come full circle by relating the WGAN components to the components of our sequence-to-sequence model. In essence, we assume that the data analysed by the model are generated by a latent class variable $\mathbf{y}$ that comes from a categorical distribution as well as a latent style variable $\mathbf{s}$ that comes from a Gaussian distribution

$$p_\tau(\mathbf{y}) = Cat(\mathbf{y}) \qquad\qquad p_\tau(\mathbf{s}) = \mathcal{N}(\mathbf{s}|\mathbf{0}, \mathbf{I}),$$

which become our target distributions in the WGAN. The generator function of the latent class variable G_{θ_y} is the encoder LSTM combined with the softmax function (eq. 4.3).

Moreover, the generator function of the latent style variable G_{θ_s} is the encoder LSTM for the section of memory converted to the **s**-representation.

Two additional neural networks $f_{\omega_y}(\mathbf{y})$ and $f_{\omega_s}(\mathbf{s})$ perform the role of the critic for each generator. These feedforward neural networks have two ReLU activated (Nair and Hinton, 2010) hidden layers with 50 and 20 units respectively. The output of the network is a scalar, which is a linear combination of the final layer activations. The class critic f_{ω_y} has to distinguish between samples $\mathbf{y}$ from G_{θ_y} and samples from a categorical distribution $\mathbf{y}'$. Similarly, the style critic f_{ω_s} has to distinguish between samples $\mathbf{s}$ from G_{θ_s} and samples from a Gaussian distribution $\mathbf{s}' \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. For both critics we experimented with up to 200 units on three layers, but found the combination of 50 and 20 units to yield the best results.

This regularization approach transforms our standard sequence-to-sequence model (figure 4.3) into the model illustrated in figure 4.4. Where training the standard sequence-to-sequence model is relatively straightforward with a single update of all parameters during each iteration, training of the sequence-to-sequence Wasserstein adversarial network is more complicated. In algorithm 1 we provide the training procedure for this model.

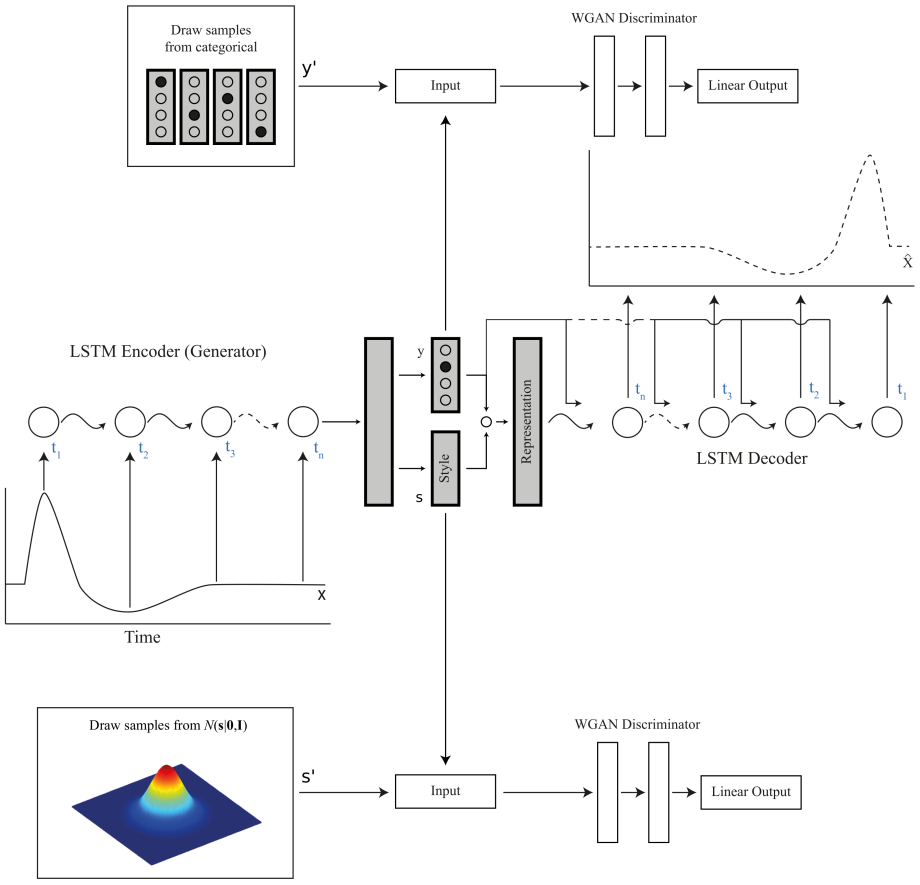


Fig. 4.4 The sequence-to-sequence Wasserstein adversarial network, or SWAN model for short. The sequence-to-sequence model (figure 4.3) is regularized by two discriminative networks. The top discriminator encourages the **y**-representation to be categorically distributed. The bottom discriminator encourages the **s**-representation to be Gaussian distributed.

Algorithm 1 Sequence-to-sequence Wasserstein adversarial network (SWAN) algorithm. All experiments in this study used the default values $\eta = 0.001$, $c = 0.01$, $m = 150$, $n_{critic} = 3$, $E = 100$.

Require: :η, the learning rate. c, the clipping hyperparameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration. E, the number of training epochs.

Require: : $\omega_y^{(0)}$, initial label critic parameters. $\omega_y^{(0)}$, initial style critic parameters. $\theta_y^{(0)}$, initial label generator parameters. $\theta_s^{(0)}$, initial style generator parameters. ${\omega_{seq}}^{(0)}$, initial sequence-to-sequence parameters.

for $j = 1$ **to** E **do**
  Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{m} \sim p_x$ a batch of data.
  With probability 0.5 use teacher forcing in the decoder f^{dec}.
  for $n = 1$ **to** n_{critic} **do**
    Sample $\{\mathbf{y}'^{(i)}\}_{i=1}^{m} \sim Cat(\mathbf{y})$ a batch from the target label distribution (random standard basis vectors).
    Sample $\{\mathbf{s}'^{(i)}\}_{i=1}^{m} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ a batch from the target style distribution.
    $g_{\omega_y} \leftarrow \nabla_{\omega_y} \left[\frac{1}{m} \sum_{i=1}^{m} f_{\omega_y}(\mathbf{y}'^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_{\omega_y}(G_{\theta_y}(\mathbf{x}^{(i)})) \right]$
    $g_{\omega_s} \leftarrow \nabla_{\omega_s} \left[\frac{1}{m} \sum_{i=1}^{m} f_{\omega_s}(\mathbf{s}'^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_{\omega_s}(G_{\theta_s}(\mathbf{x}^{(i)})) \right]$
    $\omega_y \leftarrow \omega_y + \eta \cdot \mathrm{Adam}(\omega_y, g_{\omega_y})$
    $\omega_s \leftarrow \omega_s + \eta \cdot \mathrm{Adam}(\omega_s, g_{\omega_s})$
    $\omega_y \leftarrow \mathrm{clip}(\omega_y, -c, c)$
    $\omega_s \leftarrow \mathrm{clip}(\omega_s, -c, c)$
  end for
  $g_{\theta_y} \leftarrow -\nabla_{\theta_y} \frac{1}{m} \sum_{i=1}^{m} f_{\omega}(G_{\theta_y}(\mathbf{x}^{(i)}))$
  $g_{\theta_s} \leftarrow -\nabla_{\theta_s} \frac{1}{m} \sum_{i=1}^{m} f_{\omega}(G_{\theta_s}(\mathbf{x}^{(i)}))$
  $g_{\omega_{seq}} \leftarrow -\nabla_{\omega_{seq}} \frac{1}{m} \sum_{i=1}^{m} \left\| \mathbf{x}_{T:1}^{(i)} - f^{dec}(f^{enc}(\mathbf{x}_{1:T}^{(i)})) \right\|_2^2$
  $\bar{g} \leftarrow \mathrm{avg}(g_{\theta_y}, g_{\theta_s}, g_{\omega_{seq}})$ average the overlapping parameter gradients
  $\theta_* \leftarrow \theta_* - \eta \cdot \mathrm{Adam}(\theta_*, \bar{g})$
end for

The Wasserstein generative adversarial network (WGAN) provides an elegant approach to the stable training of generative adversarial networks and to the regularization of multiple latent representations. However, the difficulties of optimization are not completely mitigated by WGANs. In the following section, we describe the additional remedies we employed.

4.2.3 Additional requirements for generative adversarial networks

Sequence-to-sequence models, without the additional complications introduced by generative adversarial networks (GANs), are sensitive to vanishing and exploding gradients during back-propagation. Although the Wasserstein implementation reduces the mode collapse tendency of the GAN, we used two other techniques to further ensure stability during training.

First, batch normalization (section 3.3.2) was applied to all the layers of the critic network, f_{ω_*}, and reduced gradient problems during training. Batch normalization on the layers of the encoder LSTM could further stabilize training, but we found this would make computation prohibitively expensive.

Second, minibatch discrimination (Salimans et al., 2016) was employed to reduce the tendency of mode collapse. This technique computes the similarity of hidden layer activations over the samples in the minibatch and provides the critic with access to this similarity measure. The intuition behind minibatch discrimination is that if the critic was able to observe an entire minibatch of data points from the generator or from the true distribution, it would notice when all the data points are the same for the generator minibatch and penalize the generator for this mode collapse behaviour. Hence, in our model we implemented a minibatch discrimination layer on the final layer activations of the critic network. A concatenation of the minibatch discrimination output and the final layer activations is then linearly mapped into a scalar, the output of the critic network.

Note that we also experimented with the standard GAN aided by minibatch discrimination. In our experiments, however, this did not sufficiently prevent mode collapse, and the model often generated inaccurate class assignments.

As mentioned before, sequence-to-sequence models are a type of autoencoder. What we have described here is an autoencoder with adversarial networks regularizing the latent space, namely an **adversarial autoencoder**. Next, we describe work relevant to this approach.

4.3 Related work

The design of our model is partly inspired by the work of Makhzani et al. (2015), in which they proposed adversarial autoencoders with one or two latent representations. Their autoencoders made use of convolutional neural networks (CNNs) for the encoder and decoder functions, whereas our model employs LSTMs for the compression of sequences. CNNs are neural networks that are specialized for processing a grid of values, such as the pixels in an image, whereas LSTMs are specialized for processing a sequence of values $\mathbf{x}_1, ..., \mathbf{x}_T$ and are therefore more suitable to neural signals.

Kingma et al. (2014) proposed an end-to-end trainable variational autoencoder (described in section 6.4) with two latent representations, one categorical and one Gaussian representation. For unlabelled data, difficulties arise because the categorical distribution is not reparametrizable and therefore not differentiable. Their solution is to marginalize out the categorical representation over all classes. However, when the number of classes becomes large, this marginalization becomes prohibitively expensive.

Fortunately, a differentiable approximation to the categorical distribution was recently proposed and is known as the *concrete relaxation* (Jang et al., 2016; Maddison et al., 2016). This approximate categorical distribution allows us to train our favourite autoencoder model with stochastic gradient descent and without the costly marginalization over all categories; it was shown to train faster and achieve better accuracies than the marginalization approach (Kingma et al., 2014) on a semi-supervised classification task. Furthermore, Jang et al. (2016) achieved a classification accuracy of 93.6% on the binarized MNIST dataset (Salakhutdinov and Murray, 2008) with concrete relaxation (Gumbel-softmax), whereas Makhzani et al. (2015) achieved 98.1% on the standard MNIST dataset with adversarial autoencoders. Although the datasets are not exactly the same, we proceed with the adversarial autoencoder due to its significantly higher accuracy.

On another note, multiple studies have applied recurrent neural networks to neural signals from the brain (Carnevale et al., 2015; Mante et al., 2013; Rajan et al., 2016; Sussillo and Abbott, 2009; Sussillo et al., 2015). One such study that is closely related to ours is that by Pandarinath et al. (2017), who proposed the latent factor analysis via a dynamical systems method. The method uses a complex variational autoencoder (see section 6.4), with an encoder consisting of two bidirectional gated recurrent unit (GRU) networks and another one-way GRU, called the controller. A decoder GRU then generates dynamic factors and firing rates of the neural signals from samples of the latent distribution. The dynamic factors generated at each step of the decoder are used as additional input to the following step of controller GRU. They apply this method to a variety of monkey and human motor cortical datasets, demonstrating the model's unprecedented accuracy for modelling neural population activity. Compared with our data, their datasets were recorded for notably shorter durations and were well-labelled by having participants closely monitored during the recording of their neural signals.

Electromyography (EMG) provides an alternative to peripheral neural recording for obtaining brain control signals at the peripheries of the body. Instead of recording neural signals, EMG records electrical activity produced by skeletal muscles; for human-machine interfacing this provides the major benefit of being non-invasive. Yousefi and Hamilton-Wright (2014) thoroughly reviewed machine learning classification techniques for EMG

signals. To highlight relevant findings, classification accuracies above 90% were achieved with feedforward neural networks (Christodoulou and Pattichis, 1999) and above 97% with support vector machines (Dobrowolski et al., 2012). Although these high accuracies are promising, EMG signals suffer from muscle activity cross-talk and as a result, only relatively simple classification tasks have been studied (Ahsan et al., 2009). Our direct access to the peripheral nervous system should theoretically enable human-machine interfacing with a higher accuracy and resolution than the EMG approach. Additionally, the peripheral nervous system allows precise feedback methodologies for motor sensitivities, making the analysis of peripheral neural signals an interesting study.

4.4 Datasets

In this section, we introduce the datasets analysed in this study.

4.4.1 Synthetic dataset

Before introducing the dataset of peripheral neural signals, we describe a synthetic dataset used to confirm that the model works as intended. This dataset contained signals from four different classes; sine, cosine, saw-tooth, and square waves. All of the signals had an amplitude of one, with a period randomly selected from a uniform distribution $\mathcal{U}[5, T-5]$, where T is the total number of time steps in the sequence. The dataset comprised 150,000 signals, with a data split of 70:10:20 (train:validation:test).

To relate this to our label-inference goal, a signal in this dataset could be assumed to have a single latent class variable responsible for the generation of the signal, e.g., a sine wave. The style representation would then include the latent variables that correspond to the frequency of the signal.

4.4.2 Peripheral nervous system dataset

Cambridge Bio-augmentation Systems[3], a Cambridge-based startup, design neural interfaces allowing long-term neural recordings. An IT2 PNS implant was used to continuously record peripheral nerve data during the first trial of the device. Over a six-week period, at a sampling rate of 30 kHz, fifteen channels of extracellular recordings were gathered from the tibial nerve of a naturally mobile porcine specimen. These recorded signals could contain artefacts due to the biological changes or activity of the specimen. Therefore, we chose to analyse

[3] `https://cbas.global/`

data from a 2-hour window three days after surgical implantation, which was known to be of sufficient quality. Spikes in the signals were determined by means of a 26 μV threshold in the *Neuroware* software package `http://www.trianglebiosystems.com/neuroware.html`.

A first approach to inferring labels from the neural signals would be to analyse the signal profiles on all 15 channels after a spike has been detected on any channel. The hope is that both the correlation between channels and the dynamics of each channel would be indicative of the latent generating variable, which stems from the performed action (class). Hence, a dataset was constructed which contains the raw signals for all 15 channels over the $T = 50$ time steps after a spike was detected on any channel. With the signal values rescaled, this **raw-neural** dataset contained data points $\{\mathbf{x}_{1:T}^{(n)}\}_{n=1}^{N}$, $\mathbf{x}_t^{(n)} \in [-1,1]^{15}$ with $N = 250,911$.

When inferring actions from neural signals, the raw signals would most likely be too noisy and not contain all of the required information in short-term subsequences. Each data point in the raw-neural dataset spans a time of only 0.00167s. It might be more useful to infer actions from the firing rates on the different channels. Thus, we constructed a dataset consisting of the spike counts on each channel using a 0.1s time window; i.e., a new sequence is created from the number of spikes detected in every contiguous 0.1s interval. From this new spike count sequence, subsequences with the same length as those in the raw neural dataset were created. The resulting **spike-count** dataset, with rescaled counts, contained data points $\{\mathbf{x}_{1:T}^{(n)}\}_{n=1}^{N}$, $\mathbf{x}_t^{(n)} \in [0,1]^{15}$ where $N = 6,840$ and $T = 50$.

Two 4-minute video recordings of the porcine specimen were taken during the two hours of the analysed dataset. Retrospective analysis of the video allowed us to identify five apparent actions performed by the porcine specimen; walking forwards, standing, shuffling, reversing, and turning. With the video synchronized to the extracellular recordings, we were able to label subsequences of the neural signals according to the video with a precision of 0.1s. This resulted in a total of 3,003 and 74 labelled data points in the raw-neural and spike-count datasets respectively. Via this test dataset, we are able to assign actions of the specimen to the $\mathbf{y}$-representation, and in doing so, obtain a proxy for how well we would be able to infer actions of the specimen solely from their peripheral neural signals.

In this case, the latent class variables responsible for the generation of each subsequence are assumed to be related to the action performed by the porcine specimen during that interval. Thus, through identification of the latent class variables, we could infer the corresponding action of the specimen. Note that the peripheral nervous system activity could be controlling other physiological processes in addition to motion. Over a long enough duration of natural mobility, our model may be able to find regularities in the neural signals that solely pertain to motion.

For both variations of the neural data, the labelled data points were held out during training. Additionally, a random 20% of the remaining data points were selected for validation purposes.

4.5 Results

Several experiments were performed in the process of developing the proposed model. Before we report the model performance, we provide a brief description of relevant avenues of the in-development experiments and the ensuing model design choices:

- Following Bowman et al. (2015), we made use of a single layer LSTM for both the encoder and decoder. More layers are known to exacerbate gradient problems during training, and this particular sequence-to-sequence model is not extremely gradient friendly.
- Arjovsky et al. (2017) recommended the *RMSProp* (Hinton et al., 2012) optimizer instead of the Adam optimizer for improved stability of Wasserstein GANs. We experimented with both optimizers and with standard stochastic gradient descent. No discernible difference was observed, and we decided to proceed with the *Adam* optimizer and a learning rate of 0.001.
- For gradient clipping (eq. 2.25) we experimented with maximum norms τ of 1, 2, 3, 5, 10, and ∞; setting $\tau = 3$ yielded the most stable training regime.
- The squared error (eq. 4.6) increases very slowly near the origin. Our input values were scaled to range from 0 to 1 and as a result, the squared error would quickly vanish. We experimented with the absolute error, which grows at the same rate in all locations. This is akin to modelling the noise in the reconstruction as a Laplacian distribution (Murphy, 2012, §7.4). No significant difference was observed, and we decided to proceed with the conventional squared error.
- With the standard regularized sequence-to-sequence model (section 4.2.1), performance was found to be sensitive to the importance λ_1 and λ_2 placed on the regularization terms. We experimented with values ranging from 0.1 to 0.00001 and found 0.001 for both hyperparameters to provide the best performance.
- In order to prevent overfitting, a smaller latent representation was used for the smaller spike-count dataset. Maintaining the notation of section 4.2, we denote the number of elements in vectors $\mathbf{y}$ and $\mathbf{s}$ by A and B respectively. For the spike-count dataset

we set $A = 20$ and $B = 44$. For the synthetic, MNIST (explained below), and neural-raw datasets we set $A = 30$ and $B = 98$. Note that the number of labels in the latent representation is larger than the number of true labels (5). Here we assume that there might be many more important latent generating variables than the actions we identified. However, if we only care about the identified actions, multiple latent variables can be assigned to the same action. Moreover, Makhzani et al. (2015) showed that having more elements in the **y**-representations increases accuracy, which supports our motivation.

In addition to the datasets introduced in section 4.4 we evaluated the performance of the proposed model on another well-understood dataset. This dataset is the MNIST dataset (section 3.4.3), downsampled to a resolution of 7×7 with the same original data split. For all datasets, the best model was chosen based on the lowest reconstruction error achieved on the validation data within 100 epochs of training. The following section reports the accuracies achieved on the four datasets.

4.5.1 Classification accuracy

We evaluated the classification accuracy with the following procedure:

1. Let $\mathcal{X}$ be the set of N test sequences with corresponding one-hot labels $\{c_k^{(n)} = e_k^{(k)}\}_{n=1}^{N}$, $k \in 1, ..., K$, where K is the number of categories[4].

2. For each dimension $b \in \{1, ..., B\}$ in the latent class representation, $\mathbf{y} \in [0,1]^B$, find the subset $\mathcal{X}_b \subset \mathcal{X}$ of sequences that have maximum probability $p(y = b|\mathbf{x}^{(n)})$

$$\mathcal{X}_b = \{\mathbf{x}^{(n)} | p(y = b|\mathbf{x}^{(n)}) \geq p(y = a|\mathbf{x}^{(n)})\}, \quad \mathbf{x}^{(n)} \in \mathcal{X}, \quad \forall a \in \{1, ..., B\}. \tag{4.15}$$

3. For each of the M sequences, $\mathbf{x}^{(m)}$ in the subset $\mathcal{X}_b$, scale the corresponding true label $c_k^{(m)}$ by the probability $p(y = b|\mathbf{x}^{(m)})$. Then assign to y_b the maximum label of the average weighted true labels

$$y_b = \underset{k}{\operatorname{argmax}} \frac{1}{M} \sum_{m=1}^{M} c_k^{(m)} p(y = b|\mathbf{x}^{(m)}), \quad \mathbf{x}^{(m)} \in \mathcal{X}_b. \tag{4.16}$$

4. Once each element of **y** has a corresponding true label, we can compute the accuracy of the assigned labels.

[4] $\mathbf{e}^{(k)}$ is a standard basis vector $[0, \ldots, 0, 1, 0, \ldots, 0]$ with a 1 at position k.

In table 4.1 we show the classification accuracies obtained for the various datasets. The top row indicates the regularization method used for the latent space of each sequence-to-sequence model; a model without regularization, a model with standard regularization (section 4.2.1), and a model regularized by adversarial networks (section 4.2.2). Mean and standard deviation of the accuracies for ten independent runs are shown.

Table 4.1 Unsupervised labelleling accuracies [%] for different regularization techniques

Dataset	None	Standard	Adversarial
Synthetic	82 ± 4.5	70 ± 9.1	86 ± 1.5
Low-resolution MNIST	38 ± 2.0	37 ± 3.1	80 ± 2.9
Raw-neural	56 ± 0.5	57 ± 1.0	63 ± 0.4
Spike-count	70 ± 3.6	70 ± 2.3	76 ± 3.9

First, the results confirm that our model works as intended – the accuracies obtained for the synthetic and MNIST datasets are well above random guessing. A lower classification accuracy on MNIST than the 85.6% reported by Makhzani et al. (2015) was expected because we used a low-resolution version of MNIST, making the digits harder to distinguish. Second, the results indicate that among the regularization approaches, the adversarial approach yields the best classification accuracies. The sequence-to-sequence model without regularization occasionally yielded good classification results but did so very inconsistently. Moreover, although GANs are notorious for their instabilities, the final adversarial approach produced no numerical issues during experimentation, whereas the other approaches occasionally did.

The accuracies obtained for the neural data exceeded our expectations, given the crude approach for assigning "true" labels to the subsequences. Results confirm that the motivations for constructing the spike-count dataset were indeed correct – the classification accuracies for the spike-count dataset are higher for all the regularization approaches. It remains difficult to discern exactly which factor of the spike-count dataset construction contributes the most to the improved performance. It could be that counting spikes reduces noise significantly, or it could be the longer temporal dependencies that are captured in each data point.

An important metric to consider is the average reconstruction error on the test dataset, which we tabulate in table 4.2. As with the accuracies, we show the reconstruction errors for the three different regularization approaches; no regularization, standard regularization, and regularization with adversarial networks. The mean and standard deviation of the squared error for ten independent runs are tabulated.

From the reconstruction results, it is evident that the adversarial approach results in larger reconstruction errors. This supports our use of the adversarial regularization approach,

Table 4.2 Sequence-to-sequence reconstruction errors for different regularization techniques

Dataset	None	Standard	Adversarial
Synthetic	0.01290 ± 0.01017	0.12891 ± 0.11364	0.15167 ± 0.08092
Low-resolution MNIST	0.06412 ± 0.04837	0.20110 ± 0.18074	0.31454 ± 0.10185
Raw-neural	4.52337 ± 0.41495	4.86144 ± 0.08559	6.17025 ± 0.06735
Spike-count	5.78619 ± 0.14712	6.34200 ± 0.09203	6.56994 ± 0.17788

All values are magnified $\times 1000$

which enables increased control of the latent space and could, therefore, provide stronger regularization.

On average, the reconstruction error for the raw-neural and spike-count datasets is higher than that of the synthetic and MNIST datasets. It was found that the model is able to reconstruct the MNIST and synthetic datasets almost perfectly, as shown in figure 4.5. For comparison, we illustrate the reconstructions of the raw-neural and spike-count datasets in figure 4.6. We show reconstructions with and without teacher forcing, with the difference between the two providing an indication of how much information is captured in the latent representation which has now been learned.

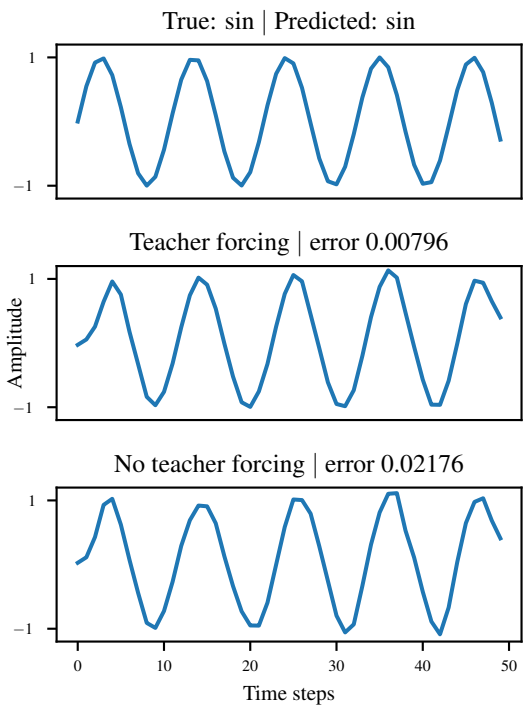
If we compare the reconstructions in figure 4.5 to those in figure 4.6, it is clear that the model had much greater difficulty with the neural data. The reconstructions of the neural data seem to be a smoothed version of the original inputs. This could mean that the smoothed trends were found to be the most important features of the dataset. Another reason for the reconstruction difficulty could be that the neural data are multivariate, making compression much harder.

So far we have only looked at metrics of our latent space, but an important aspect to consider is what the learned latent space looks like. In the following section, we take a closer look at the generated latent representations.

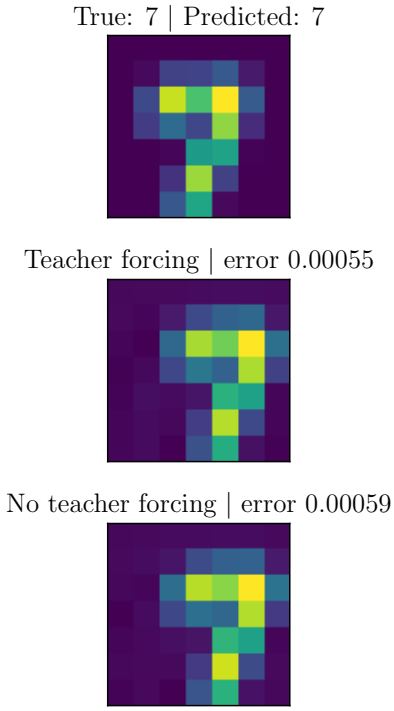
4.5.2 Cluster separation in the latent space

Owing to the **y**-representation not being a fully one-hot encoded representation, we are able to extract more information from it. To visualize these high-dimensional representations we use the t-sne method introduced in section 3.5.2.

A natural question is whether there is truly a need for the additional regularization on the latent style representation **s**. It turns out that this question is answered by addressing one of our concerns with the peripheral nervous system dataset – subsequences of the signals that are close to each other in time are expected to be more similar than subsequences that



(a) Synthetic reconstructions

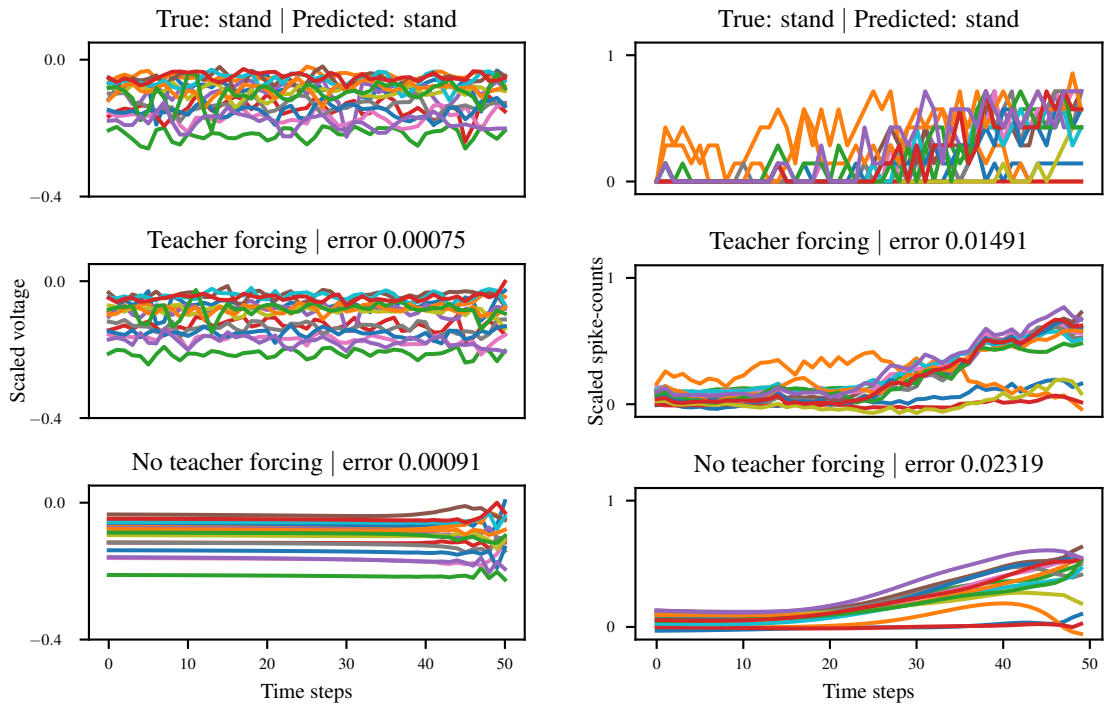


(b) MNIST reconstructions

Fig. 4.5 Reconstruction examples for the synthetic dataset (a) and for the low-resolution MNIST dataset (b). The top row of each image is the original input, with the true and predicted labels indicated in the title. The middle row shows the reconstructions of the input with teacher forcing, and the bottom row illustrates the reconstructions without teacher forcing. The squared error for each reconstruction is also shown. A more similar reconstruction with and without teacher forcing means that most information is conveyed in the latent representation.

are further apart in time. Thus, the model could merely learn to cluster subsequences that are close together in time. Instead, we seek a model that clusters subsequences based on the content that is independent of location in time.

