Stockholms universitet

Predicting Drug Effects

Using Bayesian Artificial Neural Networks for Drug Discovery

Carl Samuelsson

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2018:18
Matematisk statistik
Juni 2018

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Bachelor Thesis **2018:18**
http://www.math.su.se

Predicting Drug Effects

Using Bayesian Artificial Neural Networks for Drug Discovery

Carl Samuelsson*

June 2018

Abstract

Challenges like using predictive modelling of pharmaceutical drug effects can become noticeably more difficult if the model fails to express predictive uncertainty. Taking this into account is of great importance when developing more complex models. In this paper, we cover the theoretical framework of Bayesian Artificial Neural Networks (BANNs). The aim is to be able to apply this class of models in problem domains like drug discovery, where predictive uncertainty is a crucial aspect. In order to compute the intractable posterior distribution for a network model, we use variational inference to approximate the true posterior distribution by a variational posterior distribution. Moreover, our work covers other central areas for using BANN models such as hyperparameter optimisation and pre-processing of the data, e.g. by Principal Component Analysis. Although our empirical results did not suggest satisfactory performance for the explored BANN models, we humbly believe future research oriented more heavily towards model fitting could yield considerably better results.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: c.samuelsson94@gmail.com. Supervisor: Ola Hössjer and Disa Hansson.

Sammanfattning

Komplexiteten i att kunna tillämpa prediktiv modellering för läkemedelseffekter är starkt präglat av modellens förmåga att kvantifiera den prediktiva osäkerheten. Att beakta detta är således önskvärt vid utveckling av mer komplexa prediktionsmodeller. I denna uppsats avhandlas teorin för Bayesianska Artificiella Neurala Nätverk (BANN) i syfte att kunna tillämpa dessa modeller i problemområden som läkemedelsframtagning där den prediktiva osäkerheten är en viktig aspekt. Då a posteriori-fördelningen för en nätverksmodell är svårbehandlad, används variationsmetoder för att approximera den sanna a posteriori-fördelningen med en enklare typ av fördelning. Uppsatsen täcker även andra aspekter som är av central betydelse för användandet av BANN-modeller som hyperparametersoptimering och förbehandling av data, exempelvis genom principalkomponentanalys. Våra resultat gav ingen empirisk grund för att godta BANN-modeller som adekvata för problemändamålet. Vi tror emellertid att framtida arbete, orienterat mer kring modellanpassning, kan generera markant bättre resultat.

Foreword and Acknowledgements

This work constitutes a bachelor's thesis of 15 ECTS in Mathematical Statistics at Stockholm University.

First and foremost, I would like to thank my supervisors Ola Hössjer and Disa Hansson for theoretical discussions, encouragement and feedback that have been crucial for my work. I would also like thank my external supervisors Thomas Arctaedius and Birger Moëll at Ayond AB for formalising the project on business grounds and for providing great insights within the industry. In the same context, I am thankful towards Prosilico AB for letting me use their dataset for this project.

I want to emphasise that I have been lucky to participate in numerous interesting discussions and learning from the very best within the industry; much thanks to the non-profit organisation Stockholm AI. While too many to mention explicitly, I would like to thank friends, co-workers and fellow students who directly or indirectly have contributed to my learning path. In particular, I would like to acknowledge Amir Hossein Rahnama, Daniel Collin, Mihai Chiru, Arthur Pesah, Pontus Eklund and my brother Daniel Samuelsson for either theoretical advice and discussions, technical help with implementation or for proof reading. Lastly, the support gained from my closest friends, family and my girlfriend Julia has been essential for my work.

Contents

Abbreviations

ADME	Absorption, Distribution, Metabolism, Excretion
ANN	Artificial Neural Network
BANN	Bayesian Artificial Neural Network
EI	Expected Improvement
ELBO	Evidence Lower Bound
GLM	General Linear Model
MAP	Maximum-a-posteriori
MCMC	Markov Chain Monte Carlo
MLE	Maximum Likelihood Estimate
MSE	Mean Squared Error
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
SMBO	Sequential Model-based Global Optimisation
TPE	Tree-structured Parzen Estimator

Chapter 1

Introduction

1.1 Drug discovery

The development and discovery of new efficient drugs has arguably played an essential part for the advancements of modern society. Pharmacology, the study of drug effects, can mainly be divided into two subfields; pharmacodynamics and pharmacokinetics. While pharmacodynamics describes the effects of the drug on the body (therapeutic effects), pharmacokinetics describes the opposite: the effect of the biological system on the drug (McFadden, 2013, p.42). In the context of developing drugs, the knowledge from the pharmacokinetics is crucial for deciding route of administration, frequency of administration, dosage and even *if* a given substance can successfully interact with the body under regulatory guidelines (Czarnik and Mei, 2007, p.401).

