ABSTRACT

WARRIER,SANGEETA Predicting Biomedical Timeseries Data using Neural Networks.(Under the direction of Dr. Kevin Flores).

Multitask learning (MTL) trained neural networks have been shown to improve accuracy over vanilla neural networks in modeling population level time series data. This can be accomplished by assigning the prediction for each individual in the population as a separate task, thereby leveraging the heterogeneity of population level data. We apply the MTL framework to the task of predicting bladder pressure from an external urethral sphincter electromyograph (EUS EMG) signal [46]. Involuntary voiding of the bladder, as a result of spinal cord injury or a neurological disease, can be avoided by applying electrical stimulation at the onset of the bladder contraction that causes voiding. Thus, accurately predicting bladder pressure from EUS EMG would help a enable feedback control system to prevent the onset of involuntary bladder contractions. It is important to develop a model that is generalizable to the population to increase the usability of the model. To this end, we investigate a novel approach by training recurrent neural networks (RNNs) in a multitask setting. We found that the multitask models make more accurate individual level predictions than their single tasking counterparts. We observed that multitask learning trained recurrent neural networks (MTLRNNs) generalized better than single tasking models for predicting bladder pressure and bladder contractions.
We further our examination of MTL networks by applying this modeling framework to a WaveNet inspired CNN model to detect sepsis five hours before its onset. This is motivated by research in [2, 55] that suggests the effectiveness of early treatment therapies to lower the risk of death due to septic shock. This evokes the need to accurately predict the onset of sepsis as early as possible so that the treatment therapies can be administered. We train our MTL WaveNet models and their single tasking counterparts on EHR data from the MIMIC-III [38] database, to predict the occurrence of sepsis five hours into the future. We use the methods and results from [26] to guide the framework of our problem. With the goal of developing a generalizable model, we validate and test MTL and STL WaveNet models on out of sample data to first predict features that help determine if a patient has sepsis. We also train our models to directly predict labels indicating sepsis. We observed that the MTL WaveNet has a higher overall sensitivity of 70% and yields the most generalizable model. Overall, our results suggest that MTL models could be used to leverage heterogeneous population time series data for making individualized predictions.

Predicting Biomedical Timeseries Data using Neural Networks

by
Sangeeta Warrier

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2020

APPROVED BY:

Dr. Hien Tran	Dr. Alen Alexanderian
Dr. Arvind Saibaba	Dr. Kevin Flores Chair of Advisory Committee

BIOGRAPHY

Sangeeta Warrier, born and raised in India, came to the United States for her undergraduate studies, in 2011. During her four years in Mount Holyoke College (Massachusetts), she completed a double major with honors in Mathematics and Physics. After graduating in 2015, she started her graduate program in Applied Mathematics at NC State. While at NCSU, she started working on machine learning research at the Flores Lab and also interned at Teradata Corp and the Institute for Systems Biology.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Kevin Flores for guiding me through and introducing me to such fascinating research. I am very grateful for his honest and persistent feedback on research and writing papers over the last four years.
I would also like to thank my thesis committee, Dr Alen Alexanderian, Dr Arvind Saibaba, Dr. Hien Tran and Dr. Leslie Sombers for their attention to detail and commitment to helping me better my thesis. The manuscript would not be what it is without their feedback.
I would like to acknowledge and thank Dr Erica Rutter for her contribution to the Neurostimulation (Bladder pressure prediction) project. Without her input the project would not have reached its goal.
Graduate school can seem like a strenuous journey with no end in sight. During such times, dependable colleagues and friends are so important. I'm grateful to have had Hiwot Tesfaye, Nick Myers and Mallory McMahon by my side for study sessions, coffee breaks and much more.
I wish to acknowledge the unwavering support, encouragement and love of my parents. I am also grateful to my loving husband who has been my cheerleader throughout.

TABLE OF CONTENTS

LIST OF TABLES . . . . . **vi**

LIST OF FIGURES . . . . . **vii**

Chapter 1 INTRODUCTION **1**
1.1 Motivation and Problem Statement 1
1.2 Thesis Outline 3
1.3 Contributions 4
1.3.1 Predicting bladder pressure using MTL RNNs 4
1.3.2 Early detection of sepsis using MTL CNNs 4

Chapter 2 Neural Networks **5**
2.1 Overview 5
2.1.1 Neurons and the perceptron 6
2.1.2 Multilayer Perceptrons (MLPs) 9
2.1.3 Activation Functions 14
2.1.4 Loss functions 16
2.2 Closer look at specific NNs: Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM) networks 20
2.2.1 Overview 20
2.2.2 Forward propagation and back propagation 21
2.2.3 LSTMs 23
2.3 Closer look at specific NNs: Convolutional Neural Networks (CNNs) 26
2.3.1 Overview 26
2.3.2 Architecture 28
2.4 Regularization and Multitask Learning 30
2.4.1 Overview 30
2.4.2 Bias-variance trade-off 31
2.4.3 Validation sets 33
2.4.4 Regularization methods 34
2.4.5 Modeling Inter-individual and intra-individual variation 36
2.4.6 Multitask Learning 42

Chapter 3 Estimating bladder pressure and bladder contraction in rats **48**
3.1 Introduction 48
3.1.1 Previous Work 49
3.2 Methods 52
3.2.1 Data 52
3.2.2 Neural Network Architectures and Training 52
3.2.3 Multitasking Algorithm 53
3.3 Results 55
3.3.1 Bladder Pressure Predictions 55
3.3.2 Bladder Contraction Predictions 58

3.4 Discussion 61

Chapter 4 Neural Networks for the early detection of sepsis 62
4.1 Introduction 62
4.1.1 Previous Work 63
4.2 Methods 66
4.2.1 Data 66
4.2.2 Neural Network Architectures and Training 67
4.3 Results 71
4.3.1 Feature Predictions 71
4.3.2 Sepsis Prediction 73
4.4 Discussion 77

Chapter 5 Conclusion and Future Work 79
5.1 Predicting Bladder Pressure and Bladder Contraction in rats 79
5.2 Early detection of Sepsis 81

BIBLIOGRAPHY 82

LIST OF TABLES

Table 3.1 Recall and PPV for all the four models trained on the first three post PGE-2 trials and tested on the same 59
Table 3.2 Recall and precision for all the four models trained on the first three post PGE-2 trials and tested on the remaining trials 59

Table 4.1 Features used to train models to predict sepsis 68
Table 4.2 Sensitivity and Specificity for the sepsis predictions from all models, tested on all the patients in the test cohort, using feature predictions and the sepsis definition from section 4.2 75
Table 4.3 Sensitivity and Specificity for the sepsis predictions from all the "sepsis patient models," tested on all the patients in the test cohort using feature predictions and the sepsis definition from section 4.2 76
Table 4.4 Sensitivity and Specificity for the WaveNet models trained on data from the entire cohort to predict labels. The models were tested on all 20 patients in the test cohort 77
Table 4.5 Sensitivity and Specificity for both of the WaveNet "sepsis patient models" trained to predict labels, tested on all the patients in the test cohort 77

LIST OF FIGURES

Figure 2.1 Model of a perceptron. [25] 7
Figure 2.2 Illustration of the hyperplane as decision boundary for a 2-D, binary pattern classification problem. [25] 9
Figure 2.3 Fully connected FFN with two hidden layer. [25] 10
Figure 2.4 Signal Flow diagram for the j^{th} neuron in a larger MLP. [25] 12
Figure 2.5 a)Sigmoid function b) Tanh function 15
Figure 2.6 a)ReLu b)Leaky ReLu 16
Figure 2.7 A recurrent neural network and the unfolding in time of the computation involved in its forward computation. [60] 21
Figure 2.8 LSTM network [60] 23
Figure 2.9 Convolution operation [20] 27
Figure 2.10 An example of a convolutional layer, with the input volume in red, the volume of neurons in the first convolutional layer in blue. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially [42] 28
Figure 2.11 On the far right, in the green, is the filter of size 3. On the far left is a neuron with a receptive field of $F = 3$ striding one step at a time, $S = 1$, over the input of size $W = 5$ which is in yellow. The input has been padded with zeros on either side, with $P = 1$. This yields an output size of $\frac{(W-F+2P)}{S+1)}$, which is 5, in this case. The center panel shows the same neuron with the same receptive field size, but a longer stride of $S = 2$, yielding an output of size 3. [42] 29
Figure 2.12 As the capacity (model complexity) increases (x-axis), bias (dotted) tends to increase and the variance (dashed) tends to increase resulting in a U-shaped curve for generalization error (bold curve). [20] 33
Figure 2.13 NLME framework indicating shared and individual parameters, for n^i measurements for $m+1$ individuals, where $z^i_{t_j}$ is the response, Γ is a shared parameter and b^i is the vector of individual specific parameters. 40
Figure 2.14 Single tasking models of four tasks [9] 43
Figure 2.15 Multitasking model of four tasks [9] 43
Figure 2.16 Hard parameter sharing network [73] 44
Figure 2.17 Soft parameter sharing network[73] 44

Figure 3.1 (a) Schematic of multi-task learning recurrent neural network. The shared layers are in grey and the individual-specific layers are in color. (b) Network architectures for each model. “Shared” indicates the shared layers in the multitask models. 53
Figure 3.2 End-to-end representation of how the MTL training occurs 54

Figure 3.3 (a) Intra-individual mean squared error and (b) Inter-individual mean squared error for all the four models. (** indicates P < 0.005, and * indicates P< 0.05 Mann-Whitney U-test.) 56
Figure 3.4 Example of bladder pressure predictions +/- 30 second around bladder contraction activity, for trial 10 in rat 3 using (a) the Vanilla NN model or (b) the MTLNN model. 57
Figure 3.5 Out of sample correlation coefficients between the actual and predicted bladder pressure for the MTLNN model (panel a) and for the MTLRNN model (panel b). 57
Figure 3.6 Out of sample sensitivities for the MTLRNN model (panel a) and the MTLNN model (panel b). 60
Figure 3.7 Out of sample precision for the MTLRNN model (panel a) and for the MTLNN model (panel b). 60

Figure 4.1 Schematic of multitask learning WaveNet architecture. The shared layers are in red and the individual-specific layers are in green. . . . 70
Figure 4.2 End-to-end representation of how the MTL training occurs 71
Figure 4.3 Example of feature predictions 5 hours ahead for the heart rate, fluid intake, lactic acid and systolic blood pressure, for the patient with id 10315, using the STL model trained on data from the patient with id 154 72
Figure 4.4 Example of feature predictions 5 hours ahead for the heart rate, fluid intake, lactic acid and systolic blood pressure, for patient with id 10315, using the MTL model trained on data from patient with id 154 73
Figure 4.5 Example of feature predictions 5 hours ahead for the heart rate, fluid intake, lactic acid and systolic blood pressure, for the patient with id 10315, using the baseline, Vanilla NN model trained on data from a patient with id 154 74
Figure 4.6 Inter-individual mean squared error for the four features (heart rate, lactic acid, fluid uptake and systolic blood pressure), for both the models (** indicates P < 0.005 and * indicates P< 0.05 Mann-Whitney U-test). Note, error bars in the SBP and Lactic acid panels were too small for the scale. 75

CHAPTER

1

INTRODUCTION

1.1 Motivation and Problem Statement

> Precision medicine is a term used to describe individualized treatment that encompasses the use of new diagnostics and therapeutics, targeted to the needs of a patient based on his/her own genetic, biomarker, phenotypic, or psychosocial characteristics. [33].

The challenges that the field of precision medicine faces include the current treatment techniques that try to understand the disease instead of the patient, large population sizes, and variation within populations along with abundant, partially unused data. However, given current advances in applied machine learning [71, 41, 75, 26] the electronic health records (EHR) data for each patient can potentially be used to facilitate the delivery of individualized treatments. This evokes the need to understand similarities among patients and within a population. We attempt to understand a facet of these challenges in this thesis

by using machine learning and multitask learning (MTL) trained neural networks (MTL NN), to make predictions with biomedical time series data.

One of the applications we study is the early detection of sepsis. We formulate a predictive model to determine how, if ever, ICU patients succumb to septic shock. Early detection of sepsis is especially necessary to administer early treatment therapies to reduce the risk of death due to septic shock [2, 55]. We use EHR data from the MIMIC 3 [38] database to do so. We seek to understand the sepsis problem by applying MTL neural networks (inclduing vanilla neural networks and convolutional neural networks (CNNS)) to model our data. Our MTL algorithm is novel in that it is trained to consider the prediction for a specific individual as a single task, while also leveraging the time-series nature of the training data through a CNN, specifically the WaveNet [61] architecture. By training every individual in tandem, the MTL architecture forces a neural network to learn a representation that accounts for differences in each individual while at the same time being generalizable to other individuals. This encapsulates the idea of trying to capture similarities amongst patients, to deliver a treatment that is individual and best suited for a specific patient.

Another application we examine is the prediction of bladder pressure and bladder contractions from measurable signal data around the pelvic nerve. Bladder overactivity and urinary incontinence caused by a neurological disease or a spinal cord injury can be treated by applying electrical stimulation to the pelvic nerve at the onset of a bladder contraction. Accurately predicting bladder pressure and bladder contractions from measurable signal data, such as external urethral sphincter electromyograph (EUS EMG), is essential to the development of stimulation treatment [46, 75]. We apply similar MTL trained neural networks (recurrent neural networks (RNNs), in particular, to predict bladder pressure from the EUS EMG signal data. We trained and tested four neural network architectures to compare and evaluate their ability to predict bladder pressure, given the EUS EMG data, for each of the fifteen rats. We expect the MTL trained RNNs to learn a representation that captures the time-series behavior of the population data, to make accurate predictions for individual rats. The data that we use has few samples (12 rats) but a very dense feature matrix. We recognized the challenge of multimodality of the data and attempted to remedy this by implementing our MTL RNN algorithm. The methods we propose should translate well to

other medical datasets with high heterogeneity, low sample size, and large dimensionality of the data.

Areas where advancements in precision medicine could be applied are detailed in [33]. These include applications where sequential data need to be modeled, such as examining a person's microbiome to determine risk for cancer, cardiovascular disease and, cystic fibrosis. DNA sequences are dense sources of information and have been modeled by RNNS [24] and CNNs [90, 56]. Our methods of training RNN and CNN architectures using MTL to overcome the challenge of population heterogeneity could be applied to studying individuals' microbiomes. Given the positive results, we observed with applying MTL to the early detection of sepsis, another interesting opportunity to leverage this training scheme is to predict premature labor or preeclampsia, much like automated defibrillators detect and interrupt cardiac arrhythmias. Blood glucose prediction to detect type-II diabetes is also an important problem that could be better understood by applying deep learning methods. Current research [69, 89] implements RNNs and also regularized learning methods [54]. The abundance of EHR and medical data available, the prevalence of the disease, and the inability of current methods to overcome the challenge of high population variance make this problem ideal to be investigated via MTL trained deep learning networks.

1.2 Thesis Outline

To illustrate our work in this thesis, we begin by introducing neural networks and explain how they function in Chapter 2. We discuss different architectures, including RNNs and CNNs, and their applications in sections 2.2 and 2.4. We also detail the principles of multitask learning and how they can be applied to the problems we are studying, in section 2.4. In Chapter 3, we present our work and results in predicting bladder pressure using multitask neural networks. Chapter 4 delineates our efforts in developing an MTL trained deep learning neural network model for the early detection of sepsis. Finally, in Chapter 5, we present our conclusions and propose avenues for future work.

1.3 Contributions

In this section we summarize the key contributions of this thesis.

1.3.1 Predicting bladder pressure using MTL RNNs

Previous work to predict bladder pressure from neurostimulation data includes using LASSO models [75], neural networks [71] and LSTMS [31]. While providing relatively accurate intra-individual results, authors suggest there is room to improve inter-individual predictions. We train RNNs using a multitask learning algorithm, with individuals as tasks to predict bladder pressure. This is the first time multi-task learning has been applied, along with RNNs to learn from and make predictions with neurostimulation data. Additonally, we also use RNNs (single task learning (STL) and multitask learning trained) to predict bladder contractions from bladder pressure, for the first time, in this thesis.

1.3.2 Early detection of sepsis using MTL CNNs

The early detection of sepsis has been a relevant subject of study for researchers, owing to the complexity of and high mortality rate caused by the condition. The state of the art methods include the LiSep model [17] which uses LSTMs to predict sepsis. Other models include the InSight model [8] and the TrewScore model [26]. However, we take a precision medicine approach to study the early detection of sepsis and implement multitask learning neural networks to predict sepsis in ICU patients, i.e., we leverage population level EHR data to make individual predictions. We use a novel combination of 1-D convolutions in a WaveNet architecture and a multitask learning with individuals as tasks training scheme, to predict sepsis for individual patients. This is the first time CNNs and MTL trained CNNs have been used for sepsis prediction.

CHAPTER

2

NEURAL NETWORKS

This chapter provides a basic overview of neural networks, including their architectures, and their applications.

2.1 Overview

Neural networks are a set of algorithms developed with the intention of modeling biomedical learning, i.e., modeling the way the human brain learns and are hence, initially named artificial neural networks (ANNs) [20]. While ANNs and their successors are not realistic representations of the human brain, the inherent goal was to develop an "intelligent" system that is capable of learning from experience and making decisions, given new instances of data. To this end, it is useful to reverse engineer the computational steps of the brain and replicate its functionality [20].
More generally, a neural network (NN) attempts to model a task of interest, in a way similar

to the brain, for instance, by learning from other examples of the same task. A NN consists of building blocks, called neurons that are all connected and learn a given task in tandem. In other words,

> *a neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. The procedure used to perform the learning task is called a learning algorithm, the function of which is to modify the synaptic weights of the neuron connections to achieve the desired outcome of the network.* [25].

Neural networks are similar to statistical models, performing similar tasks of regression and classification. NNs serve as non -parametric alternatives for more complicated problems [62]. In this context, we use non-parametric to mean that neural networks are non-mechanistic. The parameters and weights are not attributed physical meaning. Being able to model complex nonlinear relationships is not the only advantage of NNs. NNs are a group of models that work under the supervised learning paradigm, in that a model is provided with training data with sample input and output pairs. The network learns a response as close as possible to the desired response by modifying the synaptic weights at each neuron by learning from each sample input-output pair. [25]. NNs also exhibit adaptivity because they can change the synaptic weights according to changes in the environment, i.e., changes in the relationship between the input and the output. The learning process of neural networks is discussed in the following sections.

2.1.1 Neurons and the perceptron

NNs are employed with the task of finding a function (either linear or nonlinear), f that maps a set of inputs to a set of outputs. There are a variety of input-output relationships to be determined, and a myriad of neural network architectures to do just that. All neural nets can be grouped under a class called Deep Neural Networks (DNNs), where depth is a measure of the size of the network. DNNs include a set of different architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), each with their ability to model different mathematical relationships. A subset of DNNs is feedforward

neural networks (FFNs). They are called feedforward because the flow of information is always forward, i.e., from the function being evaluated through the inputs, through intermediate computations used to define f, and finally to the outputs. [20]. The building blocks of all neural networks and FFNs are neurons. Before we delve into the detailed functioning of FFNs, consider a perceptron: the most basic neural network, consisting of only a single neuron with adjustable synaptic weights and bias Figure 2.1.

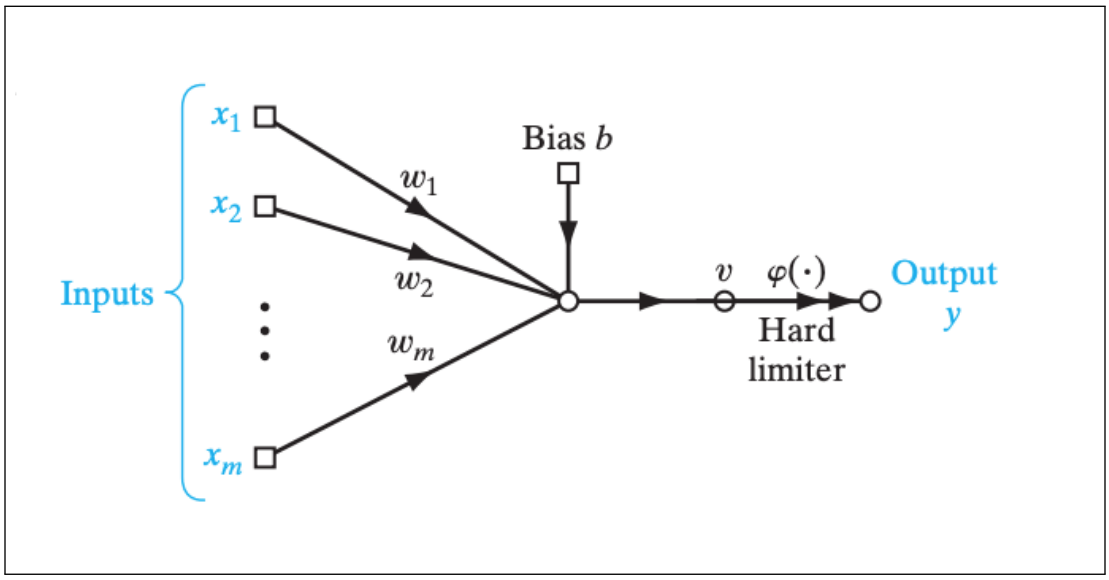


Figure 2.1 Model of a perceptron. [25]

The perceptron was developed to be used as a linear classifier, i.e., to classify patterns that are linearly separable. In other words, the perceptron attempts to find the equation of a hyperplane that separates the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$. The forward pass of information through a perceptron can be summarized by

$$\begin{aligned} v^i = \sum_0^m w_{ik} x_i + b_i \quad \text{and} \\ y_i = \varphi(v_i). \end{aligned} \tag{2.1}$$

In the equations in (2.1), v_i is the result of weighted linear combination of the $m+1$ input signals into the i^{th} neuron added to an external bias, b_i. There are $m+1$ synaptic weights for each i^{th} neuron, each denoted by w_{ik}. The final output y_i is given by applying the activation function $\varphi(.)$, a hard limiter in this case, to v_i. The perceptron, being a linear classifier, classifies the set of inputs $\mathbf{x} = \{x_0, .., x_m\}$, into one of class $\mathscr{C}_1$, if $y_k = 1$ or $\mathscr{C}_2$ if $y_k = -1$, since the dividing hyperplane is defined by $\sum_0^m w_{ik} x_i + b_i = 0$ as in Figure 2.2.
After computing the output and making the classification, the neuron output is compared to the desired output and the synaptic weights are adjusted to ensure that in the next iteration, the outputs are closer to the desired output. An error signal given by $e(N)$, is computed which in this case is determining if the classification is correct or not. The weights are then adjusted by an amount determined by η. This "learning algorithm" of the perceptron was developed by Rosenblatt (1958,1962) [25] and is called the perceptron convergence algorithm as outlined in Algorithm 1.
The first two steps of the error correction algorithm are the "forward pass" equations and the third step of updating the weights is the "backward pass" where the node receives information from the error signal about the amount of adjustment to the synaptic weights.

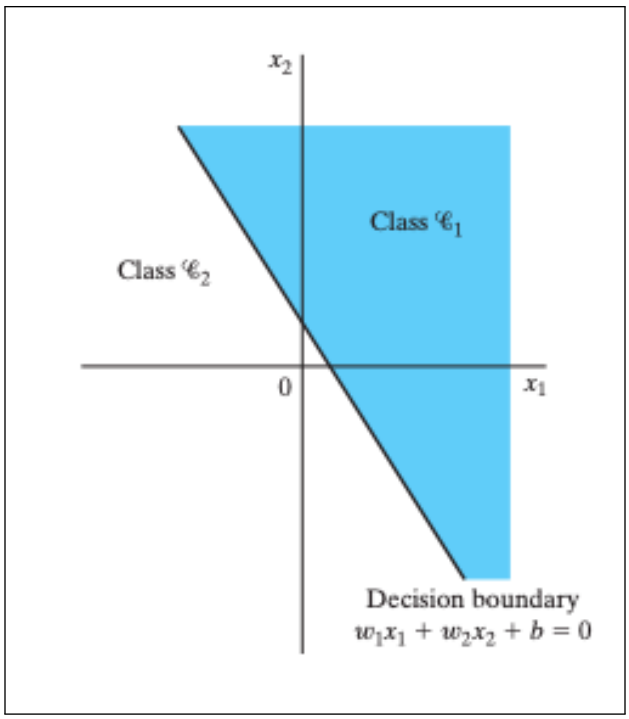


Figure 2.2 Illustration of the hyperplane as decision boundary for a 2-D, binary pattern classification problem. [25]

2.1.2 Multilayer Perceptrons (MLPs)

While the perceptron and its convergence algorithm (1) were breakthroughs, the main use case for a perceptron is when the patterns to be classified are linearly separable. However, for non-linearly separated classes, we have multilayer perceptrons (MLP), which are some of the more commonly used NNs and they differ from perceptrons in that they have additional hidden layers of neurons between the input and the output as in Figure 2.3.