To ensure that the model does not merely learn to cluster subsequences that are contiguous in time, we plot the latent label representation $\mathbf{y}$ using t-sne and label the different data points based on when they were recorded. Furthermore, we trained a model with the same hyperparameters as those used for the best raw-neural dataset model, but without regularization on the latent style representation $\mathbf{s}$. This model achieved accuracies similar to that where both representations are regularized. However, a closer look at the latent space, annotated by the time of recording, revealed that regularization on the latent style representation is required to obtain clustering that is less dependent on the time of the recording.



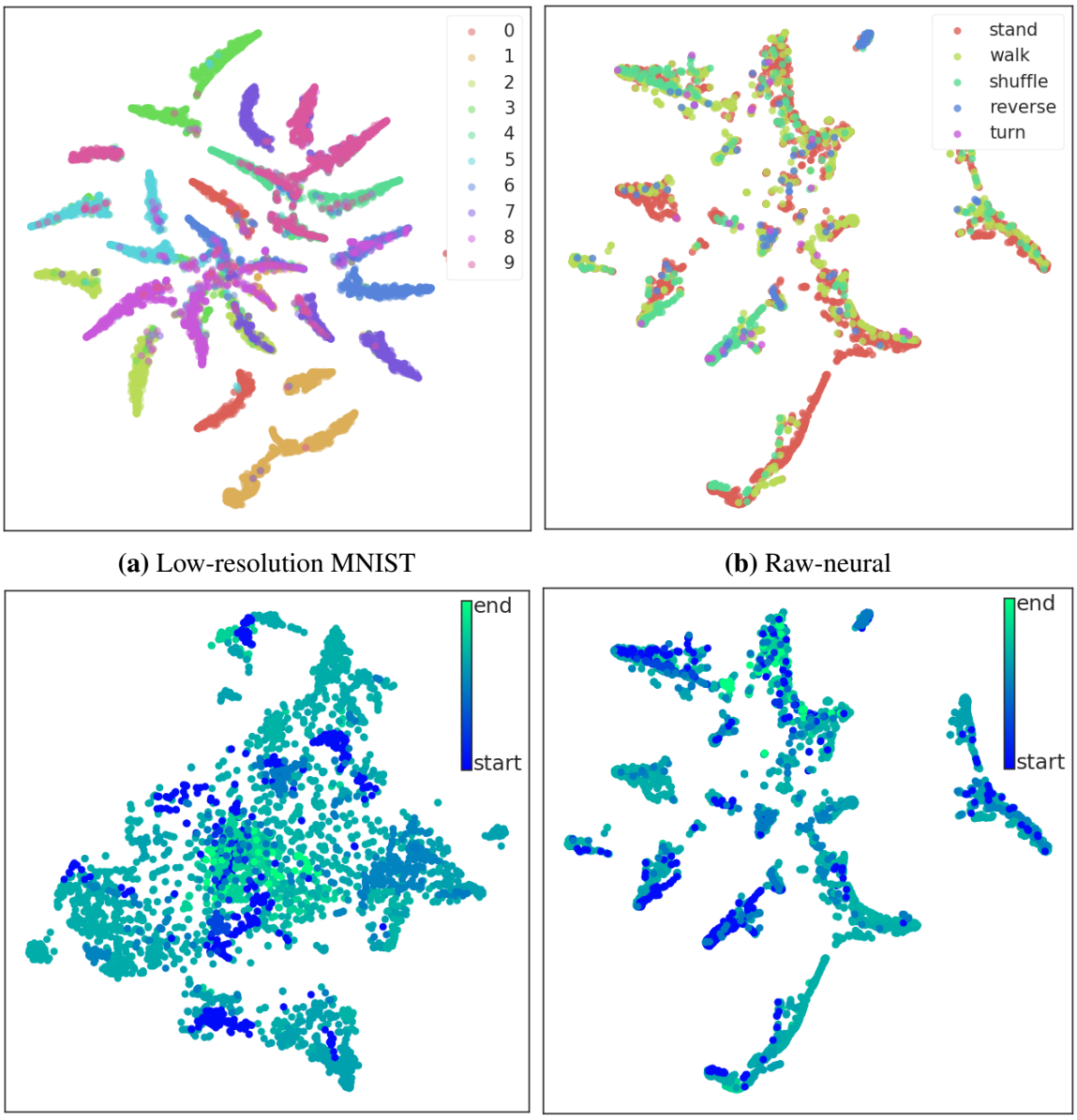
(a) Raw-neural reconstructions

(b) Spike-count reconstructions

Fig. 4.6 Reconstruction examples for the raw-neural dataset (a) and for the spike-count dataset (b). The top plots are the original inputs, with the true and predicted labels indicated in the title. The middle plots are the reconstructions with teacher forcing and the bottom plots are the reconstructions without teacher forcing. Different colour lines denote the 15 different recording channels (electrodes). The squared error of each reconstruction is also shown. Comparing the reconstructions with and without teacher forcing reveals the information conveyed by the latent representation, with more similar reconstructions indicating an increase in latent representation information. These reconstructions are not as accurate as those in figure 4.5 because the data are much harder to model.

Figure 4.7 shows the t-sne plots of the latent label representations $\mathbf{y}$ for different models. The latent representation of the MNIST model clusters well into 30 (size of the latent representation) different areas, with each cluster relatively homogeneous. In comparison, the difficulty of generating informative representations for the raw-neural dataset is elucidated in figure 4.7b. Here, it is much harder to find 30 distinct clusters, and clusters are heterogeneous.

In figures 4.7c and 4.7d we illustrate the distribution of the time of recording over the data points in the latent space. It is clear that the clusters obtained when regularizing the latent style representation are more independent of time, with clusters often containing data points from both the start and end of the entire recorded signal. Note that the specimen reversed once during the test dataset period, which means that all of the reverse-labelled subsequences are contiguous in time, and in turn, explains the homogeneity of the reverse cluster.



(a) Low-resolution MNIST

(b) Raw-neural

(c) Raw-neural time distribution
(no regularization on the **s**-representation)

(d) Raw-neural time distribution

Fig. 4.7 t-sne embedding of the **y**-representations generated for each data point in the test datasets. Representations for the low-resolution MNIST dataset are shown in (a) and for the raw-neural data in (b). Plots in (c) and (d) show the distribution of time over the data points. The regularization of the **s**-representation results in clusters that are less dependent on the time of recording; clusters occasionally contain subsequences from both the start and end of the recording period.

Although the clusters for the raw-neural data are heterogeneous, there is some discernible structure. One reason for the heterogeneity could be our crude labelling of the subsequences, where each subsequence could actually indicate something different to the assigned label. For example, it is highly probable that while the porcine specimen was walking forwards, signals indicative of standing might have been present as well.

4.6 Discussion

In this chapter we proposed a new solution for the important problem of learning with limited labelled data. The proposed solution entails the regularization of a sequence-to-sequence model with adversarial networks in order to generate latent representations that have the desired, label-like characteristics. When labels are sparse, are incorrect, or lack precision, the proposed model allows inference of enhanced labels. We demonstrated the efficacy of this approach on a naturally mobile porcine specimen.

Quantifying the benefits gained from this unsupervised approach is difficult for the dataset of peripheral nervous system signals. We admit that the actions sought to be inferred from the data are perhaps too high-level for the tibial nerve to convey. The leg of the porcine specimen performs multiple different movements during any one of our identified actions. Moreover, the subsequences of the signals cannot be labelled with high precision, which probably makes the assigned labels incorrect for many subsequences. Given the difficulties, the results are impressive, but a more rigorous labelling approach will need to be followed during subsequent data collection.

To our knowledge, this is the first machine learning study on such a peripheral neural signal dataset. In a downstream task, access and understanding of such data could equip amputees with devices that interface with their peripheral nervous system. Moreover, due to the label representations generated by our model not being fully one-hot encoded, these representations could provide machines with combinations of actions from the user. For example, instead of signalling to the device that the subject is either walking or turning, the representation could signal a little bit of both, allowing the subject to turn and walk.

A few interesting avenues of research are presented by the peripheral nervous system dataset. One question is whether a trained model will maintain the same predictions with the biological changes at the location of the implant, as the surgical site heals and the subject develops over time. On the other hand, the design of a suitable training approach, which is able to update the model over the lifetime of the subject, could ensure that the biological changes and plasticity in the subject's peripheral nervous system are reflected in the model. Moreover, when data from more specimens are available, an important goal would be to

develop a modular model that is easily transferred to different specimens, as attempted in Pandarinath et al. (2017).

Effectively, the LSTM offers flexibility in the way we can model inputs and latent representations. This flexibility has lead to a creative model that would be infeasible for conventional machine learning techniques. Such unsupervised methods are currently less-developed than their supervised counterparts, but as shown in this study, they have some of the greatest potential due to the significant cost of labelling large amounts of biomedical data (Ching et al., 2018). Our proposed model could mitigate this labelling cost by summarizing large amounts of neural signals into interpretable clusters.

This chapter demonstrated the power of LSTMs when used in a sequence-to-sequence fashion. In chapter 3 we demonstrated the applicability of LSTMs to medical data and showed how it is possible to know when the LSTM is uncertain. In the following chapter, we show that well-designed visualization techniques can improve our understanding of LSTMs.

Chapter 5

Visualization techniques for LSTMs applied to medical time series

The recent spurt of progress in deep learning (Krizhevsky et al., 2012; Lanchantin et al., 2017; Sutskever et al., 2014; Zhou and Troyanskaya, 2015) has enabled computers to tackle many problems which were previously beyond them. The result has been a machine learning boom, with computers moving into everything from self-driving cars to insurance and medical diagnosis.

There is a snag, however. People struggle to understand exactly how the self-adapted deep learning models do what they do (Economist, 2018). When algorithms are handling relatively low-risk tasks, such as playing Go or recommending a film to watch, this "black box" problem can be safely ignored. When they are deciding how to steer a car through a crowded city or what treatment a patient should receive, it is potentially harmful. And when things go wrong – as, even with the best system, they inevitably will – then customers and regulators will want to know why. Regulators have already stepped in; the European Union recently adopted new rules regarding the use of personal information, the General Data Protection Regulation (Goodman and Flaxman, 2016). A component of these rules is summarized by the phrase "right to an explanation", implying that users must be able to explain how their machine learning algorithms make decisions. For example, a clinician aided by a machine learning model should be able to explain how decisions are based on the patient's data. Although this regulation is mainly aimed at recommendation and categorization systems, it provides a hint of similar future ramifications. The main concern, and rightly so, is that machine learning models base their decisions on the right information in the data.

From a different perspective, understanding the patterns in data may be just as important as fitting the data. A model that achieves breakthrough performance may have identified

patterns in the data that practitioners in the field would like to understand – which will not be possible if the model is a black box.

In 1996, Tu (1996) argued that despite the advantageous representational capacity of neural networks, their adoption into medical practice would be limited by their opaqueness. Twenty-two years later, clinicians still distrust these models due to their opaqueness (Rajkomar et al., 2018). It is difficult to establish trust in a model of which the logic is not understood – especially when we opt for a combination of human experts and model-based decision-making systems.

Peering into the workings of the human brain requires expensive brain-scanning machines. Fortunately, inspecting the mechanisms of deep learning techniques are not as expensive, but it does require an active effort nonetheless. Research has revealed fundamental insights into the operation of convolutional neural networks (CNNs) by means of deconvolution, which elucidates filter operations (Zeiler and Fergus, 2014), and via saliency maps projected on to the input image (Simonyan et al., 2013). While image recognition systems have the benefit of easily visualizable data transformations (especially if based on CNNs), the typical medical dataset analysed by LSTMs lacks this luxury.

LSTMs are complicated and still contain many components whose roles are not well understood (Balduzzi and Ghifary, 2016). In this chapter, we show how visualization techniques can illustrate what LSTMs consider salient when analysing continuous-valued medical time series, with a specific focus on single-lead electrocardiograms. We learn from and compare recently proposed visualization techniques, which we describe next.

5.1 Related work

An inspiring study on the interpretability of convolutional neural networks (CNNs) was done by Zeiler and Fergus (2014). They proposed the deconvolutional network approach, which projects activations on each layer of a CNN onto the pixel space. This technique showed that CNNs learn simple filter-like operations on the lower layers, which are combined in higher layers to extract complex features in an image. The deconvolutional approach provides an interpretation of both how CNNs analyse images, and what they look for. Answering the "what" question means determining the properties of the functional mapping. The "how" question investigates the internal mechanisms that enable the mapping to achieve these properties. For LSTMs the "how" question remains a research challenge; here we show that the "what" question can be described effectively by their input-output relationships.

A common form of interpretability for deep learning models is through attention mechanisms. These mechanisms have been used in diverse problems such as image captioning

and machine translation to select portions of the input to focus on for generating a particular output. By revealing which input features are used for different outputs, the attention mechanisms provide insights into the model's decision-making process. Multiple studies have used attention mechanisms to determine salient inputs (Bahdanau et al., 2015; Ching et al., 2018; Deming et al., 2016; Rajkomar et al., 2018; Xu et al., 2015). In the clinical domain, Choi et al. (2016b) leveraged attention mechanisms to highlight which aspects of a patient's medical history were most relevant for making diagnoses. Choi et al. (2017) later extended this work to take into account the structure of disease ontologies and found that the concepts represented by the model aligned with medical knowledge – an exercise we strive to imitate. However, these interpretation strategies that rely on an attention mechanism do not provide insight into the logic used by the attention layer.

Back-propagation-based methods are also popular for interpreting deep learning models. They have the signal from a target output unit back-propagated to the input layer. The simplest form of this was proposed by Simonyan et al. (2013), a technique that we also investigate (section 5.2.3). Improved versions of the back-propagation approach have been proposed; the work by Bach et al. (2015) employed a strategy called Layerwise Relevance Propagation, which was shown to be equivalent to the element-wise product of the gradient and input (Kindermans et al., 2016). Moreover, guided backprop (Mahendran and Vedaldi, 2016; Springenberg et al., 2014) combines back-propagation techniques with the deconvolutional network approach to produce sharper visualizations. However, as with the attention mechanisms, many of these visualization techniques require architectural modifications (Fong and Vedaldi, 2017).

On the other hand, perturbation-based interpretation approaches do not require changes in the model architecture; instead, they change parts of the input and observe the impact on the output of the network. These include visualizing the drop in classification score as constant value masks are applied to different input patches of images (Zeiler and Fergus, 2014). A recent study by Fong and Vedaldi (2017) proposed to use gradients to learn the minimal input "deletions" that minimize the class score. This technique requires access to only the model's inputs and outputs and provides aesthetically pleasing explanations of image input salience. We modify this technique to visualize continuous inputs that are analysed by LSTMs.

Visualizations of recurrent neural networks (RNNs) have been explored in natural language processing. Li et al. (2016) visualized RNN embedding vectors to show how RNNs achieve compositionality in natural language for sentiment analysis as well as visualizing the influence of input words on classification. Similarly, Karpathy et al. (2015) analyzed the interpretability of RNNs for language modelling, demonstrating the existence of interpretable network units which are able to focus on specific language structures such as quotation marks

in text. In both studies, the aim was to improve the understanding of RNNs by leveraging human knowledge of the structured language. Such well-understood discrete input lends itself to interpretable isolated explanations for the importance of each input to the model. In the medical paradigm, firstly, we are less certain about the underlying biological processes that govern the generation of physiological signals, and secondly, due to inputs being continuous, isolation of salient features is more difficult.

RNNs have also been visualized by Lanchantin et al. (2017) in the domain of DNA sequences. Given a DNA sequence, the model is trained to predict whether there is a binding site for a particular transcription factor. Compared with our work, the DNA inputs are discrete instead of the continuous-valued medical time series that we analyse. We explore the same three techniques that they investigated and two additional techniques, which we show to be more effective. These five visualization techniques are described in the following section.

5.2 Visualization techniques

Before describing the five visualization techniques explored, some preliminaries are required. Each input sequence $\mathbf{x}_{1:T}$ with T time steps has a corresponding class label c. An LSTM provides pre-softmax activations $\mathbf{s} \in \mathbb{R}^C$ for each input sequence, with s_c denoting the score for the correct class c of the input sequence, and C denoting the number of classes.

5.2.1 Temporal output score

A first approach to understanding LSTM classifications is to illustrate the progression of model decisions over time for a specific input sequence; i.e., incrementally longer subsequences of the original time series are classified and visualized. More formally, for a sequence $\mathbf{x}_{1:T}$ with length T we classify the prefix[1] $\mathbf{x}_{1:t}$ of the original sequence as t ranges from 1 to T (Lanchantin et al., 2017). Fortunately, the classifications for all these possible prefixes of a sequence are efficiently provided by an LSTM; we can simply record the softmax vector given by $\text{softmax}(\mathbf{V}\mathbf{h}_t)$, at each time step t, where $\mathbf{h}_t \in (-1,1)^d$ is the hidden output of the LSTM with d units, as defined in eq. 2.26, and $\mathbf{V} \in \mathcal{R}^{C\times d}$ is the output weight matrix. Note that nothing changes during training where the single label of an input sequence is compared to the classification only at the final time step T.

These predictions at each time step can then be superimposed onto the original signal, either as colour-coded predicted categories or as the probability of being in the correct class. For the latter, we record the element in the softmax vector corresponding to the true label

[1] We use the terms prefix to refer to a subsequence at the start of the original sequence.

of the input sequence. As we show later, this temporal output score technique is limited by its cumulative nature; it shows the tipping points of model decision making, but it fails to indicate which input features were most salient. A better approach could be to find the most likely input sequence for each class, which we describe next.

5.2.2 Class mode visualization

In an attempt to visualize the most likely input for a specific class, the input is optimized with respect to the class score s_c while keeping the model parameters fixed (Simonyan et al., 2013). Formally, we would like to find an L_2 regularized input sequence $\mathbf{x}_{1:T}$ that maximizes the score s_c of the class c

$$J = \underset{\mathbf{x}_{1:T}}{\operatorname{argmax}}\, s_c(\mathbf{x}_{1:T}) - \lambda \left\|\mathbf{x}_{1:T}\right\|_2^2, \tag{5.1}$$

where λ specifies the importance of the regularizer.

We optimize this objective function with *RMSProp* (Hinton et al., 2012) for 2000 iterations at a learning rate of 0.001. Several values for λ were experimented with and setting $\lambda = 0.3$ produced reasonable results. Additionally, we experimented with the use of the total variation norm (see eq. 5.4) as a regularizer to encourage smoother class mode signals, but this yielded less interpretable results.

In the work by Simonyan et al. (2013), the CNN is trained on zero-centred images, therefore they initialize the optimization with a zero image and add the mean training set image to the optimized input. Our model is trained with data that have been rescaled to a specific range. We clip the optimized values accordingly, and found the most interpretable results when initializing $\mathbf{x}_{1:T}$ to be distributed as a Gaussian distribution, $\mathcal{N}(\mathbf{x}_t; \mu, 0.01\mathbf{I})$, with μ denoting the mean of the training data.

Note that we specifically use the pre-softmax activated class scores s_c. When using the softmax layer, the maximization of eq. 5.1 could be achieved by minimizing the scores of other classes. By using s_c, we ensure that optimization concentrates on only the class c in question (Simonyan et al., 2013). Experimentation with the use of the softmax layer confirmed that it produces visualizations less sensible than those produced when using s_c.

Deep learning models, however, are susceptible to artefacts (Fong and Vedaldi, 2017; Goodfellow et al., 2014; Hu and Tan, 2017; Nguyen et al., 2015; Xiao et al., 2018), often leading to uninterpretable class modes, as shown in section 5.3.1. To elucidate what we mean by **artefacts**, a type of adversarial input, we present two examples in figure 5.1. Here, seemingly insignificant arbitrary noise added to the image completely confuses the model. To circumvent the problem of artefacts, we could instead narrow the search by finding the

relationship between the correct class score s_c and a specific input via differentiation – a technique that we describe in the following section.

Fig. 5.1 Examples of artefacts for images. Above each image is the probability assigned to the correct class by GoogLeNet (Szegedy et al., 2015). Small arbitrary injections of noise seem to completely confuse the model. We refer to such perturbations as **artefacts**, where less visible versions are commonly known as adversarial inputs (Goodfellow et al., 2014). (Adapted from Fong and Vedaldi (2017).)

5.2.3 Input derivatives

Owing to the LSTM being highly nonlinear, it is difficult to determine the influence of each input $\mathbf{x}$ on s_c. We can, however, approximate $s_c(\mathbf{x})$ with a linear function in the neighbourhood of a specific input sequence $\mathbf{x}^{(0)}$ by using the first-order Taylor expansion[2]

$$\begin{aligned} s_c(\mathbf{x}) &\approx s_c(\mathbf{x}^{(0)}) + \left(\nabla s_c(\mathbf{x})|_{\mathbf{x}^{(0)}}\right)^T \left(\frac{\mathbf{x}-\mathbf{x}^{(0)}}{1!}\right) \\ &= \left(\nabla s_c(\mathbf{x})|_{\mathbf{x}^{(0)}}\right)^T \mathbf{x} + s_c(\mathbf{x}^{(0)}) - \left(\nabla s_c(\mathbf{x})|_{\mathbf{x}^{(0)}}\right)^T \mathbf{x}^{(0)} \\ &= \boldsymbol{\omega}\mathbf{x} + \mathbf{b} \end{aligned} \tag{5.2}$$

This linear formulation allows us to see that the magnitude of the elements in $\nabla s_c(\mathbf{x})|_{\mathbf{x}^{(0)}}$ determines the importance of the corresponding elements in $\mathbf{x}$. Visualizing these magnitudes enables interpretation of the model decisions. The gradient $\nabla s_c(\mathbf{x})|_{\mathbf{x}^{(0)}}$ can, fortunately, be computed with a single step of back-propagation, and as in section 5.2.2, we back-propagate the error signal with respect to pre-softmax activations.

In section 5.3.1 we show that this is a bad approximation for LSTMs. This interpretation of salience breaks for a linear classifier; if $s_c(\mathbf{x}) = \boldsymbol{\omega}\mathbf{x} + \mathbf{b}$, then $\nabla s_c(\mathbf{x})|_{\mathbf{x}^{(0)}} = \boldsymbol{\omega}$, which is independent of the input $\mathbf{x}^{(0)}$ and therefore cannot be interpreted as salience (Fong and

[2] s_c is assumed to be smooth at $\mathbf{x}^{(0)}$.

Vedaldi, 2017). This approach determines the change in s_c for arbitrary variations $\Delta_x = \mathbf{x} - \mathbf{x}^{(0)}$ from $\mathbf{x}^{(0)}$; for a linear classifier, this change is the same regardless of the starting point $\mathbf{x}^{(0)}$. The problem is reduced but not solved for nonlinear models such as the LSTM. Later, in figures 5.4 and 5.6, we show that salient features produced by this technique are diffuse and have strong responses where no obvious information can be found in the input sequence. Instead of the arbitrary variations explored with this approach, we could follow a more controlled method – perturbing the input by deleting (occluding) subsections.

5.2.4 Occlusion

Zeiler and Fergus (2014) proposed iteratively occluding regions of an image in order to visualize which regions are salient. We apply this to the sequential paradigm of LSTMs by occluding subsequences of an input sequence with a predetermined occlusion value.

This technique iteratively occludes subsequence $\mathbf{x}_{t':t'+w}$ of $\mathbf{x}_{1:T}$ by some constant k as t' ranges from 1 to T, where w denotes the width of the occlusion. At the start and end of the sequence, the occlusion is shortened such that each element of $\mathbf{x}_{1:T}$ is occluded the same number of times. Over these iterations, a new sequence $\mathbf{v}_{1:T}$ accumulates the class score s_c of a sequence $\mathbf{x}_{1:T}$ in element $\mathbf{v}_t$ if element $\mathbf{x}_t$ is not occluded. If an input $\mathbf{x}_t$ is more important for classification, then its corresponding sum of class scores $\mathbf{v}_t$ will be high. For visualization, the sequence $\mathbf{v}_{1:T}$ is scaled to range from zero to one and superimposed onto the original input sequence. In section 5.3.1, we illustrate results from our experiments with different occlusion widths w and constants k.

This relatively manual process allows us to filter the input with more creative occlusions that could highlight bespoke input-output dependencies. For example, we might want to know the interdependence of subsequences and therefore choose to simultaneously occlude the signal by width w every 20 time steps. For the MNIST dataset, processed in scanline order (section 3.4.3), it is sensible to occlude a 5×5 pixel block of the 28×28 pixel images. In this case, the tailored filter would occlude five contiguous pixels every 28 time steps, repeated five times. In the 2D space of the image, this amounts to the same occlusions used for CNNs (Zeiler and Fergus, 2014), but the resulting salient features are different due to the model being an LSTM.

It is exactly this manual nature that also limits the efficacy of the occlusion technique. First, the correct occlusion value k that resembles a "deletion" of information is difficult to determine. Second, the width w and the structure of the occluded regions have to be specified and are fixed, leaving endless possible combinations unexplored. Next, we introduce a method that mitigates these issues by employing gradients to efficiently explore the space of possible input perturbations.

5.2.5 Learning an input mask

With focus on convolutional neural network (CNN) applications, Fong and Vedaldi (2017) recently proposed a method for learning a mask that optimally deletes information in an image. Here we describe an adaptation that makes their proposal suitable for LSTMs applied to physiological signals.

The aim is to find a mask $\mathbf{m}_{1:T}$ that minimally deletes information in the input sequence $\mathbf{x}_{1:T}$ while minimizing the class score s_c. With d denoting the number of inputs per time step t, we have $\mathbf{m}_t \in [0,1]^d$. The function ϕ that masks each element of the input is defined as

$$\phi(\mathbf{x}_{1:T};\mathbf{m}_{1:T}) = \mathbf{m}_{1:T} \odot \mathbf{x}_{1:T} + k(1-\mathbf{m}_{1:T}), \tag{5.3}$$

where k is a constant resembling a deletion of information. We seek a mask with elements that are near binary, either deleting completely or not at all, and with as few zero-elements as possible. The L_1 norm provides a suitable technique to encourage this property, but also provides enticing conditions for the mask to trigger artefacts in the LSTM (see figure 5.1). To prevent these artefacts, the total variation of $\mathbf{m}_{1:T}$ is added to the objective function, which encourages the mask to be smooth. The resulting objective function is given by

$$J = \underset{\mathbf{m}_{1:T}}{\operatorname{argmin}} \lambda_1 \left\| 1-\mathbf{m}_{1:T} \right\|_1 + \lambda_2 \sum_{t=1}^{T-1} \left| \mathbf{m}_{t+1} - \mathbf{m}_t \right| + s_c\big(\phi(\mathbf{x}_{1:T};\mathbf{m}_{1:T})\big), \tag{5.4}$$

where the first term uses the L_1 norm to minimize the size of the region masked, the second term is the total variation, and λ_1 and λ_2 weight these respective terms. For visualization the learned mask is scaled to range from zero to one, subtracted from one, and superimposed onto the original input.

Fong and Vedaldi (2017) additionally upsample the input images and jitter them to further mitigate the triggering of artefacts. Essentially, this jitter randomly translates the input by a few elements. By minimizing the expected class score over these random translations, the mask is required to have a similar effect on neighbouring elements. In our experiments, we did not find signs of artefacts, and applying jitter to the input resulted in larger-than-required masks.

All of our experiments used *Adam* (Kingma and Ba, 2015) with a learning rate of 0.001 to optimize the mask over 500 iterations. Among initial values of 0, 0.5, and 1 for the mask, 1 convincingly yielded the best results. We experimented with various values for λ_1 and λ_2, and setting $\lambda_2 = 0.001$ provided good results over all the datasets analysed. Learning an input mask was found to be insensitive to the choice of lambdas; changing them modifies the size of the important regions but the important features remain the same. The optimal value

of λ_1 for each dataset is shown in section 5.3.1 along with some of the values for k that we experimented with. In the next section, we demonstrate that this method is superior to those introduced earlier in this work.

5.3 Results

In order to apply these visualization techniques, we require a trained LSTM. Thus we begin by training three LSTM classifiers, each on a different dataset, using the cross-entropy objective function. The datasets comprise the ICU dataset (section 3.2.3), the MNIST dataset and the MIT-BIH arrhythmia dataset (section 3.4.3). Each model was trained with Adam at a learning rate of 0.001 and a minibatch size of 200. Optimization was run for 100 epochs and the validation loss was used to determine the best model. For the ICU dataset, we used a sequence length of 100 at a resolution of 40 Hz. In table 5.1 the classification accuracies are presented along with the hidden layer structure used for each model. These hyperparameters were optimized via manual search.

Table 5.1 Accuracies of the visualized LSTMs

Dataset	Hidden structure	Dropout	Weight decay	Accuracy [%]
MNIST	128x128	0.1	10^{-6}	98.5
MIT-BIH	128	0.1	10^{-3}	87.6
ICU	64	0.35	10^{-1}	99.2

As a first sanity check, the accuracy achieved on MNIST is similar to those reported by Cooijmans et al. (2016) (98.9%) and Arjovsky et al. (2016) (98.1%). Similar to medical time series, the MNIST dataset, processed in scanline order, consists of continuous-valued sequences. Being well-understood data, MNIST digits thus enable intuitive evaluation of the visualization techniques for LSTMs applied to medical-like time series.

5.3.1 Qualitative evaluation

Here we qualitatively show the efficacy of the five visualization techniques by means of examples from the test datasets. To juxtapose the techniques described in sections 5.2.3 to 5.2.5, we jointly refer to them as input feature salience methods. Following the same order of introduction, we start with the temporal output score technique.

Temporal output scores

In figure 5.2 we illustrate the temporal output scores for the MNIST and MIT-BIH datasets. For both datasets, the probability of being in the correct class, as well as the most probable class, is displayed at each time step.