As we see in Figure 2.3, the MLP is stimulated by $m+1$-dimensional input signals, $\mathbf{x}(N)$ at the N^{th} epoch. Training a network involves the forward and backward pass of data through all the neurons. One such pass is an epoch. The input signals are weighted, linearly combined and passed through an activation function before being fed to each of the hidden layer neurons. This process continues until the output neuron, at which, an output $y_j(N)$ is computed to compare to the desired output $z_j(N)$. (The j subscript refers to the output

Algorithm 1: Perceptron Convergence Algorithm

Variable and parameter definitions:
$x(N) = [+1, x_1(N), ..., x_m(N)]$, the (m+1)-dimensional input at the N^{th} epoch,
$w(N) = [b, w_1(N), ..., w_m(N)]$, is the (m+1)-dimensional weight vector,
b is the bias,
$y(N)$ is the actual output at the N^{th} epoch,
$d(N)$ is the desired output at the N^{th} epoch and,
η is the learning rate The error correction algorithm is as follows:

- Initialize w
- $w(0) = 0$, and perform the following steps for each epoch or time step $n = 1, 2, ..$
- Compute actual output $y(N) = \text{sgn}[\mathbf{w}^{\mathbf{T}}(N)\mathbf{x}(N)]$
- Update w to get $\mathbf{w}(n+1) = \eta[\mathbf{z}(N) - \mathbf{y}(N)]\mathbf{x}(N)$, where $\mathbf{z}(N) - \mathbf{y}(N)$ is the error signal.
- Repeat the second and third steps after for each incremental time step.

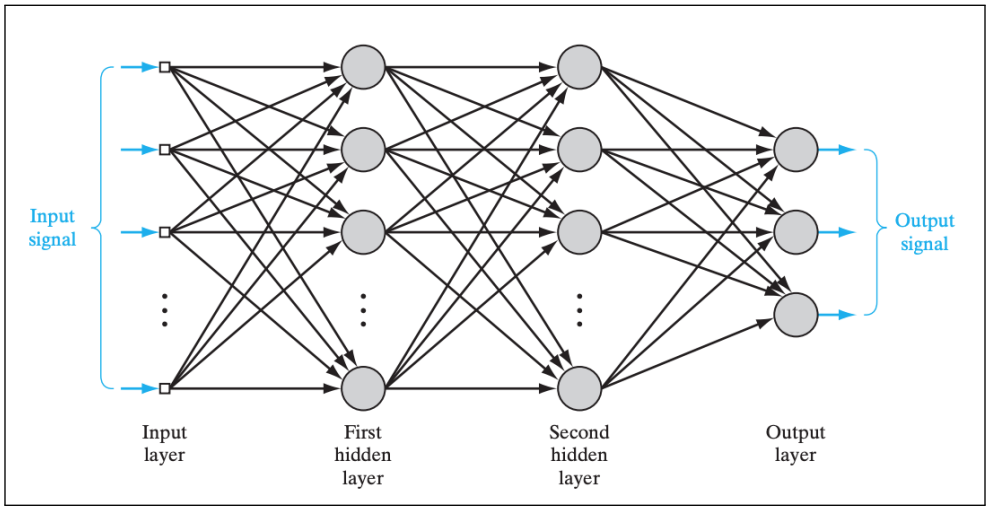


Figure 2.3 Fully connected FFN with two hidden layer. [25]

neuron being the j^{th} neuron.) The next pass is a backward pass of the error signal $e_j(N) = z_j(N) - y_j(N)$, from the output nodes, back through the hidden layer nodes to the input nodes where the goal is to modify the synaptic weights. This process can be repeated on an

example-to-example basis, where at each epoch, the network is presented with a training sample $T(N) = \mathbf{x}(N), \mathbf{z}(N)$, where each $\mathbf{x}(j)$ is $m+1$ dimensional and $\mathbf{z} = (z_1, \cdots, z_p)$ because there are p training samples.

For the perceptron, the weights were adapted according to the convergence theorem. The learning process of an MLP is similar but gradient-based. The error energy of the j^{th} neuron, at a specific instant is defined as

$$\mathscr{E}_j(N) = \frac{1}{2} e_j^2(N), \tag{2.2}$$

and that of the whole network is

$$\mathscr{E}(N) = \sum_{j \in P} \frac{1}{2} e_j^2(N), \tag{2.3}$$

where P is the set of all neurons. The average error energy is simply the sum over all training examples divided by the number of training examples p as in

$$\mathscr{E}_{\text{avg}}(N) = \frac{1}{2p} \sum_{k=1}^{p} \sum_{j \in P} e_{kj}^2(N). \tag{2.4}$$

where $e_{kj}^2(N)$ is the squared error signal of the k^{th} training example at the j^{th} neuron and N^{th} epoch.

We define the total error energy of the network also to highlight that it is a function of the synaptic weights since each error signal is a function of the weights. We adapt the weights intending to minimize the total error energy and do so using a gradient descent algorithm, often called the backpropagation algorithm. Consider a single neuron in Figure 2.4, which is part of the larger MLP in Figure 2.3.

The forward pass is defined by the equations in (2.1), where the output at neuron j, at the N^{th} epoch, is given by

$$\begin{aligned} y_j(N) &= \varphi_j(v_j(N)), \\ v_j(N) &= \sum_{i}^{m} w_{ji}(N) y_i(N) \end{aligned} \tag{2.5}$$

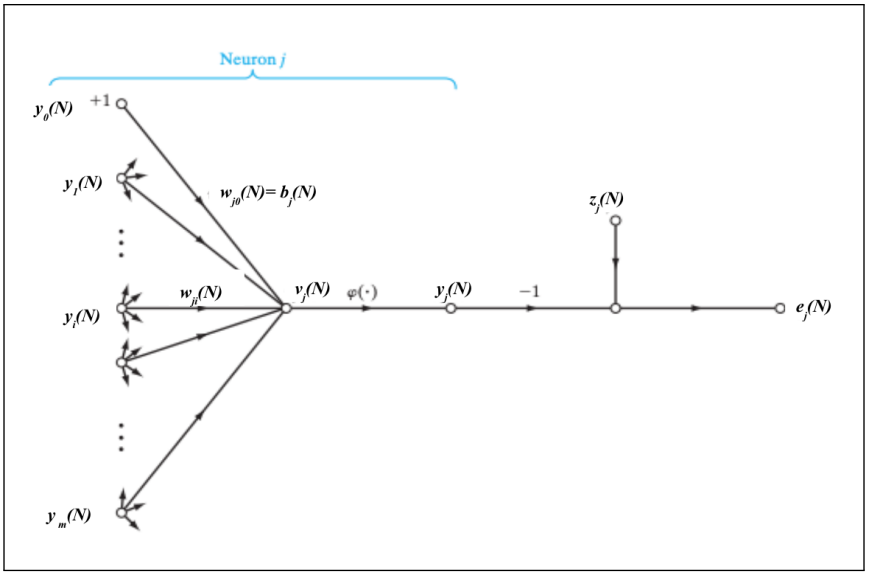


Figure 2.4 Signal Flow diagram for the j^{th} neuron in a larger MLP. [25]

where $y_i(N)$ is the output from the previous neuron i. We need to now compute the weight correction $\Delta w_{ji(N)}$, applied to each synaptic weight. This correction is proportional to the partial derivative $\frac{\mathscr{E}(N)}{w_{ji}(N)}$. Given the total error of the network to be defined as given in Equation (2.3) we have,

$$\frac{\partial \mathscr{E}(N)}{\partial w_{ji}(N)} = \frac{\partial \mathscr{E}(N)}{\partial e_j(N)} \frac{\partial y_j(N)}{\partial v_j(N)} \frac{\partial v_j(N)}{\partial w_{ji}(N)}. \tag{2.6}$$

Next, we differentiate both sides of equation (2.4) w.r.t $e_j(n)$to get,

$$\frac{\partial \mathscr{E}(N)}{\partial e_j(N)} = e_j(N). \tag{2.7}$$

We also have,

$$\frac{\partial e_j(N)}{\partial y_j(N)} = -1. \tag{2.8}$$

When we take the partial derivative of $y_j(N)$ w.r.t $v_j(N)$ from Figure 2.4, we get

$$\frac{\partial y_j(N)}{\partial v_j(N)} = \varphi_j^{'}(v_j(N)). \tag{2.9}$$

Finally we also compute

$$\frac{\partial v_j(N)}{\partial w_{ji}(N)} = y_i(N). \tag{2.10}$$

Putting together all the derivatives from equations (2.7)-(2.9), with equation (2.6) we get

$$\frac{\partial \mathscr{E}(N)}{\partial w_{ji}} = -e_j(N)\varphi_j'(v_j(N))y_i(N). \tag{2.11}$$

From equation (2.10), we get the correction $\Delta w_{ji}(N)$ to be,

$$\Delta w_{ji}(N) = -\eta \frac{\partial \mathscr{E}(N)}{\partial w_{ji}(N)}, \tag{2.12}$$

where η is the learning rate parameter. We use the negative sign because we are moving in the direction that causes the weights to decrease which allows for the fastest decrease in the error with respect to the weights, i.e. in the direction of the negative gradient. We can rewrite equation (2.12) as

$$\Delta w_{ji}(N) = -\eta \delta_j(N) y_i(N), \tag{2.13}$$

where $\delta_j(N)$ is the local gradient and is computed as

$$\begin{aligned} \delta_j &= \frac{\partial \mathscr{E}(N)}{\partial v_j(n)} \\ &= \frac{\partial \mathscr{E}(N)}{\partial e_j(N)} \frac{\partial e_j(N)}{\partial y_j(N)} \frac{\partial y_j(N)}{\partial v_j(N)} \\ &= e_j(N)\varphi_j'(v_j(N)) \end{aligned} \tag{2.14}$$

The local gradient will point in the direction of the required change in weights. The backward pass starts at the output neuron, by passing the error signals along through the network computing the local gradient at each node. Factors that influence the convergence of the back propagation algorithm to a solution of optimal weights, $\mathbf{w}^*$, include the learning rate and activation functions. It is important to monitor the convergence of the algorithm to identify when the iterations can be stopped. The back propagation algorithm involves

computing the gradient of the overall error w.r.t weights are every neuron, so the algorithm can be stopped when the synaptic weights equal $\mathbf{w}^*$, and indicate that the error has reached a global minimum, or in practice, it is sufficiently small.

2.1.3 Activation Functions

We see from section 2.1.2, that activation function φ is applied at each neuron before the weighted signal is pushed to the next node. What separates the perceptron from more complex architectures is the presence of hidden layers and nonlinear activation functions. The function of the hidden layers is to help understand the complex relationship between the features and output. The type of activation function is important to achieve this. Activation functions play a large role in determining the linearity and value of the output, as detailed in section 2.1.1. They "activate" the weighted combinations of inputs at each layer before feeding them to the next layer. The choice of an activation function depends on a few factors, including non-linearity, range, and differentiability, since we need to compute the gradient at every node. [**Goodfellow-et-al-2016**, 25]
The perceptron applies a linear activation function. However, to leverage the ability of neural networks to model nonlinear relationships, a nonlinear activation function can be used. As per the universal approximation theorem [29], if a 2-layer neural network has a nonlinear activation function, the network acts as a universal function approximator. To that end, there are a few popular activation functions and they are plotted in Figure 2.5 and Figure 2.6.
The sigmoid function in panel a) of Figure 2.4, can be represented mathematically as

$$\sigma(x) = 1/(1+e^{-x}), \tag{2.15}$$

and is often used in logistic regression and binary classification problems. They essentially force an input to be between 0 and 1. Some of the problems networks with sigmoid activated layers face, include saturated gradients and non-zero centered outputs. When the neurons activation saturates at 0 or 1, the gradient at these regions is almost zero. This gradient is multiplied with the gradient at the output gate. Therefore, if the local gradient is very small,

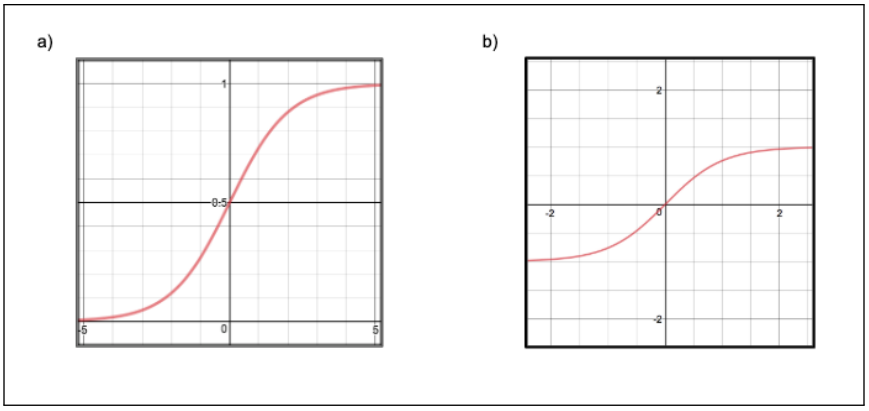


Figure 2.5 a)Sigmoid function b) Tanh function

it will effectively “kill” the gradient and almost no signal will flow through the neuron to its weights and recursively to its data [42]. Sigmids result in the gradients having a zig-zag pattern causing instability in the network [58]. Another activation function is the Tanh function, in panel b) of Figure 2.5. The tanh function is a shifted sigmoid function where

$$\tanh(x) = 2\sigma(2x) - 1, \tag{2.16}$$

with zero-centered outputs squashed between -1 and 1. While the it still has the saturated gradient problem, the unstable gradient problem that the sigmoid has is essentially taken care of.

The Rectified Linear Unit (ReLU) function, Figure 2.6 panel a), has become popular recently. The output of the ReLU function $f(x) = \max(0, x)$. It has a non-saturating gradient, which converges quicker than the sigmoid or tanh functions. Another advantage of the ReLU function is that it is relatively, computationally inexpensive compared to other activation functions. Advantages aside, the ReLU function has the issue of "killing neurons." Since any input less than zero is squashed to zero when passing through the ReLU function, neurons can be turned off or killed during training, especially if the learning rate is too high. However, the Leaky ReLU in panel b of Figure 2.6 is an attempt to overcome this challenge. The Leaky RelU essentially performs

$$f(x) = \mathbb{1}(x < 0)(\alpha x) + \mathbb{1}(x \geq 0)(x), \tag{2.17}$$

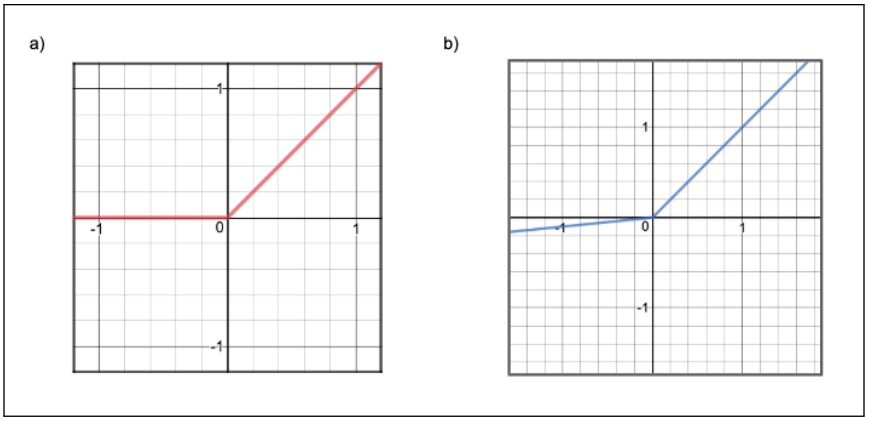


Figure 2.6 a)ReLu b)Leaky ReLu

where α is a small constant and $\mathbb{1}$ is the indicator function.
The choice of an activation function also comes down to the where in the network it is applied and the response variable. For instance, while it is discouraged to use the sigmoid function owing to its issues described in this section, it can be used in the output layer of a network designed for a binary classification problem. Similarly, the final layer of a linear regression problem would require a linear activation function.

2.1.4 Loss functions

Sections 2.1.2 and 2.1.3 detailed the learning process of a perceptron and a MLP. For their learning algorithms, the start of the backward pass involved comparing the network output $\mathbf{y}y(N)$ with the desired output $\mathbf{z}(N)$. For the perceptron algorithm, the comparison was if the network correctly classified the input or not and the weight correction applied was based on the same. For the MLP, the weight correction was applied according to the backpropagation algorithm where the optimal weight vector, $\mathbf{w}^*$ minimized the average error energy of the network. This average error energy can be thought of as a loss or cost function, and its definition is integral to the backpropagation algorithm since the algorithm focuses on minimizing this loss.
There is a multitude of loss functions, and the choice of a loss function for a network depends on the type of features that form the input and the type of output, along with the

relationship between the input and the output. The difference between linear models and neural networks is that the more complicated loss functions for the neural networks don't have closed-form solutions [20]. This lack of closed solutions evokes the need for a gradient descent based training as described in section 2.1.3, where applying weight corrections proportional to the gradient of the loss function, yields the most optimal weight vector with the minimal cost.

A general cost function for neural networks is computed using maximum likelihood. This can be generalized using the Maximum Likelihood Estimation (MLE) principle. Consider a set of $m+1$ input data, $\mathbb{X} = \{x^{(1)}, \cdots, x^{(\mathrm{m})}\}$, drawn independently from the unknown data distribution $\mathrm{p}_{\mathrm{data}}$. We also have $p_{\mathrm{model}}(\mathbf{x};\mathbf{w})$, a parametric probability distribution that maps any x to a real number that estimates $\mathrm{p}_{\mathrm{data}}$, via the parameters $\mathbf{w}$. Then, the ML estimator $\mathbf{w}$ is defined as

$$\mathbf{w}_{\mathrm{ML}} = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^{m} p_{\mathrm{model}}(\mathbf{x}^{(\mathbf{i})};\mathbf{w}). \tag{2.18}$$

Since the product of probabilities in equation (2.18) is prone to underflow, we take the log on both sides which yields a sum as in

$$\mathbf{w}_{\mathrm{ML}} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\mathbf{x}^{(\mathbf{i})};\mathbf{w}). \tag{2.19}$$

Equation (2.19) can be further re-scaled by dividing the sum by the number of training examples $m+1$ to get as an expectation where

$$\mathbf{w}_{\mathrm{ML}} = \underset{\mathbf{w}}{\operatorname{argmax}}\, \mathbb{E}_{x\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\mathbf{x};\mathbf{w}). \tag{2.20}$$

The ML estimation minimizes the dissimilarity between the empirical distribution $\hat{p}_{\mathrm{model}}$ defined by the training data and the model distribution $\mathrm{p}_{\mathrm{model}}$ [20]. The goal is to find the set of parameters $\mathbf{w}_{ML}$ that minimizes the dissimilarity between the model distribution and the empirical data distribution. So, maximizing the likelihood as in equation (2.19) is the equivalent of minimizing $-\mathbb{E}_{x\sim\hat{p}_{\mathrm{data}}}[\log \mathrm{p}_{\mathrm{model}}(\mathbf{x})]$. We can also use the maximum likelihood estimator to estimate a conditional probability $P(\mathbf{y}|\mathbf{x};\mathbf{w})$, and by extension the maximum likelihood estimator is

$$\mathbf{w}_{ML} = \underset{\mathrm{w}}{\operatorname{argmax}} P(\mathbf{Y}|\mathbf{X};\mathbf{w}) \quad \text{and,} \tag{2.21}$$

$$\mathbf{w}_{ML} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(\mathbf{y}^i|(\mathbf{x}^{(\mathbf{i})};\mathbf{w}). \tag{2.22}$$

So, the generalized cost function following the idea of the MLE estimating the conditional distribution in equation (2.21), is defined as

$$J(\mathbf{w}_{\mathrm{ML}}) = -\mathbb{E}_{x,y\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\mathbf{y}|\mathbf{x};\mathbf{w}). \tag{2.23}$$

where $p_{\mathrm{model}}(\mathbf{y}|\mathbf{x};\mathbf{w})$ is the conditional probability of the response Y, given the features X and parameters $\mathbf{w}$. The exact form of the cost function $J(\mathbf{w})$ changes with the task at hand. For instance for a regression problem, where $p_{\mathrm{model}}(\mathbf{y}|\mathbf{x})$ is defined by the normal distribution as in $p_{\mathrm{model}}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y};\hat{y}(\mathbf{x};\mathbf{w}),\mathbf{I})$, MLE is the minimizer of the mean squared difference or the mean squared error (MSE) between the desired output y and the predicted output $\hat{y}$ yielded by the function $\hat{y}(\mathbf{x};\mathbf{w})$. We define the MSE of the training stage as,

$$\mathrm{MSE}_{\mathrm{train}} = \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2. \tag{2.24}$$

The advantage of deriving the cost function from the ML estimate is that it can be used for all models even if its form changes. The specific form of the cost function derived from the MLE depends on the probability distribution, $p(\mathbf{y}|\mathbf{x})$ being estimated, which in turn depends on the output units of the network. For a regression problem, we often use a gradient-based method to minimize the MSE, as shown in equation (2.24). Consider the problem of estimating Bernoulli Output distributions, using sigmoid units. This problem is similar to a binary classification task of classifying inputs into class 0 or 1.

To estimate the Bernoulli distribution of y conditioned over x, we need to predict $P(\mathrm{y}=1|]\mathrm{x})$. This means the network needs to predict a number in the interval $[0,1]$. We will represent the output of a network, before passing it through the activation function as a weighted sum of the outputs of the hidden layers, $\mathbf{h}$, like, $\mathbf{w}^T\mathbf{h}+\mathbf{b}$. This means the output from the neuron can be stated as $\varphi(\mathbf{w}^T\mathbf{h}+\mathbf{b})$, where $\varphi(.)$ is the activation function. To allow for gradient descent to train the network effectively, we also need the activation function to

be differentiable and not have its gradient w.r.t the parameters to plummet to zero as with a linear function. An appropriate architecture for the output unit would be the sigmoid unit σ, where the first step is to compute $z = \mathbf{w}^T\mathbf{h} + \mathbf{b}$, and pass z through the sigmoid activation function to get the prediction $\hat{y}$ as

$$\hat{y} = \sigma(z). \tag{2.25}$$

The result of passing $\mathbf{w}^T\mathbf{h}+\mathbf{b}$ through the sigmoid activation function results in a probability. To derive the full loss function, let us first consider an unnormalized probability distribution $\tilde{P}(y)$ that does not sum to 1. If we assume that the unnormalized log probabilities are linear in y and z, we can exponentiate them to get the unnormalized probabilities that can be normalized by division by a constant. [20]. We have

$$\begin{aligned} &\log\tilde{P}(y) = yz, \\ \implies &\tilde{P}(y) = \exp(yz), \\ \implies &P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1}\exp(yz)}, \\ \implies &P(y) = \sigma((2y-1)z). \end{aligned} \tag{2.26}$$

Finally, the loss function for the maximum likelihood learning of a Bernoulli distribution via a sigmoid also called the binary cross-entropy loss is,

$$\begin{aligned} J(\mathbf{w}) &= -\log P(y|x), \\ &= -\log\sigma((2y-1)z). \end{aligned} \tag{2.27}$$

There are more loss functions specific to the output units for different tasks. This thesis discusses only a variation of the MSE loss and the sigmoid loss discussed in this section.

2.2 Closer look at specific NNs: Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM) networks

2.2.1 Overview

RNNs and LSTMs are a set of neural networks used to model sequential data like time series data. To model sequential data, parameter sharing is important to carry forward certain information from the beginning of the sequence to the end. For instance, consider the task of classifying whether a patient is at risk for a heart attack or not. This classification problem might involve using information like the heart rate and blood pressure or other vital signs from the past day or so to determine the risk. If we are using a NN to make this prediction, the NN needs to be able to use information from many hours prior to the current time. The advent of the RNN was brought about by developments in the concept parameter sharing [74], explained in [20].

> *Parameter sharing makes it possible to extend and apply the model to examples of different forms and generalize across them. Such sharing is particularly important when a specific piece of information can occur at multiple positions of a sequence.*

A traditional MLP has separate sets of parameters for each layer, so after passing through a layer and the activation function, some information is not carried forward. The RNN tackles this problem by having hidden units that are functions of preceding layers and help carry forward information. Consider an RNN as a way to model a dynamical system where

$$\mathbf{s}_{(t)} = f(\mathbf{s}_{(t-1)}; \mathbf{w}) \tag{2.28}$$

In equation (2.28), $\mathbf{s}_{(t)}$ is called the state of the system and the the equation is recurrent since the definition of s at any time t depends on the previous timestep $t-1$. If we also add an external input signal $\mathbf{x}_{(t)}$, we get

$$\mathbf{s}_{(t)} = f(\mathbf{s}_{(t-1)}; \mathbf{x}_{(t)}; \mathbf{w}). \tag{2.29}$$

Most RNNs are defined using

$$\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}; \mathbf{x}_{(t)}; \mathbf{w}), \tag{2.30}$$

where $\mathbf{h}_{(t)}$ is the hidden state at any given time, and is a function of the input signals at that time and the hidden state from the previous time. This is a recursive definition and when unfolded, looks like Figure 2.7. After this overview of the fundamentals of a recurrent

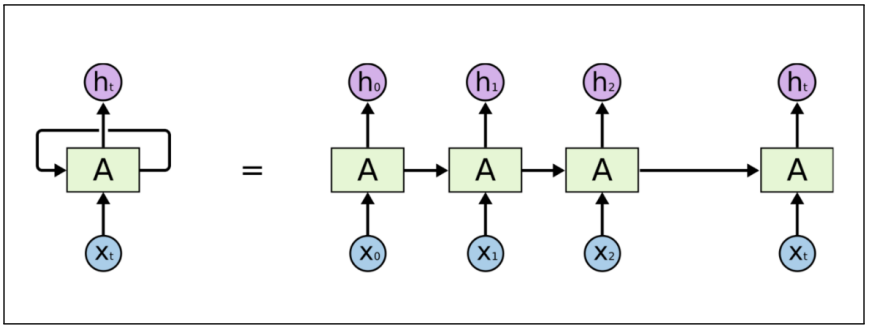


Figure 2.7 A recurrent neural network and the unfolding in time of the computation involved in its forward computation. [60]

network, we will discuss the architecture and the training of an RNN (with a single output at the end), along with its drawbacks in the following sections.

2.2.2 Forward propagation and back propagation

To derive the forward and backward propagation, we consider units in Figure 2.7. At each time point, the hidden state ($h(t)$ in the equations, but $s(t)$ in figure 2.7) is a function of the linear combination of the weighted input at that time step and the weighted hidden state of the previous time step. We have

$$\mathbf{a}_{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}_{(t-1)+\mathbf{U}}\mathbf{x}_{(t)}), \tag{2.31}$$

$$\mathbf{h}_{(t)} = \tanh(\mathbf{a}_{(t)}), \tag{2.32}$$

$$\mathbf{o}_{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}_{(t)} \quad \text{and} \tag{2.33}$$

$$\hat{\mathbf{y}} = \text{sigmoid}(\mathbf{o}_{(t)}). \tag{2.34}$$

From the equations (2.31)-(2.33), **b** and **c** are biases applied at different stages. The conventional activation function for hidden units is the sigmoid or tanh activation function. RNNs propagate the input signals and hidden units through the entire network until the output neuron, to yield a single output $\hat{y}^{(t)}$, which is a function of the weighted hidden state passed through an activation function. Equations (2.31)-(2.33), are repeated through all time steps, and the output or prediction in equation (2.34) is computed at the final time step. The activation function for the output neuron depends on the problem and could be a linear function for a regression problem or a sigmoid function for a classification task. The total cross-entropy loss for the sequence over all time t is defined by

$$J(\mathbf{w}) = -\sum_{t} y_t \log(\hat{y}_t) \tag{2.35}$$

where $\hat{y}_t$ is the prediction and y_t is the desired output at time t and **w** is the set of all the parameters in the network. For the total cross-entropy loss, we calculate the loss at each time step and sum all the losses for the final time step. Backpropagation happens recursively at each node, at every time step where gradients of the loss at that time w.r.t the parameters **U**, **V**and, **W** are taken to make the appropriate weight corrections. For an RNN with only a single output at the end, the forward propagation is the same, except there is an output $o^{(t)}$ only at the last time step, and the backpropagation is also similar to that of an FFN. [20]

The presence of the hidden state is integral to the way an RNN models sequences. It is equivalent to that of memory, and it propagates that memory through the network to yield a prediction at time t, which is a function of what happened a few time steps ago. Notice that unlike regular MLPs, the parameters **U**, **V**and, **W** are shared across all time-steps.

While training MLPs, at every time step, the parameters are different, and time step t does not "see" parameters at time step preceding it. The shared parameters not only make the training process less computationally expensive but also allow the hidden state to have a "memory" since they are trained with different inputs at each time step. [25]
While RNNs were developed to model sequential data, they suffer from the exploding and vanishing gradient problem as first discussed in [27], [6] and further in [64]. They are unable to model long-term temporal dependencies because some long-term components that either grow or vanish exponentially [64]. The gradients are computed recursively, and gradients of long-term components are a result of a long product that are either prone to overflow or underflow. Solutions to the vanishing gradient problem include gradient clipping [64] or modified architectures designed to deal with this problem like gated recurrent units (GRUs) [41]. One such architecture implemented and described in this thesis is called the Long Short Term Memory network (LSTM).

2.2.3 LSTMs

LSTMs developed first by [28], were designed to overcome the vanishing gradient problem. They do this via their gated structure.

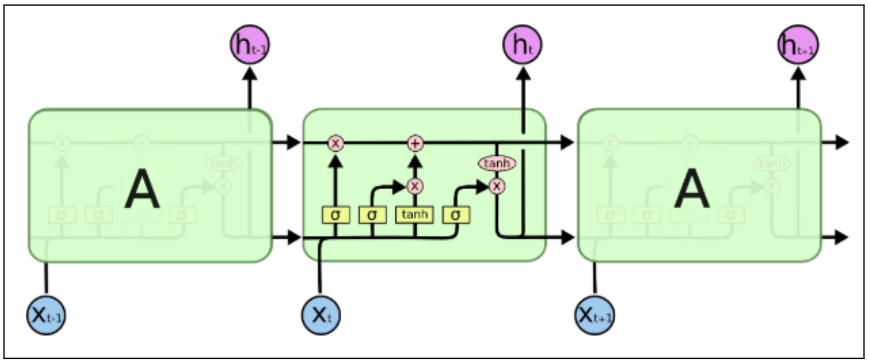


Figure 2.8 LSTM network [60]

In an LSTM network, the LSTM unit is repeated layer after layer to create a network. Each LSTM unit consists of gates, including and input gate, a forget gate and the output gate. The

yellow boxes in the LSTM unit in Figure 2.8 refers to a neural network layer, i.e. a neuron with some activation function and the pink boxes indicate piecewise operations. To understand how LSTMs function, we will develop the forward propagation equations.

The backbone of an LSTM unit is the cell state, which runs straight down the entire chain to allow for specific information to pass through unchanged and is denoted as $\mathbf{C}_t$. The cell state carries forward relevant information from the previous hidden state $\mathbf{h}_{t-1}$ and cell state $\mathbf{C}_{t-2}$ and the input signal $\mathbf{x}_t$ at that time step.
We start by determining what information must be excluded from the cell state. This is facilitated by the "forget gate layer" in the LSTM unit which utilizes the sigmoid functions. The sigmoid function takes on a value of 0 or 1, and the sigmoid layer is applied to determine if a signal should or should not pass forward. Consider

$$F_t = \sigma(\mathbf{W}_f[h_{t-1}, x_t] + b_f), \tag{2.36}$$

where $\mathbf{W}_f$ is the parameter matrix for the forget layer f_t, and b_f is bias for the forget layer. Now, the next step is to determine which information is carried forward through the network via the cell state. This is called the input gate and has two components i_t, which is a sigmoid layer that decides what values are updated and $\tilde{C}_t$ which is a tanh layer that creates potential new values to be added. We have,

$$\begin{aligned} i_t &= \sigma(\mathbf{W}_i[h_{t-1}, x_t] + b_i), \\ \tilde{C}_t &= \tanh(\mathbf{W}_i[h_{t-1}, x_t] + b_c). \end{aligned} \tag{2.37}$$

The update to the cell state is made by combining the forget gate, the old cell state from the previous time step and the input gate components like

$$C_t = F_t \times C_{t-1} + i_t \times \tilde{C}_t. \tag{2.38}$$

The final step is the output gate. This, too, has two components o_t and the hidden state for that time step h_t. The output at that step is the result of a weighted linear combination of the old hidden state and current input signal passed through a sigmoid function like

$$o_t = \sigma(\mathbf{W}_o[h_{t-1}, x_t] + b_0). \tag{2.39}$$

The next component is the hidden state at this time step which is passed on to the rest of the network. This is combined with the cell state at the current time step to get

$$h_t = o_t \times \tanh(C_t). \tag{2.40}$$

The division of labor by each gate allows updates to the information carried forward and a solution to the vanishing gradient problem. LSTMs can model long-term temporal dependencies via the constant updates made to the cell state, which carries forward only the relevant information. As a result, they have found their use in language modeling where the information at the beginning of the sentence is relevant to the end [82]. In this thesis, we explore how LSTMs can be used to make predictions with biomedical time series data and contribute to existing research [50, 17, 31, 65, 48].

2.3 Closer look at specific NNs: Convolutional Neural Networks (CNNs)

2.3.1 Overview

Convolutional neural networks (CNNs) [47] are a kind of neural network that processes and model data that have a grid-like topology like 2-D images or even time series data [20]. They are similar to FFNs, except performing some specialized operations like convolutions instead of general matrix multiplication, and pooling that allows them to process specific kinds of data.

A convolution is similar to a weighted average and can be written as

$$y_{jt} = (x * w)_{(t)} \tag{2.41}$$

where w is a weighting function. The first argument x is the input, w is called the kernel and the output y_{jt} at the j^{th} neuron, is called a feature map. The convolution operation leverages three important ideas: sparse interactions, parameter sharing and equivariant representations [20]. We will explain this further. Consider an example of a convolution operation in Figure 2.9.

The computations within most neural networks involve matrix multiplications of parameter matrices at one node with those at other nodes to represent the interaction among those nodes. For a fully connected NN, this means several thousands of parameters making the network's training computationally expensive. In Figure 2.9 we see a filter, in the form of a 2×2 matrix working over a 4×4 input matrix. This means, not every parameter in the filter interacts with every entry in the input matrix. So, a convolution creates sparse connections or sparse weights. This allows the network to efficiently describe complicated interactions between many variables, with fewer parameters than traditional networks [20].

Another factor that improves the efficiency and performance of CNNs includes parameter sharing, where each member of the kernel is used for all portions of the input. This allows the network to learn a single set of parameters for the entire input. Finally, the convolution operation is also equivariant, which implies that if the input changes, the output changes, in the same way, [20].

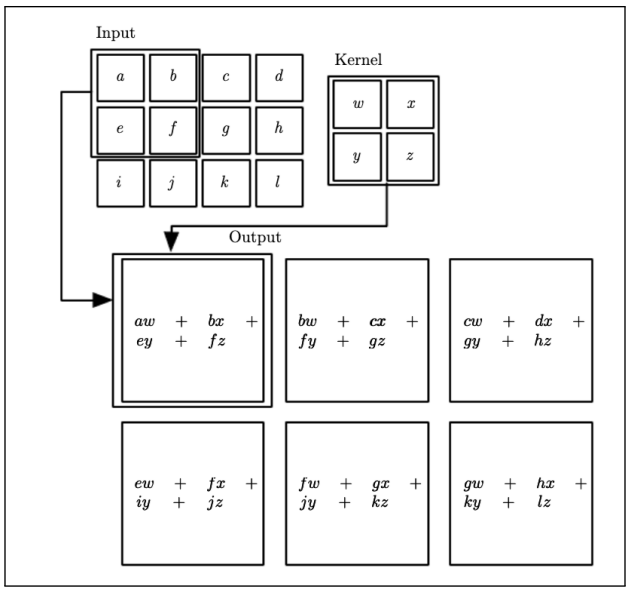


Figure 2.9 Convolution operation [20]

The next step after a convolution, the result is passed through a nonlinear activation, usually the ReLu function. This is followed by the detector stage, where a pooling function over nearby outputs replaces single outputs. Max pooling is a common pooling function, where the maximum of a certain number of neighboring outputs replaces the output itself, and this continues through the network. So, we can think of pooling as a summarizing operation that summarizes over a given area. This allows the network to estimate important features efficiently, like detecting edges in an image. Pooling can also be used to downsample, where the result after a pooling layer is a smaller-sized output. The convolution, nonlinear activation, and pooling layers are repeated to create a large network. The final layer before the output is a fully connected layer where the neurons have full connections to all activations in the previous layer. This result is passed through the output neuron activation function to provide an output, which can either be a sigmoid function for a classification task or a linear function for a regression task.

2.3.2 Architecture

In this section, we consider all the operations of the CNN in more detail while also discussing important hyper-parameters that determine the architecture of the network.
First, let us consider how the convolutional layer of the CNN works with high-dimensional input data like images. As we discussed in the previous section, a CNN implements local spatial connectivity by ensuring that every neuron is connected to a small portion of the input data. This is unlike a fully connected layer, as shown in Figure 2.10. The size of this connectivity is determined by the hyper-parameter called the receptive field and the depth of the input. The receptive field is as deep as the input.

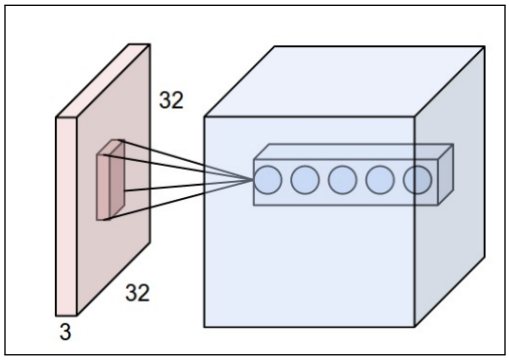


Figure 2.10 An example of a convolutional layer, with the input volume in red, the volume of neurons in the first convolutional layer in blue. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially [42]

The next piece to consider is the arrangement of the neurons in the output neuron of the convolutional layer of the CNN. Since a CNN processes multi-dimensional input data, the output volume can also be multi-dimensional. However, we want to leverage the CNN's ability to learn and output relevant information in an efficient manner. The hyper-parameters that determine the size of the output volume include the depth and stride. The depth corresponds to the number of filters one wants to use to glean different kinds of information like edges and pops of color. The next hyper-parameter is the stride of the filter and that decides how the filter moves, i.e., does it move one step at a time or does it skip a few steps.

Finally, we also have the option of padding the input with zeros. This allows us to control the size of the output. Suppose the result of striding a filter over the input is an output of a smaller size than the input. If we want the output and input sizes to match, we can use zero padding to remedy the discrepancy in sizes. This, along with how a filter is strided over the input, is illustrated in Figure 2.11.

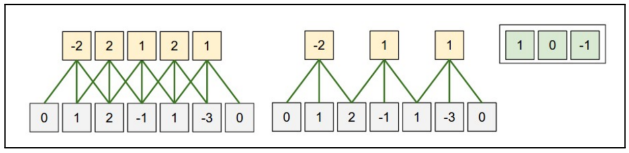


Figure 2.11 On the far right, in the green, is the filter of size 3. On the far left is a neuron with a receptive field of $F = 3$ striding one step at a time, $S = 1$, over the input of size $W = 5$ which is in yellow. The input has been padded with zeros on either side, with $P = 1$. This yields an output size of $\frac{(W-F+2P)}{S+1)}$, which is 5, in this case. The center panel shows the same neuron with the same receptive field size, but a longer stride of $S = 2$, yielding an output of size 3. [42]

To summarize, the convolutional layer receives an input of shape $W_{ci} \times H_{ci} \times D_{ci}$, where W_{ci} is the width of the input into the convolutional later, H_{ci} is the height of the input and D_{ci} is the depth of the input. The convolutional layer is defined by the number of filters K, the receptive field of each filter F, the stride length of each filter S, and finally, the number of zeros for padding on either side P. The output of the convolutional layer is $W_{co} \times H_{co} \times D_{co}$ where, $W_{co} = \frac{W_{ci}-F+2P}{S+1}$ and $H_{co} = \frac{H_{ci}-F+2P}{S+1}$ and $D_{co} = K$. Conventionally, the convolutional layer is followed by a pooling layer. The defining hyper-parameters include the receptive field size F and the stride length S. The output size can be calculated the same way as with convolutional layers. The backpropagation for convolutional layers is a convolution so that the gradients can be easily computed.

CNNs are largely used to process images and time series data owing to their grid-like structure [47, 93]. However, in this thesis we explore a specific architecture of CNNs called the WaveNet [61] architecture, initially developed to process audio signals, to predict sepsis in ICU patients.

2.4 Regularization and Multitask Learning

A deep learning algorithm, like any other machine learning algorithm, is crafted with a few key steps in mind that include: the task (example: classification, regression, etc.), the performance metric (i.e., the cost function to evaluate the training process) and the experience (i.e., training with labeled data or supervised learning vs. training with unlabeled data or unsupervised learning) [20]. We described the gradient based and supervised learning process of FFNs, RNNs and CNNs in sections 2.1.2, 2.2, 2.3 respectively. Section 2.1.4 detailed the importance of identifying an appropriate task-based loss function. In this section, we discuss the concept of overfitting and how to control it, i.e., regularization.

2.4.1 Overview

While a deep learning algorithm is prioritized with minimizing the training loss or maximizing training accuracy, an important consideration and challenge is to have an algorithm that also generalizes well, i.e performs well (with high accuracy and low loss) when fed new, previously unseen inputs. For instance, for a regression problem, the loss function works on minimizing, $\text{MSE}_{\text{train}}$, where

$$\text{MSE}_{\text{train}} = \frac{1}{\text{m}^{(\text{train})}} \left\| \mathbf{X}^{(\text{train})}\mathbf{w} - \mathbf{y}^{(\text{train})} \right\|_{\mathbf{2}}^{\mathbf{2}} \tag{2.42}$$

and $\frac{1}{\text{m}^{(\text{train})}}$ is the number of training observations $\mathbf{X}^{(\text{train})}$ with targets $\mathbf{y}^{(\text{train})}$. However, one is concerned with minimizing MSE_{test}, where the same expression as in equation (2.42) is computed for an unseen test set. In practice, a set of observations is divided into a training set, and testing set, where, for instance 80% of the observations are used for training an algorithm and the remaining 20% are set aside for testing. While training a deep neural network, there is a balance struck between minimizing the training error and the testing error. When one exceeds the other, we are faced with two challenges: overfitting (when the training error continues to decrease and is much smaller than the testing error) and underfitting (when the model fails to obtain a low enough training error).
The free lunch theorem for machine learning [88] states that averaged over all possible data-generating distributions, every classification algorithms has the same error rate when

classifying previously unobserved points [20]. This means if we make no assumptions about the data distribution of the features, all machine learning algorithms perform the same. Another interpretation is that all classification algorithms have high generalization error when tested on generalized distributions. So, instead of identifying a universal learning algorithm, the goal is to identify the data distributions and hypothesis space of functions to design a machine learning algorithm to perform well for a specific task. Picking an algorithm to approximate the function $f(\mathbf{x};\mathbf{w})$ that represents the relationship between the response y and the feature set may involve giving preference to some functions in the hypothesis space over others. There are ways to modify existing algorithms defined for specific relationships (e.g., linear functions), such as weight decaying if for instance, one may want a linear function with small weights. These different approaches are known as regularization.

> *Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error [20].*

2.4.2 Bias-variance trade-off

Regularization strategies include penalizing the values of parameters and penalizing the number of parameters. These strategies usually involve regularizing estimators with the goal of trading bias for variance [20]. The bias-variance trade-off is a challenge presented while fitting any and every model to some data. The bias of an estimator is defined as

$$\text{bias}(\hat{\mathbf{w}}_\mathbf{m}) = \mathbb{E}(\hat{\mathbf{w}}_\mathbf{m}) - \mathbf{w}, \tag{2.43}$$

where $\mathbb{E}(\hat{\mathbf{w}}_\mathbf{m})$ is the expectation value of the parameter estimated from the distribution and $w_\mathbf{m}$ is the true value. An estimator is unbiased when $\text{bias}(\hat{\mathbf{w}}_\mathbf{m}) = 0$, and therefore, the expected value equals the true value of the estimator. Similarly, the variance of an estimator is simply the variance $\text{Var}(\hat{\mathbf{w}})$. The two concepts are important and tied together because both of them are measures of different sources of error in a parameter. The bias measures how far the expected value of the predicted parameters deviates from the true value. The variance estimates the overall deviation of all the predicted values for a parameter, from the expected value. Consider the mean squared error of a regression model as

$$\begin{aligned} \mathrm{MSE}(\mathbf{w}) &= \mathbb{E}[(\hat{\mathbf{w}}_{\mathbf{m}} - \mathbf{w})^2] \\ &= \mathrm{Bias}(\hat{\mathbf{w}}_{\mathbf{m}})^2 + \mathrm{Var}(\hat{\mathbf{w}}). \end{aligned} \tag{2.44}$$

The training of a neural network involves minimizing the MSE, and that often comes with the choice of minimizing the bias or the variance of the model. The bias of a model is also closely linked and inversely proportional to the model complexity or capacity. As we increase the complexity of a model, we reduce the error due to bias. Bias, therefore, can be interpreted as the error incurred as a result of an excessively simple model used to represent a relationship that is inherently more complex [32]. The bias cannot be reduced despite an abundance of training data because the model is incorrect and not complex enough. This is an example of underfitting. The variance of a model, on the other hand, refers to the amount by which the predicted function $\hat{f}(\mathbf{x};\mathbf{w})$ changes as a result of using a different training set. Ideally, we would want a function or model that does not change much regardless of the training set, i.e., a model with low variance. Variance can be reduced by using a varied and inclusive training data set.

One of the challenges we face is the bias-variance trade-off. For instance, increasing the model complexity by adding parameters to allow for a better fit is a slippery slope. While this might reduce the error by bias, this could increase the variance and lead to overfitting to the training data. A highly complex model tends to fit a function that almost perfectly captures the behavior of the training data (including random noise) but performs poorly on a test set. However, a less complex model might have a low variance but is likely to underfit the training data owing to the model's high bias. This means one must find the right a model that hits the minimum generalization error as a result of a low enough bias and variance. This relationship among bias, variance, and model capacity is represented in Figure 2.12.

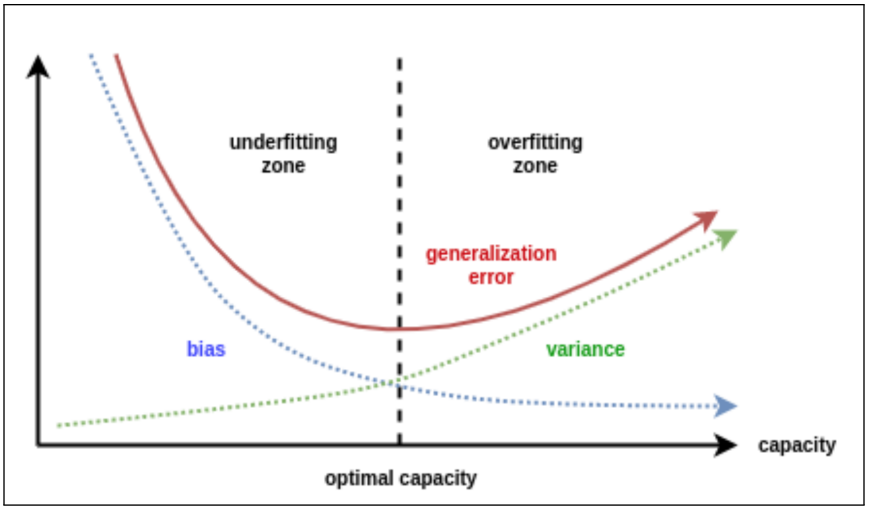


Figure 2.12 As the capacity (model complexity) increases (x-axis), bias (dotted) tends to increase and the variance (dashed) tends to increase resulting in a U-shaped curve for generalization error (bold curve). [20]

2.4.3 Validation sets

Testing error is used as a way to quantify how well the model generalizes. In practice, test sets may not be available and re-sampling methods are used to allow one to approximate the testing error. This includes the method of a validation set, where a portion of the training data is held out to test the trained model on. While the validation set approach is computationally simple, it has some drawbacks. The first one is that the validation estimate of the test error rate can be highly variable since it depends on which observations are randomly selected for training and which observations are selected for testing. Additionally, the model is trained on fewer observations since some of them are held out for validation. This could result in a worse trained model, and an overestimate of the testing error. [32]. Another approach that is often used is called k-fold cross-validation. This involves dividing the observations into k sets, where one set is left for validation and the remaining $k-1$ sets are used for training. The MSE is computed on the validation set. This process is repeated k times until so that every one of the k sets have had the chance to be a validation set. This reduces the variation in testing error and allows for a more accurate estimate. [32]. These method may not be useful for time-series data, as re-sampling of the data does not preserve

the time series nature of the observations.