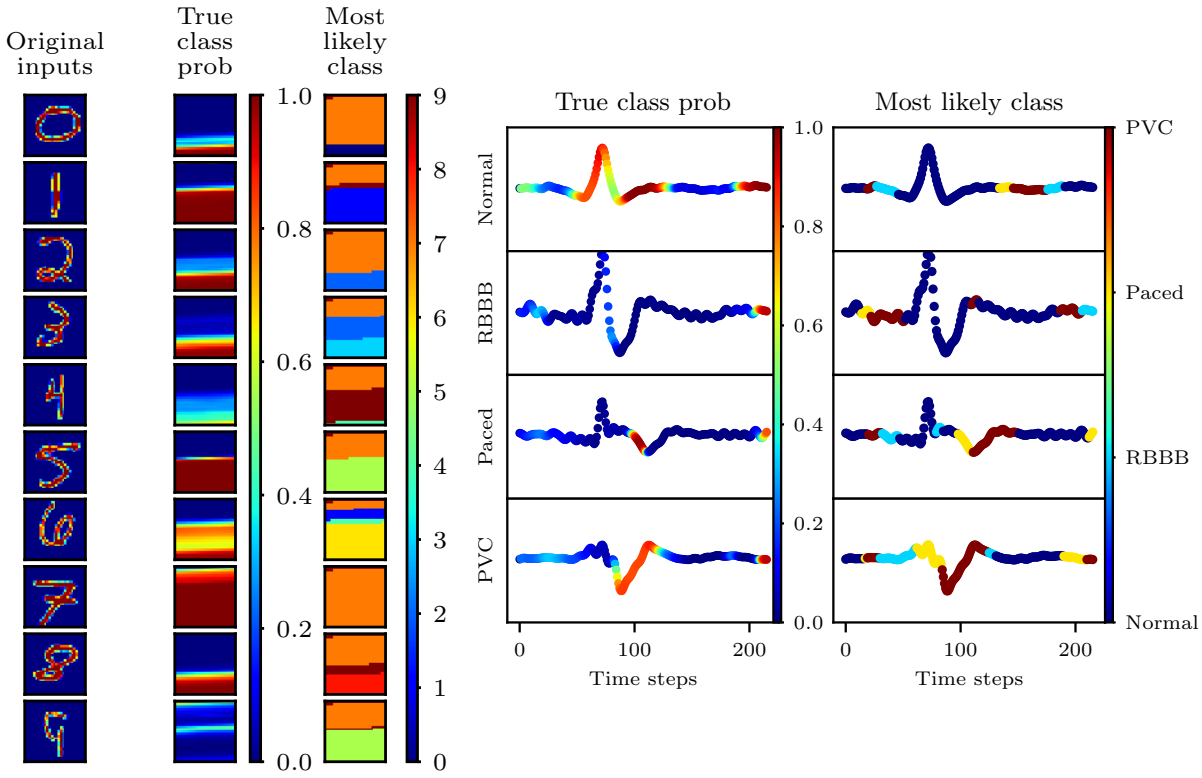


Fig. 5.2 Temporal output scores for the MNIST dataset (*left*) and the MIT-BIH dataset (*right*). The probability that the LSTM assigns to the true class is shown, along with the most likely class as predicted by the LSTM. The different classes are colour-coded according to the colour bar shown to the right of the plots. For the MIT-BIH dataset, the four classes of heartbeats are: normal, right bundle branch block (RBBB), paced, and premature ventricular contraction (PVC). The electrocardiograms all have a length of 216 time steps and are plotted with the same y-scale. This visualization technique depicts the LSTM decisions over time, and we are able to see what priors the model has learned. The minimum length required for classification is also made evident; see, for example, the four-digit of the MNIST dataset. Note, the model incorrectly classified the nine-digit example and all ECG examples were correctly classified.

Usefully, this technique allows tracking of LSTM decisions over time. Considering the most likely class on MNIST, the digits are all classified as nine for the first six time steps before the prediction switches to seven – an interesting prior learned by the model. The one-digit marginally constitutes the largest class of the MNIST training and validation datasets, with seven being the close second. We speculate that the model is quicker to identify a seven (it has a long flat region at the top) than a one, which could explain the learned prior.

With such balanced datasets, the prior could also depend on the order of the data points seen during the final epoch of training. On the other hand, the normal class of the MIT-BIH dataset is much larger than the other categories, which explains why the model initially assumes all electrocardiograms to be normal.

A neat feature of this technique, not provided by any of the other visualization techniques explored, is the ability to determine the minimum sequence length required for classification. Compare, for example, classification of the five-digit, which is confidently correct from halfway through the sequence, to the classification of the four-digit, which switches to the correct class at the very last moment – it would be detrimental to shorten the MNIST digit sequences. Similarly, for the MIT-BIH dataset, the LSTM switches to the correct class at the very last moment for all of the examples.

Although this is a fun and easy-to-implement technique, it is limited to cumulative sequential explanations. For example, in the right bundle branch block heartbeat example, *what* the model looks at is not apparent. The technique depicts when the predictions are changed, but not the extent to which each time step contributed to the prediction.

Class mode visualization

Learned optimal inputs of the MNIST and MIT-BIH LSTMs are presented in figure 5.3. Here, the susceptibility of LSTMs to artefacts causes the technique to produce illogical class modes for both models. The learned paced beat shows a glimmer of hope; there is a large spike at the start of the QRS complex (see figure 5.5), which is usually indicative of a paced heartbeat. Nevertheless, the majority of the class modes presented makes no sense. As described in section 5.2.2, deep learning models can easily be fooled to classify arbitrary noise with high confidence; a consequence of the model not having seen something similar to the arbitrary noise during training. Modern deep learning models mitigate this by means of adversarial networks, described in section 4.2.2.

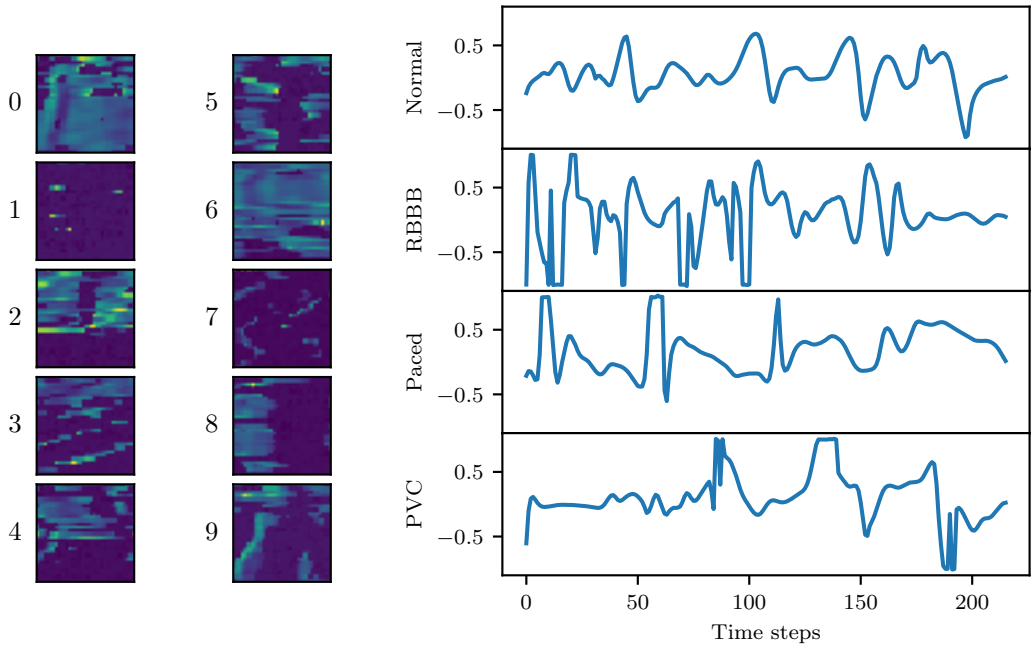


Fig. 5.3 Class mode visualizations for the MNIST dataset (*left*) and the MIT-BIH dataset (*right*). Essentially, the learned optimal input for each class is illustrated, with the classes indicated to the left of each example. For the MIT-BIH dataset, the four heartbeat categories are normal, right bundle branch block (RBBB), paced, and premature ventricular contraction (PVC). Evidently, this technique produces illogical class modes.

Input feature salience

In this section, the input derivative, occlusion, and mask learning visualization techniques (sections 5.2.3 to 5.2.5) are visually compared. Of the various hyperparameters explored for each technique, we present the set of most informative results. We start the comparison with the easy-to-understand MNIST dataset in figure 5.4. For MNIST digits, the different stroke patterns should constitute some of the salient input features, making it a good dataset for comparing the utility of different visualization techniques.

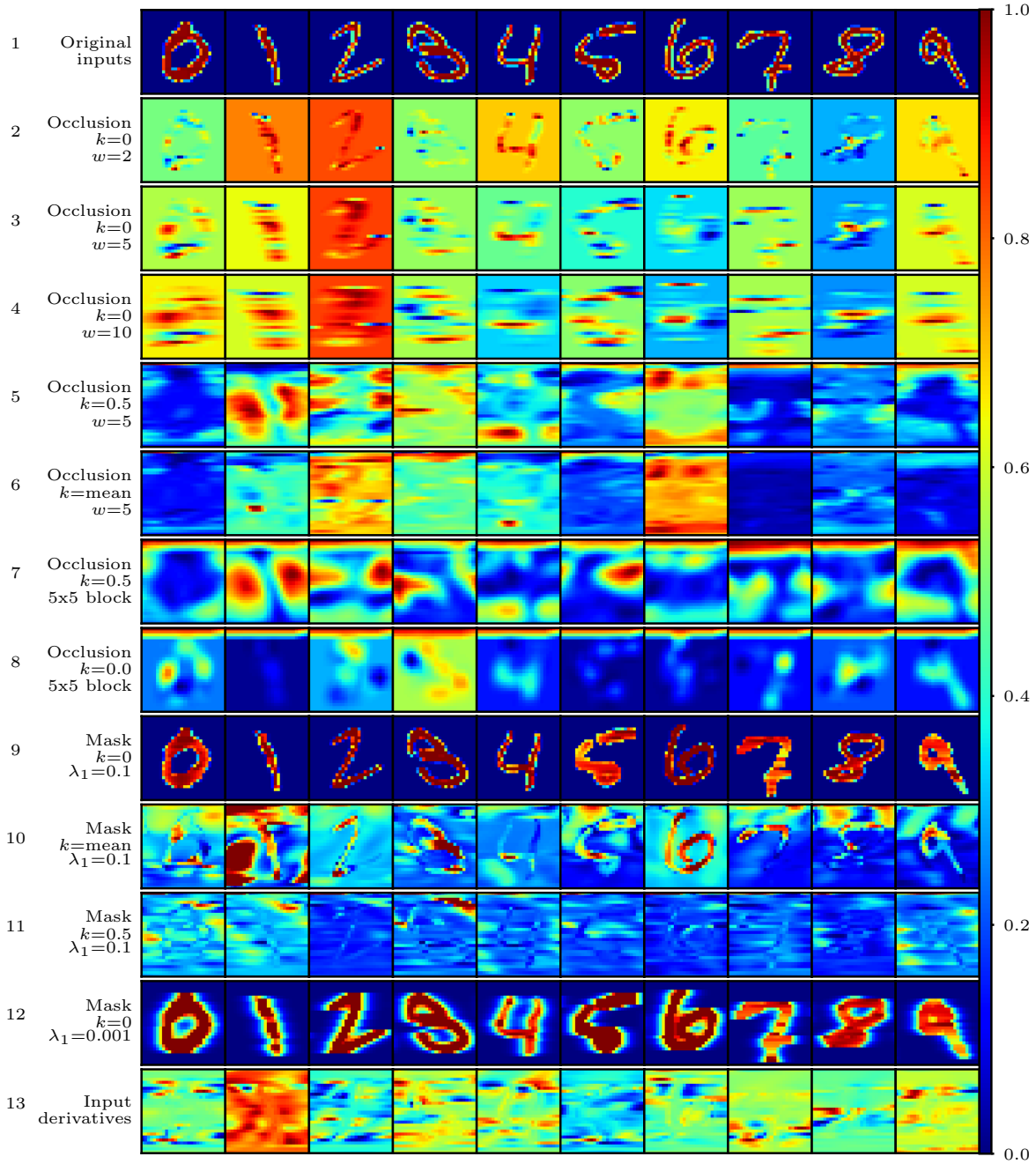


Fig. 5.4 Input feature salience for examples of the MNIST dataset. The different features that are important according to the occlusion, mask, or input derivative techniques, are displayed on a scale of zero to one, where one is important. Each column shows the analysis of the same, correctly classified, original input. The occlusion width is denoted by w and the deletion value by k, with their respective values indicated on the left-hand side. The 5×5 block refers to a carefully structured occlusion, which is the equivalent of a 5×5 pixel block being occluded for each iteration (see section 5.2.4). Most of the techniques find some interpretable salient features, with the mask learning technique, row 10, producing the best results and purely marking digit strokes as important. Interestingly, occlusion with $k = 0.5$ results in the negative of the original input being salient.

For starters, the standard occlusion technique successfully finds interpretable salient features. When the occlusion width $w = 2$, however, many seemingly important features go unnoticed, for example, the seven-digit in row 2. On the other hand, setting $w = 10$ results in overly elongated salient regions and exemplifies the difficulty of hyperparameter selection for this technique.

In the domain of MNIST digits, setting the deletion value $k = 0$ is sensible because, given the digit strokes, the background is unimportant and zero-valued. Nevertheless, occluding with $k = 0.5$ yields interpretability by producing the negative of the original input. As seen in row 7, the most effective occlusion with $k = 0.5$ is achieved when using a carefully structured 5×5 block occlusion (see section 5.2.4). When $k = 0$, however, the 5×5 block occlusion is less effective than the standard $w = 5$ occlusion (compare rows 3 and 8).

A striking property of the visualization results is how much better the masking technique performs compared with the occlusion and input derivative techniques. Setting the deletion value $k = 0$, results in a mask that covers most of the digit stroke patterns, which is intuitively the best option. Fong and Vedaldi (2017) use the mean of the input as the deletion value, which in our case yields less interpretable results (compare rows 9 and 10). It would seem that the selection of the deletion value k is task-specific.

With a better understanding of how deletion values k and occlusion widths w influence visualizations, we proceed to visualize salient features for the LSTM on the MIT-BIH dataset of electrocardiograms (ECGs). A standard electrocardiogram, with annotated cardiologist interest points, P, Q, R, S, and T, is shown in figure 5.5. With occasional reference to these interest points, we discuss the comparison of the different visualization techniques on the MIT-BIH dataset (figure 5.6).

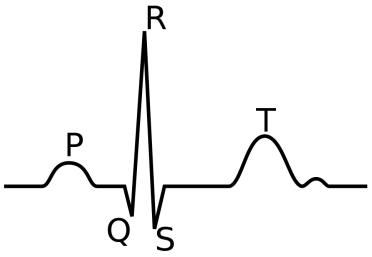


Fig. 5.5 A standard single-lead electrocardiogram with the interest points labelled.

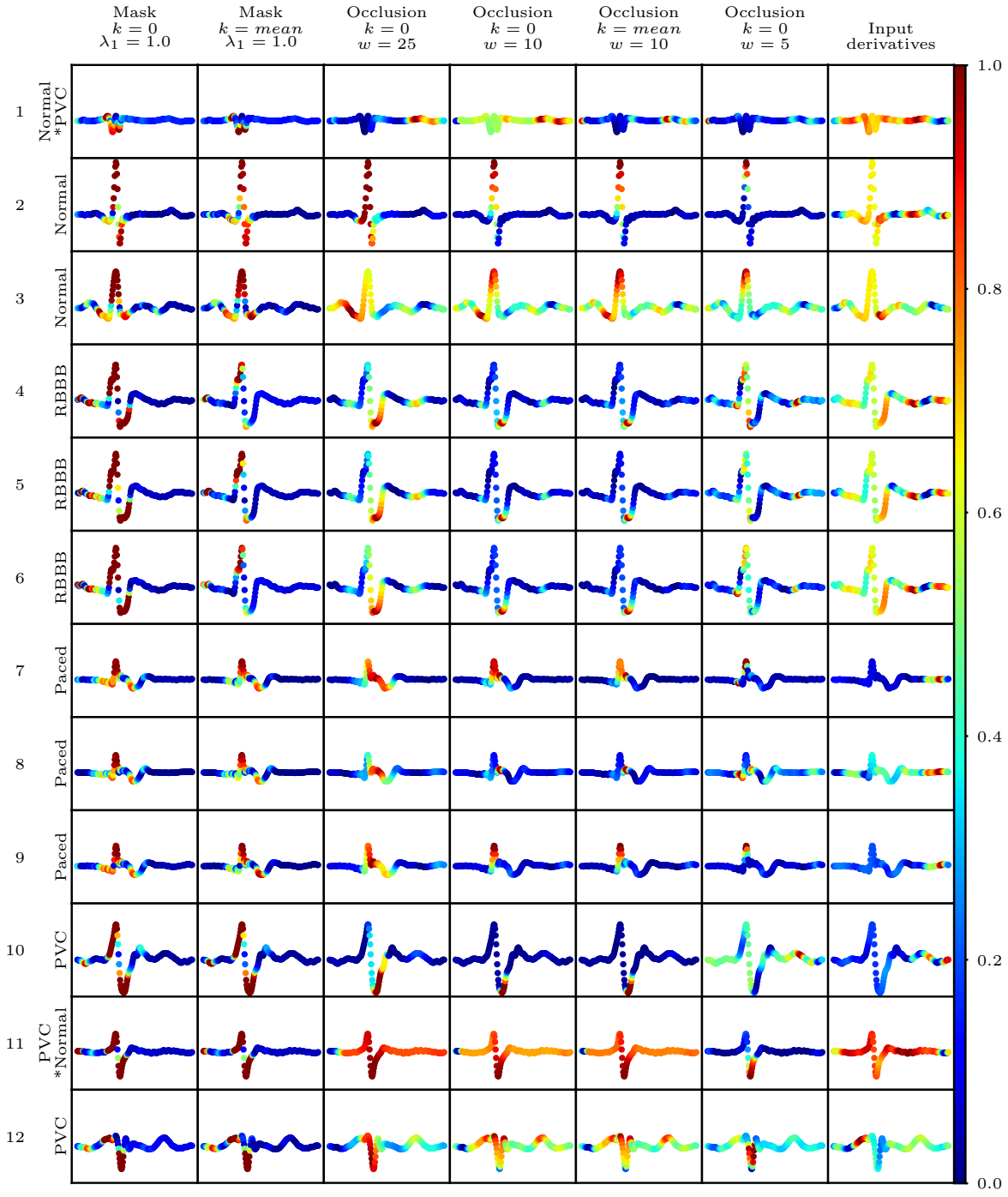


Fig. 5.6 Input feature salience for electrocardiograms (ECGs) of the MIT-BIH dataset. The input feature importance for LSTM classification, as given by the occlusion, mask, and input derivative techniques, are displayed on a scale of zero to one, where one is important. The different techniques and their settings are indicated at the top, with k denoting the deletion value, w the occlusion width, and λ_1 our preference for important-only elements in the learned mask. On the left-hand side, the true label of the ECG is indicated, with the four heartbeat categories being normal, right bundle branch block (RBBB), paced, and premature ventricular contraction (PVC). Where the LSTM incorrectly classified an input, the correct class is indicated by *. All the signals are displayed with the same y-scale and have a length of 216 time steps (x-axis).

At first glance, it's evident that the occlusion and mask techniques have some overlap (rows 2, 3, 6, 7, 9, 10, and 11). However, later in our discussion, it becomes apparent that the additional regions explained by the learned mask render it superior to the other techniques. Over different examples in each class, we also see similar features found salient by the learned mask – a property less apparent for the other techniques. Furthermore, as expected in clinical practice (Moody and Mark, 2001), the ECG leads varied among subjects (see the difference between rows 1 and 2), which makes generalization harder for LSTMs.

Whereas the MNIST digits are easy to understand, understanding the important features of electrocardiograms requires expert knowledge of cardiology. Hence, we consulted a cardiologist to help identify the salient features detected by LSTMs that align with medical theory. The analysis is summarized as follows:

- We begin with the normal heartbeat; the model correctly identifies the QRS complex as salient, with the learned mask indicating that more importance is placed on the R-peak and S-wave (rows 2 and 3). Because the signal in row 1 is from a different ECG lead, it has a low R-peak, which, given the importance of the R-peak for normal heartbeats, could explain the misclassification. Whereas the learned mask correctly identifies the Q-, R-, and S-points as important, the occlusion and derivation techniques seem to be limited to one or two points of interest. The model finds Q-wave to be less helpful, which is similar to the findings in practice because usually, the Q-wave is barely visible for normal heartbeats.

- A wide S-wave is usually seen in right bundle branch block (RBBB) heartbeats, which was correctly identified as salient by the model – highlighted by the mask ($k = 0$) and occlusion techniques for all the RBBB examples (rows 4 to 6). The learned mask additionally shows that the LSTM correctly identifies the extra bump leading up to the R-peak (a characteristic of an RSR pattern) to be important for classifying RBBB heartbeats. In rows 4 to 6, we see that setting $k = mean$ is detrimental to the learned mask, whereas the occlusion technique is largely unaffected. Note that RBBB heartbeats can look similar to LBBB heartbeats, depending on the ECG lead.

- Depending on where the pacing leads are placed within the heart, the R-wave (or sometimes Q-wave) follows the pacing spike (see rows 7 to 9). Thus, identifying the narrow upstroke of the pacing spike could provide a means of detecting paced heartbeats. According to the learned mask, the LSTM learns to identify this pacing spike, whereas the occlusion technique, with $w = 25$, seems to find the R-to-S transition important. These paced heartbeat examples demonstrate the difficulty of choosing the

occlusion width w; for larger values, the importance lies on the R-to-S transition, and the importance moves to the sharp peak as the width, w, is decreased.

- Lastly, in rows 10 to 12, the learned mask elucidates that the model considers the ratio of the R-peak and S-wave as an important feature of premature ventricular contraction (ventricular ectopics). Medically this is relevant for some ECG leads, with the S-wave being deep relative to the R-peak. In practice, the duration of the QRS complex is primarily used to determine premature ventricular contraction, but it's difficult to justify whether this is something the model finds salient. The relatively normal R-peak to S-wave ratio in row 11, compared to row 12, and the short QRS complex, could explain the misclassification.

The analysis demonstrates that the input features considered salient align well with medical theory. Note, however, cardiologists usually classify heartbeat arrhythmias by means of multiple ECG leads and not a single lead, as in this case. When they are uncertain of their classification they often look at other leads for more information. Additionally, the patient's current health condition (such as chest pain) contributes largely to a cardiologist's classification – an input not considered by our LSTM. Such additional inputs could improve the LSTM performance.

A visualization technique that has not been given much attention thus far is the input derivative. On both of the before mentioned datasets, this technique yields the least interpretable salient features. On the other hand, both the occlusion and masking techniques produce interpretable results. The input derivatives elucidate the magnitude of influence that each feature has, whereas the occlusion and masking techniques indicate the consequence of varying the input. In the medical time series domain, the latter two techniques double as a means to determine the effect of unwanted perturbations on classification, such as missing values in physiological signals.

Up to now, we have considered only univariate inputs. To assess these input feature salience techniques on multivariate time series we turn to the ICU dataset (section 3.2.3) in figure 5.7. Again we indicate the different techniques and their settings at the top of the image, and the signal category on the left. Similar to the 5×5 block occlusion, we can occlude all of the signals simultaneously over time, which is indicated by "all channels" in the figure. The default procedure is to occlude each channel separately, that is, considering all signals as a single stitched sequence.

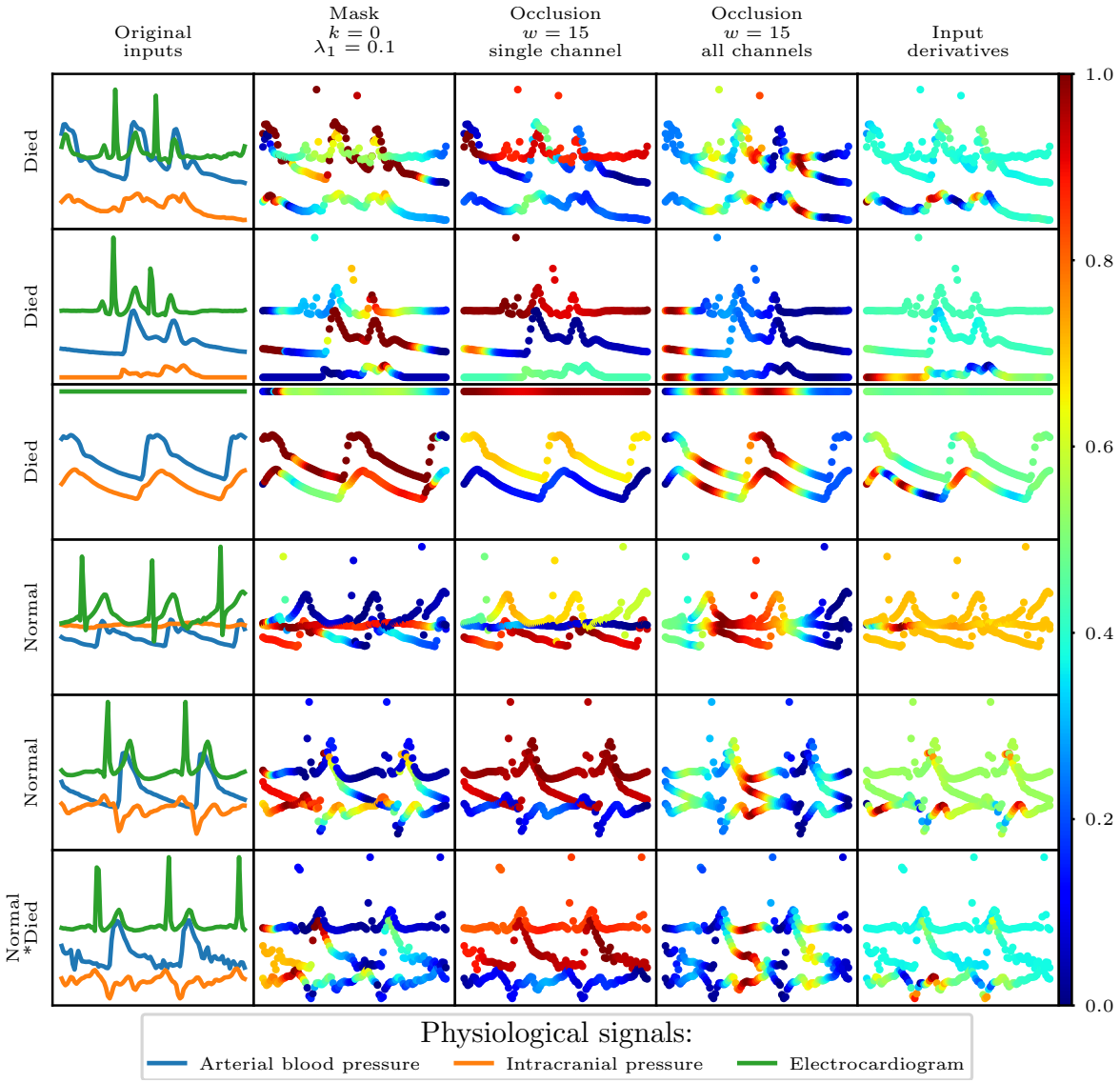


Fig. 5.7 Input feature salience for the multivariate time series of the ICU dataset. The input feature importance for LSTM classification, as given by the occlusion, mask, and input derivative techniques, are displayed on a scale of zero to one, where one is important. The different techniques and their settings are indicated at the top, with k denoting the deletion value, and w the occlusion width. Occlusion on all channels means that all the signals were simultaneously occluded, whereas the occlusion is propagated through each signal separately in the single channel case. On the left-hand side the true label of the input is indicated, where an incorrect classification by the LSTM is indicated by *. The different physiological signals, indicated at the bottom of the figure, are normalized to have zero mean and unit variance, based on dataset statistics. All inputs are displayed with the same y-scale and have a length of 100 time steps (x-axis). There is little overlap of salient features between the single channel occlusion and the learned mask. The learned mask pays attention to the blood pressure and the occlusion focusses on the electrocardiogram. Occluding all channels does not seem to yield interpretable results, and the input derivatives provide clarity but probably not clinical relevance.

Compared to the univariate salience visualizations, here, there is less overlap between the salient features of the learned mask and those of the standard occlusion (single channel). For dying patients, the mask identifies blood pressure as important, whereas the single channel occlusion focusses on the electrocardiogram (ECG). Similarly, for the normal patients, the mask focusses on the start of the blood pressure signal, whereas the occlusion technique finds the entire blood pressure sequence salient and again some of the ECG signal.

The clarity granted earlier by the 5×5 block occlusion is unfortunately not as evident here; the simultaneous occlusion of all the signals does not produce interpretable visualizations. On the other hand, the input derivatives seem to effectively hone in on specific features, making visualization aesthetically pleasing, although probably not clinically relevant.

Clearly, when dealing with multivariate time series, the visualizations become convoluted and important features are more difficult to pin down. Additionally, because the intensive care unit dataset is small, subsequences from all patients were mixed before we split the data. Therefore, salient features for the model could distinguish patients instead of medical outcomes. We explored visualizations on data with even more signals, such as the traumatic brain injury dataset (section 3.2.2), but too many signals render visualizations impractical. Hence, visualizing and understanding LSTMs on multivariate signals remains an open problem.

We observed the visualization techniques on a significant portion of all the datasets and found the salient regions to be consistent for each class – the examples presented in this section represent their entire corresponding dataset. The following section describes a quantitative effort to find a measure of efficacy for the entire test dataset.

5.3.2 Quantitative evaluation

In this section, we compare the efficacy of the input feature salience techniques by calculating how much they reduce the class score s_c on average. Here the total size of the salient regions is not a concern, instead, the important input regions should be as relevant as possible regardless of their size. To compare the different techniques, however, we need to penalize larger deletion areas because simply deleting the whole input could yield the largest score reduction. Therefore, we scale the score reduction for each input sequence by the ratio $\frac{T-M^{(n)}}{T}$, where $M^{(n)}$ is the number of input elements occluded of the sequence $\mathbf{x}^{(n)}_{1:T}$, and T is the total number of time steps. Before occluding a sequence and computing the reduced class score, a threshold denoted by $\alpha \in [0,1]$ has to be specified, above which, the input features are considered important enough to occlude. The consequent average class score reduction is obtained by the following protocol:

1. Start by computing the original class score $s_c^{(n)}$ for $\mathbf{x}_{1:T}^{(n)}$ via the LSTM.
2. Let $\mathbf{r}_{1:T}^{(n)}$, $r_t^{(n)} \in [0,1]$, be the scaled importance of input features for $\mathbf{x}_{1:T}^{(n)}$.
3. For all elements $r_t^{(n)} > \alpha$, set $x_t^{(n)} = k$, where k is the deletion value.
4. Obtain the new class score $\hat{s}_c^{(n)}$ for the occluded input.
5. Let $M^{(n)}$ be the number of elements occluded; $M^{(n)} = \sum_{n=1}^{N} \mathbb{I}(r_t^{(n)} > \alpha)$.
6. Scale the class score reduction and compute the average over the entire test dataset as $\frac{1}{N}\sum_{n=1}^{N} \frac{T-M^{(n)}}{T}(s_c^{(n)} - \hat{s}_c^{(n)})$.