2.4.4 Regularization methods

There are three broad classes of regularization methods. These are subset selection, shrinkage and dimension reduction. We discuss these methods within the context of a regression problem evaluated with a mean squared error loss.

- **Subset Selection**: Subset selection involves identifying a set of p predictors or features that we believe to be related to the response and is usually used for linear models [32]. The goal is to find not only a model that generalizes well but is also interpretable. Interpretability is often not of importance when fitting a deep learning model.
  Subset selection involves fitting a model on a reduced set of predictors and computing the loss to identify the set with the lowest loss. This can be performed in a few ways. One of these is called best subset selection. It involves fitting all 1 feature models, followed by all 2 feature models, 3 feature models until we reach the full set of p predictors. This yields a set of 2^p possibilities and can be very computationally expensive. Another method is called forward stepwise selection, where one begins with a model with no predictors and adds features, one feature at a time. The variable that improves the fit the most is added to the model at each step. This process involves fitting a null model with $p-k$ models at every k^{th} iteration which results in a total of $1+\frac{p(p+1)}{2}$ models. This allows for a computationally inexpensive alternative to the method of best subset selection [32].

- **Shrinkage methods** : An alternative to finding a subset of features with the best fit is to use a technique that regularizes the coefficient estimates or shrinks the estimates to zero [32]. These methods have been shown to significantly reduce the variance of the model and control overfitting. Shrinkage methods involve applying norm penalties to the cost function. This can be generalized as

$$\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y}) = J(\mathbf{w};\mathbf{X},\mathbf{y}) + \alpha\Omega(\mathbf{w}), \tag{2.45}$$

  where $\alpha \in [0,\infty)$ is a hyperparameter that weights the norm penalty term Ω and is called the regularization strength. Additionally, Ω penalizes only the weights at each

layer, not the biases. One of the options for Ω is to use the L-2 norm of the weights. This is also called ridge regression. The cost function is now

$$\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y}) = J(\mathbf{w};\mathbf{X},\mathbf{y}) + \frac{\alpha}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}, \tag{2.46}$$

and a single update at a gradient step is written as

$$w \leftarrow w - \eta(\alpha w + \Delta_w J(\mathbf{w};\mathbf{X},\mathbf{y})) \tag{2.47}$$

where η is the learning rate and determines how large a change in the weight is made. This happens over many steps until and in practice a quadratic approximation to the objective function, that minimizes the cost, in the neighborhood of the weights is made [20].
Apart from the L-2 weight decay, the L-1 norm of the weights is also used as a penalty. In this case the penalty term Ω is

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_i |w_i|, \tag{2.48}$$

which is the sum of the absolute values of the weights and the cost function is now

$$\tilde{J}(\mathbf{w};\mathbf{X},\mathbf{y}) = J(\mathbf{w};\mathbf{X},\mathbf{y}) + \alpha \|\mathbf{w}\|_1 . \tag{2.49}$$

The application of the L-1 norm penalty to the weights, also called the Lasso model has an advantage over Ridge regression in that it forces some of the parameters to zero and perform variable selection. Lasso regression yields sparse models [20].
The choice of Ridge regression over Lasso, depends on a case to case basis. For instance, when the goal is to develop a sparser model, Lasso regression would be the obvious choice. Ridge regression would perform better when there are a large number of predictors that are all roughly equally weighted and have a high variance [20].

- **Dimension reduction**: Another method to regularize a model is called dimension reduction, where the set of features is transformed to yield a more informative set of predictors. One of these methods is called Principal Component Analysis (PCA). Here,

the first component direction of the data is along which the observations have the highest variance. PCA hypothesizes that directions of greater variance have greater predictive power. The second principal component direction is the one where the observations have the second largest variance, and so on. The steps to perform PCA include computing the covariance matrix of the feature matrix X (Σ), followed by finding the eigenvalues of Σ. The eigenvalues are ordered according to their magnitude, with the eigenvector corresponding to the largest eigenvalue of Σ indicating the direction of largest variance, and therefore the first principal component. Finally, the feature set X is projected onto the eigenspace of the covariance matrix and the resultant transformation is used as the new feature set. PCA can be very useful for lower-dimensional data sets. In higher dimensions, PCA tends to overfit [32].

2.4.5 Modeling Inter-individual and intra-individual variation

2.4.5.1 Hierarchical Models

Modeling physical and biological systems from repeated measurements for samples within a population, involves taking into account both within individual (intra-individual) and within population (inter-individual) variance. Most common statistical models are linear models that incorporate "fixed effects" or parameters for an entire population and "random effects" which model individual experimental units. Models that incorporates both fixed effects and random effects are called mixed effects or hierarchical models. Hierarchical models can be applied to grouped data like longitudinal data, repeated measures data and multilevel data. [68]. Mixed effects modeling applied to describe nonlinear relationships between the features and response are called Non-linear mixed effects (NLME) modeling.

NLME models offer interpretability and flexibility which explains their utilization in applications like pharmacokinetics analysis [77] where series of blood samples post doses of drugs for cancer patients are analyzed to understand the pharmacological processes within the body and quantify the time-concentration relationship for individual subjects as well as the population of subjects. Other applications include toxicokinetics [18], the study of circadian rhythms [52] and fisheries science [67]. These problems have a few things in com-

mon as detailed in [14], including repeated observations of a continuous response on each of several individuals, inter-individual variability in the relationship between the response and an observed condition or time and, a scientifically relevant, parametric model defining intra-individual and inter-individual behaviour over time. NLME models are constructed with the intentions of understanding the biological or physical phenomenon of interest and its variation across a population, through a set of parameters and also utilizing population data to make individual level predictions.

We now formulate a general model incorporating mixed effects, as in [14] and follow their notation. We consider a population with i individuals, where $1 \leq i \leq p$ and $\mathbf{z}^{\mathbf{i}}_{\mathbf{t_j}}$ is the t_j^{th} measurement of the response under condition $j = 1, \cdots, n^i$ and other possible conditions or covariates u^i. In many cases t_j is the time at which measurement j was taken for individual i and for brevity we can say $\mathbf{x}^{\mathbf{i}}_{\mathbf{j}} = (\mathbf{t}^{\mathbf{i}}_{\mathbf{j}}, \mathbf{u}^{\mathbf{i}})$. We also consider a vector of characteristics that are constant and specific to individual i, a^i, like the age, gender, etc. We now have $\mathbf{z}^{\mathbf{i}} = (z^i_1, \cdots, z^i_{n^i})^T$ and the tuples $(\mathbf{z}^{\mathbf{i}}, \mathbf{u}^{\mathbf{i}}, \mathbf{a}^{\mathbf{i}})$ are independent across i, to indicate that the individuals are unrelated. The dynamics for the i^{th} individual can be usually described by

$$\frac{dz^i}{dt} = g^i(\mathbf{x}^{\mathbf{i}}, \mathbf{z}^{\mathbf{i}}_{\mathbf{t_j}}; \beta^i) \tag{2.50}$$

where $\beta^{\mathbf{i}}$ is a vector of individual specific parameters (for instance, the rate of drug clearance, growth rate). The NLME model is described in two stages: 1) the Individual-Specific Model and 2) the Population model. This is how the hierarchical modeling framework takes into account the interplay between individual data and population level characteristics. The first stage or the individual-level model is a solution to the ODE in equation 2.50 where

$$\mathbf{Z}^{\mathbf{i}}_{\mathbf{t_j}} = f(\mathbf{x}^i_{t_j}, \beta^i) + e^i t_j, \quad j = 1, \cdots, n^i. \tag{2.51}$$

In equation 2.51, f is a function modeling intra-individual behavior and depends on the individual level parameters β^i. If we continue with the pharmacokinetics example from [77] where, we have a compartment model for the concentration of the drug theophylline, the solution of which yields

$$C(t) = \frac{D k_a}{V(k_a - Cl/V)} \Big\{ \exp(-k_a t) - \exp\Big(\frac{-Cl}{V} t\Big) \Big\} \tag{2.52}$$

where $C(t)$ is the concentration of the drug at time t following an oral dose D at time 0. Additionally, k_a is the fractional rate of absorption, V is the volume required to account for all the drug in the body and Cl is the clearance rate. So, the set of individual specific parameters $\beta^i = (k_a^i, V^i, Cl^i)^T = (\beta_1^i, \beta_2^i, \beta_3^i)^T$ for each subject i. We also note that $e_{t_j}^i = \mathbf{Z}_{t_j}^i - f(x_{t_j}^i, \beta^i)$ are the intra-individual deviations and are assumed to have $\mathbb{E}(e_{t_j}^i | u^i, \beta^i) = 0$ for all j. Next we state the second state or the population level model of the NLME framework. We have,

$$\beta^i = d(a^i, \Gamma, b^i) \quad i = 1, \cdots, p, \tag{2.53}$$

where d is a $p-$dimensional function (since there are p individuals) depending upon an $(r \times 1)$ vector of fixed effects shared by all individuals, ` and a $(k \times 1)$ vector of random effects b^i, specific for each individual i. In 2.53, β^i includes variations due to individual attributes through a^i and unexplained variation in a population of individuals, through b^i. We also assume $\mathbb{E}(b^i | a^i) = \mathbb{E}(b^i) = 0$ and $\text{var}(b^i | a^i) = \text{var}(b^i) = D$, where D is the covariance matrix that describes the unexplained variation in the elements of β^i, and is the same for all individuals i. The standard assumption is $b^i \sim \mathcal{N}(0, D)$. Often, the function d in equation 2.53 is linear and we have

$$\beta^i = A^i \Gamma + B^i b^i, \tag{2.54}$$

where A^i is a design matrix depending on the elements of a^i and B^i is a design matrix populated with zeros and ones thereby allowing only some elements of β^i to have an associated random effect.

So far we have detailed the two stages of an NLME model in equations 2.51 and 2.53. The final consideration involves specifications for the within-individual variation. The error or deviations term in equation 2.51 is decomposed as

$$e_{t_j}^i = e_{R,t_j}^i + e_{M,t_j}^i \tag{2.55}$$

where e^i_{R,t_j} is due to the particular realization observed and e^i_{M,t_j} is possible measurement error. Another common assumption [14] is that the two errors are conditionally independent giving us,

$$\text{var}(Z^i|u^i,\beta^i) = \text{var}(e^i_{R,t_j}|u^i,\beta^i) + \text{var}(e^i_{M,t_j}|u^i,\beta^i). \tag{2.56}$$

A model for the variances is given by $R^i(u^i,\beta^i,\xi)$ and where $R^i(u^i,\beta^i,\xi) = \text{var}(Z^i|u^i,\beta^i)$ where $\xi = (\sigma^2_R,\sigma^2_M)^T$ and other parameters associated with the model describing $R^i(u^i,\beta^i,\xi)$, like the rate of exponential growth ρ if the suggested model is the exponential correlation function.

After formulating the NLME model, we now discuss the inference of the parameters. Understanding the values of the parameters in f and their variation across individuals, in order to allow for individual-specific prediction is of great interest. [3, 14] offer a variety of methods implemented to infer the parameters describing f. One of the most common methods and very well suited to the NLME framework is the maximum likelihood method. Let us consider the individual-level general NLME model described by equation 2.51 with an assumption on the distribution of Z_i (usually, normality), given (q^i, b^i), where $q^i = ((u^i)^T,(a^i)^T)$. The resultant conditional distribution $p(z^i|q^i,b^i;\Gamma,\xi)$. Further assuming $R^i(q^i,\Gamma,b^i,\xi)$ is diagonal, the density is written as a product of all the $p+1$ individual contributions. We also extend the assumption of normality to stage 2 of the model to get a k-variate density $p(b^i;D)$ for b^i. The joint density of (z^i,b^i) given q^i can now be written as

$$p(z^i,b^i|q^i;\Gamma,\xi,D) = p(y^i|q^i,b_i;\Gamma,\xi)p(b^i;D). \tag{2.57}$$

Furthermore, the likelihood for Γ,ξ,D based on the observations $y^1,\cdots,y^p$, given q^i is

$$\prod_{i=1}^{p}\int p(y^i,b^i|q^i;\Gamma,\xi,D)\text{db}^i = \prod_{i=1}^{p}\int p(y^i|q^i,b^i;\Gamma,\xi)p(b^i;D)\text{db}^i, \tag{2.58}$$

with independence across over all individuals. This means, the parameters we estimate are those that maximize the likelihood. Computational techniques to maximize the likelihood and others that estimate the parameters are detailed in [3] and [14]. Figure 2.13 is a visual representation of the results of the MLE approach applied to the NLME framework, with

the intention of estimating parameters to be used for individual prediction.

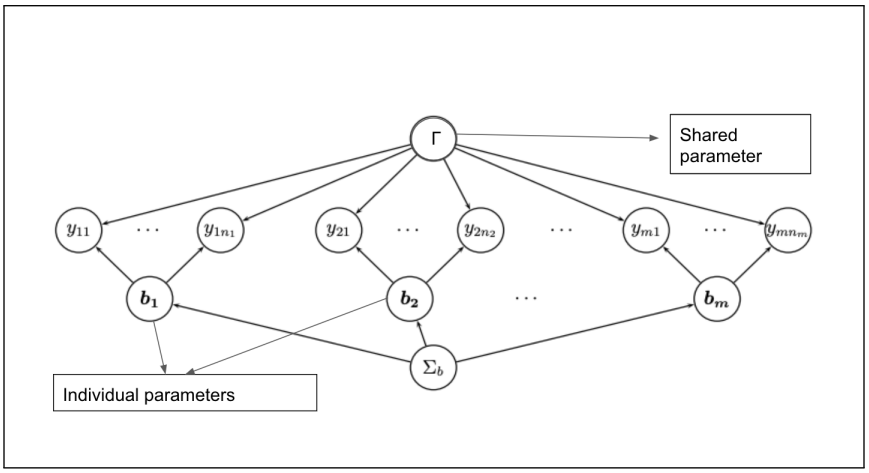


Figure 2.13 NLME framework indicating shared and individual parameters, for n^i measurements for $m+1$ individuals, where $z^i_{t_j}$ is the response, Γ is a shared parameter and b^i is the vector of individual specific parameters.

NLME models have been widely used in biomedical applications for individual predictions. Ribba et al. [70], a compartmental ODE model was used to study the reduction in size of low-grade glioma (LGG) in response to three types of therapy: procarbazine and vincistrine (PCV) chemotherapy, temozolomide (TMZ) chemotherapy, and radiotherapy. They used data measuring mean tumor diameter (MTD) from MRI scans for 21 PCV patients, 24 TMZ patients, and 25 radiotherapy patients, to fit the model. Each parameter in the compartmental model assumes log-normal random effects. The model for TMZ patients was validated on 96 other TMZ patients, and can accurately predict 5%, 50%, and 95% percentiles of the observed MTD time course using 200 virtual patients from the model. The model was further able to fit individual MTD trajectories based on treatment type and pretreatment MTD. The accuracy of this model in predicting both population-level distributions and individual trajectories suggests it is a viable tool for future use in predicting treatment efficacy for individual patients.

Staatz et al. [81] developed a population pharmacokinetic model of tacrolimus in adult liver transplant recipients, to test this model in individualizing therapy. Data was collected from 102 transplant recipients, out of which 34 patients were randomly picked to assess predictive performance. Patients were administered with tacrolimus and blood samples were collected to determine concentrations of tacrolimus until concentrations were stabilized and within the predefined concentration range and also if any symptoms of rejection were observed or to manage any suspected adverse event. The prospective and retrospective data was combined and patients were divided into index and predictive performance groups. NLME was used for population phamacokinetics modeling to determine fixed effects where a one-compartment model was used and, random effects or deviations were modeled using an exponential interindividual variability model. Finally, predictions for individual tacrolimus concentrations were made using the predictive-performance group. The results reported, indicate unbiased, but imprecise predictions owing to high interindividual variability calling for more research in the field.

Other studies also utilize mixed models, like [85], where they use a population pharmacokinetics model to understand variability affecting risperidone and 9-hydroxyrisperidone pharmacokinetics from genetics and non-genetic sources with the intention of furthering the argument to personalize drug dosages. Similarly, the work in [16] shows how a nonlinear joint model can effectively predict the risk of death in metastatic castration-resistant prostate cancer (mCRPC) patients. However, what all of these studies have in common are domain knowledge allowing for a compartment model to be used to describe the changes in drug concentrations or assumptions about the underlying distributions of the subject data which oftentimes is not available. So, while NLME models are extremely effective in the field of pharmacokinetics as indicated in [14, 77, 81, 85], they cannot be used for problems that are not as well defined, like bladder pressure prediction from neurostimulation data or the early detection of sepsis. This evokes the need for an even more flexible framework, that is generalizable and capable of using population data to make individual predictions. The implementation of multitask learning trained neural networks is a possible candidate and we discuss this further in the following section. We also present the results of applying this methodology to two applications: bladder prediction from neurostimulation data and

the early detection of sepsis, in chapters 3 and 4.

2.4.6 Multitask Learning

Multitask Learning [9] is a way to improve generalization by pooling from similar tasks. When a part of a model (parameters) is shared across tasks, this forces the model to generalize well. Multitask learning hypothesizes that areas of variation in similar tasks are shared, so by training tasks together and sharing representations, these sources of variance can be minimized [20]. To draw a parallel to the human brain, humans rarely learn new tasks from scratch. For instance, when we learn a new language, we often try to utilize grammar and semantics cues from the first language we learned. From a machine learning perspective, multitask learning was defined in [9] as

> *Multitask Learning (MTL) is an inductive transfer mechanism whose principal goal is to improve generalization performance. MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks.*

Caruana et al [9] make their first argument for MTL and by comparing backpropagation for single-tasking (ST) ANNs and that for MTL ANNs. Consider Figure 2.14, where four tasks are trained separately and don't share parameters or data. In Figure 2.15, the inputs into the network is the same as that in panel Figure 2.14, but now the networks share a hidden layer. This means it is possible for the representations for one task, learned in the hidden layer, to help another task.
Sharing what is learned by different tasks, while tasks are trained in parallel, is the central idea in multitask learning [9]. MTL used domain specific information obtained through the training process of related tasks that use a shared representation. During backpropagation, since one or more hidden layers are shared, MTL allows the hidden layer to generalize to more than one task, which would not be possible via STL. Caruana et al [9] describe MTL as an inductive transfer process because, in an MTL model, each task has access to additional sources of information (that from other tasks) apart from its own and can leverage this information to learn that specific task better, while also learning how to generalize to other, similar tasks. Furthermore, they describe that the inductive transfer improves

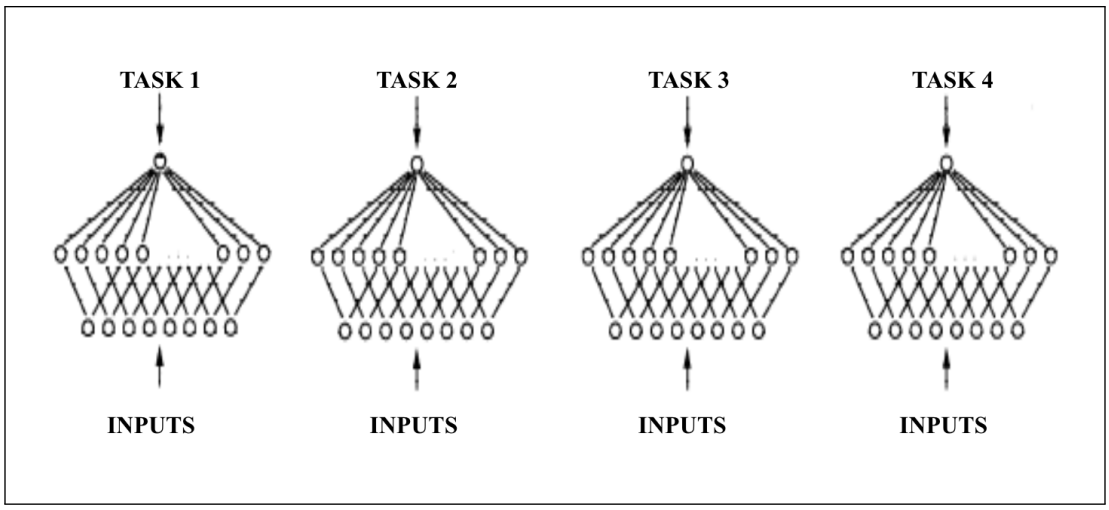


Figure 2.14 Single tasking models of four tasks [9]

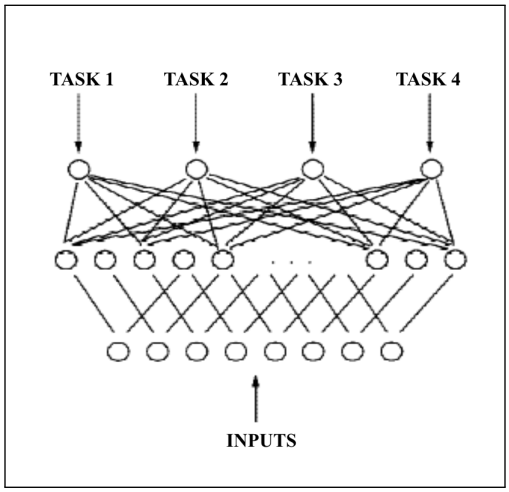


Figure 2.15 Multitasking model of four tasks [9]

generalization by providing a strong inductive bias, i.e., where the learner prefers a specific hypothesis over others. In terms of the neural network, each task prefers hypotheses and learns parameters that explain all the tasks, thereby facilitating the network's ability to generalize [9].

> For multitask bias to exist, the inductive learner must be biased to prefer hypotheses that have utility across multiple tasks [9].

MTL can be performed in two main ways, with regards to deep neural networks: hard parameter sharing and soft parameter sharing of the hidden layers [9, 73]. Hard parameter sharing, as shown in Figure 2.16 is when parameters of the hidden layers are shared across all the tasks. This allows for overfitting to be controlled greatly as the model is trained to find a representation that fits all the tasks [73, 5]. Soft parameter sharing is the method where there are separate task-specific models and no shared layers. However, the distance between the parameters is regularized in order to close the distance as much as possible, as in Figure 2.17.

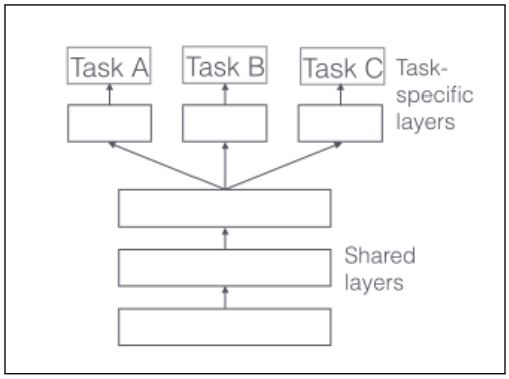


Figure 2.16 Hard parameter sharing network [73]

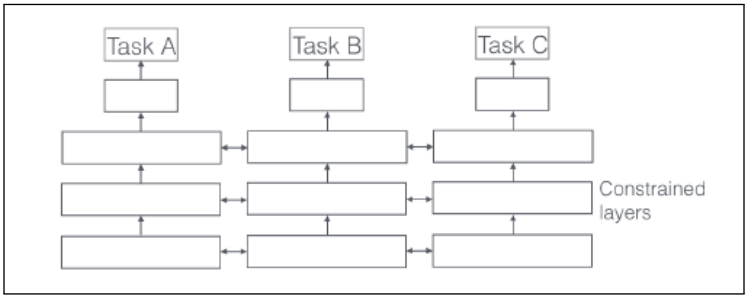


Figure 2.17 Soft parameter sharing network[73]

We now examine the forward propagation in a hard parameter sharing MTL network, i.e.,where the initial layers are shared as in Figure 2.16. The i^{th} neuron in one of the initial layers receives $m+1$ inputs each given by x_k. We have,

$$\begin{aligned} v_i &= \sum_{0}^{m} w_{ik} x_k + b_i \quad \text{and,} \\ y_i &= \varphi(v_i). \end{aligned} \tag{2.59}$$

In equation (2.59), y_i is the output of the i^{th} neuron, and w_{ik} are shared weights. Now, the network splits into P task specific sections so we get:

$$\begin{aligned} y_j^p(N) &= \varphi_j(v_j^p(N)), \\ v_j^p(N) &= \sum_{i}^{m} w_{ji}^p(N) y_i(N) \end{aligned} \tag{2.60}$$

where y_i is the output from the previous neuron and $y_j^p(N)$ is the output of the j^{th} neuron for the p^{th} individual.
There are several reasons presented in [9] and [73], for how inductive transfer takes place and inductive bias is obtained through multitask learning. We explore this by considering two tasks A and B, with a hidden layer representation F.

- **Implicit data augmentation:** MTL increases the amount of data a network has to work with since similar tasks are jointly trained. As seen when discussing the back-propagation in MTL (Figures 2.15), the training signals in the hidden layer are also shared, since the parameters are shared. This allows the network to learn a representation that captures both tasks by averaging the noise patterns.

- **Attention Focusing:** Consider the situation where task A is particularly difficult to train, either because there are limited training data available or because the available data is particularly noisy. In a case such as this, the network has a difficult time picking out relevant features out of the noisy data. However, while training tasks A and B in tandem, the network selects attributes and focuses its attention on features that are relevant to both tasks.

- **Eavesdropping:** We have two tasks A and B with the hidden layer F. While training separately, it could be the case that F is more difficult to learn for task A than it is for task B, (reasons include noisy residual errors, complicated relationship between F and other layers). In this case, while training A and B jointly, A could eavesdrop on B's training of layer F and better learn F.

- **Representation bias:** MTL encourages the model to pick a hidden layer representation that other tasks also prefer. This allows the model to generalize to new tasks. This is also how MTL acts as a regularizer by preventing the network from overfitting to a single task.

Owing to MTL's ability to improve training efficiency and allow for generalization, there have been several applications that have adopted this new method of training. [49] demonstrates the superiority of a MTL recurrent neural network model to classify text over their single-tasking counterparts. Liu et al [49] utilize MTL to jointly train four tasks, each of which classifies movie reviews either into binary classes of "objective" or "subjective" [63] or into five classes of sentiments [79]. Three of these tasks have sentence-level data, where a sentence long review is designated a class. The fourth task classifies several sentence long reviews (from the IMDB data set [51]). They implement three different ways to multitask their model by sharing different combinations of LSTM layers, bidirectional LSTM layers, and even embedding layers and then splitting into task specific layers. All the MTL models are compared to the STL models performing the same task and are shown to achieve higher classification accuracy for all the four tasks. Another application involves implementing multitask with CNNs to improve multiview face detection [91]. Zhang et al [91] built a deep CNN (DCNN) that can simultaneously learn the face/nonface decision, the face pose estimation problem, and the facial landmark localization problem. They collected face data from several databases including the Feret database [66], the BioID database [35], the web, etc. The processed images (according to [92]) were sent to the DCNN, which consists of three convolutional layers with max-pooling layers sandwiched between them. The fourth layer is a fully connected layer that splits into three task-specific branches that output three corresponding decisions. Zhang et al [91] report state of the art detection results in comparison to other architectures presented in the literature.

2.4.6.1 MTL with Recurrent Neural Networks and Convolutional Neural Networks

The two applications we consider in this thesis involve time-series predictions. Time series prediction can be difficult because many NN architectures struggle with modeling long-term temporal dependencies. The advent and use of LSTMs [28] have been instrumental. However, we are also motivated by the explanation Caruana et al [9] offer regarding the success of MTL with time-series data. It is suggested that long-term and short-term temporal learning are different processes and that predictions at different time scales have different dependencies. Using MTL is expected to learn both time scales simultaneously, thereby improving predictions. In this thesis, we apply mutlitask learning to LSTMs and CNNs to predict bladder contraction and sepsis, respectively.

In Chapter 3 we present our work in bladder pressure predictions for rats. We develop a novel combination of multitask learning, where the prediction of bladder pressure for each individual is a separate task, and recurrent neural networks to predict bladder pressure using neurostimulation data. The individual-as-task approach to MTL was first presented in [34], where they predicted wellness markers for students utilizing activity tracker data. We used this approach coupled with RNNs to leverage the time series nature of the neurostimulation data we had. We demonstrate how the MTL versions of our models yield better and more accurate predictions.

In Chapter 4, we describe our application of MTL to the problem of sepsis prediction in ICU patients. We use clinical measurements and waveform data made available on the MIMIC-III [38] database. Sepsis prediction is a difficult problem, particularly because of the variance in the population. With the goal of developing a model that can generalize to the population and make predictions on an individual level, we implemented a combination of individual as task MTL with a CNN inspired by the wavenet [61] architecture.

CHAPTER

3

ESTIMATING BLADDER PRESSURE AND BLADDER CONTRACTION IN RATS

3.1 Introduction

The ability to leverage individual-level time-series data from heterogeneous populations is an area of active biomedical research. While a myriad of statistical methods has been used to do this in the past, neural networks have been shown to make very efficient and useful models. Their ability to model nonlinear dependencies with minimal knowledge of statistical priors is an advantage that can be leveraged to predict medical outcomes [83]. Further research [65] demonstrate success with using long short term memory (LSTM) recurrent neural networks to capture long term temporal dependencies and accurately predicting missing data in electronic health records [10] or even patient diagnoses [48]. RNNs have also been used to make predictions about patients' heart health using physiological signals

like the ECG signal [84]. In this chapter, we use multitasking neural networks to predict bladder pressure in rats using the external urethral sphincter electromyograph (EUS EMG) signals from a pre-clinical study.

3.1.1 Previous Work

The inability to efficiently void the bladder, along with urinary incontinence is often a result of neurological disease or spinal cord injury (SCI) in humans. Voiding dysfunction can result in medical complications such as urinary tract infections (UTI) and damage. Kennely et al. [43] demonstrates through case studies that by controlling the stimulation parameters, intraurethral electrical stimulation could either inhibit bladder contraction (preventing involuntary bladder emptying) or excite the bladder (providing micturition). However, while electrical stimulation is becoming an accepted treatment of urinary incontinence by inhibiting neurogenic detrusor overactivity, the majority of the present methods apply continuous stimulation to the bladder [86, 12, 36]. Several studies [13, 76] have shown that conditional stimulation can allow for increased voiding efficiency. This means that accurately predicting bladder contractions is instrumental so that electrical stimulation can be rightly applied at the onset of a bladder contraction, thereby preventing involuntary emptying of the bladder.

Langdale et al. [46] proposed that overactivity and urinary incontinence caused by a neurological disease or a spinal cord injury can be treated by applying electrical stimulation to the pelvic nerve at the onset of a bladder contraction. Directly measuring bladder pressure, however, is extremely invasive and unrealistic for practical use. Wenzel et al. [87] demonstrate a relationship between the external anal sphincter (EAS) and neurogenic detrusor activity to test if the electromyogram (EMG) of the EAS could be used to detect the occurrence of reflexive bladder contractions in cats and humans. Thus, accurately predicting bladder pressure and bladder contractions from measurable signal data, such as EUS EMG signal, is essential to the development of stimulation treatment. Previous research in predicting bladder pressure from physiological signals has used Kalman filtering [71], regularized regression models [75] and, neural networks [71] along with LSTM networks [31].

Research [22, 30, 72] suggest that the afferent neurons present in the bladder, begin to fire after a certain bladder pressure is reached and increase their firing rate as the pressure

increases. Ross et al. [72] quantify the hysteresis observed in the relationship between bladder pressure and the afferent neuron firing rates. They implement and evaluate three algorithms to estimate bladder pressure from the sacral dorsal root ganglia (DRG) signals in cats. These include a linear regression model, a Kalman Filtering model and a nonlinear auto-regressive moving average (NARMA) model. Owing to the results of [72], Ross et al. [71] hypothesize that the NARMA model, which takes into account the bladder pressure and the firing rates from the afferent neurons, would yield the most accurate estimates of bladder pressure, followed by the Kalman Filtering model. Kalman Filtering is a recursive algorithm that estimates an unknown variable using measurements over a given period. It is used to estimate states given by linear dynamical systems in a state-space format [44, 78]. Ross et al. [71] combine a physical trajectory model and a neural signal-based observational model using a weighted average method to obtain a prediction for the bladder pressure. This model was evaluated against the NARMA model implemented via a NN to estimate the bladder pressure. For the NARMA model, the bladder pressure at time k, is considered to be a function of the firing rate of the neurons at that step, the firing rate at the previous step, and the bladder pressure at the previous step. Finally, the linear regression model, along with a LASSO regression, used the firing rate to estimate the bladder pressure at a given time. The results of the paper presented via the Normalized Root Mean Squared Error (NRRMSE), and the Pearson's correlation coefficient illustrated the superiority of the NARMA model over the other two.

Rutter et al. [75] implemented a spectrogram model along with a firing rate (FR) model to predict the bladder pressure from the electro-urethral sphincter electromyograph (EUS EMG) signal, in rats. As a baseline for the FR model, the EUS EMG signal was used to calculate the number of spikes per second, and then a linear regression model was trained to predict the bladder pressure using this firing rate as in [7], for fifteen female rats. For the spectrogram model, a spectrogram representation of the EUS EMG signal data, i.e., the coefficients of the fast Fourier transform for the EUS EMG signal, was used for the feature set. A LASSO regression model was trained to predict the bladder pressure from the spectrogram data. The performance metrics for comparison included the Root mean squared error (RRMSE) and the Pearson's correlation coefficient (CC) between the predicted bladder pressure and the actual bladder pressure. The results reported indicate that the LASSO model performed better, with a lower RMSE (16.12 ± 6.11)and a higher CC (0.8) than

the FR model and also other models reported in the literature (17 ± 7 [57]). They were also able to predict the bladder contractions using the bladder pressure predictions from the two models with reasonable accuracy, using the bladder contraction detection algorithm from Langdale et al. [46]. Rutter et al. [75] reported a high sensitivity (91.3%) (true positive rate) and specificity (80.2%)(true negative rate) for the spectrogram model trained on and tested on data from that rat. They also used models trained on one rat to predict the bladder pressure for another rat. The authors state that while some models yield accurate inter-individual predictions, some models do not. This could be because the models are over-trained on the data for that rat and could benefit sharing data from across the rats. However, overall, the LASSO model performs better with inter-individual predictions as well.

As suggested in [83], neural networks tend to overfit. Rutter et al. [75] also report overfitting to the training data, with the LASSO model in their paper. In this chapter, we intend to address this by implementing a multitask learning (MTL) training scheme to improve generalization by leveraging population information. By learning tasks in parallel, the model is able to learn a representation that models all the tasks while also acting as a regularization scheme [9, 73]. Multitask learning can be implemented in several ways. A commonly used method is to share parameters by sharing layers between tasks while training a network. MTL neural networks can be applied to various areas, including speech recognition, natural language processing [15, 11] and healthcare [34, 23]. Jaques et al. [34] proposed a MTL scheme in which predictions for each individual (user) in a population, is a separate task. They found the MTL model to be more accurate than state of the art statistical methods, presumably because assigning the 'users as tasks' utilizes information from population heterogeneity to better train a neural network to make accurate predictions on an individual level. By training every individual in tandem, the MTL architecture forces a neural network to learn a representation that accounts for differences in each individual while at the same time being generalizable to other individuals.

We investigate whether MTL models can be used to leverage data from the entire population to make accurate bladder pressure predictions on the individual level. Since autoencoder RNN based neural networks have been shown to yield relatively accurate bladder contraction predictions [31] with a mean NRRMSE of $17.6 \pm 2.7\%$, we propose a novel combination of RNNs with MTL in the form of individuals as tasks (see Figure 3.1a) for predicting bladder

pressure in a population of rats. We hypothesize that combining a multitask strategy with an RNN architecture will yield a more generalizable predictive model. Our goal for this study is to develop a model of bladder pressure from non-invasive EUS EMG so that the model can be used to predict the onset of bladder contractions using the predicted bladder pressures.

3.2 Methods

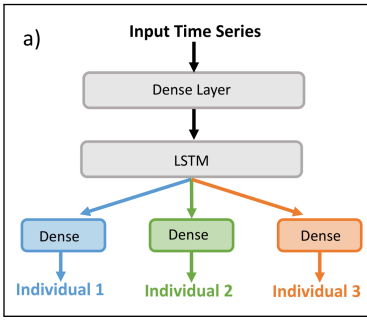
3.2.1 Data

We used data from 12 rats that had been dosed with Prostaglandin E2 to induce bladder contractions [21], similar to bladder dysfunction found in SCI, each with 5 to 8 trials in which the bladder was filled, and bladder contractions were observed, as described in [46]. Each trial contained the EUS EMG signal and measured bladder pressure. We used a time-dependent spectrogram representation of EUS EMG signal, as in [75], as a feature vector for our machine learning models. The spectrogram was calculated in a 0.25 second sliding time window and is a representation of the EUS EMG signal in the frequency domain. To focus on training and evaluating the models on bladder pressure, the data used for training and testing models to time intervals 30 seconds around bladder contraction activity in each trial [75]. The coefficients of the Fast Fourier Transform (FFT) within each window of the EUS EMG signal are used as time-dependent covariates for prediction. The 1500 FFT coefficients for each quarter second-time point results in a dense feature matrix of size $T \times 1500$, where T is the total number of quarter second-time points, for each of the 12 rats. We expect that, by being trained on population-level data, a multitasking approach will be able to overcome the challenge of training neural networks on a few numbers of individuals while also using a dense feature set.

3.2.2 Neural Network Architectures and Training

We trained four different neural network architectures, outlined in Figure 3.1b: a vanilla neural network which is a simple feedforward network (Vanilla NN), an MTL neural network (MTLNN), a vanilla recurrent neural network (Vanilla RNN) and a MTLRNN. We tuned the

learning rate and the regularization strength for the first dense layer as hyper-parameters. The hyper-parameters that allowed for relatively low mean squared error (RMSE) when testing on a subsequent trial, for all four architectures, were a regularization strength of 0.05 and a learning rate of 0.001 using the Adam optimizer [45].
The first trial for each individual rat was used as training data. For the RNN models, the second trial was used for validation data, and the remaining trials were used for testing data. For the NN models, the data for the first trial was split into training (85%) and validation (15%) sets.



b)

Model	# of layers	Layer types (# neurons)
Vanilla NN	2	Dense(16)-Dense(8)
MTLNN	2	Shared(Dense(16))-Dense(8)
Vanilla RNN	3	Dense(8)-LSTM(16)-Dense(8)
MTLRNN	3	Shared(Dense(8)-LSTM(16))-Dense(8)

Figure 3.1 (a) Schematic of multi-task learning recurrent neural network. The shared layers are in grey and the individual-specific layers are in color. (b) Network architectures for each model. "Shared" indicates the shared layers in the multitask models.

3.2.3 Multitasking Algorithm

The MTL models were trained by iterating through all tasks, i.e., individuals, as described in [34]. All shared neural network weights were trained on each iteration, and only individual-specific weights in the last layer were trained with the corresponding individual data set, as shown in Figure 3.1. This means, to complete a single epoch, data from every individual must have been used to train the shared layers and the rat specific layers after the network split. This approach achieves our goal to have "individuals as tasks," i.e. predicting the bladder pressure for a specific rat is a separate task. By leveraging population-level data from all the individuals to train the shared layers, we hypothesized that the multitasking

approach is able to make more accurate predictions for each individual. In contrast, a single-tasking neural network approach would train the first two LSTM and dense layers using data from only a single individual, i.e., the parameters in these first two layers are not shared since the networks are not jointly trained.
The forward propagation of information occurs as described by equations (2.59) and (2.60). At the backpropagation step, we compute the loss as

$$\text{Loss} = \left\| y_{t_j}^i - z_{t_j}^i \right\|_{\mathbf{2}} \tag{3.1}$$

where $z_{t_j}^i = f(x_{t_j}^i, \mathbf{w}) + e_j^i$ and f is the function approximated by each of the two vanilla neural network models/architectures we investigated. For the two MTL models, $z_{t_j}^i = f(x_{t_j}^i, \mathbf{w^{shared}}; \mathbf{w^i}) + e_{t_j}^i$ and the loss is computed as shown in Figure 4.2.
To summarize the training, at the start of an epoch, the data for the first individual is fed

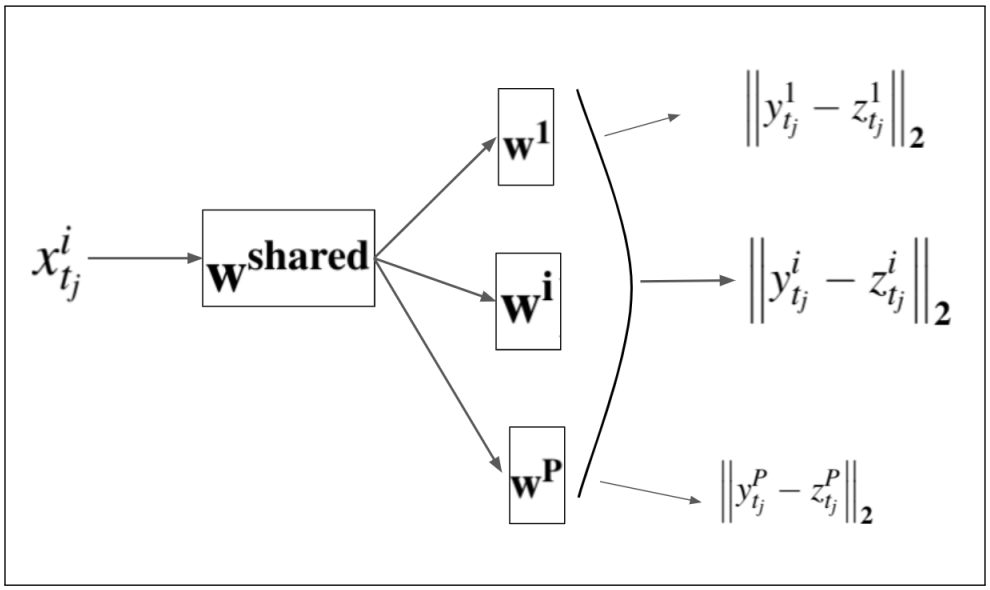


Figure 3.2 End-to-end representation of how the MTL training occurs

to the network and the shared parameters are trained along with the individual specific parameters. The loss for this individual is calculated. This step is carried out for every

individual, and once the final individual's data has been fed into the network, the parameters (shared and task-specific) have been updated and, the loss has been computed, the epoch is considered completed. This process is repeated for all the epochs.

3.3 Results

3.3.1 Bladder Pressure Predictions

3.3.1.1 Intra-individual prediction accuracy

We evaluated the performance of the four models on within-individual prediction accuracy. Prediction accuracy was quantified by calculating the RMSE over all time points for each testing trial. We found that the MTLNN, RNN, and MTLRNN models had the lower intra-individual RMSE (Figure 3.3a, Mann-Whitney U-test, $p < 0.05$), compared to the Vanilla NN model. Representative example predictions are shown in Figure 3.4. The difference among the RMSE values for the MTLRNN, Vanilla RNN and MTLNN models was not statistically significant. However, both MTL and the RNN architectures served as an improvement upon the Vanilla NN network. This means, MTL is able to generalize well using data from other rats and better model time series data, despite the underlying architecture using a neural network, just as the Vanilla RNN does. MTL does not improve the prediction accuracy of the RNN network. This could be because there is limited variation within a specific rat, and MTL is not required to regularize the performance if a Vanilla RNN is being used.

3.3.1.2 Inter-individual prediction accuracy

In order to investigate the generalizability of models trained on a single individual to other individuals, we evaluated the inter-individual prediction bladder prediction accuracy of each of the four trained models. We calculated the RMSE and Pearson correlation coefficient between the actual and predicted bladder pressure for every model trained on a single individual, using the trials from all other individuals as the testing data. We found that similar to intra-individual predictions, the MTLRNN and RNN models had the lowest RMSE overall for inter-individual prediction, both equal to each other (Figure 3.3b, Mann-Whitney U-test, $p < 0.05$) and, the Vanilla NN model highest mean correlation coefficient across all

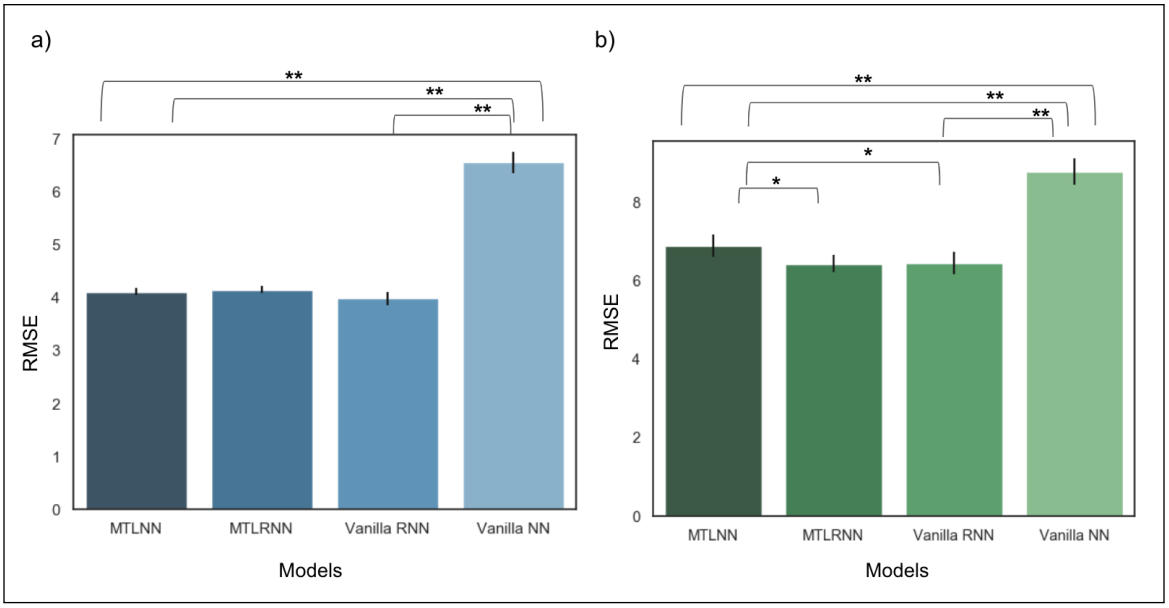


Figure 3.3 (a) Intra-individual mean squared error and (b) Inter-individual mean squared error for all the four models. (** indicates P < 0.005, and * indicates P< 0.05 Mann-Whitney U-test.)

individuals (Mann-Whitney U-test, $p < 0.05$). We note that for inter-individual predictions, an MTL RNN model leverages the time series behavior for all the rats and yields more accurate predictions than the MTL NN. In figure 3.5, we have plotted the out of sample correlation coefficients between the actual and predicted bladder pressures for both the MLNN model (panel a) and MTLRNN model (panel b). We can see that for some rats, such as rat 14, we are unable to obtain generalizable results using either model. However, the MTLRNN model , overall, has a higher correlation coefficient for rat 3 than the MTLNN model. This means, using the MTLRNN model, we can generalize rat 3's model to predict the other rats. Interestingly, the mean correlation coefficients for the MTLRNN and the MTLNN model were within the same order of magnitude (.74 and .77, respectively). This further corroborates that the MTLRNN model generalizes well and can be used to accurately model bladder pressure.