Importantly, this metric does not consider whether the occlusions are correct, for example, the electrocardiograms could be occluded at clinically irrelevant regions. It solely provides a measure of how efficiently each technique finds salient input features for the LSTM. Hence, establishing the utility of the visualization techniques still requires the qualitative evaluation. To have the average class score reductions be related to the utility of the salience technique, we select the hyperparameters based on visual analysis. The chosen hyperparameters are presented in table 5.2; the input derivative technique does not require hyperparameter selection.

Table 5.2 Hyperparameters for the different visualization techniques used to compute the average class score reduction. The values are based on the best results found during visual analysis.

Technique	MNIST	MIT-BIH	ICU
Occlusion	$w = 5,\ k = 0$	$w = 25,\ k = 0$	$w = 15,\ k = 0$
Mask	$k = 0,\ \lambda_1 = 0.01$	$k = 0,\ \lambda_1 = 1.0$	$k = 0,\ \lambda_1 = 0.1$

In figure 5.8 we illustrate the average score reductions over a range of values for α. Evidently, the learned mask most efficiently deletes information from the input. In contrast to the other techniques, when $\alpha = 0.9$, the learned mask still greatly reduces the class score, meaning that the most salient features are truly important. On the ICU dataset, the occlusion technique performs poorly, whereas the input derivatives seem to identify salient features efficiently – similar to what was found in the qualitative evaluation.

5.4 Discussion

This chapter deals with the important problem of providing insights into how black box models, specifically LSTMs, make their decisions. The premise is simple – for many practical applications, one needs justification of the decision, instead of a bare prediction. LSTMs, and other neural networks do not provide this kind of information. When these models are applied

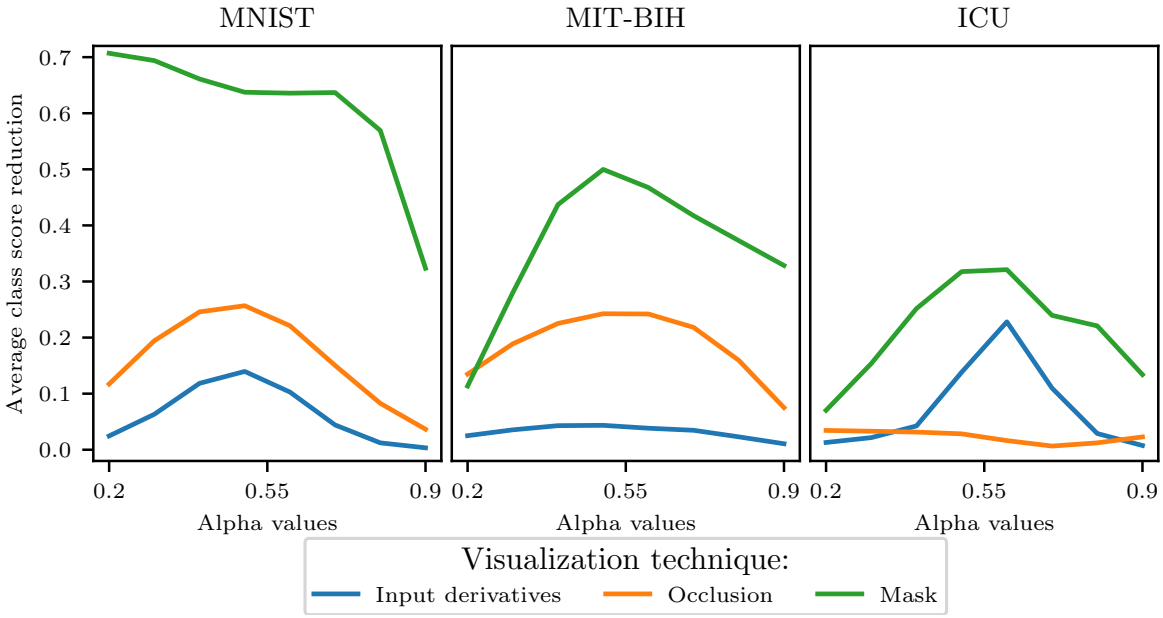


Fig. 5.8 Average class score reductions for the three input feature salience techniques. From left to right the reductions are shown for LSTMs trained on the MNIST, MIT-BIH, and ICU datasets. Each colour resembles a different visualization technique, as indicated in the legend. The y-scale is the same for each graph. Learning a mask is clearly the superior technique.

to medical data, understanding why they make certain decisions would allow clinicians to build better trust in them and could provide novel insights on medical phenomena.

We compared the efficacy of five visualization techniques for the LSTM. In this comparison, it is argued that explaining what an LSTM focusses on depends in large part on the meaning of varying the input to the model. It was found that learning a deletion mask yields the most interpretable results and the largest reduction of the class score. Performance of this technique seemed fairly unaffected by different preferences for zero-valued mask elements and mask smoothness, but this could be data-dependent. Furthermore, we found that the electrocardiogram input features considered salient by LSTM align well with medical theory. We would like to point out, however, that the visualization techniques explored also have limitations; for multivariate time series, such as intensive care unit vital signs, the visualizations can become obtuse when the number of signals is large.

This work goes some way to improving our understanding of LSTMs. In defence of deep learning model interpretability, it may be lagging behind most Bayesian models, but it is comparable to or better than that of many other widely-used machine learning methods, such as random forests and support vector machines (SVMs) (Ching et al., 2018). Training SVMs with nonlinear kernels sacrifices interpretability and the same input importance scores that can be obtained for random forests can be obtained for deep learning models. Similarly,

a decision tree with myriad nodes and branches may also be difficult to comprehend, and training simple machine learning techniques on heavily engineered features could render the approach opaque if the features are difficult to interpret.

Up to this point in the thesis, we have looked at applications where LSTMs are beneficial and have demonstrated how it is possible to gain a better understanding of these models. Next, we take a look at how we can extend the capabilities of the LSTM.

Chapter 6

LSTMs as a generative model for validity in complex discrete structures

Often real-life data are sequences of some form, such as text, mathematical expressions, and atomic structures of molecules. In these scenarios, each element in the sequence is one of any in a set, sometimes referred to as the alphabet. The space of possible sequences grows exponentially with the length of the sequences; the fraction of valid sequences in this space typically decreases with the length of the sequences. Learning the rules that govern production (the grammar) of valid sequences is stifled by this small fraction of valid sequences.

As a remedy, this chapter introduces our first extension of the LSTM. This extension leverages the sequential nature of the LSTM to allow efficient active learning of sequence grammars. Efficiency is obtained by reducing the search from that over all possible sequences – a large space – to a search over the set of possible elements at each time step. Before detailing this approach, we describe an application where the validity of sequences is important.

6.1 Generating valid discrete structures

Deep generative models have enabled the search of high-dimensional discrete spaces (Gómez-Bombarelli et al., 2016b; Kusner et al., 2017). This discrete search is at the heart of problems in drug discovery (Gómez-Bombarelli et al., 2016a), symbolic regression (Kusner et al., 2017), and natural language processing (Bowman et al., 2015; Guimaraes et al., 2017).

Optimizing or searching in a continuous space is easier than in a discrete space. Therefore, an autoencoder is employed to "lift" the search from the discrete space into the continuous space. Here the autoencoder (section 3.5.1) consists of an encoder for mapping a discrete

space into a continuous space and a decoder for mapping from a continuous space into a discrete space. The discrete space is presented to the autoencoder as a sequence in some formal language, for example, in Gómez-Bombarelli et al. (2016b) molecules are encoded as SMILES strings (see section 6.4). Your preferred deep learning model can then be used as the encoder and decoder; however, when employing these models, the decoder frequently generates invalid sequences. Kusner et al. (2017) aimed to fix this by basing the sequential models on parse tree representations of discrete structures, where externally specified grammatical rules assist the model in the decoding process. This work bolstered the ability of the model to produce valid sequences during decoding, but the method requires hand-crafted grammatical rules for each application domain and leaves room for improving the percentage of valid sequences generated.

In the spirit of this thesis, this work is concerned with an end-to-end learning approach, whereby a **validity model** learns the validity constraints of a given discrete space. Effectively, the proposed model learns the grammar of character-based sequences, which allows it to generate valid sequences by means of this grammar. As made apparent in the following sections, training such a model is challenging. Therefore, to assist in training this model we propose two data augmentation techniques.

Where no labelled dataset of valid and invalid sequences is available, we propose a novel approach to active learning for sequential tasks inspired by classic mutual-information-based approaches (Hernández-Lobato et al., 2014; Houlsby et al., 2011). Where datasets containing valid examples do exist, we propose an effective data augmentation process based on applying minimal perturbations to known valid sequences. These two techniques allow us to rapidly learn sequence validity models that can be used as; i) generative models, which we demonstrate in the context of Python 3 mathematical expressions; and ii) a validity-encouraging model for character-based sequences, that can drastically improve the ability of decoding continuous representations into valid discrete sequences for deep generative models. We demonstrate the latter in the context of molecules, but first, we define our model for sequence validity.

6.2 A model for sequence validity

To formalize the problem we denote the set of discrete sequences of length T by $\mathcal{X} = \{(x_1, ..., x_T) : x_t \in \mathcal{C}\}$ using the alphabet $\mathcal{C} = \{1, ..., C\}$ of size C. Individual sequences in $\mathcal{X}$ are denoted $\mathbf{x}_{1:T}$. We assume the availability of a **validator** $v : \mathcal{X} \to \{0, 1\}$, an oracle which can tell us whether a given sequence is valid (1) or not (0). It is important to note that such a validator provides very sparse feedback: it can only be evaluated on a *complete* sequence.

Examples of such validators are compilers for programming languages, which can identify syntax and type errors, and chemo-informatics software for parsing SMILES strings (section 6.4), which identify violations of valence constraints. Running the standard validity checker $v(\mathbf{x}_{1:T})$ on a partial sequence or subsequence (e.g., the first $t < T$ characters of a computer program) does not in general provide any indication as to whether the complete sequence of length T is valid.

Here we are concerned with learning a generative model for the set of sequences $\mathcal{X}_+ = \{\mathbf{x}_{1:T} \in \mathcal{X} : v(\mathbf{x}_{1:T}) = 1\}$, the subset of valid sequences in $\mathcal{X}$. To achieve this, we require the sequential generative process to be guided towards valid sequences. This guidance takes the form of a more informative function $\tilde{v}(\mathbf{x}_{1:t})$ which operates on prefixes [1] $\mathbf{x}_{1:t}$ of a hypothetical longer sequence $\mathbf{x}_{1:T}$ and outputs

$$\tilde{v}(\mathbf{x}_{1:t}) = \begin{cases} 1 & \text{if there exists a suffix } \mathbf{x}_{t+1:T} \text{ such that } v(c[\mathbf{x}_{1:t}, \mathbf{x}_{t+1:T}]) = 1, \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where $c[\mathbf{x}_{1:t}, \mathbf{x}_{t+1:T}]$ concatenates a prefix and a suffix to form a complete sequence. Thus the function $\tilde{v}(\mathbf{x}_{1:t})$ determines whether a given prefix can form part of a valid complete sequence. Note that we are indifferent to *how many* suffixes yield valid sequences. Essentially, a generative model of $\mathcal{X}_+$ which constructs sequences from left to right, a single character at a time, can use $\tilde{v}(\mathbf{x}_{1:t})$ to provide early feedback as to which of the next character choices lead to a "dead end" from which no valid sequence can be produced.

Training this prefix validity model $\tilde{v}(\mathbf{x}_{1:t})$ is best viewed from a reinforcement learning perspective. Hence, the problem can be framed as a Markov decision process (Sutton and Barto, 1998) for which we train a reinforcement learning agent to select characters sequentially in a manner that avoids producing invalid sequences. At time t, ($t \in \{1, ..., T\}$) an agent is in state $\mathbf{x}_{<t} = \mathbf{x}_{1:t-1}$ and can take actions $x_t \in \mathcal{C}$, i.e., the current action depends only on the current prefix. At the end of an episode (end of sequence), following action x_T, the agent receives a reward of $v(\mathbf{x}_{1:T})$. Since in practice we are only able to evaluate $v(\mathbf{x}_{1:T})$ in a meaningful way on complete sequences, the agent does not receive any reward at any of the intermediate steps $t < T$. The optimal Q-function, $Q^\star(s, a)$, a function of a state s and an action a, represents the expected reward of an agent following an optimal policy which takes action a at state s (Watkins, 1989). This optimal Q-function would assign value 1 to actions $a = x_t$ in states $s = \mathbf{x}_{<t}$ for which there exists a suffix $\mathbf{x}_{t+1:T}$ such that $c[\mathbf{x}_{1:t}, \mathbf{x}_{t+1:T}] \in \mathcal{X}_+$, and value 0 to all other state-action pairs. This behaviour exactly matches the desired prefix

[1] We use the terms prefix and suffix to refer to subsequences at the start and end of the original sequence.

validator in eq. 6.1, that is, $Q^\star(\mathbf{x}_{<t}, x_t) = \tilde{v}(\mathbf{x}_{1:t})$. Consequently, learning $\tilde{v}(\mathbf{x}_{1:t})$ corresponds to learning the Q-function.

Having access to the model of $\mathcal{X}_+$ by $Q^\star$ allows us to obtain a generative model for $\mathcal{X}_+$. In particular, an agent following any optimal policy $\pi^\star(\mathbf{x}_{<t}) = \text{argmax}_{x_t \in \mathcal{C}} Q^\star(\mathbf{x}_{<t}, x_t)$ will always generate valid sequences. If we sample uniformly at random across all optimal actions at each time $t = 1, ..., T$, we obtain the joint distribution given by

$$p(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} \frac{Q^\star(\mathbf{x}_{<t}, x_t)}{Z(\mathbf{x}_{<t})}, \tag{6.2}$$

where $Z(\mathbf{x}_{<t}) = \sum_{x_t} Q^\star(\mathbf{x}_{<t}, x_t)$ are the per-time-step normalization constants. This distribution allows us to sample sequences $\mathbf{x}_{1:T}$ in a straightforward manner by sequentially selecting characters $x_t \in \mathcal{C}$ given the previously selected characters in $\mathbf{x}_{<t}$.

Rather conveniently, we can approximate this optimal Q-function with an LSTM that has a sigmoid activated output unit for each character in $\mathcal{C}$, such that each output is in the closed interval $[0, 1]$. A schematic of the LSTM output is shown in figure 6.1. We denote the output of the LSTM unit corresponding to character x_t by $\mathrm{y}(x_t|\mathbf{x}_{<t}, \omega)$, with network weights ω and input sequence $\mathbf{x}_{<t}$. This output $\mathrm{y}(x_t|\mathbf{x}_{<t}, \omega)$ is interpreted as the probability that action x_t can yield a valid sequence given the current state $\mathbf{x}_{<t}$, that is, as $p(Q^\star(\mathbf{x}_{<t}, x_t) = 1)$.

Within our framework, a sequence $\mathbf{x}_{1:T}$ will be valid according to the model if every action during the sequence generation process is permissible, that is, if $Q^\star(\mathbf{x}_{<t}, x_t) = 1, \ \forall t$. Similarly, we consider that the sequence $\mathbf{x}_{1:T}$ will be invalid if at least one action during the sequence generation processes is not valid[2], that is, if $Q^\star(\mathbf{x}_{<t}, x_t) = 0$ at least once for $t = 1, ..., T$. This leads to the following log-likelihood function given the training dataset $\mathcal{D} = \{(\mathbf{x}^{(n)}_{1:T}, y^{(n)})\}_{n=1}^N$ of sequences $\mathbf{x}^{(n)}_{1:T} \in \mathcal{X}$ and corresponding labels[3] $y^{(n)} = v(\mathbf{x}_{1:T})$:

$$\mathcal{L}(\omega|\mathcal{D}) = \sum_{n=1}^{N} \Big(y^{(n)} \log p(y^{(n)} = 1|\mathbf{x}^{(n)}_{1:T}, \omega) + (1 - y^{(n)}) \log p(y^{(n)} = 0|\mathbf{x}^{(n)}_{1:T}, \omega) \Big), \tag{6.3}$$

[2]Note that, once $Q^\star(\mathbf{x}_{<t}, x_t)$ is zero, all the following values of $Q^\star(\mathbf{x}_{<t}, x_t)$ in that sequence will be irrelevant to us. Therefore, we can safely assume that a sequence is invalid if $Q^\star(\mathbf{x}_{<t}, x_t)$ is zero at least once in the sequence.

[3]Note the difference in notation for the model output $\mathrm{y}(x_t|\mathbf{x}_{<t}, \omega)$ and that of the sequence labels y.

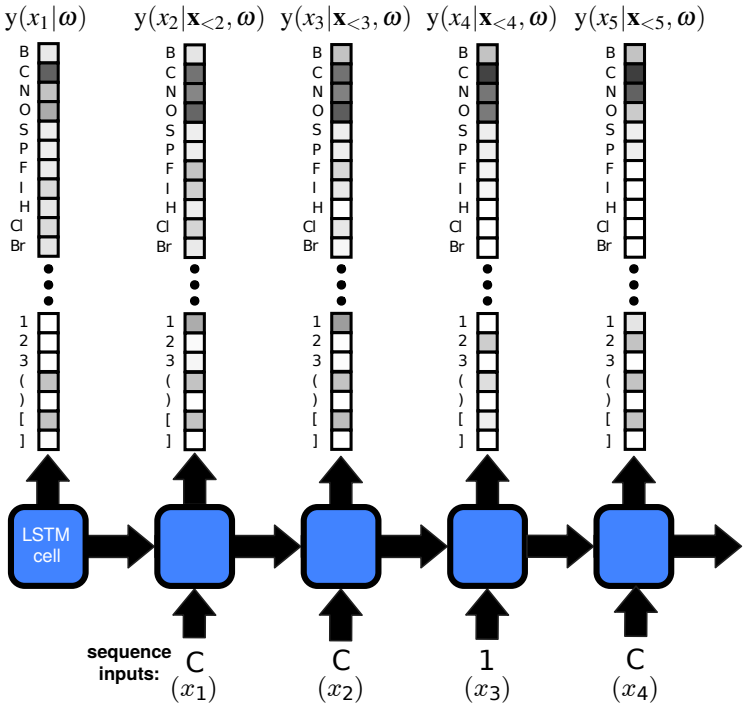


Fig. 6.1 The LSTM model used to approximate the Q-function. For each element x_t in the sequence, a vector of sigmoid activations is produced, which indicates the likelihood of the next element leading to a valid sequence. The first prediction is generated from the initial hidden state of the LSTM. The output columns shown above each LSTM unit depict hypothetical sigmoid activations predicting the likelihood of a valid sequence for each possible next character given the previous sequence elements and the weights; $\mathrm{y}(x_t|\mathbf{x}_{<t},\omega)$. Darker blocks indicate characters that have a higher likelihood of leading to a valid complete sequence. Here the set of characters is the SMILES alphabet (table 6.4); a specific form of molecular structures that allow them to be denoted as a sequence of characters. Letters represent different atoms, numbers represent ringbonds, and brackets represent branches in the molecule. We show an example input by using the first four characters of an arbitrary molecule (see figure 6.3 for the full molecule).

where, following from the above characterization of valid and invalid sequences, we define

$$p(y^{(n)}=1|\mathbf{x}^{(n)}_{1:T},\omega)=\prod_{t=1}^{T}\mathrm{y}(x^{(n)}_t|\mathbf{x}^{(n)}_{<t},\omega) \tag{6.4}$$

$$p(y^{(n)}=0|\mathbf{x}^{(n)}_{1:T},\omega)=1-\prod_{t=1}^{T}\mathrm{y}(x^{(n)}_t|\mathbf{x}^{(n)}_{<t},\omega). \tag{6.5}$$

The log-likelihood (eq. 6.3) can be optimized using stochastic gradient descent and back-propagation, resulting in a maximizer $\widehat{\omega}$ such that $\mathrm{y}(x_t|\mathbf{x}_{<t},\widehat{\omega})\approx Q^{\star}(\mathbf{x}_{<t},x_t)$.

Notice the difference between the formulation above and that of a maximum likelihood solution to T independent binary classifications, which would require eq. 6.5 to be

$$p(y^{(n)} = 0|\mathbf{x}_{1:T}^{(n)}, \boldsymbol{\omega}) = \prod_{t=1}^{T} \big(1 - \mathrm{y}(x_t^{(n)}|\mathbf{x}_{<t}^{(n)}, \boldsymbol{\omega})\big). \tag{6.6}$$

Here, the prediction that a single element x_t of a sequence is invalid, $\mathrm{y}(x_t|\mathbf{x}_{<t}, \boldsymbol{\omega}) = 0$, would not strictly result in a prediction of the complete sentence being invalid, $p(y^{(n)} = 0|\mathbf{x}_{1:T}^{(n)}, \boldsymbol{\omega}) \leq 1$, which is the case in eq. 6.5 and empirically found to be important. Essentially, compared to eq. 6.5, eq. 6.6 encourages $\mathrm{y}(x_t|\mathbf{x}_{<t}, \boldsymbol{\omega})$ to be low for all t, which in turn leads to the value of eq. 6.4 vanishing with an increase in sequence length. Thus, eq. 6.5 allows the model to assign 0 to only those elements that lead to invalid sequences.

To summarize, this section showed how concepts from reinforcement learning may be used to define a suitable generative model and how this model can be approximated using sequence-based deep learning techniques. As mentioned before, datasets of discrete sequences are often highly unbalanced, containing larger fractions of valid sequences. The following section describes two approaches that mitigate this issue and thus permits the learning of a grammar.

6.3 Ensuring a balanced dataset

One critical aspect of learning $\boldsymbol{\omega}$ as described in section 6.2 is the generation of a training dataset $\mathcal{D}$. A naive approach would be to draw elements from $\mathcal{X}$ uniformly at random. However, in many cases, $\mathcal{D}$ contains only a tiny fraction of valid sequences and the uniform sampling approach produces extremely unbalanced datasets which contain very little information about the structure of valid sequences. While rejection sampling can be used to increase the number of valid samples, the resulting additional cost makes such an alternative infeasible in most cases. This problem is exacerbated by longer sequence lengths T. With $|\cdot|$ denoting the cardinality of a set, $|\mathcal{X}|$ will always grow as $|\mathcal{C}|^T$, while $|\mathcal{X}_+|$ will typically grow at a lower rate.

In light of this, we employ two approaches for artificially constructing balanced datasets that permit learning these models with far fewer samples than $|\mathcal{C}|^T$. The first approach applies to settings where we do not have a collection of known valid sequences and makes use of active learning; the second applies to settings where a dataset of valid sequences is available and employs a sequence perturbation technique.

6.3.1 Active learning

Bayesian active learning enables automatic construction of a training dataset $\mathcal{D}$. This method works by iteratively selecting sequences in $\mathcal{X}$ that are most informative about the parameters ω given the data collected thus far (MacKay, 1992a).

As we had before, we denote an arbitrary sequence by $\mathbf{x}_{1:T}$ and the corresponding unknown binary label by y, which indicates whether the sequence is valid or not. Our model's predictive distribution $p(y|\mathbf{x}_{1:T},\omega)$ is given by eq. 6.4 and 6.5. The aim here is to add sequence-label pairs to our dataset that provide us with the biggest gain in information. The amount of information that we expect to gain on the weights ω by labelling and adding $\mathbf{x}_{1:T}$ to $\mathcal{D}$ can be measured in terms of the expected reduction in entropy of the posterior distribution $p(\omega|\mathcal{D})$

$$\alpha(\mathbf{x}_{1:T}) = \mathbb{H}[p(\omega|\mathcal{D})] - \mathbb{E}_{p(y|\mathbf{x}_{1:T},\omega)}\big[\mathbb{H}[p(\omega|\mathcal{D}\cup(\mathbf{x}_{1:T},y))]\big], \tag{6.7}$$

where $\mathbb{H}[\cdot]$ computes the entropy of a distribution. It turns out that this formulation of the entropy-based active learning criterion is problematic because computing $p(\omega|\mathcal{D}\cup(\mathbf{x}_{1:T},y))$ requires training on the new data point $\mathbf{x}_{1:T}$. Fortunately, Houlsby et al. (2011) showed that $\alpha(\mathbf{x}_{1:T})$ is equal to the mutual information between y and ω given $\mathbf{x}_{1:T}$ and $\mathcal{D}$

$$\alpha(\mathbf{x}_{1:T}) = \mathbb{H}\big[\mathbb{E}_{p(\omega|\mathcal{D})}[p(y|\mathbf{x}_{1:T},\omega)]\big] - \mathbb{E}_{p(\omega|\mathcal{D})}\big[\mathbb{H}[p(y|\mathbf{x}_{1:T},\omega)]\big], \tag{6.8}$$

which is easier to work with as the required entropy is now that of Bernoulli predictive distributions, an analytic quantity. Let $\mathcal{B}(p)$ denote a Bernoulli distribution with probability p, with probability mass $p^z(1-p)^{1-z}$ for values $z \in \{0,1\}$. The entropy of $\mathcal{B}(p)$ can easily be obtained as

$$\mathbb{H}[\mathcal{B}(p)] = -p\log p - (1-p)\log(1-p) =: g(p) \tag{6.9}$$

The expectation with respect to $p(\omega|\mathcal{D})$ can easily be approximated by Monte Carlo estimation. We could attempt to sequentially construct $\mathcal{D}$ by optimizing eq. 6.8. However, this optimization process would still be difficult, as it would require evaluating $\alpha(\mathbf{x}_{1:T})$ exhaustively on all the elements of $\mathcal{X}$. The sequential nature of LSTMs provide us with a greedy approach that circumvents this issue; instead of performing active learning over the entire space of sequences $\mathcal{X}$, active learning can be performed over the much smaller space of the alphabet $\mathcal{C}$, at each time step. In particular, at each time step $t = 1,...,T$, we select x_t by maximizing the mutual information between ω and $Q^\star(\mathbf{x}_{<t},x_t)$, where $\mathbf{x}_{<t}$ denotes the prefix already selected at previous steps of the optimization process. This mutual information

quantity is denoted by $\alpha(x_t|\mathbf{x}_{<t})$ and its expression is given by

$$\alpha(x_t|\mathbf{x}_{<t}) = \mathbb{H}\big[\mathbb{E}_{p(\omega|\mathcal{D})}[\mathcal{B}(\mathrm{y}(x_t|\mathbf{x}_{<t},\omega))]\big] - \mathbb{E}_{p(\omega|\mathcal{D})}\big[\mathbb{H}[\mathcal{B}(\mathrm{y}(x_t|\mathbf{x}_{<t},\omega))]\big], \tag{6.10}$$

An informative sequence can then be generated efficiently by sequentially maximizing eq. 6.10, an operation that requires only $|\mathcal{C}| \times T$ evaluations of $\alpha(x_t|\mathbf{x}_{<t})$.

In order to approximate eq. 6.10, we first approximate the posterior distribution $p(\omega|\mathcal{D})$. Recall from section 3.4.2 that we can use dropout to stochastically zero-out units in the layers of the LSTM to obtain estimates of uncertainty in the predictions. Under the assumption of a Gaussian prior $p(\omega)$ over the weights, the stochastic process yields an implicit approximation $q(\omega)$ to the posterior distribution $p(\omega|\mathcal{D}) \propto \exp(\mathcal{L}(\omega|\mathcal{D}))p(\omega)$. We then draw K samples from $q(\omega)$ by means of Monte Carlo dropout to estimate the expectations in eq. 6.10. With $\omega_1, ..., \omega_K \sim q(\omega)$ and g defined in eq. 6.9, the resulting estimator is given by

$$\widehat{\alpha}(x_t|\mathbf{x}_{<t}) = g\Big(\frac{1}{K}\sum_{k=1}^{K}\mathrm{y}(x_t|\mathbf{x}_{<t},w_k)\Big) - \frac{1}{K}\sum_{k=1}^{K} g\Big(\mathrm{y}(x_t|\mathbf{x}_{<t},w_k)\Big). \tag{6.11}$$

We find that reasonable estimates can be obtained even for small K and use $K = 16$ in our experiments.

The iterative procedure just described is designed to produce a single informative sequence. In practice, we would like to generate a batch of informative and diverse sequences because when training neural networks, the processing of a minibatch of data is computationally more efficient than processing multiple data points individually. To construct a minibatch with M informative sequences, we propose to repeat the previous iterative procedure M times. Importantly, to introduce diversity in the batch-generation process, we "soften" the greedy maximization at each step by injecting a small amount of noise in the evaluation of the objective function (Finkel et al., 2006). In addition to introducing diversity, this can also lead to better overall solutions than those produced by the noiseless greedy approach (Cho, 2016). Noise is introduced into the greedy selection process by sampling from

$$p(x_t|\mathbf{x}_{<t},\theta) = \frac{\exp\big(\alpha(x_t|\mathbf{x}_{<t})/\theta\big)}{\sum_{x'_t\in\mathcal{C}}\exp\big(\alpha(x'_t|\mathbf{x}_{<t})/\theta\big)}, \tag{6.12}$$

for each $t = 1, ..., T$, which is a Boltzmann distribution with a sampling temperature θ. Using a higher value for θ increases the diversity of the sequences in the minibatch. Setting $\theta = 1e-7$ provided the ideal diversity-validity trade-off during training.

Comment: It's quite spectacular to note that the active learning technique described here enables learning "without" data. Essentially, to train a model with this approach only the rules of the grammar are required; for example, the compiler of a programming language. Via these grammar rules, this approach allows efficient exploration of the data space in order to generate an informative dataset of sequences to train on. The result, as in our case, could be a validity model that is potentially more informative than the original oracle (compiler) because it can also operate on incomplete sequences. This "data-less" training is reminiscent of the approach used for AlphaGo Zero (Silver et al., 2017). The AlphaGo Zero model is not trained on human-play examples, but rather it generates its own set of plays by using the rules of the game, and then continues to train against itself, all the while selecting the most informative strategies.

In this section, we introduced an active learning technique for efficiently generating a training dataset with an approximately optimal ratio of valid and invalid sequences. When a set of known valid sequences is available, there is an easier approach to achieve this same goal. In the following section, we describe how to use these existing valid sequences to seed a process for generating balanced datasets.

6.3.2 Sequence perturbation

In some settings, such as the molecule domain we will consider later, we have datasets of known valid examples (e.g., collections of known drug-like molecules), but rarely are datasets of invalid examples available. Obtaining invalid sequences may seem trivial, as invalid samples may be garnered by sampling uniformly from $\mathcal{X}$. However, these are almost always so far from any valid sequence that they carry little information about the boundary between valid and invalid sequences. Moreover, training on a single known dataset carries the danger of overfitting to the subset of $\mathcal{X}_+$ covered by the data.