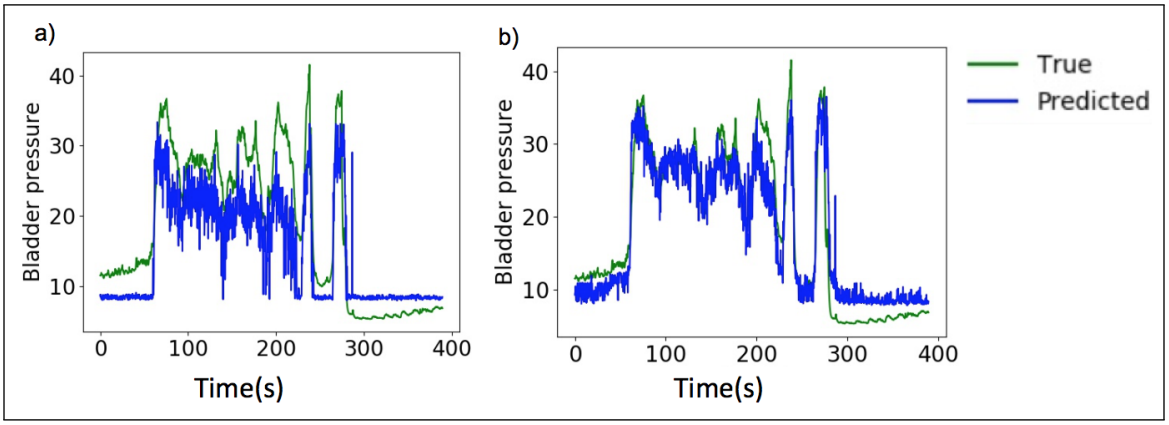


Figure 3.4 Example of bladder pressure predictions +/- 30 second around bladder contraction activity, for trial 10 in rat 3 using (a) the Vanilla NN model or (b) the MTLNN model.

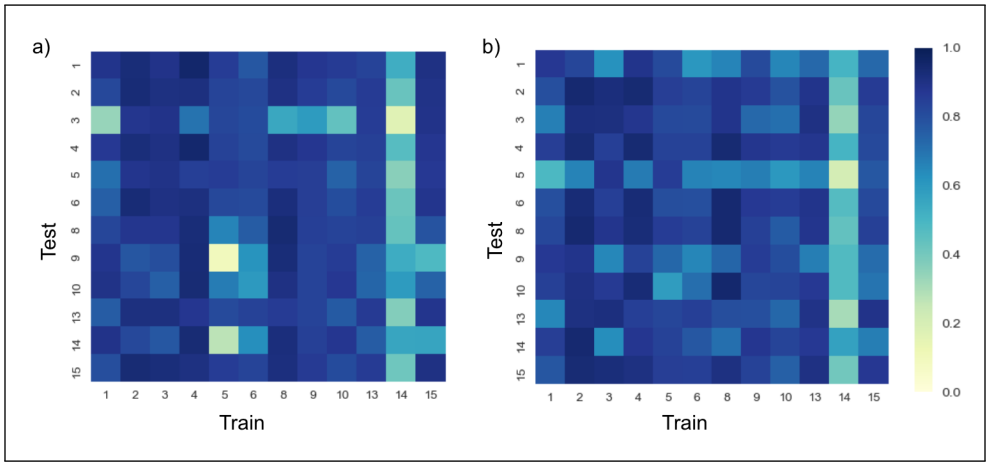


Figure 3.5 Out of sample correlation coefficients between the actual and predicted bladder pressure for the MTLNN model (panel a) and for the MTLRNN model (panel b).

3.3.2 Bladder Contraction Predictions

3.3.2.1 Intra-individual Predictions

We also classified bladder contractions using the bladder contraction algorithm from [46] and [75] and the bladder pressure predictions from all the four models. We trained the bladder contraction algorithm on the first three post-PGE2 trials and tested it on the rest. The performance of the algorithm using predictions for each of the models was evaluated by computing the sensitivity (recall) and precision (positive prediction value (PPV)) as outlined in

$$\text{sensitivity} = \frac{\text{Number of correctly identified contractions}}{\text{Total number of contractions}} \times 100\% \tag{3.2}$$

and,

$$\text{precision} = \frac{\text{Number of correctly identified contractions}}{\text{Number of positive predictions}} \times 100\%. \tag{3.3}$$

The results are reported in tables 3.1 and 3.2, respectively. The Vanilla NN model performed the worst with the lowest sensitivity and PPV, and the Vanilla RNN model performed the best on both the training and test sets. For both the Vanilla NN and Vanilla RNN model, the majority of the sensitivities were 1 on the training set. However, on the testing set, only 3 out of 12 rats had a sensitivity of 1, for the Vanilla NN model. The Vanilla RNN model performed better with a sensitivity of 1 for 7 out of 12 rats. The RNN model is able to leverage the time-series nature of the data to identify contractions correctly. We note that the MTL NN model has a sensitivity and PPV comparable to the MTL RNN and RNN models, and is a significant improvement over the Vanilla NN model. This indicates that multitasking is able to improve the network's ability to detect contractions and make better predictions.
The MTLRNN model had one of the highest sensitivity and precision on the testing set. Most of the rats had a precision greater that 0.7 on the training and testing set. Furthermore, 8 out of 12 rats had a sensitivity equal to 1 on the testing set. This indicates that while the MTL RNN model had a high RMSE for bladder predictions, it is able to predict spikes in bladder pressure accurately and, therefore, correctly identify bladder contractions.

Table 3.1 Recall and PPV for all the four models trained on the first three post PGE-2 trials and tested on the same

Metric	Vanilla NN	MTLNN	RNN	MTLRNN
Recall	78%	97.2%	97.2%	94.4%
Precision	58.1%	82.3%	82.7%	77.3%

Table 3.2 Recall and precision for all the four models trained on the first three post PGE-2 trials and tested on the remaining trials

Metric	Vanilla NN	MTLNN	RNN	MTLRNN
Recall	50%	81.5%	79.2%	77.1%
Precision	38.7%	53.1%	53.3%	62.9%

3.3.2.2 Inter-individual Predictions

We also tested the models' transferability by using models trained one rat to predict bladder contractions for other rats. The MTLRNN and Vanilla RNN models have the highest inter-individual recall and precision, with the MTLNN model following closely. The Vanilla NN model performs relatively poorly, with several rats having recall and precision of 0. The results for the MTLRNN and MTLNN models are summarized in Figures 3.6 and 3.7.
The MTL NN and MTL RNN models demonstrate generalizability because their predictive power transfers over to other rats. For instance, for the MTL NN model, the sensitivity is greater than 0.7 for 78 out of 144 cases, and for the MTL RNN model, the sensitivity is greater than 0.7 for 55 out of 144 cases. In contrast, the Vanilla NN model has a sensitivity greater than 0.7 only for 16 cases, where the models were trained on one rat and tested on others.

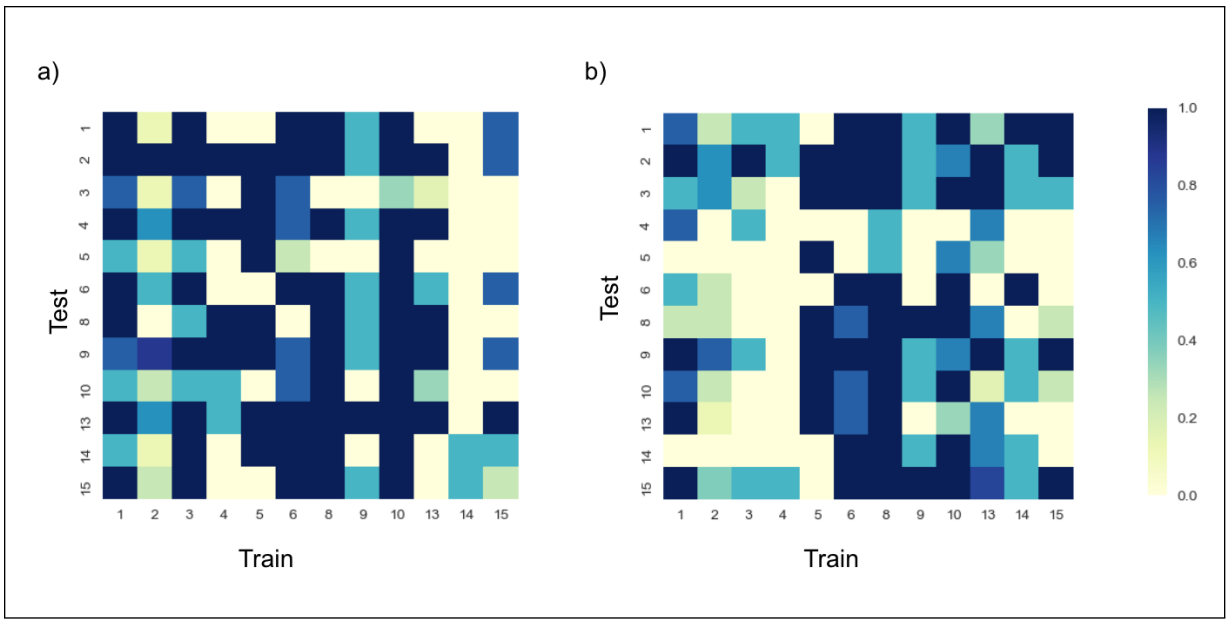


Figure 3.6 Out of sample sensitivities for the MTLRNN model (panel a) and the MTLNN model (panel b).

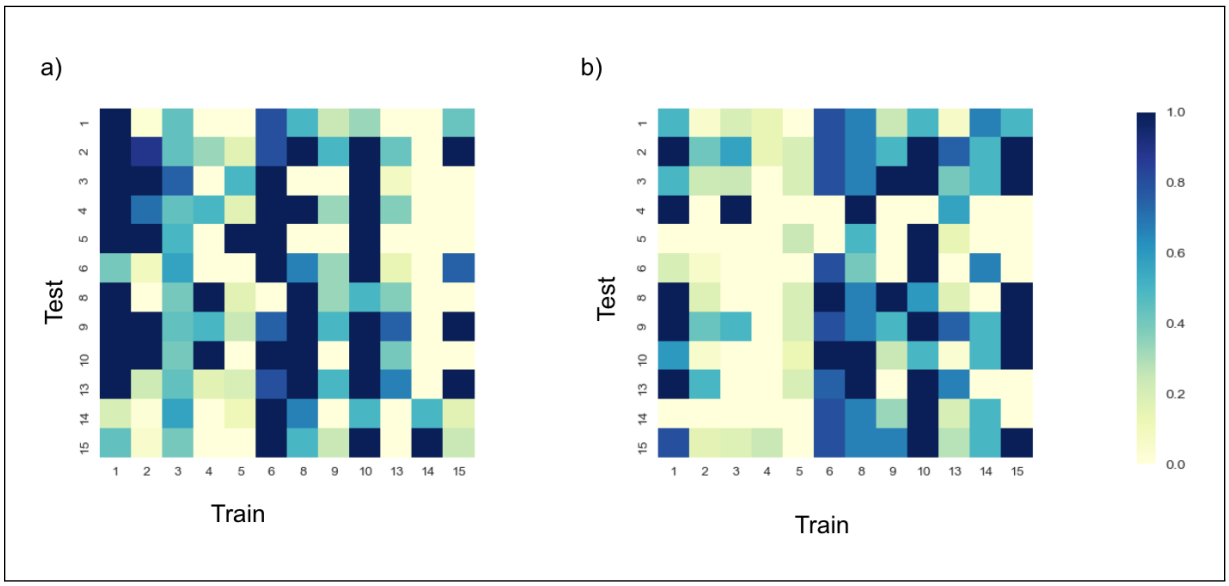


Figure 3.7 Out of sample precision for the MTLRNN model (panel a) and for the MTLNN model (panel b).

3.4 Discussion

Our results showed that the MTLNN model is able to produce the most accurate intra-individual and inter-individual predictions. This suggests that the MTL architecture can utilize data from all the individuals to learn shared features for enhancing predictive accuracy overall. We observed that RNN models were less accurate than their Vanilla NN counterparts. This challenges our expectations of RNNs to yield accurate predictions for time series data, given their established ability to accurately model sequential data. However, the mean inter-individual correlation coefficient was reasonably high for the MTLRNN model. In future work, we will investigate whether this implies that the MTLRNN model can be used to reliably predict changes in bladder pressure from the EUS EMG signal. If so, using RNNs would be preferable to using vanilla neural networks because RNNs could be used to forecast future bladder pressure levels without an EMG signal, e.g., using an encoder-decoder architecture [82].

CHAPTER

4

NEURAL NETWORKS FOR THE EARLY DETECTION OF SEPSIS

4.1 Introduction

The abundance of Electronic Health Records (EHR) has motivated their widespread use in machine learning models for medical diagnosis applications [1, 80]. ANNs, owing to their ability to model complex, non-linear relationships, have been used in biomedical applications [83, 1, 53]. Moein et al. [53] detail the complicated process of making a medical diagnosis. They selected eight different diseases and developed a fuzzy-FFN to classify patients and make medical diagnoses like a tumor, earache, and heart disorder to demonstrate the effectiveness (increases classification accuracy) of efficient symptom selection and data fuzzification on a neural network. [1] demonstrate the ability of a simple FFN, with a single hidden layer, to obtain high classification accuracy with two medical diagnoses:

acute nephritis disease and heart disease. Other studies [80, 23]) leverage the time-series nature of clinical signals and utilize LSTM networks and RNNs to identify the correct medical diagnoses. In this chapter, we discuss the important medical condition sepsis and use a novel combination of a WaveNet [61] inspired CNN architecture trained using an individual-as-task [34] multitask learning (MTL) scheme, to predict clinical signals and sepsis in ICU patients.

4.1.1 Previous Work

Sepsis is a clinical syndrome defined by a systemic response to infection and is a leading cause of mortality, morbidity and healthcare utilization [59, 55, 2]. With the number of cases of sepsis exceeding 750000 a year, it is still an "elusive syndrome" [2], having multiple possible causes. However, sepsis can be broadly defined to be the result of infections acquired from the community, hospital, or elsewhere and is the evidence of a systemic response called the systemic inflammatory response syndrome (SIRS) [55]. A patient qualifies as having SIRS if two or more of the following criteria are satisfied: 1) temperature greater than 100.4° F or less than 96.8°; 2) pulse rate greater than 90 beats per minute; 3) respiratory rate greater than 20 breaths/min ; and 4) white blood cell (WBC) count less than 4000/mm^3 or greater than 12000/mm^3. While the SIRS criteria provide some information, they have been replaced by more recent definitions of sepsis detailed in [55] and [2]. Severe sepsis includes instances of one or more organ dysfunctions and clinical markers associated with failing organs and is followed by septic shock as a result of hypotension and increased fluid replacement, which eventually leads to death. The treatment for severe sepsis, as described in [2], largely involves a management plan six hours after identifying sepsis and a plan for if or when the patient is in the ICU. The basic steps are to provide cardio-respiratory resuscitation and managing infection. However, the exact steps to ensure effective resuscitation are not defined. Owing to the lack of new therapies for sepsis and the number of cases of mortality it causes, there have been several clinical studies to understand the condition better. The rise in clinical trials to study sepsis and septic shock have seen a decline in the number of deaths they cause [55] because of early detection and treatment. There have also been several studies that have been dedicated to identifying a more data-driven approach utilizing machine learning, for the early detection of sepsis.

While there are scores to assess illness severity like Sequential Organ Failure Assessment (SOFA) score, Modified Early Warning Score (MEWS), and Simple Clinical Score (SCS), these only point to the general deterioration of health [19, 4, 39]. Henry et al. [26] developed one of the first models to detect patients at risk for sepsis and septic shock several hours before the onset of septic shock. They trained a Cox-hazard model on physiological signals and vital signs for ICU patients, from the MIMIC-II [38] data set and computed a Targeted Early Warning Score (TREW Score) to identify the early onset of sepsis and septic shock. The TREW score was compared to the MEWS score (used in ICU triage) and a severity score that identifies a person with sepsis if they have at least two conditions from the SIRS criteria, suspicion of infection, and either hypotension or hyperlactatemia. The Cox-hazard model is stated in

$$\lambda(t|X) = \lambda_o(t)e^{\beta^T X} \tag{4.1}$$

where $\lambda(t|X)$ estimates the risk of septic shock at a certain time t, given the set of features X and $\lambda_o(t)$ is the time-dependent baseline hazard function which quantifies the instantaneous probability that the onset of septic shock occurs at that given time. Furthermore, β is the vector of regression coefficients used to weight the features X of the patient. A patient was classified as having sepsis when all the SIRS [55] conditions were met, and the patient had an infection as indicated by any ICD9 code for infection. A patient was classified as having septic shock when along with sepsis, they had hypotension (Systolic BP $< 90 mmHg$) for over 30 minutes and more than 20 ml/kg of fluid replacement for 24 hours or more. The features to predict sepsis and septic shock were selected based on how their inclusion influenced the accuracy of the model. Henry et al. [26] demonstrate that the TREW score from the Cox hazard model, stated in equation 4.1, was capable of identifying patients at risk of septic shock at a specificity of 0.67 and a sensitivity of 0.85, and that too at a median of 28.2 [interquartile range (IQR), 10.6 to 94.2] hours before their onset. The number of predicted patients at the risk of septic shock, by the TREW score, was 58.5% higher than other methods presented in the literature.

Calvert et al. [8] followed a similar computational approach to allow for the early detection of sepsis. Their goal was to predict the onset of the patient's first five-hour SIRS episode,

three hours before its onset. The early warning algorithm is called InSight and demonstrated a sensitivity of 0.90(95%CI : 0.89˜0.91) and a specificity of 0.81(95%CI : 0.80˜0.82). They used data from the MIMIC-II [38] database, from where they extracted nine vital signs and physiological signals (systolic blood pressure, pulse pressure, heart rate, temperature, respiration rate, white blood cell count, pH, blood oxygen saturation, and age). The time-series nature of these measurements was leveraged to compute the mean, correlation, and other trend measurements in a five-hour sliding window. These trend features were weighted to compute a risk score, which would then be used to classify a patient into the appropriate category. The optimal weights of the trend features were obtained after the supervised training of the model. The InSight model was able to achieve high sensitivity in comparison to other biomarker detection methods and existing early detection models. An improvement (higher sensitivity and specificity) upon the InSight model came about with the study by Kam et al [40], where they used deep learning models (FFNs and LSTMs) to predict the same outcome as the InSight model.

Deep learning methods have continued to be used for the early detection of sepsis. Fagerstrom et al conducted one such study [17], where they built upon the early detection framework set up by Henry et al. in [26]. Fagerström et al [17] developed "LiSep Sepsis," an LSTM network for the early detection of sepsis. They used the same feature set and definition of sepsis as in Henry et al. [26]. The LiSep LSTM model can predict sepsis with the same Area Under the Receiver Operator Characteristic (AUROC) curve as the TREW Score model; however, it does so 48 hours [IQR (20.0, 135.0)], before the onset of sepsis. Given the success of deep learning models with predicting the early onset of sepsis [17, 8, 40], in this study we implement a WaveNet [61] inspired CNN model trained with an individual as task multitask learning scheme [34] to predict the clinical signals and vital signs that determine the onset of sepsis. We then use the definition of sepsis presented in [26, 17] to classify patients.

Fagerström et al. [17] discuss the use of a more varied patient set to allow for a generalizable model, since the MIMIC-III data set is localized to the ICU at the Beth Israel Deaconess Medical Center in Boston. Deep learning models, while accurate, tend to overfit to the training data. Models trained for medical diagnoses would benefit from being generalizable

to all patients, while also making accurate predictions on the individual level. Multitask learning models have been used, with success, to predict clinical benchmarking tasks [23]. Harutyanan et al. [23] developed four standardized benchmarks, including in-hospital mortality, length of stay and phenotype prediction using multitask trained LSTM models. Jaques et al. [34], provide a novel method to multitask where the prediction of wellness markers obtained from wearable devices, is considered a separate task and multitask learning is implemented by training as many tasks as the number of individuals simultaneously through shared hidden layers. Sharing parameters allows for the model to learn a representation that generalizes to all the patients, while making accurate individual predictions. We expect this MTL learning scheme combined with CNNs, which have shown to be accurate while modeling data with a grid-like structure (time series, images) [61, 93], to provide a model that leverages population-level data to make individual-level classifications for patients.

4.2 Methods

4.2.1 Data

We utilized data from the MIMIC III [38] database which contains deidentified EHR data including vital signs, waveform measurements for vital signs, clinical data, and lab tests, for over 59000 patients in the ICU of the Beth Israel Deaconess Medical Center (Boston). We used the feature set used by Henry et al. [26], listed in Table 4.1. However, we performed our computational experiment with a small cohort of 20 patients over the age of 40, where 10 had sepsis, and 10 did not. To define sepsis, we used the same definition for sepsis to classify patients and assign labels. The first requirement to classify a patient with sepsis is to satisfy all the SIRS [2] criteria for sepsis (1-4) and septic shock (5). These are:

1. Temperature >= 100.4◦ F or <= 96.8◦ F
2. Respiratory Rate >= 20 breaths/min
3. Heart rate >= 90 bpm
4. Total WBC count < 4000/mm^3 or >12000/mm^3

5. Systolic BP <= 90mm Hg OR Lactic Acid >0 2mg/L.

The second requirement is to have an ICD-9 code indicating any infection or any mention of infection in clinical notes. For a patient to have septic shock, along with the two criteria for sepsis a third criterion must be satisfied which is

1. Hypotension or Systolic BP <= 90mm Hg for > 30 minutes and,

2. Fluid replacement, i.e., 20ml/kg for >= 24 hours.

The data was normalized by removing the mean and dividing by the standard deviation for non binary data. We trained our network for each patient, where the training data consisted of a 27 feature matrix per patient to predict those 27 features. Missing data was imputed by carrying forward the most recent measurement if available, or with the patient or population mean. We used data for other patients to validate and test our models. For instance, patient with id 135 was trained on their own data, but validated and tested on data from other patients. We did this to test how well the single tasking (STL) model was able to generalize in comparison to its multitasking counterpart. Both the models were trained to predict two targets:

1. All the features 5 hours ahead

2. The label for sepsis in a sliding window fashion

4.2.2 Neural Network Architectures and Training

The WaveNet architecture was introduced in [61], to process raw audio signals and to be applied to text to speech synthesizers. WaveNet is an autoregressive model that models the probability $p(x_t|(x_{t-1},\cdots x_1)$, for each element x_t in an audio stream $x = \{x_1,\cdots,x_t\}$. This means an element at every time step depends on elements from the preceding time steps, much like LSTMs and RNNs. However, given the high frequency at which speech is sampled, LSTMs and RNNs struggle with modeling the long term temporal dependencies within the signal, and are computationally expensive to train. CNNs are have been shown to model time series effectively [93, 37]. The WaveNet architecture is a further improvement because of the causal dilated convolutions that are integral to its architecture. Performing

Table 4.1 Features used to train models to predict sepsis

Features
Heart Rate
Respiratory Rate
WBC
Urine Output
Systolic Blood Pressure
Diastolic Blood Pressure
Sp02
FiO2
Bicarbonate
Arterial pH
Glasgow Coma Scale (GCS)
Hematocrit
Hemoglobin
PaO2
PaCO2
Platelet Count
Potassium
Creatinine
Fluid uptake
Temperature
Lactic Acid
Systolic Blood Pressure (waveform data)
Chronic Liver Disease
Chronic Heart Disease
Chronic Organ Failure
HIV
ICD9 Codes for Infection

convolutions without dilations involves sliding a filter or weight matrix along the input. Causal dilated convolutions involve using filters with their receptive field inreasing every layer, in a way that mainitains the temporal structure of the data. This is done to make sure that data at time $t-1$ does not depend on data at time t, which may happen if convolutions

are stacked without dilations. The WaveNet model takes care of this via stacked dilated convolution layers as shown in Figure 4.1. In Figure 4.1, the dilation rate is two and we see how the filter convolves to include every 2^{ith} signal in the i^{th} layer. This means, at every output signal is a function of elements before it. This efficient handling of long term temporal dependencies makes the WaveNet. Given the time series nature of the features for sepsis prediction and that we intend to develop a model for the early prediction of sepsis, we intend to leverage the WaveNet model to do so.