We address this by perturbing sequences from a dataset of valid sequences, such that approximately half of the generated sequences are invalid. These perturbed sequences $x'_{1:T}$ are constructed by setting each x'_t to be a symbol selected independently from $\mathcal{C}$ with probability γ, while keeping the original x_t with probability $1-\gamma$. In expectation, this changes γT entries in the sequence. We choose $\gamma = 0.05$, which results in a generated dataset that is approximately 50% valid.

One may wonder whether the active learning approach already provides a technique superior to the one described here. Theoretically, the active learning approach would explore a space much wider than the sequence perturbation technique. However, when the validator $v : \mathcal{X} \to \{0,1\}$ is not entirely correct the active learning approach ends up

exploring uncommon boundary conditions which are approved by the validator, but invalid in reality. We found this would happen when using *rdkit* (`http://www.rdkit.org/`), a chemical informatics software package, as the validator for molecules.

In this section two approaches that allow the generation of balanced datasets for training a validity model have been described. We proceed by showing the empirical efficacy of the proposed techniques.

6.4 Experiments

In order to evaluate the proposed technique for training a validity model, we make use of two different sequential datasets. First, we look at fixed length Python 3 mathematical expressions, where we derive lower bounds for the support of our model and compare the performance of active learning with that achieved by a simple passive approach. Second, we look at molecular structures encoded into string representations, where we utilize existing molecule datasets with our proposed sequence perturbation method to learn the rules governing molecular validity. To come full circle, we evaluate the efficacy of our validity model on the downstream task of decoding valid molecules from a continuous latent representation given by a variational autoencoder.

6.4.1 Mathematical expressions

Owing to the Python 3 compiler only approving code that is valid in reality, it serves as a great environment to test our proposed active learning approach. Here, $\mathcal{X}$ consists of all length 25 sequences that can be constructed from the alphabet of numbers and symbols show in table 6.1. The validity of any given expression is determined by means of the Python 3 *eval* function: an invalid expression is one that raises an exception when evaluated.

Table 6.1 Python 3 expression alphabet

Digits	Operators	Comparisons	Brackets
`1234567890`	`+-*/%!`	`=<>`	`()`

Measuring model performance

Within this problem domain, we do not assume the existence of a dataset of positive examples. Therefore we evaluate the models based on their generated sequences. We cannot simply measure the validity of the generated sequences because it's trivial for a model to repeatedly

generate the same valid sequence. Instead, we design a new measure of performance, whereby models are compared in terms of their capability to provide a diverse set of valid sequences, i.e., a high entropy distribution over valid sequences. To achieve this we sample sequences from the validity model and measure the validity and entropy of the samples. To sample stochastically, we use a Boltzmann policy, in other words, a policy that samples the next action (or element) according to

$$\pi(x_t = c|\mathbf{x}_{<t}, \omega, \tau) = \frac{\exp\big(\mathrm{y}(c|\mathbf{x}_{<t}, \omega)/\tau\big)}{\sum_{j\in\mathcal{C}} \exp\big(\mathrm{y}(j|\mathbf{x}_{<t}, \omega)/\tau\big)}, \tag{6.13}$$

where τ is a temperature constant that governs the trade-off between exploration and exploitation. For example, higher values of τ would result in more diverse sequences sampled from the model, which leads to a higher entropy, but possibly a lower validity. Note that this is not the same as the Boltzmann distribution in eq. 6.12, which is used as a proposal generation scheme during active learning and is defined on the estimated mutual information instead of the Q-function values.

For a set of M sequences of length T with a sampling temperature τ_i the entropy is given by

$$\mathbb{H}_{\tau_i} = -\frac{1}{M}\sum_{m=1}^{M}\sum_{t=1}^{T}\sum_{c}\pi(x_t^{(m)} = c|\mathbf{x}_{<t}^{(m)}, \omega, \tau_i)\log\pi(x_t^{(m)} = c|\mathbf{x}_{<t}^{(m)}, \omega, \tau_i). \tag{6.14}$$

Once we have obtained the entropy and the set of sample sequences $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(M)}\}_{\tau_i}$ for a range of temperatures τ_i, we compute the fraction of valid sequences for each set of samples. By plotting the trade-off between validity and entropy of a model, we can compute the area under this validity-entropy curve (V-H AUC) to provide a metric for model quality. Similar to the AUC introduced in section 3.1, this metric provides a single measure that combines both the entropy and validity of a model.

Experimental setup and results

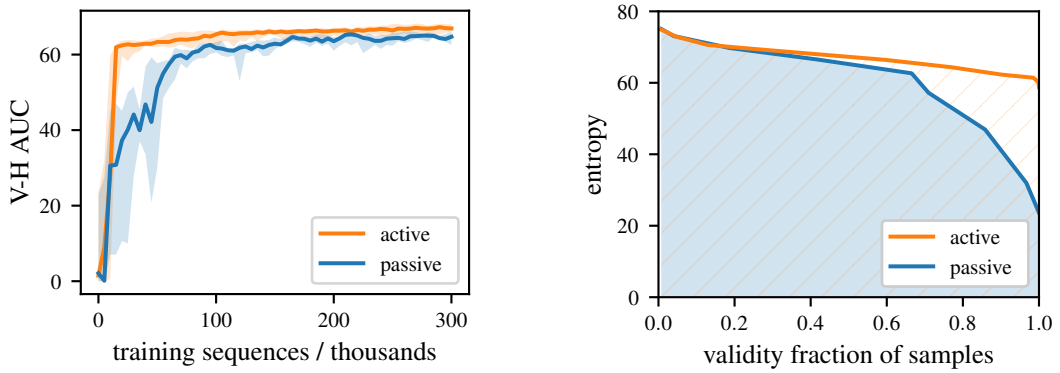
We train two versions of the validity model proposed in section 6.2. The first uses a **passive** method, by training on sequences sampled from a uniform distribution over $\mathcal{X}$. The second uses an **active** method, where the active learning procedure described in section 6.3 is employed to select the training sequences. In all other regards, the two models are identical; the hyperparameters are provided in table 6.2.

In figure 6.2 we show the validity-entropy trade-off at the different iterations of training. Positively, both models yield a diverse output distribution over valid sequences. However,

Table 6.2 Validity model hyperparameters

Hyperparameter	Value
Single hidden layer of size	512
Dropout probability	0.2
Batch size	20
Learning rate	1e-4
Weight decay	1e-8
Optimizer	*Adam*

from figure 6.2a it is clear that the active method is able to learn a model for sequence validity much quicker than the passive method. Furthermore, the validity-entropy plot shown in figure 6.2b elucidates that the converged model using the active method is capable of generating a more diverse range of valid sequences than the model trained using the passive method.



(a) V-H AUC as training progresses

(b) Entropy-validity trade-off

Fig. 6.2 Experiments with length 25 Python 3 expressions. (a) Area under validity-entropy curve (V-H AUC) as training progresses, with the 10^{th} and 90^{th} percentiles shaded. Active learning converges faster and reaches a higher maximum. (b) Entropy versus validity for median active and median passive model after 200k training sequences. Both models have learned a high entropy distribution over valid sequences, with active learning reaching a higher entropy, which is better.

One may wonder what exactly the entropy values relate to. To provide some context for the entropy values, we estimate an information theoretic lower bound for the number of distinct sequences that our model is able to generate. For different temperatures τ we can obtain the fraction of valid samples, and an approximation of the size of the support of $\prod_t \mathrm{y}(x_t|\mathbf{x}_{<t}, \omega)$ over $\mathcal{X}$. As before, let $\mathcal{X}_+$ denote the valid subset of $\mathcal{X}$ and $|\cdot|$ denote the cardinality of a set. Suppose we estimate the fraction of valid sequences $f_+ = |\mathcal{X}_+|/|\mathcal{X}|$ by Monte Carlo sampling uniformly from $\mathcal{X}$. We can then estimate $N_+ = |\mathcal{X}_+|$ by $f_+|\mathcal{X}|$,

where $|\mathcal{X}| = |\mathcal{C}|^T$, a known quantity. A uniform distribution over N_+ sequences would have an entropy of $\log N_+$. Therefore, if our model was perfectly uniform over the sequences it can generate, the number of distinct sequences it can generate can be calculated by

$$N_{model} = \exp(\mathbb{H}_{\tau_i}), \tag{6.15}$$

where $\mathbb{H}_{\tau_i}$ is given in eq. 6.14. Owing to the model, at its optimum, not being uniform over the sequences it can generate, this is a lower bound coverage; i.e., to yield the same entropy, a model that is not uniform over the data points has to generate at least as many distinct data points as a model that is uniform over the data points[4]. In table 6.3 we present the lower bounds on the coverage of the two respective models.

Table 6.3 Estimated lower bound of the coverage N for the passive and active models, defined as the size of the set of Python 3 expressions on which the respective model places positive probability mass. Evaluation is on models trained until convergence ($800,000$ training points, which is beyond the scope of figure 6.2)

Temperature τ	Passive model		Active model	
	validity	N	validity	N
0.100	0.850	9.7×10^{27}	0.841	8.2×10^{28}
0.025	0.969	2.9×10^{25}	0.995	4.3×10^{27}
0.005	1.000	1.1×10^{22}	1.000	1.3×10^{27}

The coverage estimates show that the number of diverse sequences generated by the active model is a few orders of magnitude larger than that of the passive model. Note that the overhead of the active learning data generating procedure is minimal: processing 10,000 sequences takes 31s with the passive method versus 37s with the active method. Hence, the results in this section demonstrate the benefits provided by the proposed active learning approach. Next, we evaluate the sequence perturbation technique on molecules.

6.4.2 SMILES molecules

SMILES (simplified molecular input line entry system) strings (Weininger, 1988) are one of the most common representations for molecules, which are an ordering of atoms and bonds. It is attractive for many applications because it maps the graphical representation of a molecule to a sequential representation, capturing not just the chemical composition, but

[4]With the cardinality of the set required in the uniform and non-uniform cases denoted by N_1 and N_2 respectively, we have $\mathbb{H} = \log N_1 = -\sum_{n=1}^{N_2} p(x^{(n)}) \log p(x^{(n)}) \leq \log N_2 \implies N_1 \leq N_2$.

also structure. This structural information is captured by intricate dependencies in SMILES strings based on the chemical properties of individual atoms and valid atom connectivities.

For the purposes of this study, it is not necessary to give a full description of the SMILES language, however, to accustom the reader to this language, we describe the essentials via an example molecule illustrated in figure 6.3. The SMILES string corresponding to this molecule is given by

```
CC1CN(C(=O)C2=CC(Br)=CN2C)CCC1[NH3+]
```

First, neighbouring characters (Atoms) are singly-bonded to each other. Second, the `H` atoms are omitted except inside square brackets. Thus the start of the molecule, `CC`, means that three `H` atoms and the second `C` atom are bonded to the first `C` in order to make 4 bonds. Third, double bonds are denoted by `=` and triple bonds by `#`. Having two bond characters neighbour each other is invalid. Fourth, round brackets, `()`, indicate where the molecule branches off into a sub-molecule. Every open bracket has to be matched with a closing bracket, and there must be at least one atom between any two brackets. Lastly, digits denote rings in a molecule; each digit has to have a matching digit somewhere else in the molecule, marking where it closes the ring, and if there is a ring inside another ring a different digit is used (incremented by one).

Proceeding with the same example molecule we can elucidate atomic bond constraints in SMILES strings. The atom Bromine can only bond with a single other atom, meaning that it may only occur at the beginning or end of a SMILES string, or within a so-called 'branch', denoted by a bracketed expression `(Br)`. We present the SMILES alphabet in table 6.4 and the number of bonds per atom in table 6.5. Additionally, figure 6.3 illustrates examples of how a string may fail to form a valid SMILES molecule representation.

Table 6.4 SMILES alphabet

Atoms/Chirality	Bonds/Ringbonds	Charges	Branches/Brackets
`B C N O S P F I H Cl Br @`	`=#/\12345678`	`-+`	`()[]`

The intricacy of SMILES strings makes them a suitable testing ground for our method. There are two technical distinctions to make between this experiment and the previously considered Python 3 mathematical expressions. First, as there exist databases of SMILES strings, we leverage those by using the sequence perturbation technique described in section 6.3. The main data source considered is the ZINC dataset (Irwin and Shoichet, 2005), as used by Kusner et al. (2017). We also used the USPTO 15k reaction products data (Lowe, 2014) and a set of molecule solubility information (Huuskonen, 2000) as withheld test data. Second,

Table 6.5 Atomic bond constraints

Atoms	Bonds
`H,F,Cl,Br,I`	1
`O`	2
`N,B`	3
`C`	4
`P`	3,5
`S`	2,4,6

whereas we used fixed length Python 3 expressions in order to obtain coverage bounds, molecules are inherently of variable length. To deal with this, we pad all the molecule sequences to a fixed length.

Evaluating the accuracy of the validity model

As a first test of the suitability of our validity model, we train it on perturbation-augmented ZINC data and examine the accuracy of its predictions on a withheld test partition of that same dataset as well as the two unseen molecule datasets. Here, accuracy defines the ability of the model to accurately recognise which perturbations make a certain SMILES string valid, and which leave it invalid. Effectively, this measures how well the model has captured the grammar of SMILES strings in the vicinity of the data manifold. Again, we would like to point out that using the active learning approach on this task would result in a performance worse than that of the sequence perturbation technique because the active learning explores a space much larger than what it is tested on.

We visualize the predictions $\mathrm{y}(x_t|\mathbf{x}_{<t}, \omega)$ at each time step of the trained model on a test molecule in figure 6.3 and highlight some of the learned rules. Clearly, the model has learned the SMILES grammar; it knows that the oxygen atom `O` at position 10 requires two bonds, and since it is preceded by a double bond, the model knows that most characters would be invalid here except for the closing bracket). The same can be seen for bromine `Br` at position 18, which can only form a single bond. At the bottom, positions 32-35 show that the model has also learned that closing square brackets] can only follow an atom that follows an open bracket [. The complete prediction for this molecule is presented in figure 6.4. Here, more of the learned rules become evident, such as the improbable occurrence of the `H` atom, except inside square brackets, and that closing round brackets cannot immediately follow an open round bracket (position 7). Knowing that the model produces sensible predictions, we proceeded to evaluate its accuracy on the held-out test datasets.

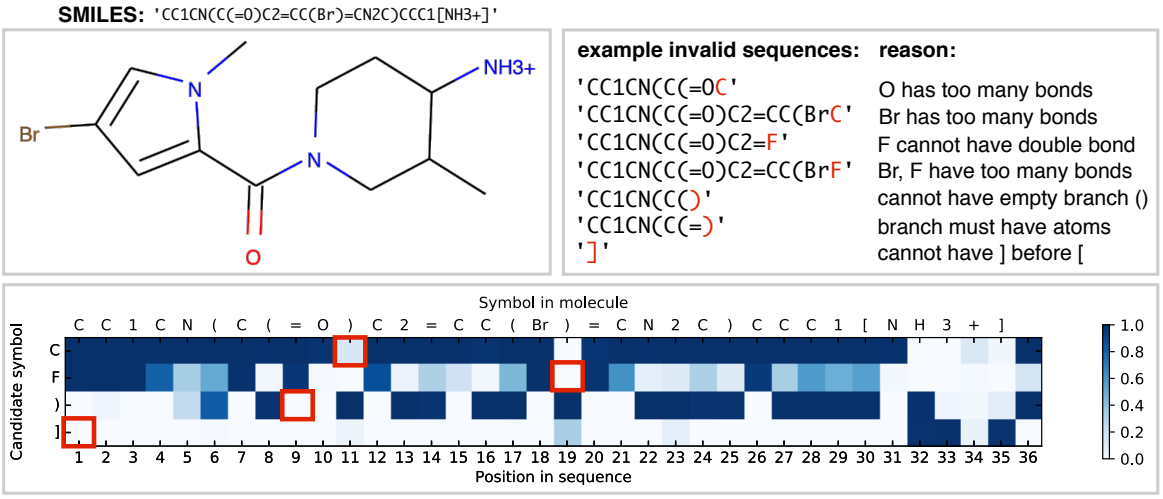


Fig. 6.3 Predictions $\mathrm{y}(x_t|\mathbf{x}_{<t},\omega)$ of the validity model at each step t for the valid test molecule shown in the top left, for a subset of possible actions selecting as next character `C`, `F`, `)`, or `]`. Each column shows which actions the trained agent believes are valid at each t, given the preceding characters $\mathbf{x}_{<t}$. We see that the validity model has learned basic valence constraints: for example the oxygen atom `O` at position 10 requires two bonds, and since it is preceded by a double bond, the model knows that neither carbon `C` nor fluorine `F` can immediately follow it at position 11; we see the same after the bromine `Br` at position 18, which can only form a single bond. The model also correctly identifies that closing branch symbols `)` cannot immediately follow opening branches (after positions 6, 8, and 17), as well as that closing brackets `]` cannot occur until an open bracket has been followed by at least one atom (at positions 32–35). The full output heatmap for this example molecule is shown in Figure 6.4. In the top right image, we show some examples of invalid SMILES strings.

Recalling that a sequence is invalid if $\tilde{v}(\mathbf{x}_{1:t}) = 0$ at any $t \leq T$, we consider the model prediction for molecule $\mathbf{x}_{1:T}$ to be $\prod_{t=1}^{T} \mathbb{I}[\mathrm{y}(x_t|\mathbf{x}_{<t},\omega) \geq 0.5]$, and compare this to its true label as given by *rdkit*, a chemical informatics software package. The results confirm that the model is able to learn the grammar of molecules; accuracies of 99,8%, 100%, and 100% were achieved on the perturbed ZINC (test), perturbed USPTO, and perturbed solubility data respectively. Given the successful learning of the molecule grammar, we now continue to show how standard generative models can be augmented by this validity model.

Improving generative models with the validity model

In order to demonstrate the validity model's capability of improving existing generative models for discrete structures, we show how it can be used to improve the results of previous work – specifically, the character variational autoencoder (CVAE) applied to SMILES strings (Gómez-Bombarelli et al., 2016b). Much like the autoencoder described in section 3.5.1, here an encoder maps points in $\mathcal{X}_+$ to a continuous latent representation $\mathcal{Z}$ and a paired decoder maps $\mathcal{Z}$ back to $\mathcal{X}_+$. A reconstruction-based error is minimized such that training

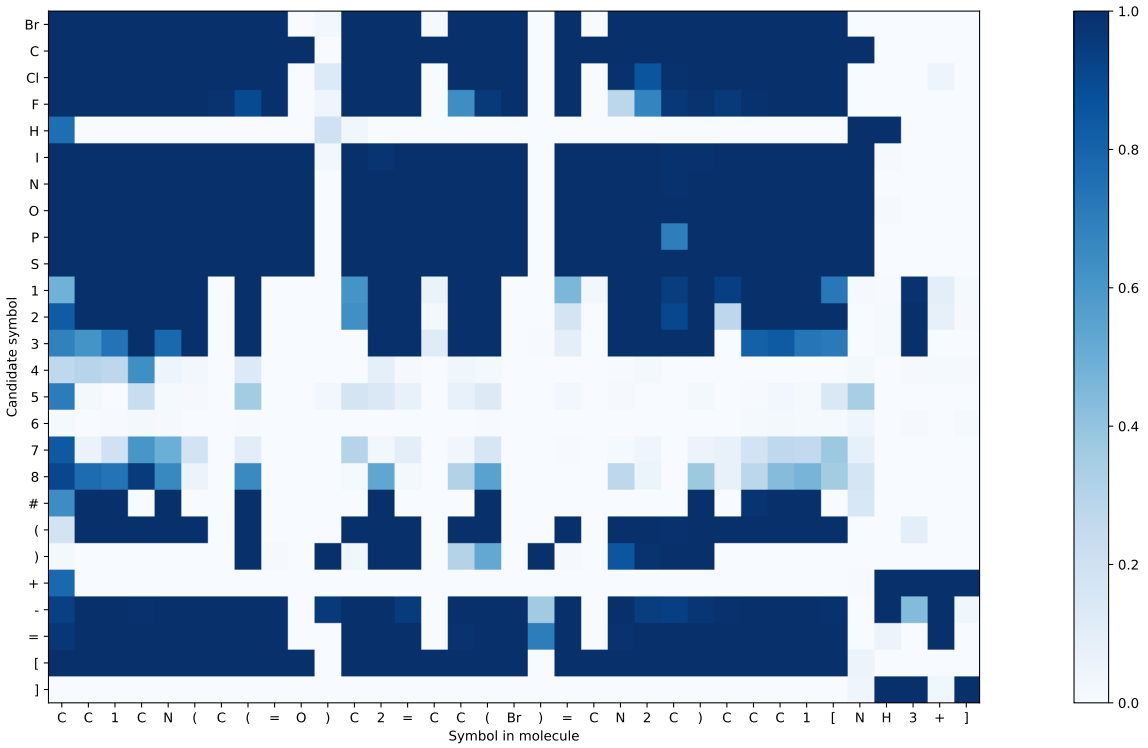


Fig. 6.4 Full heatmap showing predictions $\mathrm{y}(x_t|\mathbf{x}_{<t},\omega)$ for the molecule in Figure 6.3. Darker blocks indicate higher probabilities of a valid complete sequence with the corresponding symbol.

points mapped to the latent space decode back into the same SMILES strings. Consequently, the reconstruction accuracy is the fraction of output sequences that are the same as their corresponding input sequences.

Compared to the standard autoencoder, the **variational autoencoder** (Kingma and Welling, 2013) has an additional loss term that encourages the posterior over $\mathcal{Z}$ to be close to some prior, typically a normal distribution. We can interpret $\mathbf{z} \in \mathcal{Z}$ as a latent variable in a probabilistic generative model; a probabilistic decoder is defined by a θ parametrized likelihood function $p_\theta(\mathbf{x}|\mathbf{z})$. Alongside the latent prior distribution $p_\theta(\mathbf{z})$, the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x}) \propto p_\theta(\mathbf{z})p_\theta(\mathbf{x}|\mathbf{z})$ can then be interpreted as a probabilistic encoder.

To admit efficient inference the posterior is approximated by $q_\phi(\mathbf{z}|\mathbf{x}^{(n)}) = \mathcal{N}(\mathbf{z};\mu^{(n)},\sigma^{2(n)}\mathbf{I})$ which employs the reparametrization trick to enable back-propagation through samples of $\mathbf{z}$,

$$\mathbf{z} = \mu + \sigma \odot \varepsilon, \quad \varepsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I}), \tag{6.16}$$

and deep learning models are used to learn mappings from $\mathbf{x}$ to μ and $\mathbf{x}$ to σ. The aim is then to maximize the expected reconstruction log-likelihood and concurrently minimize the KL-divergence between the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and the latent prior $p_\theta(\mathbf{z})$, which

results in the following objective function

$$\mathcal{L}(\theta,\phi) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\big). \tag{6.17}$$

Owing to the KL term being non-negative, this objective function is a lower bound on the log-likelihood of the data $\mathcal{L}(\theta,\phi) \leq \log p_\theta(\mathbf{x})$, known as the evidence lower bound (ELBO). Using this objective function the variational autoencoder can be trained in the same way as the standard autoencoder, via stochastic gradient descent, and the posterior over $\mathcal{Z}$ can be encouraged to have some desired continuous distribution.

For molecules, a key performance indicator of the variational autoencoder model is the fraction of representations sampled from the prior over $\mathcal{Z}$ that decode into valid molecules. If many sampled representations correspond to invalid molecules, any sort of predictive modelling on the $\mathcal{Z}$ space would most likely also yield many invalid SMILES strings.

In order to explain how the validity model can augment this character variational autoencoder, we first describe the decoding in more detail. The decoder functions by outputting a set of weights $p_\theta(x_t|\phi_t,\mathbf{z})$ for each character x_t in the reconstructed sequence conditioned on the latent representation $\mathbf{z} \in \mathcal{Z}$ produced by the encoder and the latent state ϕ_t of the recurrent neural network; the sequence is recovered by sampling from a multinomial according to these weights x_t. To integrate our validity model into this framework, we take the decoder output for each step t and mask out the options that the validity model predicts to lead to an invalid sequence. Hence, we sample characters from a multinomial according to the new weights given by $p_\theta(x_t|\phi_t,\mathbf{z}) \cdot \mathbb{I}[\mathrm{y}(x_t|\mathbf{x}_{<t},\omega) \geq 0.5]$.

In table 6.6 we compare our approach to previous models. We use a Kekulé format of the ZINC data in our experiments – a specific representation of aromatic bonds that our model handled particularly well. The results reported for the grammar variational autoencoder (VAE) are taken directly from Kusner et al. (2017) and on non-Kekulé format data. The character variational autoencoder (CVAE) model is trained for 100 epochs, as per previous work, with *Adam* at a learning rate of 0.001. The structure of all these VAEs are identical to the optimal structure found in Gómez-Bombarelli et al. (2016b). The encoder uses three 1D convolutional neural network layers of filter sizes 9, 9, 11 and 9, 9, 10 convolutional kernels, respectively, followed by two fully-connected layers of width 435 and 292. The decoder starts with a fully-connected layer of width 292, fed into a gated recurrent unit (GRU) network with three hidden layers of 501 units. The validity model used here has the same hyperparameters as before (table 6.2).

From the sample validities, it is evident that the addition of our validity model to the character VAE tremendously bolsters the VAE's ability to decode a continuous latent representation into a valid molecule. An added bonus is that the action of our model is completely

Table 6.6 Performance of different variational autoencoders applied to molecules. The three models are the character variational autoencoder (CVAE), the grammar VAE, and the CVAE with the proposed validity model overlaid at test time. Sample validity is the percentage of samples from the prior over $\mathcal{Z}$ that decode into valid molecules.

Model	Reconstruction accuracy [%]	Sample validity [%]
CVAE + validity model	50.2	22.3
Grammar VAE	53.7	7.2
CVAE	49.7	0.5

post-hoc, and is thus applicable to any pre-trained character-based generative model where elements of the latent space correspond to a structured discrete sequence. Note that the binary nature of the proposed validity model means that it should not affect the reconstruction accuracy. In fact, some modest gains are present, as shown by the reconstruction accuracies in table 6.6.

6.5 Discussion

In this chapter, we proposed a modelling technique for learning the validity constraints of discrete spaces. Particularly, the proposed log-likelihood (eq. 6.3) makes the model easy to train, is unaffected by the introduction of padding for variable length sequences and, as its optimum is largely independent of the training data distribution, it allows for the utilization of active learning techniques. Empirically we found it essential to present the model with boundary examples of grammar constraints. Where no such informative dataset exists, we proposed a mutual-information-based active learning scheme that uses model uncertainty to select training sequences. Where datasets of positive examples are available, we proposed a simple method of perturbations to create informative examples of validity constraints being broken.

The efficacy of the active learning approach was demonstrated on Python 3 expressions. In the context of molecules, the model was able to almost perfectly learn the validity of independently perturbed molecules. When applied to the variational autoencoder benchmark on molecules, the proposed method outperformed previous results by a large margin on prior sample validity – the relevant metric for the downstream utility of the latent space.

Prudent selection of the objective function for the LSTM and leveraging the sequential nature of the LSTM for active learning proved to be a useful modification in this chapter. Just like the convolutional neural network (CNN), the LSTM network is a powerful model tailored to the structure of the input. Since the original CNN, there have been several novel algorithms

and architectures which considerably improve the performance of the conventional CNN; these include the Residual network (He et al., 2016), the Inception network (Szegedy et al., 2015), and the Highway network (Srivastava et al., 2015b). Similarly, LSTMs have seen several modifications that have made them better, such as the phased LSTM (Neil et al., 2016), the tensorized LSTM (He et al., 2017), and the stochastic LSTM (Fraccaro et al., 2016). Nevertheless, as we have shown here, there remains room for improving LSTMs. In the following section, we propose our second extension of the LSTM, which provides some interesting benefits.

Chapter 7

The unreasonable effectiveness of the forget gate

Good engineers ensure that their designs are practical. To make the sequence-to-sequence model proposed in chapter 4 more practical for implementation in a neuroprosthetic device, a hardware efficient version of the model is desired. Given the success of the gated recurrent unit (GRU) (section 2.5), which uses two gates, the first approach to a more hardware efficient LSTM could be the elimination of redundant gates, if there are any. Because we seek a model more efficient than the GRU, only a single-gate LSTM model is a worthwhile endeavour. To motivate why this single gate should be the forget gate, we begin with the LSTM genesis.

In an era where training recurrent neural networks (RNNs) was notoriously difficult, Hochreiter and Schmidhuber (1997) argued that having a single weight (edge) in the RNN to control whether input or output of a memory unit needs to be accepted or ignored, creates conflicting updates (gradients). Essentially, the long and short-range error act on the same weight at each step, and with sigmoid activated units, this results in the gradients vanishing faster than the parameters can grow. They proceeded to propose the long short-term memory (LSTM) network, which had multiplicative input and output gates. These gates would mitigate the conflicting update issue by "protecting" the hidden units from irrelevant information, either from the input or from the output of other units.

This first version of the LSTM had only two gates; it was Gers et al. (2000) who realized that if there is no mechanism for the memory units to forget information, they may grow indefinitely and eventually cause the network to break down. As a solution, they proposed another multiplicative gate for the LSTM architecture, known as the forget gate – completing the version of the LSTM that we know today.

Comment: It's interesting to note the difference between the motivations that lead to the LSTM and the chain-of-thought that yielded the gated recurrent unit (GRU). Cho was "not well aware" (Cho, 2015, §4.2.3) of the LSTM when he, together with collaborators, designed the GRU. In contrast to the conflicting update problem (Hochreiter and Schmidhuber, 1997) and the indefinite state growth (Gers et al., 2000) arguments, Cho (2015) approached the RNN problem by thinking of it as a computer processor with memory registers. In the case of computers, we do not want to overwrite all the registers (memory values) at each step. Therefore, the RNN requires an update gate, which controls the hidden states (registers) that are overwritten (the update gate in the GRU is akin to the combined function of the input and forget gates in the LSTM). Furthermore, we do not necessarily need to read all the registers at each time step, only the important ones. Thus another gate is required in the RNN (the reset gate) to regulate the registers considered. Ideally, all of the gating operations would be binary values, but such values would result in zero gradients. Fortunately, the sigmoid or tanh nonlinearities provide leaky versions of these gating mechanisms and have smooth gradients.

It wasn't until many years later that Greff et al. (2015) and Jozefowicz et al. (2015) simultaneously discovered the forget gate to be the crucial ingredient of the LSTM. Gers et al. (2000) proposed initializing the forget gate biases to positive values and Jozefowicz et al. (2015) showed that an initial bias of 1 for the LSTM forget gate makes the LSTM as strong as the best of the explored architectural variants (including the GRU) (Goodfellow et al., 2016, §10.10.2). Given the new-found importance of the forget gate, would the input and output gates have been found necessary if the LSTM was conceived with only a forget gate?

In this chapter, we take the liberty of exploring the gains introduced by the sole use of the forget gate. We empirically demonstrate that for two medical and four LSTM benchmark datasets, only the forget gate is required and provides a better solution than the use of all three LSTM gates. After some discussion in Cambridge University's Signal Processing Laboratory's coffee room, we decided to name this architecture JANET (Just Another NETwork)[1]. Before we provide the details of this network, we review some of the many modifications that have been made to the LSTM.

[1]Not to be confused with Jos's Awesome NETwork!

7.1 Related work

With some success, many studies have improved the LSTM by making the hidden units more complex (Fraccaro et al., 2016; Graves, 2011; He et al., 2017; Krueger et al., 2017; Neil et al., 2016), with classic examples being peephole connections (Gers and Schmidhuber, 2000) and depth gated LSTMs (Yao et al., 2015). Similarly, several studies have proposed recurrent neural networks (RNN) simpler than the LSTM yet still competitive, such as the skip-connected RNN (Zhang et al., 2016), the unitary RNN (Arjovsky et al., 2016), the Delta-RNN (Ororbia II et al., 2017), and the identity RNN (Le et al., 2015). However, one of the most thorough studies on the architecture of the LSTM is probably the study by Greff et al. (2015) (5,400 experiment simulations). They explored the following LSTM variants individually:

- No input gate
- No forget gate
- No output gate
- No input activation function
- No output activation function
- No peepholes
- Coupled input and forget gate
- Full gate recurrence

The first five variants are self-explanatory. Peepholes (Gers and Schmidhuber, 2000) connect the memory unit to the gates; the gate activations in the LSTM (eq. 2.26) become

$$\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t = \sigma(\mathbf{p}_{i,o,f} \odot \mathbf{c}_{t-1} + \mathbf{U}_{i,o,f}\mathbf{h}_{t-1} + \mathbf{W}_{i,o,f}\mathbf{x}_t + \mathbf{b}_{i,o,f}), \tag{7.1}$$

where $\mathbf{p}_{i,o,f}$ are the peephole weight vectors (Greff et al., 2015). The coupled input and forget gate variant uses only one gate for modulating the input and the recurrent self-connections, i.e., $\mathbf{f} = \mathbf{1} - \mathbf{i}$. Full gate recurrence is the initial setup of Hochreiter and Schmidhuber (1997), wherein all the gates receive recurrent inputs from all gates at the previous time step. This cumbersome architecture requires nine additional recurrent weight matrices and did not feature in any of their later papers. Interestingly, the results in Greff et al. (2015) indicate that none of the variants significantly improve on the standard LSTM. The forget gate was found to be essential, but a forget-gate-only variant was not explored.

Two studies that are closely related to ours are those by Zhou et al. (2016) and Wu and King (2016). The former successfully implemented a similar gate reduction to the gated recurrent unit (GRU); they couple the reset (input) gate to the update (forget) gate and show

that this minimal gated unit (MGU) achieves a performance similar to the standard GRU with only two-thirds of the parameters. The study by Wu and King (2016) proposes a gate reduction similar to that of ours for LSTMs. They demonstrate that their *simple* LSTM achieves the same performance as the standard LSTM on a speech synthesis task. Compared with our work, they keep the hyperbolic tangent activation function on the memory unit, and their implementation did not employ the same bias initialization scheme, which we show is paramount for successful implementation of these models over a wide range of datasets. We became aware of these studies after having completed most of our work; our simplification of the LSTM provides a network that yields classification accuracies at least as good as the standard LSTM and often performs substantially better – a result not achieved by the models proposed in the afore-mentioned studies.

7.2 Just Another NETwork

Recurrent neural networks (RNNs) typically create a lossy summary $\mathbf{h}_T$ of a sequence. It is lossy because it maps an arbitrarily long sequence $\mathbf{x}_{1:T}$ into a fixed length vector. As mentioned before, recent work has shown that this forgetting property of LSTMs is one of the most important (Greff et al., 2015; Jozefowicz et al., 2015). Hence, we propose a simple transformation of the LSTM that leaves it with only a forget gate, and since this is just another network (JANET), we name it accordingly. Recall from eq. 2.26 that the LSTM is defined as

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t + \mathbf{b}_o) \\
\mathbf{f}_t &= \sigma(\mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t + \mathbf{b}_f) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t + \mathbf{b}_c) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned} \tag{7.2}$$

To transform the above into the JANET architecture, the input and output gates are removed. It seems sensible to have the accumulation and deletion of information be related, therefore we couple the input and forget modulation as in Greff et al. (2015), which is similar to the leaky unit implementation (Jaeger, 2002, §8.1). Furthermore, the tanh activation of $\mathbf{h}_t$ shrinks the gradients during back-propagation, which could exacerbate the vanishing gradient problem, and since the weights $\mathbf{U}_*$ can accommodate values beyond the range [-1,1], we can remove this unnecessary, potentially problematic, tanh nonlinearity. The resulting JANET is

given by

$$\begin{aligned}\mathbf{f}_t &= \sigma(\mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t + \mathbf{b}_f)\\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + (\mathbf{1} - \mathbf{f}_t) \odot \tanh(\mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t + \mathbf{b}_c)\\ \mathbf{h}_t &= \mathbf{c}_t \end{aligned} \tag{7.3}$$

Intuitively, allowing slightly more information to accumulate than the amount forgotten would make sequence analysis easier. We found this to be true empirically by subtracting a pre-specified value β from the input control component[2], as given by

$$\begin{aligned}\mathbf{s}_t &= \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t + \mathbf{b}_f\\ \tilde{\mathbf{c}}_t &= \tanh(\mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t + \mathbf{b}_c)\\ \mathbf{c}_t &= \sigma(\mathbf{s}_t) \odot \mathbf{c}_{t-1} + (\mathbf{1} - \sigma(\mathbf{s}_t - \beta)) \odot \tilde{\mathbf{c}}_t\\ \mathbf{h}_t &= \mathbf{c}_t. \end{aligned} \tag{7.4}$$

We speculate that the value for β is dataset dependent, however, we found that setting $\beta = 1$ provides the best results for datasets analysed in this study, which have sequence lengths varying from 200 to 784.

If we follow the standard parameter initialization scheme for LSTMs, the JANET quickly encounters a problem. The standard procedure is to initialize the weights $\mathbf{U}_*$ and $\mathbf{W}_*$ to be distributed as $\mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}}, \frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}}\right]$, where n_l is the size of each layer l (Glorot and Bengio, 2010b; He et al., 2015), and to initialize all biases to zero except for the forget gate bias $\mathbf{b}_f$, which is initialized to one (Jozefowicz et al., 2015). Hence, if the values of both input and hidden layers are zero-centred over time, $\mathbf{f}_t$ will be centred around $\sigma(1) = 0.7311$. In this case, the memory values $\mathbf{c}_t$ of the JANET would not be retained for more than a couple of time steps. This problem is best exemplified by the MNIST dataset (LeCun, 1998) processed in scanline order (Cooijmans et al., 2016); each training example contains many consecutive zero-valued subsequences, each of length 10 to 20. In the best case scenario – a length 10 zero-valued subsequence – the memory values at the end of the subsequence would be centred around

$$\mathbf{c}_{t+10} = \mathbf{f}_t^{10} \odot \mathbf{c}_t \leq 0.7311^{10}\mathbf{c}_t \leq 0.0436\mathbf{c}_t. \tag{7.5}$$

Thus, with the standard initialization scheme, little information would be propagated during the forward pass and in turn, the gradients will quickly vanish.

[2] β is a constant-valued column vector of the appropriate size.

Fortunately, the recent work by Tallec and Ollivier (2018) proposed a more suitable initialization scheme for the forget gate biases of the LSTM. To motivate this initialization scheme we start by re-writing the leaky RNN (Jaeger, 2002, §8.1)

$$\mathbf{h}_{t+1} = \alpha \odot \tanh(\mathbf{U}\mathbf{h}_t + \mathbf{W}\mathbf{x}_t + \mathbf{b}) + (\mathbf{1} - \alpha) \odot \mathbf{h}_t, \tag{7.6}$$

as its continuous time version, by making use of the first order Taylor expansion $h(t+\delta t) \approx h(t) + \delta t \frac{\mathrm{d}h(t)}{\mathrm{d}t}$ and a discretization step $\delta t = 1$,

$$\frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \alpha \odot \tanh\Big(\mathbf{U}\mathbf{h}(t) + \mathbf{W}\mathbf{x}(t) + \mathbf{b}\Big) - \alpha \odot \mathbf{h}(t). \tag{7.7}$$

Tallec and Ollivier (2018) state that in the free regime, when inputs stop after a certain time t_0, $\mathbf{x}(t) = \mathbf{0}$ for $t > t_0$, with $\mathbf{b} = \mathbf{0}$ and $\mathbf{U} = \mathbf{0}$, eq. 7.7 becomes

$$\begin{aligned} \frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} &= -\alpha\mathbf{h}(t) \\ \int_{t_0}^{t} \frac{1}{\mathbf{h}(t)}\,\mathrm{d}\mathbf{h}(t) &= -\alpha \int_{t_0}^{t} \mathrm{d}t \\ \mathbf{h}(t) &= \mathbf{h}(t_0)\exp(-\alpha(t-t_0)). \end{aligned} \tag{7.8}$$

From eq. 7.8 the hidden state $\mathbf{h}$ will decrease to $\exp(-1)$ of its original value over a time proportional to $1/\alpha$. This $1/\alpha$ can be interpreted as the characteristic forgetting time, or the time constant, of the recurrent neural network. Therefore, when modelling sequential data believed to have dependencies in a range $[T_{\min}, T_{\max}]$, it would be sensible to use a model with a forgetting time lying in approximately the same range, i.e., having $\alpha \in [\frac{1}{T_{\max}}, \frac{1}{T_{\min}}]^d$, where d is the number of hidden units.

In the LSTM the input gate $\mathbf{i}$ and the forget gate $\mathbf{f}$ learn time-varying approximations of α and $(1-\alpha)$, respectively. Obtaining a forgetting time centred around T requires $\mathbf{i}$ to be centred around $1/T$ and $\mathbf{f}$ to be centred around $(1-1/T)$. Assuming the shortest dependencies to be a single time step, Tallec and Ollivier (2018) propose the **chrono initializer**, which initializes the LSTM gate biases as

$$\begin{aligned} \mathbf{b}_f &\sim \log(\mathcal{U}[1, T_{\max} - 1]) \\ \mathbf{b}_i &= -\mathbf{b}_f, \end{aligned} \tag{7.9}$$

with $T_{\max}$ the expected range of long-term dependencies and $\mathcal{U}$ the uniform distribution. Importantly, these are only the initializations, and the gate biases are allowed to change independently during training.

Applying chrono initialization to the forget gate $\mathbf{f}$ of the JANET[3], mitigates the memory issue (eq. 7.5). With the values of the input and hidden layers zero-centred over time, the forget gate corresponding to a long-range ($T_{\max}$) hidden unit will have an activation of

$$\begin{aligned}\sigma(\log(T_{\max}-1)) =& \frac{1}{1+\exp(-\log(T_{\max}-1))} \\ &\xrightarrow[T_{\max}\to\infty]{} 1.\end{aligned} \tag{7.10}$$

Consequently, for the MNIST memory problem ($T_{\max}-1=783$), these long-range units would retain most of their information, even after 20 consecutive zeros

$$\begin{aligned}f^{long} &= \frac{1}{1+\exp(-\log(783))} \geq 0.9987 \\ c_{t+20}^{long} &= (f^{long})^{20} c_t^{long} \geq 0.9987^{20} c_t^{long} \geq 0.97 c_t^{long}.\end{aligned} \tag{7.11}$$

For the JANET, chrono initialization provides an elegant implementation of skip-like connections between the memory units over time. It has long been known that skip connections mitigate the vanishing gradient problem (Lin et al., 1996; Srivastava et al., 2015b). A systematic study of recurrent neural networks (RNNs) by Zhang et al. (2016) found that explicitly adding skip connections in the RNN graph improves performance by allowing information to be transmitted directly between non-consecutive time steps. For RNNs, they devise the recurrent skip coefficient, a value that measures the number of time steps through which unimpeded flow of information is allowed, and argue that higher values are usually better. Furthermore, skip connections are responsible for much of the boom in machine learning; they are the pith of the powerful residual networks (He et al., 2016), highway networks (Srivastava et al., 2015b), and the WaveNet (Van Den Oord et al., 2016). A natural question that follows, is how the skip-connections influence the gradients of the JANET and the LSTM.

7.2.1 A comparison of gradients

Before comparing the gradients of the LSTM and the JANET we provide some preliminaries. We denote the derivatives of the element-wise nonlinearities by the following:

$$\begin{aligned}\sigma'(x) &= \sigma(x)(1-\sigma(x)), & 0 < \sigma'(x) \leq 0.25 \\ \tanh'(x) &= 1-\tanh^2(x), & 0 < \tanh'(x) \leq 1\end{aligned} \tag{7.12}$$

[3]The memory unit biases $\mathbf{b}_c$ are initialized to zero.

For brevity, we denote the pre-activation vectors in eq. 7.2 and 7.3 as

$$\mathbf{s}_{i,o,f,c} = \mathbf{U}_{i,o,f,c}\mathbf{h}_t + \mathbf{W}_{i,o,f,c}\mathbf{x}_{t+1} + \mathbf{b}_{i,o,f,c}. \tag{7.13}$$

Lastly, we consider a diagonal matrix as a vector of its diagonal elements. Thus, a derivative of an element-wise multiplication of two vectors is written as a vector. Consider the following derivative of an element-wise multiplication of vectors $\{\mathbf{a},\mathbf{b}\} \in \mathcal{R}^3$

$$\begin{aligned}
\frac{\partial \mathbf{v}}{\partial \mathbf{a}} &= \frac{\partial}{\partial \mathbf{a}}\mathbf{b} \odot \mathbf{a} \\
&= \begin{bmatrix} \frac{\partial v_1}{\partial a_1} & \frac{\partial v_1}{\partial a_2} & \frac{\partial v_1}{\partial a_3} \\ \frac{\partial v_2}{\partial a_1} & \frac{\partial v_2}{\partial a_2} & \frac{\partial v_2}{\partial a_3} \\ \frac{\partial v_3}{\partial a_1} & \frac{\partial v_3}{\partial a_2} & \frac{\partial v_3}{\partial a_3} \end{bmatrix} \\
&= \begin{bmatrix} b_1 & 0 & 0 \\ 0 & b_2 & 0 \\ 0 & 0 & b_3 \end{bmatrix},
\end{aligned} \tag{7.14}$$

which we write as

$$\frac{\partial \mathbf{v}}{\partial \mathbf{a}} = \mathbf{b}. \tag{7.15}$$

Here we compare the gradient propagation through the memory cells of a single-layer JANET with that of a single-layer LSTM. To analyse this flow of information we can compute the gradient $\partial J / \partial \mathbf{c}_t$ of the objective function J with respect to some arbitrary memory vector $\mathbf{c}_t$. Starting with the JANET (eq. 7.3), we re-write it as

$$\begin{aligned}
\mathbf{f}_{t+1} &= \sigma(\mathbf{s}_f) \\
\mathbf{c}_{t+1} &= \mathbf{f}_{t+1} \odot \mathbf{c}_t + (\mathbf{1} - \mathbf{f}_{t+1}) \odot \tanh(\mathbf{s}_c).
\end{aligned} \tag{7.16}$$

For this architecture the gradient of the objective function is given by

$$\frac{\partial J}{\partial \mathbf{c}_t} = \frac{\partial J}{\partial \mathbf{c}_T} \prod_{k=t}^{T-1} \left[\frac{\partial \mathbf{c}_{k+1}}{\partial \mathbf{c}_k} \right], \tag{7.17}$$

with

$$\begin{aligned}
\frac{\partial \mathbf{c}_{t+1}}{\partial \mathbf{c}_t} = \mathbf{U}_f \sigma'(\mathbf{s}_f) \odot \mathbf{c}_t + \sigma(\mathbf{s}_f) + (1 - \sigma(\mathbf{s}_f)) \odot (\mathbf{U}_c \tanh'(\mathbf{U}_c \mathbf{c}_t)) \\
- \sigma'(\mathbf{s}_f) \odot (\mathbf{U}_f \tanh(\mathbf{U}_c \mathbf{c}_t)).
\end{aligned} \tag{7.18}$$

Assuming that the input and hidden layers are zero-centred over time (as for the memory problem eq. 7.5) and all the forget gate biases are initialized to the longest range (eq. 7.11), $\sigma(\mathbf{s}_f)$ will typically take values of one[4] and $\sigma'(\mathbf{s}_f)$ values near zero. In this scenario, we see that all but one of the terms in eq. 7.18 reduce to zero and we have

$$\frac{\partial \mathbf{c}_{t+1}}{\partial \mathbf{c}_t} = 1, \tag{7.19}$$

meaning that gradients from distant memory cells $\mathbf{c}_t$ are largely unaffected by the sequence length.

Moving on to the LSTM, we re-write eq. 7.2 as

$$\begin{aligned} \mathbf{i}_{t+1}, \mathbf{o}_{t+1}, \mathbf{f}_{t+1} &= \sigma(\mathbf{s}_{i,o,f}) \\ \mathbf{c}_{t+1} &= \mathbf{f}_{t+1} \odot \mathbf{c}_t + \mathbf{i}_{t+1} \odot \tanh(\mathbf{s}_c) \\ \mathbf{h}_{t+1} &= \mathbf{o}_{t+1} \odot \tanh(\mathbf{c}_{t+1}). \end{aligned} \tag{7.20}$$

Here the gradient of the objective function is

$$\frac{\partial J}{\partial \mathbf{c}_t} = \frac{\partial J}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t} + \frac{\partial J}{\partial \mathbf{c}_{t+1}}\frac{\partial \mathbf{c}_{t+1}}{\partial \mathbf{c}_t} = \frac{\partial J}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t} + \frac{\partial J}{\partial \mathbf{c}_{t+1}}\mathbf{f}_{t+1}. \tag{7.21}$$

With a forget gate chrono-initialized to a hypothetical value of one and with $\frac{\partial J}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t} = 0$, the LSTM would permit unhindered gradient propagation. Under standard and chrono-initialization schemes, however, this $\frac{\partial J}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t}$ term is unlikely to be zero. First,

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t} = \mathbf{o}_t \odot \tanh'(\mathbf{c}_t), \tag{7.22}$$

which is non-zero with $0 \leq \mathbf{o}_t \leq 1$ (centred around 0.5 under the memory problem assumptions eq. 7.5) and $0 \leq \tanh'(\mathbf{c}_t) \leq 1$. Second,

$$\frac{\partial J}{\partial \mathbf{h}_t} = \frac{\partial J}{\partial \mathbf{h}_{t+1}}\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial J}{\partial \mathbf{h}_{t+1}}\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{c}_{t+1}}\frac{\partial \mathbf{c}_{t+1}}{\partial \mathbf{h}_t}, \tag{7.23}$$

where under chrono-initialized assumptions

$$\frac{\partial \mathbf{c}_{t+1}}{\partial \mathbf{h}_t} = \mathbf{U}_f \sigma'(\mathbf{s}_f) \odot \mathbf{c}_t + \mathbf{U}_i \sigma'(\mathbf{s}_i) \odot \tanh(\mathbf{s}_c) + \sigma(\mathbf{s}_i) \odot (\mathbf{U}_g \tanh'(\mathbf{s}_c)) \tag{7.24}$$

[4] With the biases large enough for $\sigma(\mathbf{s}_f) \approx \sigma(\mathbf{s}_f - \beta)$

would typically take values of zero because $\sigma(\mathbf{s}_i)$, $\sigma'(\mathbf{s}_f)$ and $\sigma'(\mathbf{s}_i)$ are centred near zero, but $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$ depends on gradients w.r.t. the output gate and new-input functions ($\mathbf{o}_{t+1}$ and $\tilde{\mathbf{c}}_{t+1}$), resulting in a summation of non-zero gradients. Initializing the biases of these two gates such that $\frac{\partial J}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t} = 0$ could provide a better solution for the LSTM and we leave exploration of this for future work.

In practice the gradients are not as ill-conditioned as we have described here because the gate activations are not homogeneous; some gate combinations track short-term dependencies and others track long-term dependencies. However, with all the initializations kept the same, these derivations could explain why the JANET could be easier to train than the LSTM.

We have shown how the simplification of the LSTM could lead to a better-conditioned training regime. We follow with the theoretical computational savings gleaned by this simplification.

7.2.2 Theoretic computational benefits

Hardware efficient machine learning is a field of study by itself (Adolf et al., 2016; Han et al., 2015; Hinton et al., 2015; Sindhwani et al., 2015; Wang et al., 2017). The general aim is to maintain the same level of accuracy but require less computational resources in the process. Usually, this applies to only the forward pass efficiency of the network, i.e., being able to run a trained network on a small device. This is the same goal we have for our simplified version of the LSTM. If we assume the accuracies of the JANET and the LSTM to be the same, how much do we save on computation?

Consider an $n_1 \times n_2$ LSTM layer that has n_1 inputs and n_2 hidden units, then we have $\mathbf{x}_t \in \mathbb{R}^{n_1}$, $\{\mathbf{c}_t\ ,\ \mathbf{h}_t\ ,\ \mathbf{b}_j\} \in \mathbb{R}^{n_2}$, $\mathbf{W}_j \in \mathbb{R}^{n_1 \times n_2}, \mathbf{U}_j \in \mathbb{R}^{n_1 \times n_2}$. For the LSTM we have $j = \{i, o, f, c\}$, and the total number of parameters is $4(n_1 n_2 + n_2^2 + n_2)$. For the JANET we have $j = \{f, c\}$, and the total number of parameters is $2(n_1 n_2 + n_2^2 + n_2)$. Thus we reduce the number of parameters by half, but what does this mean in terms of memory consumption and computational cost? A proxy for the required memory is the number of values that need to be in memory at each step; e.g., the LSTM requires $n_1 + n_2 + n_2 + 4(n_1 n_2 + n_2^2 + n_2)$ values to be stored. Since this value is dominated by the $4 \times n_2^2$ term (typically $n_2 \geq 100$), the JANET would require approximately half of the memory required by an LSTM in a forward pass. Adolf et al. (2016) showed that matrix and element-wise multiplication operations each constitute roughly half of the computation required by an LSTM. With the JANET, processing required for element-wise multiplications is reduced by one third because there are no output gate element-wise multiplications. Thus, the total processing power required by the JANET is roughly $0.5 + \frac{2}{3} \times 0.5 = \frac{5}{6}^{\text{ths}}$ of the processing power required by the LSTM.

If we assume that the electrical power consumed by the memory component of our device is 5% of that consumed by the processor (Acar et al., 2016), then the JANET will consume approximately $0.95 \times \frac{5}{6} + 0.05 \times 0.5 = 0.817$ of the electrical power consumed by the LSTM. However, this ratio is a theoretical estimation and could be different in practice.

Such computational efficiencies are particularly beneficial when applications involve resource-constrained devices. The application researched in chapter 4 would be realized as a device implanted in an amputee – an application that would require hardware efficient machine learning. If our simplification of the LSTM is able to provide the same classification accuracy as the standard LSTM, this would be an essential step towards hardware efficient LSTMs.

7.3 Experiments and results

We evaluate the performance of JANET on four datasets of which three have been introduced earlier in this thesis. Here we introduce the permuted MNIST (pMNIST) dataset, which is the same as the MNIST dataset (section 3.4.3), except, the pixels in each image have been permuted in the same random order. As stated by Arjovsky et al. (2016), the MNIST images have regular distinctive patterns much shorter than the 784-long input sequences; permuting the pixels create longer-term dependencies that are harder for LSTMs to learn. The three other datasets used are the standard MNIST dataset, the traumatic brain injury (TBI) dataset (section 3.2.2), and the MIT-BIH arrhythmia dataset (section 3.4.3).

In table 7.1 we present the test set accuracies on the four different datasets. In addition to JANET and the standard LSTM, we show the results obtained with a standard recurrent neural network (RNN) and a gated recurrent unit (GRU) RNN. For the traumatic brain injury (TBI) and MNIST datasets, all the models had two hidden layers of 128 units. For the MIT-BIH arrhythmia and pMNIST datasets, all the models had a single hidden layer of 128 units. All the networks were trained using *Adam* (Kingma and Ba, 2015) with a learning rate of 0.001 and a minibatch size of 200. Dropout of 0.1 was used on the output of the recurrent layers, and a weight decay factor of 1e-5 was used. For the LSTM and JANET chrono initialization was employed. The models were trained for 100 epochs and the best validation loss was used to determine the final model.

Surprisingly, the results indicate that the JANET yields higher accuracies than the standard, LSTM. Moreover, JANET is among the top performing models on all of the analysed datasets. Thus, by simplifying the LSTM, we not only save on computational cost but also gain in test set accuracy!

Table 7.1 Accuracies [%] for different recurrent neural network architectures. All networks have a single hidden layer of 128 units unless otherwise stated. The means and standard deviations from 10 independent runs are presented. The best accuracies of our experiments are presented in bold as well as the best cited results.

Model	MNIST	pMNIST	MIT-BIH	TBI
JANET	**99.0** $\pm$ 0.120	**92.5** $\pm$ 0.767	**88.3** $\pm$ 0.193	**86.4** $\pm$ 1.391
LSTM	98.5 $\pm$ 0.183	91.0 $\pm$ 0.518	87.4 $\pm$ 0.130	86.1 $\pm$ 1.405
RNN	10.8 $\pm$ 0.689	67.8 $\pm$ 20.176	73.5 $\pm$ 4.531	82.1 $\pm$ 1.536
uRNN (Arjovsky et al., 2016)	95.1	91.4	-	-
iRNN (Le et al., 2015)	97.0	82.0	-	-
tLSTM[a] (He et al., 2017)	**99.2**	**94.6**	-	-
stanh RNN[b] (Zhang et al., 2016)	98.1	94.0	-	-

[a] Effectively has more layers than the other networks.
[b] Single hidden layer of 128 units.

As in Zhang et al. (2016), due to the 10 to 20 long subsequences of consecutive zeros (see section 7.2), we found training of LSTMs to be harder on MNIST compared to training on pMNIST. By harder, we mean that gradient problems and bad local minima cause the objective function to have a rougher and consequent slower descent than the smooth monotonic descent experienced when training is easy. This does not mean that achieving near-perfect classification is more difficult; near-perfect classification on MNIST is relatively easy, whereas the longer-range dependencies in the pMNIST dataset render near-perfect classification difficult. This pMNIST permutation, in fact, blends the zeros and ones for each data point, giving rise to more uniform sequences, which make training easier.

In figure 7.1 we elucidate the difficulty of training on MNIST digits, processed in scanline order. From the figure, LSTMs clearly have a rougher ascent in accuracy on MNIST than on pMNIST and can sometimes fail catastrophically on MNIST. The chrono initializer prevents this catastrophic failure during training, but it results in a lower optimum accuracy. On the pMNIST dataset, there were no discernible differences between the chrono and standard-initialized LSTMs – the benefits of chrono initialization for LSTMs are not obvious on these datasets.

As described in section 7.2, the JANET allows skip connections over time steps of the sequence. In figure 7.2 we show how these skip connections result in the JANET being more efficient to train than the LSTM on the MNIST dataset. There is a recent machine learning

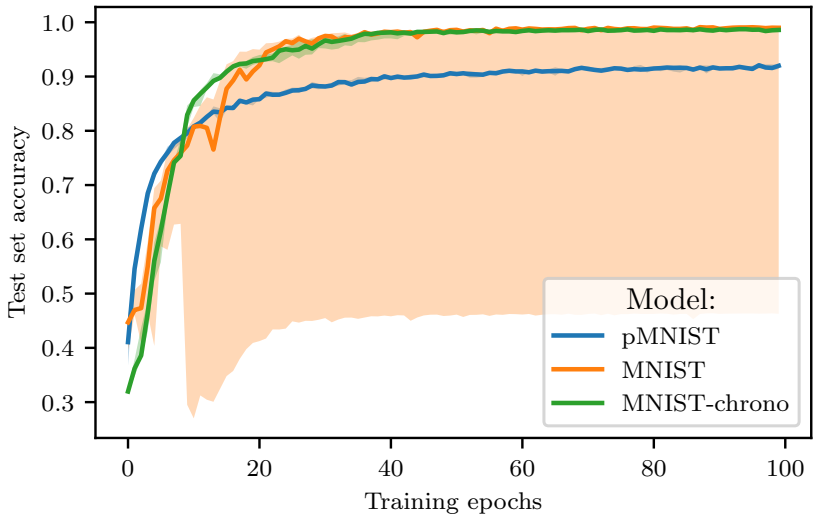


Fig. 7.1 Test accuracies during training for the LSTM on MNIST and pMNIST. The median values are shown with the 10^{th} and 90^{th} percentiles shaded (too small to see for green and blue). MNIST-chrono refers to the chrono-initialized LSTM. There was no discernible difference between a chrono-initialized and standard-initialized LSTM on pMNIST. The plots indicate that training is harder on the MNIST dataset, with both LSTM models having a rougher and slower ascent to the optimum accuracy than the model trained on pMNIST. Furthermore, the standard-initialized LSTM catastrophically failed for one of the 10 simulations.

theme of creating models that are easier to optimize instead of creating better optimizers, which is difficult (Goodfellow et al., 2016, §10.11). Being an easier to train version of the LSTM, the JANET continues this theme.

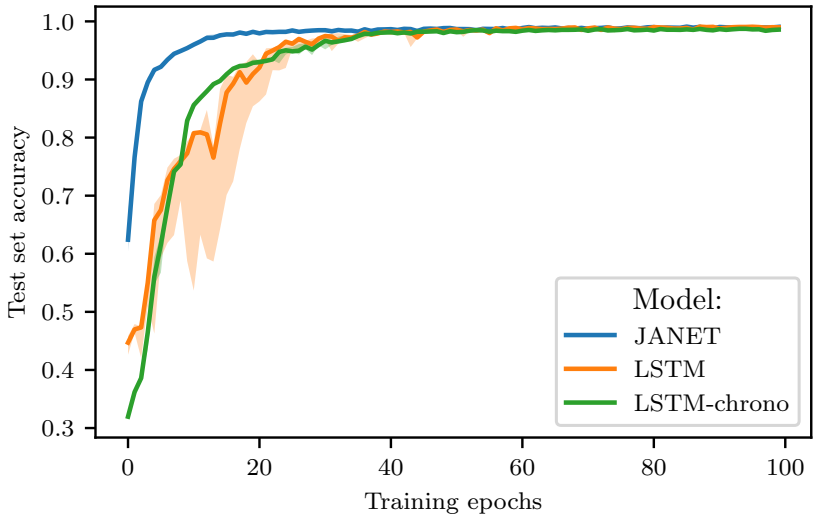


Fig. 7.2 Comparing test set accuracies over training epochs for the JANET and the LSTM on MNIST. The median values are plotted with the 25^{th} and 75^{th} percentiles shaded (too small to see for green and blue). LSTM-chrono refers to the chrono-initialized LSTM. Compared with the LSTM, the JANET has a quicker and smoother ascent of test set accuracy during training.