The novelty in our implementation is the combination of a CNN architecture trained via an individual-as-task MTL scheme. As detailed in Chapter 2 and Chapter 3, MTL can help train similar tasks together and allow for a genralizable model to make accurate predictions. To reiterate, all shared neural network weights were trained on each iteration, and only individual specific weights in the last layer were trained with the corresponding individual data set as shown in panel a) of Figure 4.1. This means, to complete a single epoch, data from every individual must have been used to train the shared layers and the rat specific layers after the network split. This approach achieves our goal to have "individuals as tasks". In this way, we leverage population level information to make predict features and sepsis on the individual level. We trained two different network architectures, a single task learning (STL) WaveNet inspired architecture with four convolutional layers with dilated convolutions followed by dense layers that yield an output for each individual as shown in panel b of Figure 4.1 and a multitask learning version of the same, where the convolutional layers are shared. In the MTL version, the shared convolutional layers split into individual dense layers to yield individual predictions, as in panel a) of Figure 4.1. We tuned the learning rate and the regularization strength for the first dense layer as hyper-parameters. The hyper-parameters that allowed for relatively low mean squared error (MSE) when training for features, and testing on another patient, for both architectures, were a regularization strength of 0.005 and a learning rate of 0.001 using the Adam optimizer [45]. To allow for a comparison to a baseline, we also trained a simple neural network with two dense layers, with the optimal hyper parameters.
The forward propagation of information occurs as described in equations (2.59) and (2.60).

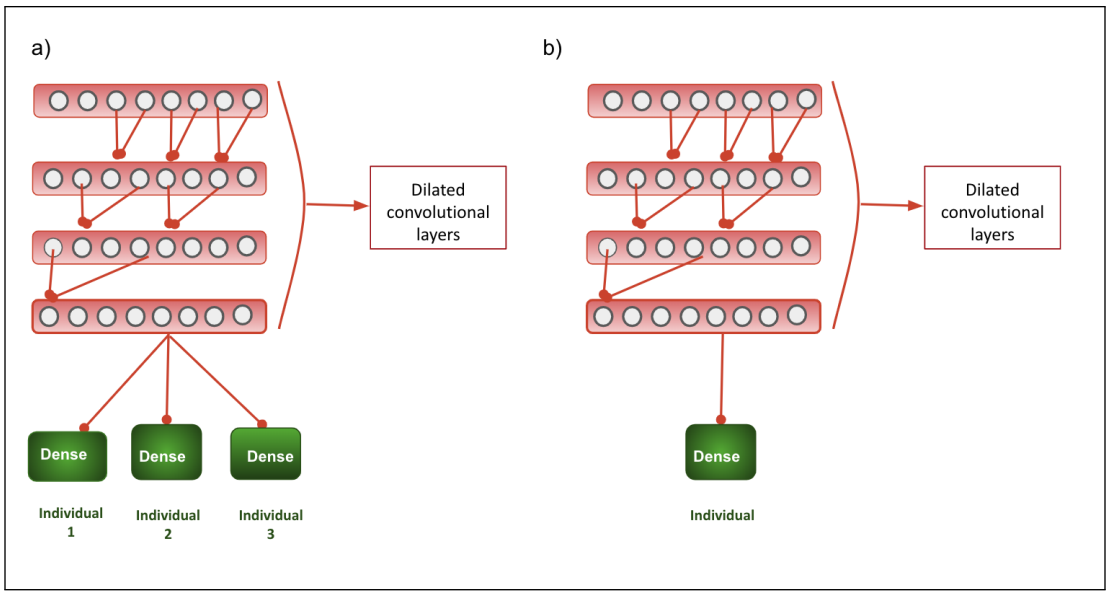


Figure 4.1 Schematic of multitask learning WaveNet architecture. The shared layers are in red and the individual-specific layers are in green.

At the backpropagation step, the loss for each individual is computed as

$$\text{CrossEntropy} = -\sum_{c=1}^{c=2} z_{t_j}^i \log(y_{t_j}^i) \tag{4.2}$$

where c is the number of classes, $z_{t_j}^i = f(x_{t_j}^i, \mathbf{w}) + e_j^i$ and, f is the function approximated by the single tasking WaveNet models/architectures we investigated. For the two MTL models, $z_{t_j}^i = f(x_{t_j}^i, \mathbf{w^{shared}}; \mathbf{w^i}) + e_{t_j}^i$.
We performed both parts of our computational experiment on a cohort of 20 patients, over the age of 40 and 10 of whom had sepsis, to test our hypothesis. We made sure there was variance within the features (i.e., the features were not static and composed of imputed mean values). This was important to allow the networks to model data that was representative of the population.

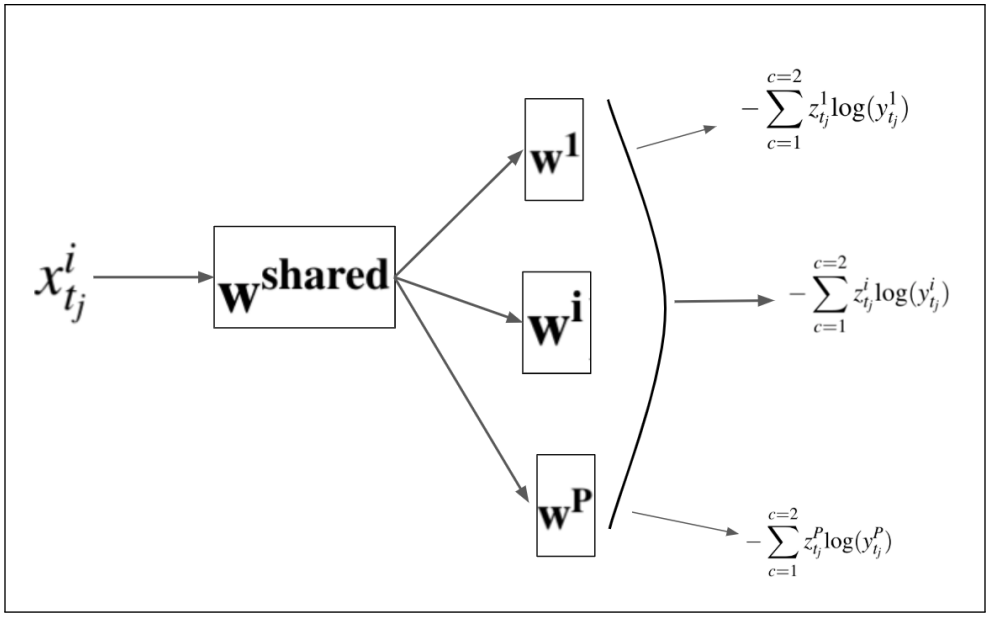


Figure 4.2 End-to-end representation of how the MTL training occurs

4.3 Results

4.3.1 Feature Predictions

We evaluated the performance all the models predicting features by computing the normalized root mean squared error. We also computed the sensitivity and specificity to determine the accuracy of the sepsis predictions made, using the features. Put simply, we first trained our models to predict features 5 hours ahead and used the predicted features to determine if the patient has sepsis or not. Figures 4.3, 4.4 and, 4.5 show examples of feature predictions for four integral features involved in predicting sepsis made by the MTL WaveNet, STL WaveNet and Vanilla NN models, respectively.

We trained ten models, both STL and MTL, on data from patients with sepsis. We shall refer to these as sepsis patient models. The remaining ten were trained on data from patients without sepsis and shall referred to as non-sepsis patient models.

Consider the plots for the heart rate (top most panel, Figure 4.3) for the STL and Vanilla NN models. We see that while the predictions vary with time (like the true heart rate), the

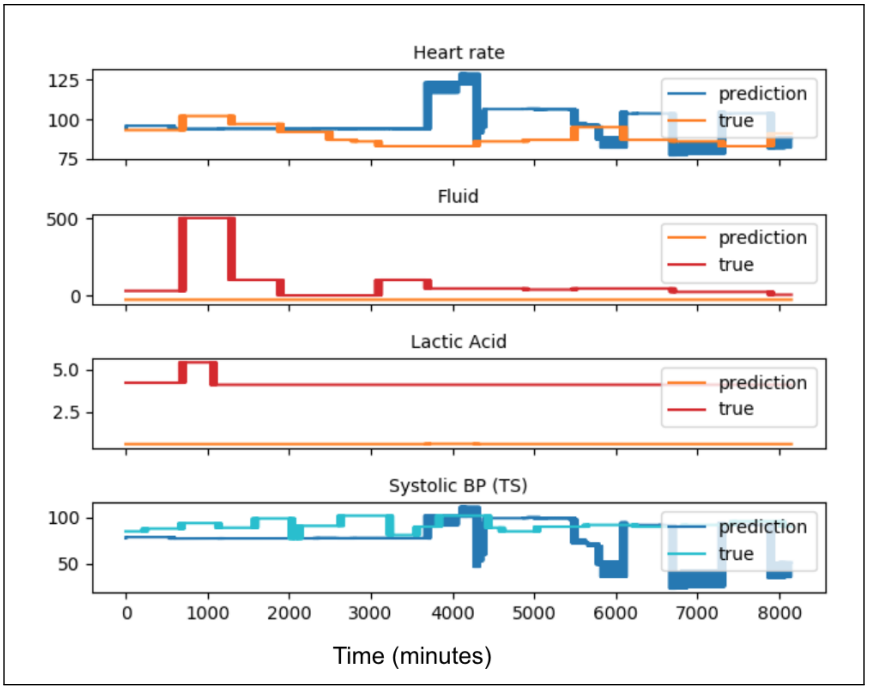


Figure 4.3 Example of feature predictions 5 hours ahead for the heart rate, fluid intake, lactic acid and systolic blood pressure, for the patient with id 10315, using the STL model trained on data from the patient with id 154

patterns in which they do so do not match the true. This property holds for the predictions for the other three features as well. On the other hand, the MTL model predicts a constant value for all the four features.

As stated in Section 4.2, we trained all the models for a patient on their own data and validated on data from other patients. To test our models, we used a cohort of 20 patients with an equal number of people with and without sepsis. Each model was tested on each patient in the test cohort. We computed and plotted the inter-individual normalized mean squared mean squared error between the prediction and the true values for all the four features, in Figure 4.6. We found that the MTL model had the lower inter-individual NRMSE (Figure 4.6, Mann-Whitney U-test,$p<0.05$) for the heart rate predictions. The difference between the NRMSE for the other features for the MTL and STL models are not statistically significant. However, the NRMSE for the baseline Vanilla NN model is consistently high for all the features, highlighting the model's poor performance in predicting features.

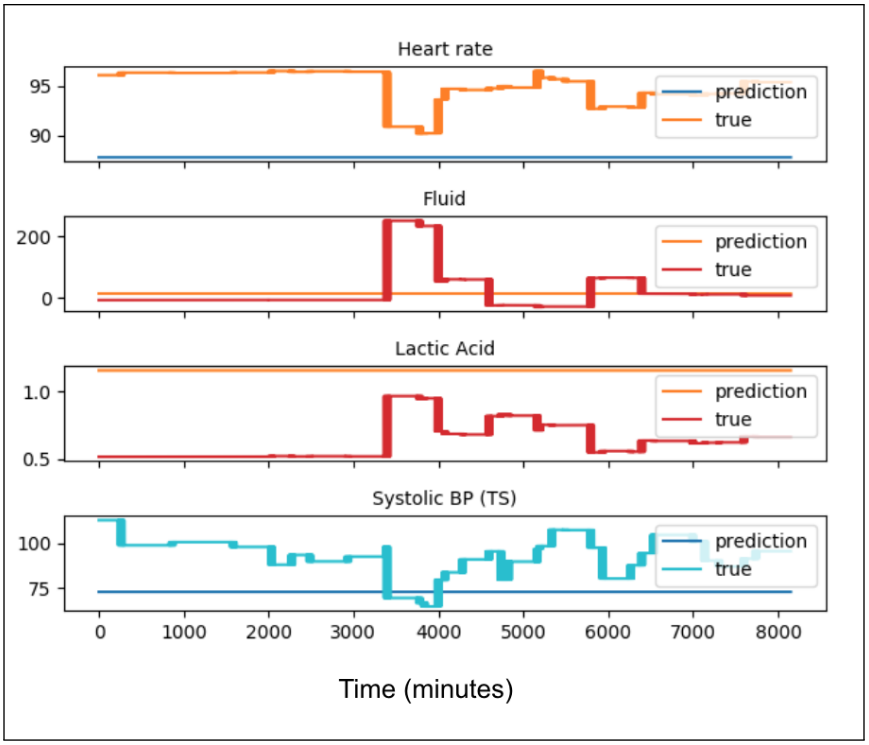


Figure 4.4 Example of feature predictions 5 hours ahead for the heart rate, fluid intake, lactic acid and systolic blood pressure, for patient with id 10315, using the MTL model trained on data from patient with id 154

4.3.2 Sepsis Prediction

We used the feature predictions for five hours ahead to determine if a patient has sepsis or not by using the same set of conditions described in section 4.2. We used the sepsis predictions to compute the sensitivity and specificity of the STL WaveNet, MTL WaveNet, and Vanilla NN models, trained for each patient. The average sensitivity and specificity for all the models over the entire test cohort are presented in Table 4.2. We define the sensitivity as

$$\text{sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False negatives}} \times 100\% \tag{4.3}$$

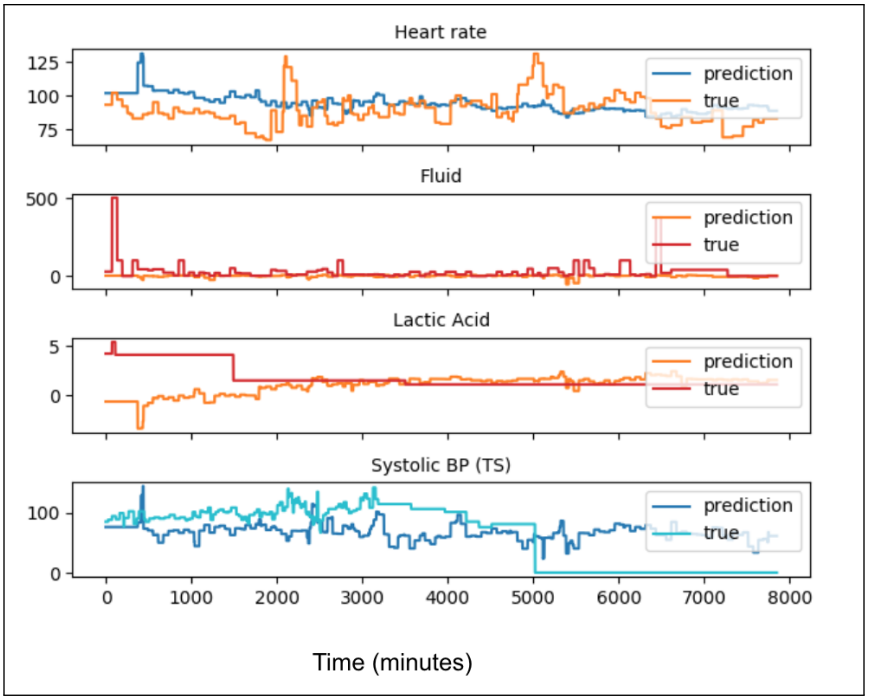


Figure 4.5 Example of feature predictions 5 hours ahead for the heart rate, fluid intake, lactic acid and systolic blood pressure, for the patient with id 10315, using the baseline, Vanilla NN model trained on data from a patient with id 154

and, the specificity as,

$$\text{specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}} \times 100\% \tag{4.4}$$

where the true positives are the number of patients correctly identified to have sepsis and, the true negatives are the number of patients correctly identified not to have sepsis. Furthermore, false positives are the number of patients the model detected to have sepsis, who do not have sepsis and false negatives are the number of patients who had sepsis, but the model failed to detect them as so.

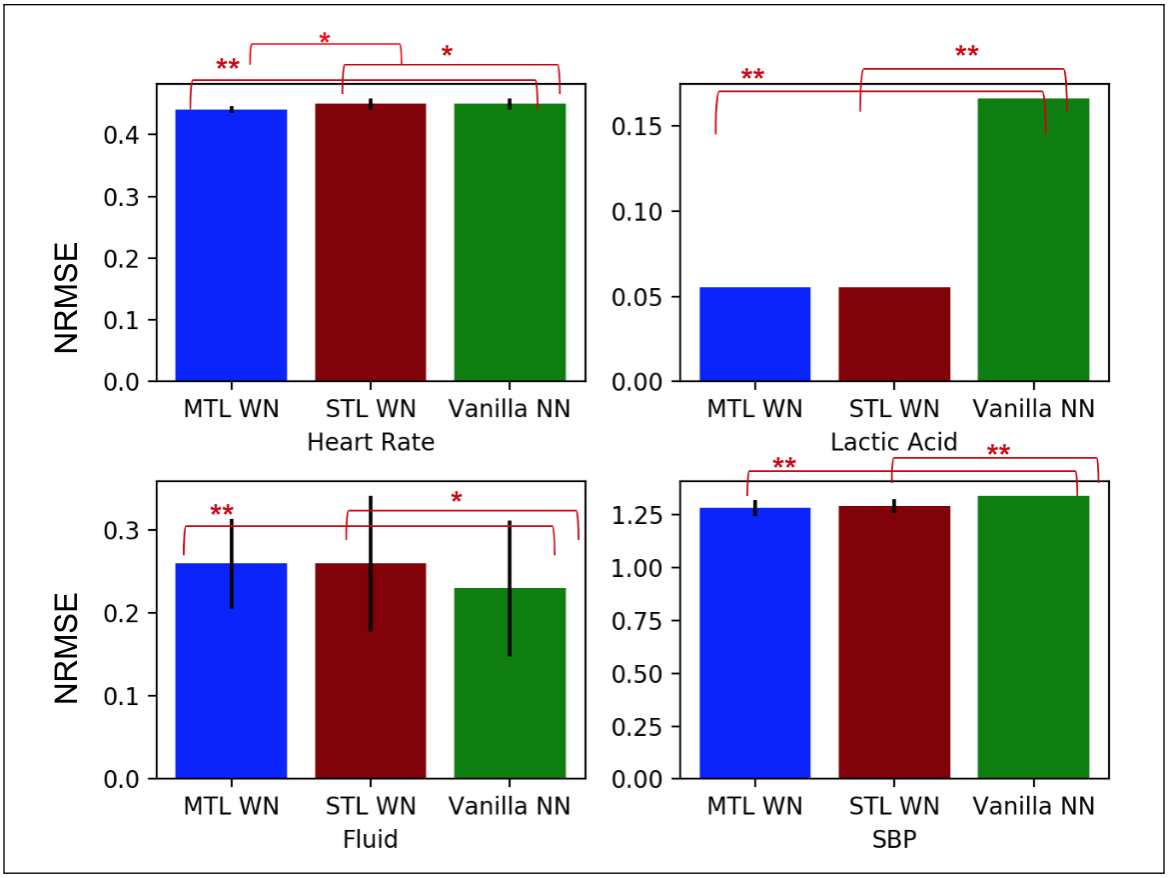


Figure 4.6 Inter-individual mean squared error for the four features (heart rate, lactic acid, fluid uptake and systolic blood pressure), for both the models (** indicates P < 0.005 and * indicates P< 0.05 Mann-Whitney U-test). Note, error bars in the SBP and Lactic acid panels were too small for the scale.

Table 4.2 Sensitivity and Specificity for the sepsis predictions from all models, tested on all the patients in the test cohort, using feature predictions and the sepsis definition from section 4.2

Metric	MTL WaveNet	STL WaveNet	Vanilla NN
Sensitivity	40.4%	38.4%	0.0%
Specificity	70.0%	72.1%	99.0%

We noticed that sepsis patient models were more accurate with distinguishing between patients who have sepsis and those who don't, than the non-sepsis patient models. Nine out of ten non-sepsis patient models predicted that no one has sepsis despite there being ten patients with sepsis, in the test set. We decided to compute the sensitivity and specificity only for the sepsis patient models, which we list in Table 4.3. We note that the sensitivity

Table 4.3 Sensitivity and Specificity for the sepsis predictions from all the "sepsis patient models," tested on all the patients in the test cohort using feature predictions and the sepsis definition from section 4.2

Metric	MTL WaveNet	STL WaveNet	Vanilla NN
Sensitivity	72.9%	72.2%	0.0%
Specificity	46.9%	47.7%	99%

of models trained on patients with sepsis see an increase in sensitivity, but decrease in specificity. The Vanilla NN model consistently predicts every patient not to have sepsis. As a result, it has a high false positive error and a sensitivity of 0.0 regardless of the data it was trained on (sepsis or no sepsis).

The second half of our computational experiment involved training the STL and MTL WaveNet models to predict the labels directly, instead of features, five hours ahead. The sensitivity and specificity for the models are detailed in Table 4.4. We note that the MTL WaveNet model trained to predict label does better than its single-tasking counterpart, and the MTL and STL WaveNet models trained to predict features. We also checked to see if the sepsis patient models better at distinguishing patients with or without sepsis and we report sensitivity and specificity for the sepsis patient models in Table 4.5. The sensitivity and specificity for the MTL model drop, but those for the STL model increase.

Table 4.4 Sensitivity and Specificity for the WaveNet models trained on data from the entire cohort to predict labels. The models were tested on all 20 patients in the test cohort

Metric	MTL WaveNet	STL WaveNet
Sensitivity	70%	31.5%
Specificity	70%	68.4%

Table 4.5 Sensitivity and Specificity for both of the WaveNet "sepsis patient models" trained to predict labels, tested on all the patients in the test cohort

Metric	MTL WaveNet	STL WaveNet
Sensitivity	40%	40.0%
Specificity	40%	60.0%

4.4 Discussion

Our results indicate that the MTL WaveNet model yields feature predictions that are as accurate as those of the STL WaveNet model. We hypothesized MTL to regularize the models and prevent their overfitting to the training data, despite the variance among patients. In turn, the MTL model regularized and predicted the mean for features and was unable to capture the time-series variation within the features. However, the MTL model outperformed the STL version with regards to the sepsis classification using the feature predictions, yielding a higher sensitivity. Since the sepsis classification is made after a series of threshold considerations for clinical signals, it is possible that the mean predicted by the MTL model was close enough to the true value to allow for accurate classifications. The overall sensitivity (averaged over sepsis and no sepsis patient models) for the MTL WaveNet model was higher than the STL models. This suggests that MTL framework is able to leverage population data to inform the individual. The representation learned by the MTL trained no sepsis patient models is one that favors all the other tasks (including the sepsis patient models), allowing them to detect patients with sepsis, better.

For the second part of our computational experiment, we trained both the architectures to predict class labels. We observe that the overall sensitivity and specificity are higher for the MTL model, in comparison to its STL counterpart. Both of these metrics drop (from 70%to40% for the MTL WaveNet), when considering only the sepsis patient models. Results in table 4.5 report the same sensitivities for both the models, and a higher specificity for the STL version. A higher overall sensitivity for the MTL model further corroborates the ability of the multitask learning algorithm to leverage data from both the sepsis and non sepsis patient models and inform the non-sepsis patient models to better identify patients who have sepsis. STL training fails in this regard, thereby yielding a lower overall sensitivity. The models we used were only trained on data from 20 patients; however, our framework has the potential to use the data from thousands of patients. This will allow the MTL model to learn from more data and deliver more accurate predictions.

CHAPTER

5

CONCLUSION AND FUTURE WORK

In this thesis, we investigated the application of different neural networks trained using a multitask learning scheme for predicting biomedical time series data. The two specific applications we considered were predicting bladder pressure and bladder contractions, and the early detection of sepsis. In this concluding chapter, we highlight our contributions, evaluate the performance of our models, and suggest possible areas of future work.

5.1 Predicting Bladder Pressure and Bladder Contraction in rats

Neurological diseases or spinal cord injuries can cause overactive bladders. This condition can be prevented by applying electrical stimulation at the onset of a bladder contraction. Therefore, accurately predicting bladder pressure and bladder contractions is important to apply the stimulation at the appropriate time. There are computational models to predict

bladder pressure [31, 75, 57]. Our investigation is an extension of the work done by Rutter et al. [75] as we used the same dataset and a similar formulation of the problem, where we used the spectrogram representation of the EUS EMG data from [46] to predict bladder pressure. We then used the pressure predictions to detect bladder contractions using the algorithm provided by [46]. However, we used individual as task MTL trained neural networks to predict the bladder pressure. Our contributions are summarized below.

1. We use RNNS to learn from and make bladder contraction predictions with the spectrogram representation of neurostimulation (EUS EMG and bladder pressure) data for the first time.

2. We use a novel combination of individuals as tasks MTL trained RNNs to predict bladder contractions and bladder predictions from neurostimulation data .