Given the success of the JANET on the pMNIST dataset (table 7.1), we experimented with larger layer sizes. In figure 7.3 we illustrate the test set accuracies during training for different layer sizes of the LSTM and the JANET. Additionally, we depict the best-reported accuracy on pMNIST (He et al., 2017) with the dashed blue line. This best accuracy of 96.7% was achieved by a WaveNet, a network with dilated convolutional neural network layers (Van Den Oord et al., 2016). The dilation increases exponentially across the layers and essentially enables skip connections over multiple time steps, as shown in figure 7.4.

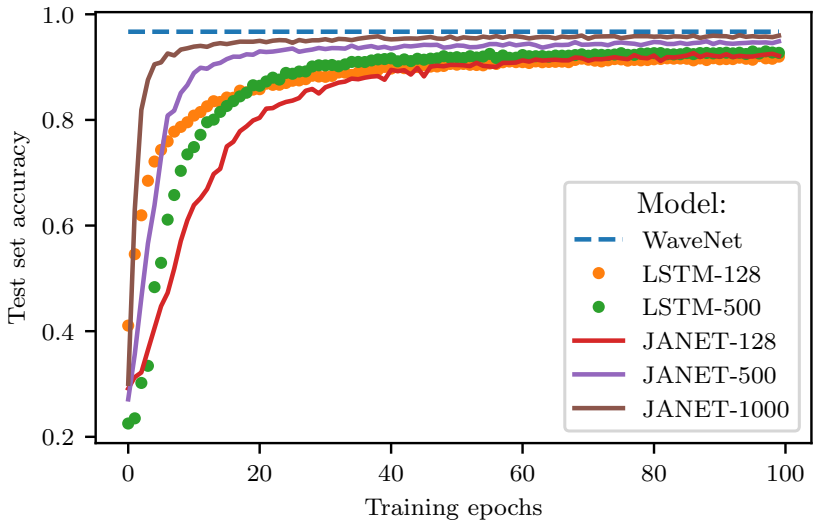


Fig. 7.3 The accuracy achieved on pMNIST for different layer sizes (indicated in the legend) of the JANET and the chrono-initialized LSTM. The dashed blue depicts the best-reported accuracy on pMNIST (He et al., 2017), which was achieved by a WaveNet (Van Den Oord et al., 2016). The JANET clearly improves with a larger layer and performs almost as well as the WaveNet.

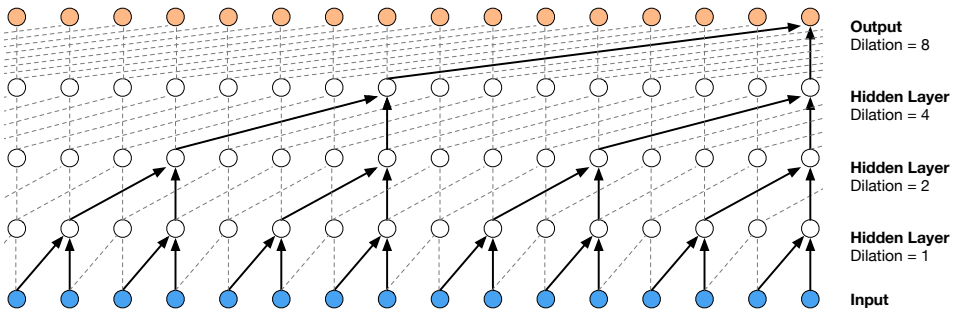


Fig. 7.4 Visualization of a stack of dilated causal convolutional layers. This creates a skip connection mechanism for information to flow from the first input to the last output. (Adapted from Van Den Oord et al. (2016))

The results show that the JANET not only outperforms the LSTM, but it competes with one of the best performing models on this dataset. With 1000 units in a single hidden layer the JANET achieves a mean classification accuracy of 95.0% over 10 independent runs with a standard deviation of 0.48%. The benefit of more units is unclear for the LSTM, which has a similar performance with 500 and 128 units to that of the JANET with 128 units. Furthermore, our models were trained on a Nvidia GeForce GTX 1080 GPU, and the largest LSTM we could train was an LSTM with 500 units. Even with a minibatch size of 1, the LSTM with 1000 units was too large to fit into the 8Gb of GPU memory.

Note that the WaveNet performed worse than the JANET on the standard MNIST dataset, achieving a classification accuracy of 98.3% compared to the JANET's 99.0%. The WaveNet results presented here were produced by Chang et al. (2017) using 10 layers of 50 units each. The WaveNet gains additional skip connections with more layers, the JANET gains additional skip connections with more units per layer.

To further ensure that the JANET performs at least as well as the LSTM, we compare the models on two commonly used synthetic tasks for RNN benchmarks. These are known as the copy task and the add task (Arjovsky et al., 2016; Hochreiter and Schmidhuber, 1997; Tallec and Ollivier, 2018).

Copy task Consider 10 categories $\{a_i\}_{i=0}^{9}$. The input takes the form of a $T+20$ length sequence of categories. The first 10 entries, a sequence that needs to be remembered, are sampled uniformly, independently, and with replacement from $\{a_i\}_{i=0}^{7}$. The following $T-1$ entries are a_8, a dummy value. The next single entry is a_9, representing a delimiter, which should indicate to the model that it is now required to reproduce the initial 10 categories in the output sequence. Thus, the target sequence is $T+10$ entries of a_8, followed by the first 10 elements of the input sequence in the same order. The aim is to minimize the average cross entropy of category predictions at each time step of the sequence. This translates to remembering the categorical sequence of length 10 for T time steps. The best that a memoryless model can do on the copy task is to predict at random from among possible characters, yielding a loss of $\frac{10\log 8}{T+20}$ (Arjovsky et al., 2016). The first $T+10$ entries are assumed to be a_8, giving a loss of $-\frac{1}{T+20}(\sum^{T+10} 0 + \sum^{10}\sum^{8}\frac{1}{8}\log\frac{1}{8})$.

Add task Here each input consists of two sequences of length T. The first sequence consists of numbers sampled at random from $\mathcal{U}[0,1]$. The second sequence, with exactly two entries of one and the remainder zero, is an indicator sequence. The first 1 entry is located uniformly at random within the first half of the sequence, and the second is located uniformly at random in the second half of the sequence. The scalar output corresponds to the sum

of the two entries in the first sequence corresponding to the non-zero entries of the second sequence. A naive strategy would be to predict a sum of 1 regardless of the input sequence, which would yield a mean squared error of 0.167, the variance of the sum of two independent uniform distributions (Arjovsky et al., 2016).

We follow Tallec and Ollivier (2018) and use identical hyperparameters for all our models with a single hidden layer of 128 units. The models were trained using Adam with a learning rate of 0.001 and a minibatch size of 50. We illustrate the results for the copy task with $T = 500$, the maximum sequence length used in (Arjovsky et al., 2016), in figure 7.5. For the add task, we present the results for $T = 200$ and $t = 500$ in figure 7.6.

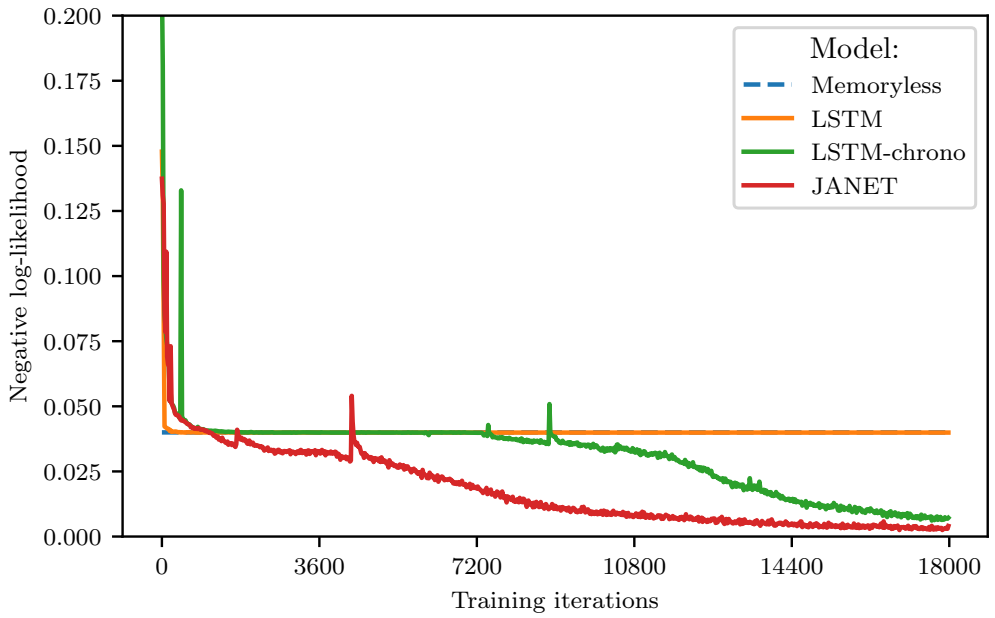


Fig. 7.5 Copy task – comparing the negative log-likelihood (lower is better) of the JANET and the LSTM on the copy task with $T = 500$. The LSTM without chrono initialization performs the same as the memoryless baseline, the same as the results in Arjovsky et al. (2016). Compared to the chrono-initialized LSTM, the JANET converges faster and to a better optimum.

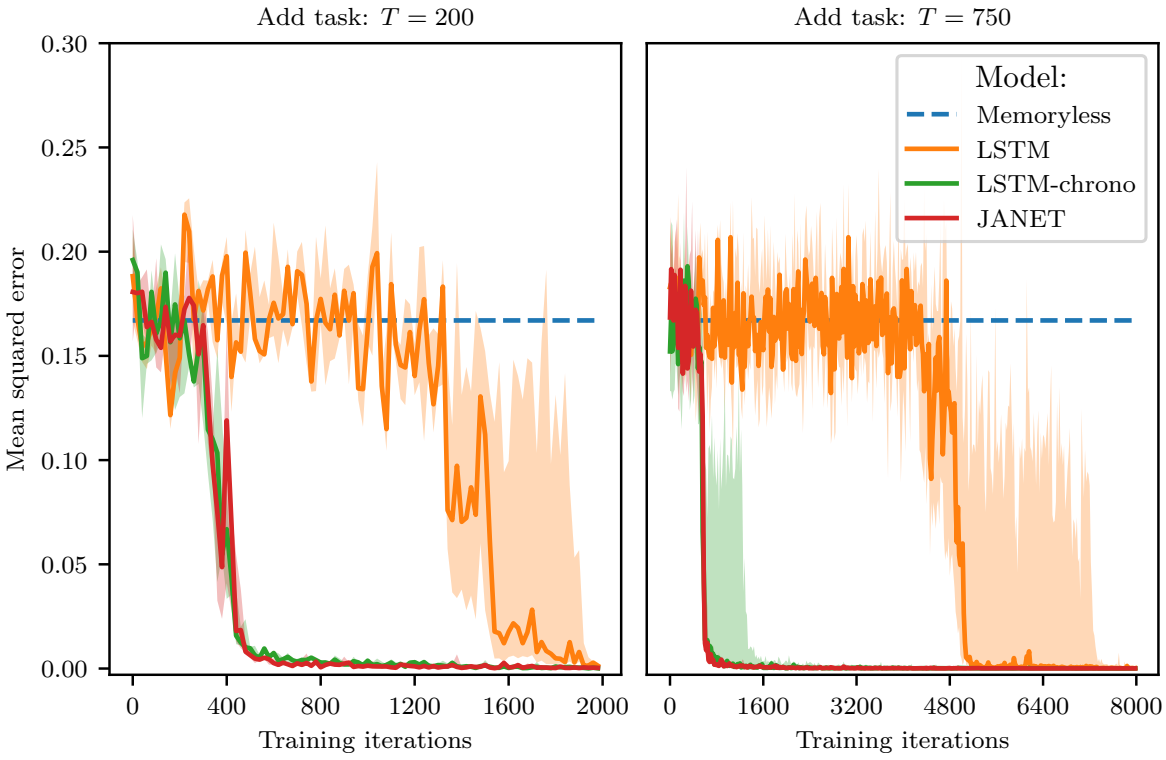


Fig. 7.6 Add task – comparing the mean squared error (lower is better) of the JANET and the LSTM on the add task. The median values of 10 independent runs are shown with the 10^{th} and 90^{th} percentiles shaded for the add task with $T = 200$ (*left*) and $T = 500$ (*right*). Both graphs are displayed with the same y-scale. In both tasks, the standard-initialized LSTM performs the worst. The JANET performs as well as the chrono-initialized LSTM, and slightly better when $T = 750$.

In both tasks, we achieve similar results to those reported by Tallec and Ollivier (2018) and Arjovsky et al. (2016), and the standard-initialized LSTM performs the worst among the three techniques. Compared to the chrono-initialized LSTM, the JANET converges faster and to a better optimum on the copy task. On the add task, the chrono-initialized LSTM and the JANET have a similar performance, with the latter being slightly better for larger T. The copy task is arguably more memory intensive than the add task. This could explain why the JANET, which has built-in long-term memory capability, would outperform the LSTM on the copy task.

In summary, our experiments have shown that the JANET, a simplification of the LSTM, can provide classification accuracies at least as good as the standard LSTM on the various datasets analysed. Bearing the computational savings in mind, the follow-question is whether the sequence-to-sequence model from chapter 4 is able to glean similar benefits from JANET.

7.4 The JANET and the SWAN

To determine whether JANET can be used for the sequence-to-sequence Wasserstein adversarial network (SWAN), we repeat the experiments from section 4.5 with a JANET used for the encoder. In table 7.2 we juxtapose the accuracies obtained for the standard SWAN model and for a SWAN model using a JANET as the encoder. Mean and standard deviation of the accuracies for ten independent runs are shown.

Table 7.2 Unsupervised labelleling accuracies [%] with the JANET

Dataset	LSTM	JANET
Synthetic	86 ± 1.5	86 ± 1.2
Low-resolution MNIST	80 ± 2.9	78 ± 3.2
Raw-neural	63 ± 0.4	63 ± 0.5
Spike-count	76 ± 3.9	75 ± 2.1

The results indicate that we achieve similar accuracies when using the JANET as the encoder of the sequence-to-sequence Wasserstein adversarial network. The results are not as positive as in section 7.3, but this does mean that we can save on computation and not sacrifice on accuracy. We present the reconstruction errors of both models in table 7.3. The reconstruction errors are slightly higher when using the JANET, which could mean that the JANET regularizes the network too strictly, but whether this is good or bad is hard to determine.

Table 7.3 Sequence-to-sequence reconstruction errors with the JANET

Dataset	LSTM	JANET
Synthetic	0.15167 ± 0.08092	0.38878 ± 0.10829
Low-resolution MNIST	0.31454 ± 0.10185	0.57675 ± 0.32116
Raw-neural	6.17025 ± 0.06735	6.66507 ± 0.01711
Spike-count	6.56994 ± 0.17788	7.03793 ± 0.29825

All values are magnified $\times 1000$

7.5 Discussion

In this work, we proposed a simplification of the LSTM that employs only the forget gate. The proposed model was shown to achieve better generalization than the LSTM on synthetic memory tasks and on the MNIST, pMNIST, TBI, and MIT-BIH arrhythmia datasets. Additionally, the model requires half of the number of parameters required by an LSTM and two-thirds of the element-wise multiplications, permitting computational savings.

The unreasonable effectiveness of the proposed model could be attributed to the combination of fewer nonlinearities and chrono initialization. This combination enables skip connections over entries in the input sequence. As described in section 7.2, the skip connections created by the long-range units allow information to flow unimpeded from the elements at the start of the sequence to memory units at the end of the sequence. For the standard LSTM, these skip connections are less apparent and an unimpeded propagation of information is unlikely due to the multiple possible transformations at each time step.

Modern neural networks move towards the use of more linear transformations (Goodfellow et al., 2016, §8.7.5). These make optimization easier by making the model differentiable almost everywhere, and by making these gradients have a significant slope almost everywhere, unlike the sigmoid nonlinearity. Effectively, information is able to flow through many more layers provided that the Jacobian of the linear transformation has reasonable singular values. Linear functions consistently increase in a single direction, so even if the model's output is far from correct, it is clear, simply from computing the gradient, which direction its output should move towards to reduce the objective function. In other words, modern neural networks have been designed so that their *local* gradient information corresponds reasonably well to moving towards a distant solution; a property also induced by skip connections. What this means for the LSTM, is that, although the additional gates should provide it with more flexibility than our model, the highly nonlinear nature of the LSTM makes this flexibility difficult to utilize and so potentially of little use.

With some success, many studies have proposed models more complex than the LSTM. This has made it easy, however, to overlook a simplification that also improves the LSTM. The JANET provides a network that is easier to optimize and therefore achieves better results. Much of this work showcased how important parameter initialization is for neural networks. In future work, improved initialization schemes could allow the standard LSTM to surpass the models described in this study.

Chapter 8

Conclusions

The aim of this thesis was to improve the state of the long short-term memory (LSTM) network for biological applications. Ultimately, this required interpreting the model, which we approached via visualizations, and improving model suitability, which we pursued with bespoke extensions of the LSTM.

Biological applications In chapter 3 we used two new high-resolution medical datasets to show that LSTMs are better than hidden Markov models for analysing medical time series. The suitability of LSTMs for such tasks is supported by their inclusion in the majority of the best performing approaches on a publicly available temporal medical dataset (Clifford et al., 2017). We proceeded to show that the LSTM holds two advantages for such medical datasets:

- Dropout allows them to be used in a Bayesian framework, which enables uncertainty measures of predictions.
- They can be used in an autoencoder, allowing clustering based on sequential features, which in turn could alleviate scant labels in medical datasets.

In chapter 4 we again focused on the problem of limited labelled data – a widespread problem for both continually monitored health and neural data. Here a new dataset with its own problems was introduced and a solution was proposed by means of a sequence-to-sequence Wasserstein adversarial network. Essentially, we demonstrated that regularizing the latent space of a sequence-to-sequence model with Wasserstein adversarial networks enables inference of the actions represented by peripheral neural signals.

Visualizations One of the main concerns for the adoption of the LSTM into mainstream practice is their interpretability. In chapter 5 we sought to explain the decisions of LSTMs

applied to continuous-valued medical time series via various visualization techniques. Among these techniques, learning the optimal input deletion mask produced the most interpretable input salience maps, and we showed for electrocardiograms that many of the identified important features align with medical theory.

Extensions Another important effort in the medical domain is drug discovery and identification of drug targets and interactions. The structure of molecules that constitute such drugs can be represented as a discrete-valued sequence. By again leveraging the Bayesian capability of the LSTM, we showed in chapter 6 that LSTMs enable a version of active learning that reduces the search over discrete-valued sequences – a large space – to a search over the set of possible elements at each time step. This active learning technique makes it possible to efficiently learn the grammar that governs the generation of discrete-valued sequences, such as molecules. An LSTM that has learned the grammar of molecules can guide generative models to generate a higher percentage of valid molecules.

Having demonstrated the utility of the LSTM for biological applications, we sought a hardware efficient version of the LSTM that would make it more enticing for practical adoption (chapter 7). It was found that a forget-gate-only version of the LSTM provides both computational savings and improved generalization compared to the standard LSTM. Moreover, this new architecture, dubbed the JANET, competes with some of the best contemporary models.

To conclude, this thesis goes some way to improving the state of LSTMs for biological data by demonstrating their utility in this domain, showing that prediction uncertainty measures are easy to obtain, proposing a visualization technique for interpretability, and proposing extensions that allow tractability, improved accuracy, and hardware efficiency.

8.1 Future work

This work and other ongoing research have shown that the problems posed by the biological domain can be solved by deep learning models such as the LSTM. Deep learning-based methods, such as LSTMs, now match or surpass the previous state-of-the-art in a diverse array of biological applications, but they do not yet solve all of the problems we encounter (Ching et al., 2018). Some of these are discussed below:

Data limitations Consider for a moment speech recognition, which has also greatly benefited from LSTMs. Since 2009 there have been drastic performance improvements with error rates dropping from more than 20% to less than 6% (Ching et al., 2018) and finally

approaching or exceeding human performance in the last year (Saon et al., 2017; Xiong et al., 2016). Combined with other factors, this performance led to widespread adoption of machine learning in speech recognition through applications such as Siri, Alexa, and Google Translate. To have biological applications reach the same ubiquity, high-quality, large-scale, correctly labelled biological datasets are crucial. The lack of large medical datasets labelled by experts remains a problem, hence, the pursuit of such datasets is desirable.

A more specific dataset limitation is evident from chapter 4, where we introduced a new dataset of peripheral neural signals. Gathering higher quality data of the same kind would enable future investigations of the sequence-to-sequence Wasserstein adversarial network that could answer the following questions:

- How well does the model cope with biological changes at the surgical site over a period of one year?
- How transferable is a trained model between different specimens?

Uncertainty estimates After completing our work on Bayesian LSTMs (section 3.4), Fortunato et al. (2017) proposed an approach to Bayesian recurrent neural networks that uses Bayes by backprop (Blundell et al., 2015; Graves, 2011). Their approach yielded better accuracies and uncertainty measures than the Monte Carlo dropout approach on a language modelling task. Although their approach requires significant adaptations of the standard recurrent neural network (or LSTM), it would be interesting to determine whether the performance benefits extend to biological data.

Interpretability In chapter 5 we showed what the LSTM looks at when classifying continuous valued inputs by visualizing salient features of the input. The aim is to understand the decision-making process of these models. To understand humans, we can query them about their decisions, thus an interesting future venture would be to teach these models to provide explanations for their decisions.

Incomplete validators in biology When actively learning the grammar of discrete-valued sequences (chapter 6), we found that a validator (compiler) having even the slightest inconsistency with the true grammar could be detrimental. As a remedy, a sequence perturbation technique, leveraging existing datasets, was proposed. Another possible solution, which we leave for future work, maintains the active learning aspect by using mutual information to find the most informative perturbations in these existing sequences.

Room for model improvement In chapter 3 our LSTM implementations were outmatched by approaches that combine LSTMs with hand-engineered feature extraction. Automatically learning the most important representations of data could reveal features that have previously been overlooked (Ching et al., 2018; Farabet et al., 2013; Goodfellow et al., 2016; Lipton et al., 2015a; Rajkomar et al., 2018). Finding a deep learning model for sequence analysis that performs better without manual feature extraction thus remains an open problem. Our improvement of the LSTM, the JANET, was conceived during the final stages of this work; it would be interesting to determine whether the JANET brings us closer to automatic feature discovery. Another model that could provide automatic feature discovery is the WaveNet (Van Den Oord et al., 2016), which has recently grown in popularity and has had widespread adoption in practice for natural language processing.

In chapter 7 we showed that the chrono initializer provides the connection structure that enables the JANET to learn long-range dependencies in sequences. For the LSTM the benefits of these initialization schemes are not as clear. In future work, improved initialization schemes for the LSTM could allow it to surpass the performance of the JANET.

The ultimate goal is to develop a deep learning model that can represent and learn the extremely complex long-range structure of real-world sequences. Doing so might require completely novel approaches.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., S. Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Abston, K. C., Pryor, T. A., Haug, P., and Anderson, J. (1997). Inducing practice guidelines from a hospital database. In *Proceedings of the AMIA Annual Fall Symposium*, page 168. American Medical Informatics Association.

Acar, H., Alptekin, G. I., Gelas, J. P., and Ghodous, P. (2016). Beyond CPU: Considering memory power consumption of software. In *5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, pages 1–8.

Adolf, R., Rama, S., Reagen, B., Wei, G., and Brooks, D. (2016). Fathom: Reference workloads for modern deep learning methods. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10.

Ahsan, M. R., Ibrahimy, M. I., and Khalifa, O. O. (2009). EMG signal classification for human computer interaction: A review. *European Journal of Scientific Research*, 33(3):480–501.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. *arXiv:1701.07875 [cs, stat]*.

Arjovsky, M., Shah, A., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K., and Samek, W. (2015). On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PloS one*, 10(7):e0130140.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*.

Balduzzi, D. and Ghifary, M. (2016). Strongly-Typed Recurrent Neural Networks. *PMLR*, pages 1292–1300.

Baskin, I. I., Winkler, D., and Tetko, I. V. (2016). A renaissance of neural networks in drug discovery. *Expert Opinion on Drug Discovery*, 11(8):785–795.

Beal, M. J., Ghahramani, Z., and Rasmussen, C. E. (2001). The Infinite Hidden Markov Model. In *Advances in Neural Information Processing Systems*, pages 577–584. MIT Press.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2013). Advances in optimizing recurrent networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628.

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

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., and others (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, volume 19, page 153. Curran Associates, Inc.

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

Bennett, C. C. and Hauser, K. (2013). Artificial intelligence framework for simulating clinical decision-making: A Markov decision process approach. *Artificial Intelligence in Medicine*, 57(1):9–19.

Berchtold, A. (1999). The double chain markov model. *Communications in Statistics - Theory and Methods*, 28(11):2569–2589.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blei, D. M. and Jordan, M. I. (2006). Variational inference for Dirichlet process mixtures. *Bayesian analysis*, 1(1):121–144.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight Uncertainty in Neural Networks. In *International Conference on Machine Learning*, pages 1613–1622.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating Sentences from a Continuous Space. *arXiv:1511.06349 [cs]*.

Carnevale, F., de Lafuente, V., Romo, R., Barak, O., and Parga, N. (2015). Dynamic Control of Response Criterion in Premotor Cortex during Perceptual Detection under Temporal Uncertainty. *Neuron*, 86(4):1067–1077.

Carreiras, C., Alves, A. P., Lourenço, A., Canento, F., Silva, H., Fred, A., et al. (2015). BioSPPy: Biosignal processing in Python. (Accessed on 03/28/2018).

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. (2017). Dilated Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 76–86. Curran Associates, Inc.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

Chiappa, S. and Bengio, S. (2004). HMM and IOHMM modeling of EEG rhythms for asynchronous BCI systems. In *12th European Symposium on Artificial Neural Networks (ESANN)*, pages 199–204.

Ching, T., Himmelstein, D. S., Beaulieu-Jones, B. K., Kalinin, A. A., Do, B. T., Way, G. P., Ferrero, E., Agapow, P.-M., Zietz, M., Hoffman, M. M., Xie, W., Rosen, G. L., Lengerich, B. J., Israeli, J., Lanchantin, J., Woloszynek, S., Carpenter, A. E., Shrikumar, A., Xu, J., Cofer, E. M., Lavender, C. A., Turaga, S. C., Alexandari, A. M., Lu, Z., Harris, D. J., DeCaprio, D., Qi, Y., Kundaje, A., Peng, Y., Wiley, L. K., Segler, M. H. S., Boca, S. M., Swamidass, S. J., Huang, A., Gitter, A., and Greene, C. S. (2018). Opportunities And Obstacles For Deep Learning In Biology And Medicine. *bioRxiv*, page 142760.

Cho, K. (2015). Natural Language Understanding with Distributed Representation. *arXiv:1511.07916 [cs, stat]*.

Cho, K. (2016). Noisy Parallel Approximate Decoding for Conditional Recurrent Language Model. *arXiv:1605.03835 [cs, stat]*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Choi, E., Bahadori, M. T., Schuetz, A., Stewart, W. F., and Sun, J. (2016a). Doctor AI: Predicting Clinical Events via Recurrent Neural Networks. *JMLR Workshop Conf Proc*, 56:301–318.

Choi, E., Bahadori, M. T., Song, L., Stewart, W. F., and Sun, J. (2017). GRAM: Graph-based Attention Model for Healthcare Representation Learning. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 787–795, New York, NY, USA. ACM.

Choi, E., Bahadori, M. T., Sun, J., Kulas, J., Schuetz, A., and Stewart, W. (2016b). RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512. Curran Associates, Inc.

Christodoulou, C. I. and Pattichis, C. S. (1999). Unsupervised pattern recognition for the classification of EMG signals. *IEEE Transactions on Biomedical Engineering*, 46(2):169–178.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs]*.

Churpek, M. M., Yuen, T. C., Winslow, C., Meltzer, D. O., Kattan, M. W., and Edelson, D. P. (2016). Multicenter Comparison of Machine Learning Methods and Conventional Regression for Predicting Clinical Deterioration on the Wards:. *Critical Care Medicine*, 44(2):368–374.

Clermont, G., Angus, D. C., DiRusso, S. M., Griffin, M., and Linde-Zwirble, W. T. (2001). Predicting hospital mortality for patients in the intensive care unit: A comparison of artificial neural networks with logistic regression models. *Critical care medicine*, 29(2):291–296.

Clifford, G. D., Liu, C., Moody, B., Lehman, L., Silva, I., Li, Q., Johnson, A., and Mark, R. G. (2017). AF Classification from a short single lead ECG recording: The PhysioNet/Computing in Cardiology Challenge 2017. *Computing in Cardiology (CinC)*, 44.

Clifford, G. D., Liu, C., Moody, B., Springer, D., Silva, I., Li, Q., and Mark, R. G. (2016). Classification of normal/abnormal heart sound recordings: The Physionet/Computing in Cardiology Challenge 2016. *Computing in Cardiology (CinC)*, pages 609–612.

Clifford, G. D., Silva, I., Moody, B., Li, Q., Kella, D., Shahin, A., Kooistra, T., Perry, D., and Mark, R. G. (2015). The PhysioNet/Computing in Cardiology Challenge 2015: Reducing false arrhythmia alarms in the ICU. *Computing in Cardiology (CinC)*, pages 273–276.

Clifton, D. A., Niehaus, K. E., Charlton, P., and Colopy, G. W. (2015). Health Informatics via Machine Learning for the Clinical Management of Patients. *Yearbook of medical Informatics*, 20(1):38–43.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. (2016). Recurrent Batch Normalization. In *International Conference on Learning Representations*.

Cooper, G. F., Aliferis, C. F., Ambrosino, R., Aronis, J., Buchanan, B. G., Caruana, R., Fine, M. J., Glymour, C., Gordon, G., Hanusa, B. H., et al. (1997). An evaluation of machine-learning methods for predicting pneumonia mortality. *Artificial intelligence in medicine*, 9(2):107–138.

Czosnyka, M., Smielewski, P., Kirkpatrick, P., Laing, R. J., Menon, D., and Pickard, J. D. (1997). Continuous assessment of the cerebral vasomotor reactivity in head injury. *Neurosurgery*, 41(1):11–19.

Datta, S., Puri, C., Mukherjee, A., Banerjee, R., Choudhury, A. D., Singh, R., Ukil, A., Bandyopadhyay, S., Pal, A., and Khandelwal, S. (2017). Identifying Normal, AF and other Abnormal ECG Rhythms using a Cascaded Binary Classifier. *Computing in Cardiology (CinC)*, 44.

Deming, L., Targ, S., Sauder, N., Almeida, D., and Ye, C. J. (2016). Genetic Architect: Discovering Genomic Structure with Learned Neural Architectures. *arXiv:1605.07156 [cs, stat]*.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Dietterich, T. G. (2002). Machine Learning for Sequential Data: A Review. In *Structural, Syntactic, and Statistical Pattern Recognition*, number 2396 in Lecture Notes in Computer Science, pages 15–30. Springer Berlin Heidelberg.

Dobrowolski, A. P., Wierzbowski, M., and Tomczykiewicz, K. (2012). Multiresolution MUAPs decomposition and SVM-based analysis in the classification of neuromuscular disorders. *Computer Methods and Programs in Biomedicine*, 107(3):393–403.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Economist, T. (2018). Humans may not always grasp why AIs act. Don't panic. *The Economist*.

Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning Hierarchical Features for Scene Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929.

Finkel, J. R., Manning, C. D., and Ng, A. Y. (2006). Solving the Problem of Cascading Errors: Approximate Bayesian Inference for Linguistic Annotation Pipelines. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP, pages 618–626.

Fong, R. C. and Vedaldi, A. (2017). Interpretable Explanations of Black Boxes by Meaningful Perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437.

Fortunato, M., Blundell, C., and Vinyals, O. (2017). Bayesian Recurrent Neural Networks. *arXiv:1704.02798 [cs, stat]*.

Fraccaro, M., Sø nderby, S. r. K., Paquet, U., and Winther, O. (2016). Sequential Neural Models with Stochastic Layers. In *Advances in Neural Information Processing Systems*, pages 2199–2207. Curran Associates, Inc.

Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2016a). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *International Conference on Machine Learning*, pages 1050–1059.

Gal, Y. and Ghahramani, Z. (2016b). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027. Curran Associates, Inc.

Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000.*, volume 3, pages 189–194. IEEE.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471.

Ghassemi, M., Pimentel, M. A., Naumann, T., Brennan, T., Clifton, D. A., Szolovits, P., and Feng, M. (2015). A Multivariate Timeseries Modeling Approach to Severity of Illness Assessment and Forecasting in ICU with Sparse, Heterogeneous Clinical Data. In *AAAI*, pages 446–453.

Glorot, X. and Bengio, Y. (2010a). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Glorot, X. and Bengio, Y. (2010b). Understanding the difficulty of training deep feedforward neural networks. In *PMLR*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323.

Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C., and Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet. *Circulation*, 101(23):e215–e220.

Gómez-Bombarelli, R., Aguilera-Iparraguirre, J., Hirzel, T. D., Duvenaud, D., Maclaurin, D., Blood-Forsythe, M. A., Chae, H. S., Einzinger, M., Ha, D.-G., Wu, T., Markopoulos, G., Jeon, S., Kang, H., Miyazaki, H., Numata, M., Kim, S., Huang, W., Hong, S. I., Baldo, M., Adams, R. P., and Aspuru-Guzik, A. (2016a). Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. *Nature Materials*, 15(10):1120–1127.

Gómez-Bombarelli, R., Duvenaud, D., Hernández-Lobato, J. M., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2016b). Automatic chemical design using a data-driven continuous representation of molecules. *arXiv:1610.02415 [physics]*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680. Curran Associates, Inc.

Goodman, B. and Flaxman, S. (2016). European Union regulations on algorithmic decision-making and a "right to explanation". *arXiv:1606.08813 [cs, stat]*.

Graves, A. (2011). Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356. Curran Associates, Inc.

Graves, A. (2012). Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 5–13. Springer.

Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850 [cs]*.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772.

Graves, A., Jaitly, N., and Mohamed, A. (2013). Hybrid speech recognition with Deep Bidirectional LSTM. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278.

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868.

Greene, B., de Chazal, P., Boylan, G., Connolly, S., and Reilly, R. (2007). Electrocardiogram Based Neonatal Seizure Detection. *IEEE Transactions on Biomedical Engineering*, 54(4):673–682.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015). LSTM: A Search Space Odyssey. *arXiv:1503.04069 [cs]*.

Guimaraes, G. L., Sanchez-Lengeling, B., Outeiral, C., Farias, P. L. C., and Aspuru-Guzik, A. (2017). Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. *arXiv:1705.10843 [cs, stat]*.

Guiza, F., Depreitere, B., Piper, I., Van den Berghe, G., and Meyfroidt, G. (2013). Novel Methods to Predict Increased Intracranial Pressure During Intensive Care and Long-Term Neurologic Outcome After Traumatic Brain Injury: Development and Validation in a Multicenter Dataset. *Critical Care Medicine*, 41(2):554–564. WOS:000314106000040.

Guiza Grandas, F., Ramon, J., and Blockeel, H. (2006). Gaussian processes for prediction in intensive care. In *Gaussian Processes in Practice Workshop*, pages 1–4.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, pages 5767–5777. Curran Associates, Inc.

Gultepe, E., Green, J. P., Nguyen, H., Adams, J., Albertson, T., and Tagkopoulos, I. (2014). From vital signs to clinical outcomes for patients with sepsis: A machine learning basis for a clinical decision support system. *Journal of the American Medical Informatics Association*, 21(2):315–325.

Hamilton, P. S. (2002). Open source ECG analysis. *Computing in Cardiology (CinC)*, pages 101–104.

Han, S., Mao, H., and Dally, W. J. (2015). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations*.

Harutyunyan, H., Khachatrian, H., Kale, D. C., and Galstyan, A. (2017). Multitask Learning and Benchmarking with Clinical Time Series Data. *arXiv:1703.07771 [cs, stat]*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE.

He, Z., Gao, S., Xiao, L., Liu, D., He, H., and Barber, D. (2017). Wider and Deeper, Cheaper and Faster: Tensorized LSTMs for Sequence Learning. In *Advances in Neural Information Processing Systems*, pages 1–11. Curran Associates, Inc.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. In *Advances in Neural Information Processing Systems*, pages 918–926. Curran Associates, Inc.

Hinton, G., Srivastava, N., and Swesky, K. (2012). Neural networks for machine learning: Overview of mini-batch gradient descent. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. (Accessed on 05/03/2016).

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507.

Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. In *Neural Information Processing Systems: Deep Learning Workshop*.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Hong, S., Wu, M., Zhou, Y., Wang, Q., Shang, J., Li, H., and Xie, J. (2017). ENCASE: An ENsemble ClASsifiEr for ECG classification using expert features and deep neural networks. *Computing in Cardiology (CinC)*, pages 1–4.

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. (2011). Bayesian Active Learning for Classification and Preference Learning. *arXiv:1112.5745 [cs, stat]*.

Hu, W. and Tan, Y. (2017). Black-Box Attacks against RNN based Malware Detection Algorithms. *arXiv:1705.08131 [cs]*.

Huuskonen, J. (2000). Estimation of Aqueous Solubility for a Diverse Set of Organic Compounds Based on Molecular Topology. *Journal of Chemical Information and Computer Sciences*, 40(3):773–777.

Imhoff, M. and Fried, R. (2009). The crying wolf: still crying? *Anesthesia & Analgesia*, 108(5):1382–1383.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456.

Irwin, J. J. and Shoichet, B. K. (2005). ZINC - A Free Database of Commercially Available Compounds for Virtual Screening. *Journal of Chemical Information and Modeling*, 45(1):177–182.

Jaeger, H. (2002). *Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the "Echo State Network" Approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn.

Jagannatha, A. N. and Yu, H. (2016). Bidirectional RNN for Medical Event Detection in Electronic Health Records. In *Proceedings of NAACL-HLT*, pages 473–482.

Jang, E., Gu, S., and Poole, B. (2016). Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *IEEE International Conference on Computer Vision*, pages 2146–2153.

Jennett, B. and Bond, M. (1975). Assessment of outcome after severe brain damage: A practical scale. *The Lancet*, 305(7905):480–484.

Johnson, A. E., Pollard, T. J., Shen, L., Li-wei, H. L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. *Scientific data*, 3:160035.

Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python. [Online; accessed 9 April 2018].

Jordan, M. I. (1986). Serial Order: A Parallel Distributed Processing Approach. Technical Report ICS-8604, UC San Diego.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350.

Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and Understanding Recurrent Networks. *arXiv:1506.02078 [cs]*.

Kashani, A. and Barold, S. S. (2005). Significance of QRS Complex Duration in Patients With Heart Failure. *Journal of the American College of Cardiology*, 46(12):2183–2192.

Kendall, A., Badrinarayanan, V., and Cipolla, R. (2015). Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding. *arXiv:1511.02680 [cs]*.

Kendall, A. and Gal, Y. (2017). What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems*, pages 5574–5584. Curran Associates, Inc.

Kim, S., Kim, W., and Park, R. W. (2011). A Comparison of Intensive Care Unit Mortality Prediction Models through the Use of Data Mining Techniques. *Healthcare Informatics Research*, 17(4):232.

Kindermans, P., Schütt, K., Müller, K., and Dähne, S. (2016). Investigating the influence of noise and distractors on the interpretation of neural networks. *arXiv:1611.07270 [cs, stat]*.

Kingma, D. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., and Welling, M. (2014). Semi-supervised Learning with Deep Generative Models. In *Advances in Neural Information Processing Systems*, pages 3581–3589. Curran Associates, Inc.

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*.

Kononenko, I. (2001). Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1):89–109.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. Curran Associates, Inc.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. (2017). Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations. In *International Conference on Learning Representations*.

Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar Variational Autoencoder. In *International Conference on Machine Learning*, pages 1945–1954.

Lanchantin, J., Singh, R., Wang, B., and Qi, Y. (2017). Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. In *Pacific Symposium on Biocomputing*, pages 254–265. World Scientific.

Längkvist, M., Karlsson, L., and Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24.

Lapuerta, P., Azen, S. P., and LaBree, L. (1995). Use of neural networks in predicting the risk of coronary artery disease. *Computers and Biomedical Research*, 28(1):38–52.

Le, Q. V. (2013). Building high-level features using large scale unsupervised learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8595–8598.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *arXiv:1504.00941 [cs]*.

LeCun, Y. A. (1986). Learning Process in an Asymmetric Threshold Network. In *Disordered Systems and Biological Organization*, NATO ASI Series, pages 233–240. Springer, Berlin, Heidelberg.

LeCun, Y. A. (1998). The MNIST Database of Handwritten Digits. *http://yann.lecun.com/exdb/mnist/*.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2016). Visualizing and Understanding Neural Models in NLP. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Lin, T., Horne, B. G., Tino, P., and Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338.

Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015a). A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv:1506.00019 [cs]*.

Lipton, Z. C., Kale, D. C., Elkan, C., and Wetzell, R. (2015b). Learning to Diagnose with LSTM Recurrent Neural Networks. *arXiv:1511.03677 [cs]*.

Liu, C., Springer, D., Li, Q., Moody, B., Juan, R. A., Chorro, F. J., Francisco Castells, Roig, J. M., Silva, I., Johnson, A. E. W., Syed, Z., Schmidt, S. E., Papadaniil, C. D., Hadjileontiadis, L., Naseri, H., Moukadem, A., Dieterlen, A., Brandt, C., Hong Tang, Samieinasab, M., Samieinasab, M. R., Sameni, R., Mark, R. G., and Clifford, G. D. (2016). An open access database for the evaluation of heart sound algorithms. *Physiological Measurement*, 37(12):2181.

Lowe, D. M. (2014). Patent reaction extraction. Available at `https://bitbucket.org/dan2097/patent-reaction-extraction/downloads`.

Lugovaya, T. S. (2005). Biometric human identification based on ECG. PhysioNet.

MacKay, D. J. C. (1992a). Information-Based Objective Functions for Active Data Selection. *Neural Computation*, 4(4):590–604.

MacKay, D. J. C. (1992b). A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*.

Mahendran, A. and Vedaldi, A. (2016). Salient Deconvolutional Networks. In *Computer Vision – ECCV*, pages 120–135. Springer, Cham.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial Autoencoders. *arXiv:1511.05644 [cs]*.

Mani, S., Ozdas, A., Aliferis, C., Varol, H. A., Chen, Q., Carnevale, R., Chen, Y., Romano-Keeler, J., Nian, H., and Weitkamp, J. (2014). Medical decision support using machine learning for early detection of late-onset neonatal sepsis. *Journal of the American Medical Informatics Association*, 21(2):326–336.

Mani, S., Shankle, W. R., Dick, M. B., and Pazzani, M. J. (1999). Two-stage machine learning model for guideline development. *Artificial intelligence in medicine*, 16(1):51–71.

Mante, V., Sussillo, D., Shenoy, K. V., and Newsome, W. T. (2013). Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.

McGlynn, E. A., Asch, S. M., Adams, J., Keesey, J., Hicks, J., DeCristofaro, A., and Kerr, E. A. (2003). The quality of health care delivered to adults in the united states. *New England journal of medicine*, 348(26):2635–2645.

Meehl, P. E. (1986). Causes and effects of my disturbing little book. *Journal of personality assessment*, 50(3):370–375.

Meyfroidt, G., Güiza, F., Ramon, J., and Bruynooghe, M. (2009). Machine learning techniques to examine large patient databases. *Best Practice & Research Clinical Anaesthesiology*, 23(1):127–143.

Miotto, R., Wang, F., Wang, S., Jiang, X., and Dudley, J. T. (2017). Deep learning for healthcare: Review, opportunities and challenges. *Brief Bioinform*.

Moody, G. B. and Mark, R. G. (2001). The impact of the MIT-BIH Arrhythmia Database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50.

Morik, K., Imboff, M., Brockhausen, P., Joachims, T., and Gather, U. (2000). Knowledge discovery and knowledge validation in intensive care. *Artificial Intelligence in Medicine*, 19(3):225–249.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pages 807–814.

Neal, R. M. (1995). *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto.

Neil, D., Pfeiffer, M., and Liu, S. (2016). Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890. Curran Associates, Inc.

Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436.

Ohmann, C., Moustakis, V., Yang, Q., Lang, K., Group, A. A. P. S., et al. (1996). Evaluation of automatic knowledge acquisition techniques in the diagnosis of acute abdominal pain. *Artificial intelligence in medicine*, 8(1):23–36.

Olah, C. (2015). Understanding LSTM networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. (Accessed on 03/28/2018).

Ongenae, F., Van Looy, S., Verstraeten, D., Verplancke, T., Benoit, D., De Turck, F., Dhaene, T., Schrauwen, B., and Decruyenaere, J. (2013). Time series classification for the prediction of dialysis in critically ill patients using echo state networks. *Engineering Applications of Artificial Intelligence*, 26(3):984–996.

Oresko, J. J., Jin, Z., Cheng, J., Huang, S., Sun, Y., Duschl, H., and Cheng, A. C. (2010). A Wearable Smartphone-Based Platform for Real-Time Cardiovascular Disease Detection Via Electrocardiogram Processing. *IEEE Transactions on Information Technology in Biomedicine*, 14(3):734–740.

Ororbia II, A. G., Mikolov, T., and Reitter, D. (2017). Learning Simpler Language Models with the Differential State Framework. *Neural Computation*, 29(12):3327–3352.

Pandarinath, C., O'Shea, D. J., Collins, J., Jozefowicz, R., Stavisky, S. D., Kao, J. C., Trautmann, E. M., Kaufman, M. T., Ryu, S. I., Hochberg, L. R., Henderson, J. M., Shenoy, K. V., Abbott, L. F., and Sussillo, D. (2017). Inferring single-trial neural population dynamics using sequential auto-encoders. *bioRxiv*, page 152884.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *PMLR*, pages 1310–1318.

Pham, V., Bluche, T., Kermorvant, C., and Louradour, J. (2014). Dropout Improves Recurrent Neural Networks for Handwriting Recognition. In *14th International Conference on Frontiers in Handwriting Recognition*, pages 285–290.

Pimentel, M. A. F., Santos, M. D., Springer, D. B., and Clifford, G. D. (2015). Heart beat detection in multimodal physiological data using a hidden semi-Markov model and signal quality indices. *Physiological Measurement*, 36(8):1717.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Potes, C., Parvaneh, S., Rahman, A., and Conroy, B. (2016). Ensemble of feature-based and deep learning-based classifiers for detection of abnormal heart sounds. In *Computing in Cardiology (CinC)*, pages 621–624.

Rajan, K., Harvey, C. D., and Tank, D. W. (2016). Recurrent Network Models of Sequence Generation and Memory. *Neuron*, 90(1):128–142.

Rajkomar, A., Oren, E., Chen, K., Dai, A. M., Hajaj, N., Hardt, M., Liu, P. J., Liu, X., Marcus, J., Sun, M., Sundberg, P., Yee, H., Zhang, K., Zhang, Y., Flores, G., Duggan, G. E., Irvine, J., Le, Q., Litsch, K., Mossin, A., Tansuwan, J., Wang, D., Wexler, J., Wilson, J., Ludwig, D., Volchenboum, S. L., Chou, K., Pearson, M., Madabushi, S., Shah, N. H., Butte, A. J., Howell, M. D., Cui, C., Corrado, G. S., and Dean, J. (2018). Scalable and accurate deep learning with electronic health records. *npj Digital Medicine*, 1(1):18.

Rajpurkar, P., Hannun, A. Y., Haghpanahi, M., Bourn, C., and Ng, A. Y. (2017). Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks. *arXiv:1707.01836 [cs]*.

Ramon, J., Fierens, D., Güiza, F., Meyfroidt, G., Blockeel, H., Bruynooghe, M., and Van Den Berghe, G. (2007). Mining data from intensive care patients. *Advanced Engineering Informatics*, 21(3):243–256.

Reddi, S. J., Kale, S., and Kumar, S. (2018). On the Convergence of Adam and Beyond. In *International Conference on Learning Representations*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Salakhutdinov, R. and Murray, I. (2008). On the Quantitative Analysis of Deep Belief Networks. In *International Conference on Machine Learning*, pages 872–879.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems*, pages 2234–2242. Curran Associates, Inc.

Saon, G., Kurata, G., Sercu, T., Audhkhasi, K., Thomas, S., Dimitriadis, D., Cui, X., Ramabhadran, B., Picheny, M., Lim, L.-L., Roomi, B., and Hall, P. (2017). English Conversational Telephone Speech Recognition by Humans and Machines. *arXiv:1703.02136 [cs]*.

Saria, S., Rajani, A. K., Gould, J., Koller, D., and Penn, A. A. (2010). Integration of Early Physiological Responses Predicts Later Illness Severity in Preterm Infants. *Science Translational Medicine*.

Sboner, A. and Aliferis, C. F. (2005). Modeling clinical judgment and implicit guideline compliance in the diagnosis of melanomas using machine learning. In *AMIA Annual Symposium Proceedings*, pages 664–668. American Medical Informatics Association.

Schaefer, A. J., Bailey, M. D., Shechter, S. M., and Roberts, M. S. (2005). Modeling medical treatment using markov decision processes. In *Operations research and health care*, pages 593–612. Springer.

Semeniuta, S., Severyn, A., and Barth, E. (2016). Recurrent Dropout without Memory Loss. *arXiv:1603.05118 [cs]*.

Sennrich, R., Haddow, B., and Birch, A. (2016). Edinburgh Neural Machine Translation Systems for WMT 16. In *Proceedings of the First Conference on Machine Translation*, volume 2, pages 371–376.

Shen, T. W., Tompkins, W. J., and Hu, Y. H. (2002). One-lead ECG for identity verification. In *Engineering in Medicine and Biology, 2002. 24th Annual Conference and the Annual Fall Meeting of the Biomedical Engineering Society EMBS/BMES Conference, 2002. Proceedings of the Second Joint*, volume 1, pages 62–63.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]*.

Sindhwani, V., Sainath, T., and Kumar, S. (2015). Structured Transforms for Small-Footprint Deep Learning. In *Advances in Neural Information Processing Systems*, pages 3088–3096. Curran Associates, Inc.

Smielewski, P. (2011). Cambridge university: Neurosurgery unit | about ICM+. `http://www.neurosurg.cam.ac.uk/pages/ICM/about.php`. (Accessed on 11/02/2016).

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*.

Springer, D. B., Brennan, T., Ntusi, N., Abdelrahman, H. Y., Zühlke, L. J., Mayosi, B. M., Tarassenko, L., and Clifford, G. D. (2016). Automated signal quality assessment of mobile phone-recorded heart sound signals. *Journal of Medical Engineering & Technology*, 40(7-8):342–355.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Srivastava, N., Mansimov, E., and Salakhutdinov, R. (2015a). Unsupervised learning of video representations using LSTMs. *CoRR, abs/1502.04681*, 2.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015b). Training Very Deep Networks. In *Advances in Neural Information Processing Systems*, pages 2377–2385. Curran Associates, Inc.

Stanculescu, I., Williams, C. K. I., and Freer, Y. (2014). Autoregressive Hidden Markov Models for the Early Detection of Neonatal Sepsis. *IEEE Journal of Biomedical and Health Informatics*, 18(5):1560–1570.

Sussillo, D. and Abbott, L. F. (2009). Generating Coherent Patterns of Activity from Chaotic Neural Networks. *Neuron*, 63(4):544–557.

Sussillo, D., Churchland, M. M., Kaufman, M. T., and Shenoy, K. V. (2015). A neural network that finds a naturalistic solution for the production of muscle activity. *Nature Neuroscience*, 18(7):1025–1033.

Sutskever, I. (2013). *Training recurrent neural networks*. PhD thesis, University of Toronto, Toronto, Ont., Canada.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112. Curran Associates, Inc.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*, volume 1. MIT press Cambridge.

Svátek, V., Ríha, A., Peleska, J., and Rauch, J. (2003). Analysis of guideline compliance–a data mining approach. *Studies in health technology and informatics*, 101:157–161.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Tallec, C. and Ollivier, Y. (2018). Can recurrent neural networks warp time? In *International Conference on Learning Representations*.

Teijeiro, T., García, C. A., Castro, D., and Félix, P. (2017). Arrhythmia classification from the abductive interpretation of short single-lead ecg records. *arXiv preprint arXiv:1711.03892*.

Tong, Y., Frize, M., and Walker, R. (2002). Extending ventilation duration estimations approach from adult to neonatal intensive care patients using artificial neural networks. *IEEE Transactions on Information Technology in Biomedicine*, 6(2):188–191.

Tsien, C. L., Kohane, I. S., and McIntosh, N. (2000). Multiple signal integration by decision tree induction to detect artifacts in the neonatal intensive care unit. *Artificial intelligence in medicine*, 19(3):189–202.

Tu, J. V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231.

Tu, J. V. and Guerriere, M. R. J. (1993). Use of a Neural Network as a Predictive Instrument for Length of Stay in the Intensive Care Unit Following Cardiac Surgery. *Computers and Biomedical Research*, 26(3):220–229.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

Vairavan, S., Eshelman, L., Haider, S., Flower, A., and Seiver, A. (2012). Prediction of mortality in an intensive care unit using logistic regression and a hidden Markov model. *Computing in Cardiology (CinC)*, pages 393–396.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499 [cs]*.

Van Der Maaten, L. (2014). Accelerating t-SNE using tree-based algorithms. *Journal of machine learning research*, 15(1):3221–3245.

Van Der Maaten, L. and Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Van Der Westhuizen, J. (2016). A review of machine learning applied to medical time series. Technical report, Cambridge University Engineering Dept. CUED/F-INFENG/TR.702:0951-9211.

Villani, C. (2009). *Optimal transport: old and new*. Grundlehren der mathematischen Wissenschaften. Springer, Berlin.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders. In *International Conference on Machine Learning*, pages 1096–1103.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164.

Wang, Z., Lin, J., and Wang, Z. (2017). Accelerating Recurrent Neural Networks: A Memory-Efficient Approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2763–2775.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge.

Wattenberg, M., Viégas, F., and Johnson, I. (2016). How to Use t-SNE Effectively. http://distill.pub/2016/misread-tsne/. (Accessed on 03/03/2016).

Weininger, D. (1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36.

Williams, C., Quinn, J., and McIntosh, N. (2006). Factorial Switching Kalman Filters for Condition Monitoring in Neonatal Intensive Care. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Wu, Z. and King, S. (2016). Investigating gated recurrent networks for speech synthesis. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5140–5144.

Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M., and Song, D. (2018). Generating Adversarial Examples with Adversarial Networks. *arXiv:1801.02610 [cs, stat]*.

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. (2016). Achieving Human Parity in Conversational Speech Recognition. *arXiv:1610.05256 [cs]*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *International Conference on Machine Learning*, pages 2048–2057.

Yao, K., Cohn, T., Vylomova, K., Duh, K., and Dyer, C. (2015). Depth-gated Recurrent Neural Networks. *arXiv:1508.03790 [cs]*.

Yousefi, J. and Hamilton-Wright, A. (2014). Characterizing EMG data using machine-learning tools. *Computers in Biology and Medicine*, 51:1–13.

Zabihi, M., Rad, A. B., Katsaggelos, A. K., Kiranyaz, S., Narkilahti, S., and Gabbouj, M. (2017). Detection of Atrial Fibrillation in ECG Hand-held Devices Using a Random Forest Classifier. *Computing in Cardiology (CinC)*, 44:1.

Zabihi, M., Rad, A. B., Kiranyaz, S., Gabbouj, M., and Katsaggelos, A. K. (2016). Heart sound anomaly and quality detection using ensemble of neural networks without segmentation. In *Computing in Cardiology (CinC)*, pages 613–616.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent Neural Network Regularization. *arXiv:1409.2329 [cs]*.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV*, pages 818–833. Springer, Cham.

Zhai, H., Brady, P., Li, Q., Lingren, T., Ni, Y., Wheeler, D. S., and Solti, I. (2014). Developing and evaluating a machine learning based algorithm to predict the need of pediatric intensive care unit transfer for newly hospitalized children. *Resuscitation*, 85(8):1065–1071.

Zhang, J., Mitliagkas, I., and Ré, C. (2017). YellowFin and the Art of Momentum Tuning. *arXiv:1706.03471 [cs, stat]*.

Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R. R., and Bengio, Y. (2016). Architectural Complexity Measures of Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1822–1830. Curran Associates, Inc.

Zhou, G.-B., Wu, J., Zhang, C., and Zhou, Z. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234.

Zhou, J. and Troyanskaya, O. G. (2015). Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934.

Appendix A

Traumatic brain injury dataset variables

Table A.1 Variables recorded for the traumatic brain injury dataset (section 3.2.2)

Temporal	Static
Intracranial pressure	Age
Cerebral perfusion pressure	Gender
Arterial blood pressure	
Heart rate	
Respiratory rate	
Systolic arterial blood pressure	
Diastolic arterial blood pressure	
5s amplitude of arterial blood pressure	
5s amplitude of respiratory rate	
5s respiratory rate pulse	
Minimum of intracranial pressure over 5s	
Maximum of intracranial pressure over 5s	
Peak-to-peak timing of arterial blood pressure	
Peak-to-peak timing of intracranial pressure	
Slow wave intracranial pressure power	
Pressure-reactivity index (Czosnyka et al., 1997)	

Appendix B

Physionet 2017 experiments

This appendix provides results for the various experiments performed to determine the optimal LSTM for the Physionet 2017 challenge, as described in section 3.3. Each table lists the model architectures and their corresponding F1 score on the validation set, which were used to find the optimal value for the hyperparameter indicated by the table title. The default model was trained for 300 epochs using Adam with a learning rate of 0.001, a minibatch size of 100, 0.1 dropout, zero weight decay and a segmentation length of 1000. We start with the segmentation length hyperparameter.

Table B.1 Segmentation length

Model architecture	Segmentation length	Valid F1
1x128	500	0.611
1x512	500	0.613
1x128	1000	0.637
1x512	1000	0.664
1x128	1500	0.632
1x512	1500	0.640

Table B.2 Number of units

Model architecture	Number of units	Valid F1
1x64	64	0.596
1x128	128	0.637
1x256	256	0.624
1x300	300	0.660
1x512	512	0.664

Table B.3 Number of layers

Model architecture	Number of layers	Valid F1
1x128	1	0.637
2x128	2	0.607
2x128, 0.2 dropout	2	0.585
2x128, 0.3 dropout	2	0.591
3x128	3	0.638
2x256	2	0.587

Table B.4 Dropout

Model architecture	Dropout	Valid F1
1x128	0	0.621
1x128	0.05	0.622
1x128	0.1	0.637
1x128	0.5	0.540
2x128	0.1	0.607
2x128	0.2	0.585
2x128	0.3	0.591

Table B.5 Weight decay

Model architecture	Weight decay	Valid F1
1x128, $\sigma^2 = 0.1$ Gaussian augmentation	1e-3	0.639
1x512	1e-3	0.579
1x512	1e-4	0.585
1x512	1e-6	0.582
1x512	0	0.664

Table B.6 Batch normalization (section 3.3.2)

Model architecture	Batch normalization	Valid F1
2x128	On	0.674
3x128	On	0.650
1x256	On	0.587
1x256, TBPTT-1000*	On	0.598
1x512	Off	0.664
1x512	On	0.583
1x512, $\sigma^2 = 0.1$ Gaussian augmentation, TBPTT-1000	On	0.601

* TBPTT-1000 – truncated back-propagation through time with $k_1 = k_2 = 1000$ (section 3.3.3)

Table B.7 Truncated back-propagation through time (section 3.3.3)

Model architecture	$k_1 = k_2$	Valid F1
3x128	1000	0.578
1x256	1000	0.597
1x256, batch normalized	1000	0.598
1x512	250	0.653
1x512	500	0.612
1x512	1000	0.660
1x512, $\sigma^2 = 0.1$ Gaussian augmentation, batch normalized	1000	0.601

Table B.8 Dark knowledge (section 3.3.4)

Model architecture	Teacher size	Valid F1
1x128, $\sigma^2 = 0.1$ Gaussian augmentation	None	0.639
1x128, $\sigma^2 = 0.1$ Gaussian augmentation	1x128	0.642
1x256	None	0.624
1x256	1x256	0.637
1x256, $\sigma^2 = 0.1$ Gaussian augmentation	1x512	0.663
1x300	1x512	0.682
1x512	None	0.664
1x512	1x512	0.690
1x512, $\sigma^2 = 0.1$ Gaussian augmentation	1x1024	0.694

Table B.9 Gaussian data augmentation

Model architecture	σ^2	Valid F1
1x128	None	0.637
1x128, 1e-3 weight decay	0.1	0.639
1x256	0.1	0.630
1x512	None	0.664
1x512	0.05	0.591
1x512	0.1	0.663
1x512	1.0	0.635

Table B.10 Shifted data augmentation

Model architecture	Shifted data augmentation	Valid F1
1x512	off	0.664
1x512	on	0.607