The data that we use has few samples (12 rats) but a very dense feature matrix. The methods we propose should translate well to other medical datasets where there is high heterogeneity, low sample size, and large dimensionality of the data. While Multitask learning considerably improves upon both the NN and RNN architectures in our study, our results, Figure 3.3, also serve as an improvement upon the results reported in [75]. We note that the RNN models outperform their NN counterparts, yielding both the lowest inter-individual and intra-individual RMSE. The mean inter-individual correlation coefficient was reasonably high for the MTLRNN model.
For future work, we will investigate how the MTLRNN model can be used to reliably predict changes in bladder pressure from the EUS EMG signal. If so, using RNNs would be appropriate because RNNs could be used to forecast future bladder pressure levels without an EMG signal, e.g., using an encoder-decoder architecture [82]. An autoencoder and LSTM decoder architecture was used in [31] and they report NRMSE scores of $14.9 \pm 4.8\%$. Applying MTL to such a model might yield improved model performance. Alternatively, we might consider implementing a WaveNet [61] architecture to process the EUS EMG signal and yield bladder pressure predictions. Training the model using an individual-as-task MTL [34] scheme could allow for a generalized MTL WaveNet model that accurately predicts bladder pressure.

5.2 Early detection of Sepsis

Early detection of sepsis is integral to allow for early treatment therapy [2, 55]. Machine Learning methods can be applied to EHR data to help achieve this goal. Current computational methods include deep learning neural networks [17], [50] and statistical models like the Cox Hazard model [26]. In this thesis, we attempt to build upon the work in [26]. We decided to use MTL trained neural networks MTL on the same set of data from the MIMIC-III [38], with a similar set of features and the same sepsis definition. We trained and tested MTL and STL versions of a WaveNet [61] inspired architecture, along with a Vanilla NN, to provide a baseline for comparison. We first predicted all the features five hours ahead and used those features to determine if a patient has sepsis or not. Next, we trained the model to predict labels indicating sepsis directly, five hours ahead. Our contributions can be summarized below:

1. We use 1-D convolutions through a WaveNet architecture , for the first time, to learn from and make sepsis predictions with EHR data.

2. We train the WaveNet model using a multitask learning algorithm, with individuals as tasks, also for the first time, to predict sepsis for each individual.

We reported a sensitivity of 70% for the MTL WaveNet models predicting labels. We noticed that the MTL algorithm improved performance and allowed more no-sepsis patient models to detect sepsis. The highest sensitivity was comparable to others reported in the literature. We recognize that the sensitivity of the MTL model we report is for out of sample predictions. To improve results, we would need to train on a cohort larger than 20 patients. We expect that increasing the sample size, by including more patients from the MIMIC-III dataset, would improve performance and allow better generalization. However, this still restricts the dataset to ICU patients. The goal is to predict sepsis for a patient before they get to the ICU. This can be achieved by using a more diverse dataset and attempting to make predictions much earlier than we currently do. We are currently making predictions five hours before onset, which is later than [17] and [26]. A richer and more diverse dataset will help achieve both goals of developing a more generalizable model that is able to make early enough predictions so that early treatment therapy [2, 55] can be administered to patients.

BIBLIOGRAPHY

[1] Amato, Filippo, López, Alberto, Peña-Méndez, Eladia Marıa, Vaňhara, Petr, Hampl, Aleš & Havel, Josef. "Artificial neural networks in medical diagnosis" (2013).

[2] Angus, Derek C & Van der Poll, Tom. "Severe sepsis and septic shock". *N Engl J Med* **369** (2013), pp. 840–851.

[3] Banks, H Thomas, Kenz, Zackary R & Thompson, W Clayton. "A review of selected techniques in inverse problem nonparametric probability distribution estimation". *Journal of Inverse and Ill-Posed Problems* **20**.4 (2012), pp. 429–460.

[4] Barriere, Steven L & Lowry, Stephen F. "An overview of mortality risk prediction in sepsis". *Critical care medicine* **23**.2 (1995), pp. 376–393.

[5] Baxter, Jonathan. "A Bayesian/information theoretic model of learning to learn via multiple task sampling". *Machine learning* **28**.1 (1997), pp. 7–39.

[6] Bengio, Yoshua, Simard, Patrice & Frasconi, Paolo. "Learning long-term dependencies with gradient descent is difficult". *IEEE transactions on neural networks* **5**.2 (1994), pp. 157–166.

[7] Bruns, Tim M, Gaunt, Robert A & Weber, Douglas J. "Estimating bladder pressure from sacral dorsal root ganglia recordings". *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, pp. 4239–4242.

[8] Calvert, Jacob S, Price, Daniel A, Chettipally, Uli K, Barton, Christopher W, Feldman, Mitchell D, Hoffman, Jana L, Jay, Melissa & Das, Ritankar. "A computational approach to early sepsis detection". *Computers in biology and medicine* **74** (2016), pp. 69–73.

[9] Caruana, Rich. "Multitask learning". *Mach. Learn.* **28**.1 (1997), pp. 41–75.

[10] Che, Zhengping, Purushotham, Sanjay, Cho, Kyunghyun, Sontag, David & Liu, Yan. "Recurrent neural networks for multivariate time series with missing values". *Sci. Rep.* **8**.1 (2018), p. 6085.

[11] Collobert, Ronan & Weston, Jason. "A unified architecture for natural language processing: Deep neural networks with multitask learning". *ICML*. ACM. 2008, pp. 160–167.

[12] Craggs, MD, Edhem, I, Knight, SL, McFarlane, JP & Shah, N. “Suppression of normal human voiding reflexes by electrical stimulation of the dorsal penile nerve”. *Journal of Physiology-London*. Vol. 507. Wiley-Blackwell Commerce Place, 350 Main ST, Malden 02148, MA USA. 1998, 19P–20P.

[13] Dalmose, AL, Rijkhoff, NJM, Kirkeby, Hans Jørgen, Nohr, M, Sinkjær, Thomas & Djurhuus, Jens Christian. “Conditional stimulatzion of the dorsal penile/clitoral nerve may increase cystometric capacity in patients with spinal cord injury”. *Neurourology and urodynamics* **22**.2 (2003), pp. 130–137.

[14] Davidian, Marie & Giltinan, David M. “Nonlinear models for repeated measurement data: an overview and update”. *Journal of agricultural, biological, and environmental statistics* **8**.4 (2003), p. 387.

[15] Deng, Li, Hinton, Geoffrey & Kingsbury, Brian. “New types of deep neural network learning for speech recognition and related applications: An overview”. *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8599–8603.

[16] Desmée, Solène, Mentré, France, Veyrat-Follet, Christine, Sébastien, Bernard & Guedj, Jérémie. “Nonlinear joint models for individual dynamic prediction of risk of death using Hamiltonian Monte Carlo: application to metastatic prostate cancer”. *BMC Medical Research Methodology* **17**.1 (2017), p. 105.

[17] Fagerström, Josef, Bång, Magnus, Wilhelms, Daniel & Chew, Michelle S. “LiSep LSTM: A Machine Learning Algorithm for Early Detection of Septic Shock”. *Scientific reports* **9**.1 (2019), pp. 1–8.

[18] Gelman, Andrew, Bois, Frederic & Jiang, Jiming. “Physiological pharmacokinetic analysis using population modeling and informative prior distributions”. *Journal of the American Statistical Association* **91**.436 (1996), pp. 1400–1412.

[19] Ghanem-Zoubi, Nesrin O, Vardi, Moshe, Laor, Arie, Weber, Gabriel & Bitterman, Haim. “Assessment of disease-severity scoring systems for patients with sepsis in general internal medicine departments”. *Critical Care* **15**.2 (2011), R95.

[20] Goodfellow, Ian, Bengio, Yoshua & Courville, Aaron. *Deep learning*. MIT press, 2016.

[21] Granato, Chiara, Korstanje, Cees, Guilloteau, Véronique, Rouget, Céline, Palea, Stefano & Gillespie, James I. “Prostaglandin E2 excitatory effects on rat urinary bladder: a

comparison between the β-adrenoceptor modulation of non-voiding activity in vivo and micro-contractile activity in vitro". *N-S Arch. Pharmacol.* **388** (2015), pp. 727–735.

[22] Häbler, HJ, Jänig, W & Koltzenburg, M. "Myelinated primary afferents of the sacral spinal cord responding to slow filling and distension of the cat urinary bladder." *The Journal of physiology* **463**.1 (1993), pp. 449–460.

[23] Harutyunyan, Hrayr, Khachatrian, Hrant, Kale, David C, Ver Steeg, Greg & Galstyan, Aram. "Multitask learning and benchmarking with clinical time series data". *Scientific data* **6**.1 (2019), pp. 1–18.

[24] Hassanzadeh, Hamid Reza & Wang, May D. "DeeperBind: Enhancing prediction of sequence specificities of DNA binding proteins". *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE. 2016, pp. 178–183.

[25] Haykin, Simon. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[26] Henry, Katharine E, Hager, David N, Pronovost, Peter J & Saria, Suchi. "A targeted real-time early warning score (TREWScore) for septic shock". *Science translational medicine* **7**.299 (2015), 299ra122–299ra122.

[27] Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **6**.02 (1998), pp. 107–116.

[28] Hochreiter, Sepp & Schmidhuber, Jürgen. "Long short-term memory". *Neural computation* **9**.8 (1997), pp. 1735–1780.

[29] Hornik, Kurt, Stinchcombe, Maxwell, White, Halbert, et al. "Multilayer feedforward networks are universal approximators." *Neural networks* **2**.5 (1989), pp. 359–366.

[30] Iggo, A. "Tension receptors in the stomach and the urinary bladder". *The Journal of physiology* **128**.3 (1955), pp. 593–607.

[31] Jabbari, Milad & Erfanian, Abbas. "estimation of Bladder pressure and Volume from the neural Activity of Lumbosacral Dorsal Horn Using a Long-Short-term-Memory-based Deep neural network". *Scientific Reports* **9**.1 (2019), pp. 1–15.

[32] James, Gareth, Witten, Daniela, Hastie, Trevor & Tibshirani, Robert. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[33] Jameson, J Larry & Longo, Dan L. “Precision medicine—personalized, problematic, and promising”. *Obstetrical & gynecological survey* **70**.10 (2015), pp. 612–614.

[34] Jaques, Natasha, Taylor, Sara, Nosakhare, Ehimwenma, Sano, Akane & Picard, Rosalind. “Multi-task learning for predicting health, stress, and happiness”. *NIPS Workshop on Machine Learning for Healthcare*. 2016.

[35] Jesorsky, Oliver, Kirchberg, Klaus J & Frischholz, Robert W. “Robust face detection using the hausdorff distance”. *International conference on audio-and video-based biometric person authentication*. Springer. 2001, pp. 90–95.

[36] Jezernik, Sašo, Craggs, Michael, Grill, Warren M, Creasey, Graham & Rijkhoff, Nico JM. “Electrical stimulation for the treatment of bladder dysfunction: current status and future possibilities”. *Neurological research* **24**.5 (2002), pp. 413–430.

[37] Jin, Lin-peng & Dong, Jun. “Ensemble deep learning for biomedical time series classification”. *Computational intelligence and neuroscience* (2016).

[38] Johnson, Alistair EW, Pollard, Tom J, Shen, Lu, Li-wei, H Lehman, Feng, Mengling, Ghassemi, Mohammad, Moody, Benjamin, Szolovits, Peter, Celi, Leo Anthony & Mark, Roger G. “MIMIC-III, a freely accessible critical care database”. *Scientific data* **3** (2016), p. 160035.

[39] Jones, Alan E, Trzeciak, Stephen & Kline, Jeffrey A. “The Sequential Organ Failure Assessment score for predicting outcome in patients with severe sepsis and evidence of hypoperfusion at the time of emergency department presentation”. *Critical care medicine* **37**.5 (2009), p. 1649.

[40] Kam, Hye Jin & Kim, Ha Young. “Learning representations for the early detection of sepsis with deep neural networks”. *Computers in biology and medicine* **89** (2017), pp. 248–255.

[41] Kanai, Sekitoshi, Fujiwara, Yasuhiro & Iwamura, Sotetsu. “Preventing gradient explosions in gated recurrent units”. *Advances in neural information processing systems*. 2017, pp. 435–444.

[42] Karpathy, Andrej et al. “Cs231n convolutional neural networks for visual recognition”. *Neural networks* **1** (2016), p. 1.

[43] Kennelly, Michael J, Bennett, Maria E, Grill, Warren M, Grill, Julie H & Boggs, Joseph W. “Electrical stimulation of the urethra evokes bladder contractions and emptying in

spinal cord injury men: case studies". *The journal of spinal cord medicine* **34**.3 (2011), pp. 315–321.

[44] Kim, Youngjoo & Bang, Hyochoong. "Introduction to Kalman Filter and Its Applications". *Introduction and Implementations of the Kalman Filter*. IntechOpen, 2018.

[45] Kingma, Diederik P & Ba, Jimmy. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980* (2014).

[46] Langdale, Christopher L., Hokanson, James A., Sridhar, Arun Raghav Mahankali & Grill, Warren M. "Stimulation of the pelvic nerve increases bladder capacity in the prostaglandin E2 rat model of overactive bladder." *American Journal of Physiology-Renal Physiology* **313 3** (2017), F657–F665.

[47] LeCun, Yann, Bengio, Yoshua, et al. "Convolutional networks for images, speech, and time series". *The handbook of brain theory and neural networks* **3361**.10 (1995), p. 1995.

[48] Lipton, Zachary C., Kale, David C., Elkan, Charles & Wetzel, Randall. *Learning to Diagnose with LSTM Recurrent Neural Networks*. 2015. arXiv: `1511.03677 [cs.LG]`.

[49] Liu, Pengfei, Qiu, Xipeng & Huang, Xuanjing. "Recurrent neural network for text classification with multi-task learning". *arXiv preprint arXiv:1605.05101* (2016).

[50] Lukaszewski, Roman A, Yates, Adam M, Jackson, Matthew C, Swingler, Kevin, Scherer, John M, Simpson, AJ, Sadler, Paul, McQuillan, Peter, Titball, Richard W, Brooks, Timothy JG, et al. "Presymptomatic prediction of sepsis in intensive care unit patients". *Clin. Vaccine Immunol.* **15**.7 (2008), pp. 1089–1094.

[51] Maas, Andrew L, Daly, Raymond E, Pham, Peter T, Huang, Dan, Ng, Andrew Y & Potts, Christopher. "Learning word vectors for sentiment analysis". *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics. 2011, pp. 142–150.

[52] Mikulich, Susan K, Zerbe, Gary O, Jones, Richard H & Crowley, Thomas J. "Comparing linear and nonlinear mixed model approaches to cosinor analysis". *Statistics in medicine* **22**.20 (2003), pp. 3195–3211.

[53] Moein, S, Monadjemi, SA & Moallem, P. “A novel fuzzy-neural based medical diagnosis system”. *International Journal of Biological & Medical Sciences* **4**.3 (2009), pp. 146–150.

[54] Naumova, Valeriya, Pereverzyev, Sergei V & Sivananthan, Sivananthan. “A meta-learning approach to the regularized learning—Case study: Blood glucose prediction”. *Neural Networks* **33** (2012), pp. 181–193.

[55] Nguyen, H Bryant, Rivers, Emanuel P, Abrahamian, Fredrick M, Moran, Gregory J, Abraham, Edward, Trzeciak, Stephen, Huang, David T, Osborn, Tiffany, Stevens, Dennis, Talan, David A, et al. “Severe sepsis and septic shock: review of the literature and emergency department management guidelines”. *Annals of emergency medicine* **48**.1 (2006), 54–e1.

[56] Nguyen, Ngoc Giang, Tran, Vu Anh, Ngo, Duc Luu, Phan, Dau, Lumbanraja, Favorisen Rosyking, Faisal, Mohammad Reza, Abapihi, Bahriddin, Kubo, Mamoru, Satou, Kenji, et al. “DNA sequence classification by convolutional neural network”. *Journal of Biomedical Science and Engineering* **9**.05 (2016), p. 280.

[57] Niederhauser, Thomas, Gafner, Elena S, Cantieni, Tarcisi, Grämiger, Michelle, Haeberlin, Andreas, Obrist, Dominik, Burkhard, Fiona & Clavica, Francesco. “Detection and quantification of overactive bladder activity in patients: Can we make it better and automatic?” *Neurourology and urodynamics* **37**.2 (2018), pp. 823–831.

[58] Nwankpa, Chigozie, Ijomah, Winifred, Gachagan, Anthony & Marshall, Stephen. “Activation functions: Comparison of trends in practice and research for deep learning”. *arXiv preprint arXiv:1811.03378* (2018).

[59] O’Brien Jr, James M, Ali, Naeem A, Aberegg, Scott K & Abraham, Edward. “Sepsis”. *The American journal of medicine* **120**.12 (2007), pp. 1012–1022.

[60] Olah, Christopher. *Understanding LSTM Networks*. 2015. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[61] Oord, Aaron van den, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew & Kavukcuoglu, Koray. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: `1609.03499 [cs.SD]`.

[62] Paliwal, Mukta & Kumar, Usha A. “Neural networks and statistical techniques: A review of applications”. *Expert systems with applications* **36**.1 (2009), pp. 2–17.

[63] Pang, Bo & Lee, Lillian. “A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts”. *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics. 2004, p. 271.

[64] Pascanu, Razvan, Mikolov, Tomas & Bengio, Yoshua. “On the difficulty of training recurrent neural networks”. *International conference on machine learning*. 2013, pp. 1310–1318.

[65] Pham, Trang, Tran, Truyen, Phung, Dinh & Venkatesh, Svetha. “Predicting healthcare trajectories from medical records: A deep learning approach”. *Journal of biomedical informatics* **69** (2017), pp. 218–229. ISSN: 1532-0464.

[66] Phillips, P Jonathon, Moon, Hyeonjoon, Rizvi, Syed A & Rauss, Patrick J. “The FERET evaluation methodology for face-recognition algorithms”. *IEEE Transactions on pattern analysis and machine intelligence* **22**.10 (2000), pp. 1090–1104.

[67] Pilling, Graham M, Kirkwood, Geoffrey P & Walker, Stephen G. “An improved method for estimating individual growth variability in fish, and the correlation between von Bertalanffy growth parameters”. *Canadian Journal of Fisheries and Aquatic Sciences* **59**.3 (2002), pp. 424–432.

[68] Pinheiro, José & Bates, Douglas. *Mixed-effects models in S and S-PLUS*. Springer Science & Business Media, 2006.

[69] Plis, Kevin, Bunescu, Razvan, Marling, Cindy, Shubrook, Jay & Schwartz, Frank. “A machine learning approach to predicting blood glucose levels for diabetes management”. *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*. 2014.

[70] Ribba, Benjamin, Kaloshi, Gentian, Peyre, Mathieu, Ricard, Damien, Calvez, Vincent, Tod, Michel, Čajavec-Bernard, Branka, Idbaih, Ahmed, Psimaras, Dimitri, Dainese, Linda, et al. “A tumor growth inhibition model for low-grade glioma treated with chemotherapy or radiotherapy”. *Clinical Cancer Research* **18**.18 (2012), pp. 5071–5080.

[71] Ross, Shani E, Ouyang, Zhonghua, Rajagopalan, Sai & Bruns, Tim M. “Evaluation of decoding algorithms for estimating bladder pressure from dorsal root ganglia neural recordings”. *Ann. Biomed. Eng.* **46**.2 (2018), pp. 233–246.

[72] Ross, Shani E, Sperry, Zachariah J, Mahar, Colin M & Bruns, Tim M. “Hysteretic behavior of bladder afferent neurons in response to changes in bladder pressure”. *BMC neuroscience* **17**.1 (2016), p. 57.

[73] Ruder, Sebastian. “An overview of multi-task learning in deep neural networks”. *arXiv preprint arXiv:1706.05098* (2017).

[74] Rumelhart, David E, Hinton, Geoffrey E & Williams, Ronald J. “Learning representations by back-propagating errors”. *nature* **323**.6088 (1986), pp. 533–536.

[75] Rutter, Erica M, Langdale, Christopher L, Hokanson, James A, Hamilton, Franz, Tran, Hien, Grill, Warren M & Flores, Kevin B. “Detection of bladder contractions from the activity of the external urethral sphincter in rats using sparse regression”. *IEEE T. Neur. Sys. Reh.* **26**.8 (2018), pp. 1636–1644.

[76] Shah, N, Edhem, I, Knight, S, Shah, J & Craggs, M. “Acute suppression of provoked detrusor hyperreflexia by electrical stimulation of the dorsal penile nerve”. *Eur Urol* **33**.Suppl 1 (1998), p. 60.

[77] Sheiner, Lewis B, Rosenberg, Barr & Marathe, Vinay V. “Estimation of population characteristics of pharmacokinetic parameters from routine clinical data”. *Journal of pharmacokinetics and biopharmaceutics* **5**.5 (1977), pp. 445–479.

[78] Simon, Dan. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[79] Socher, Richard, Pennington, Jeffrey, Huang, Eric H, Ng, Andrew Y & Manning, Christopher D. “Semi-supervised recursive autoencoders for predicting sentiment distributions”. *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2011, pp. 151–161.

[80] Song, Huan, Rajan, Deepta, Thiagarajan, Jayaraman J & Spanias, Andreas. “Attend and diagnose: Clinical time series analysis using attention models”. *Thirty-second AAAI conference on artificial intelligence*. 2018.

[81] Staatz, Christine E, Willis, Charlene, Taylor, Paul J, Lynch, Stephen V & Tett, Susan E. “Toward better outcomes with tacrolimus therapy: population pharmacokinetics and individualized dosage prediction in adult liver transplantation”. *Liver Transplantation* **9**.2 (2003), pp. 130–137.

[82] Sutskever, Ilya, Vinyals, Oriol & Le, Quoc V. “Sequence to Sequence Learning with Neural Networks”. *Proc. NIPS*. Montreal, CA, 2014. URL: http://arxiv.org/abs/1409.3215.

[83] Tu, Jack V. “Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes”. *Journal of clinical epidemiology* **49**.11 (1996), pp. 1225–1231.

[84] Übeyli, Elif Derya. “Combining recurrent neural networks with eigenvector methods for classification of ECG beats”. *Digital Signal Processing* **19**.2 (2009), pp. 320–329.

[85] Vandenberghe, Frederik, Guidi, Monia, Choong, Eva, Gunten, Armin von, Conus, Philippe, Csajka, Chantal & Eap, Chin B. “Genetics-based population pharmacokinetics and pharmacodynamics of risperidone in a psychiatric cohort”. *Clinical pharmacokinetics* **54**.12 (2015), pp. 1259–1272.

[86] Vodusek, David B, Light, J Keith & Libby, James M. “Detrusor inhibition induced by stimulation of pudendal nerve afferents”. *Neurourology and urodynamics* **5**.4 (1986), pp. 381–389.

[87] Wenzel, Brian J, Boggs, Joseph W, Gustafson, Kenneth J, Creasey, Graham H & Grill, Warren M. “Detection of neurogenic detrusor contractions from the activity of the external anal sphincter in cat and human”. *Neurourology and Urodynamics: Official Journal of the International Continence Society* **25**.2 (2006), pp. 140–147.

[88] Wolpert, David H & Macready, William G. “No free lunch theorems for optimization”. *IEEE transactions on evolutionary computation* **1**.1 (1997), pp. 67–82.

[89] Zecchin, Chiara, Facchinetti, Andrea, Sparacino, Giovanni, De Nicolao, Giuseppe & Cobelli, Claudio. “A new neural network approach for short-term glucose prediction using continuous glucose monitoring time-series and meal information”. *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2011, pp. 5653–5656.

[90] Zeng, Haoyang, Edwards, Matthew D, Liu, Ge & Gifford, David K. “Convolutional neural network architectures for predicting DNA–protein binding”. *Bioinformatics* **32**.12 (2016), pp. i121–i127.

[91] Zhang, Cha & Zhang, Zhengyou. “Improving multiview face detection with multi-task deep convolutional neural networks”. *IEEE Winter Conference on Applications of Computer Vision*. IEEE. 2014, pp. 1036–1041.

[92] Zhang, Cha & Zhang, Zhengyou. “Winner-take-all multiple category boosting for multi-view face detection” (2009).

[93] Zhao, Bendong, Lu, Huanzhang, Chen, Shangfeng, Liu, Junliang & Wu, Dongya. “Convolutional neural networks for time series classification”. *Journal of Systems Engineering and Electronics* **28**.1 (2017), pp. 162–169.