1.1.1 ADME

One common way to model complicated phenomena within the field of pharmacokinetics in a simpler way for conceptualisation, is through a qualitative model called ADME (Absorption, Distribution, Metabolism and Excretion) (McFadden, 2013, pp.43-49).

As a quick overview of the ADME scheme, we list below the four different parts of the abbreviation ADME and what they mean with regard to interpretation.

- *Absorption.* Describes how the drug is absorbed into the bloodstream in terms of administration.

- *Distribution.* Describes how a drug is distributed into different compartments of the body. For instance, a given drug may be distributed with a larger concentration into a number of compartments, such as different organs, instead of solely being distributed into and remained within the bloodstream compartment.

- *Metabolism.* Describes how the body is changing a given drug towards similar substances that are eliminated later on.

- *Excretion.* Describes how substances are removed from the body.

1.1.2 Predictive modelling

Previously, the ADME aspects of a developed drug were tested first in the later phases of development through expensive clinical trials on humans or animals (Czarnik and Mei, 2007, p.401). This was problematic in the sense that if a candidate treatment that had shown promising results regarding its therapeutic effects failed to meet up its pharmacokinetic requirements, all the work spent on developing the specific candidate became useless. Naturally, a need of predicting or simulating the ADME attributes of a given drug arose for more efficient drug discovery, where ADME properties can be accounted for earlier on in the development.

In order to use predictive modelling for the qualitative ADME attributes, a quantitative representation is needed. For this work, we will not cover the aspects of deciding a proper quantitative representation, as we lack proper domain knowledge. Instead, as will be covered in Section 1.3, we will use only one quantitative parameter that corresponds to the qualitative aspects of distribution (the letter D in the ADME abbreviation).

1.2 Bayesian Artificial Neural Networks

In a classification competition back in 2003, organised for the Conference and Workshop on Neural Information Processing Systems (NIPS), the winners used Bayesian Artificial Neural Networks to model a drug discovery problem. Their dataset, called Dorothea (Guyon et al., 2007), consisted of many more variables (100 000) than observations (1 150) with a sparse characterisation. Subsequently, their dataset resembles the dataset we will use for this thesis, which will be elaborated on later.

Furthermore, Bayesian Artificial Neural Networks exceed non-Bayesian Artificial Neural Networks in terms of depicting predictive uncertainty (Lakshminarayanan et al., 2016), which in problem domains like drug discovery is a desirable property for a predictive model.

An Artificial Neural Network can (much simplified) be thought of as a sequence of multiple General Linear Models (GLMs), where the output from one GLM is used as input to the next GLM in the sequence. The Bayesian interpretation of an Artificial Neural Network, referred to as Bayesian Artificial Neural Networks, is to view the model parameters as some multivariate random variable that follows a specific probability distribution. A more elaborate introduction on this topic can be found in an upcoming chapter.

1.3 Problem formulation and purpose

In this thesis, we look into the theory of Bayesian Artificial Neural Networks from a mathematical and statistical point of view. Since the typical reader is not expected to possess much knowledge within the specific domain of Bayesian Artificial Neural Networks, the main scope of this thesis is centred around building the necessary theoretical toolbox for using these models.

Subsequently, the application of the covered theory in the context of drug predictions should merely be seen as a proof-of-concept approach, i.e. the purpose of this work is not to find a state-of-the-art model for any drug discovery problem; instead we investigate whether Bayesian Artificial Neural Networks can be used for modelling the prediction of drug effects. Practical aspects such as more exhaustive empirical experiments should be seen as a natural next step for this work.

The specific problem of interest regarding drug effects is modelled by a set of ADME values mentioned in Section 1.1.1 and 1.1.2. In other words, our interpretation of the problem of successfully predicting drug effects is to minimise the error between model predictions of the set of ADME values and their true values measured from clinical tests. We will assume for our work that the quantitative ADME parameters can answer the qualitative question regarding drug effects in terms of pharmacokinetics.

Lastly, in order to infer the ADME values for a given drug, we will use a set of its molecular properties which we will denote as X. We can graphically visualise our objective of inferring a multivariate mapping $F : X \to Y$ where

$Y = (A, D, M, E)^T$ as in Figure 1.1. In the general case, multiple quantitative parameters could correspond to each element in the Y-vector, and then Y would be a vector of vectors. However, the used terminology for our work will implicitly assume that each element in Y is a scalar, although this view could easily be generalised.

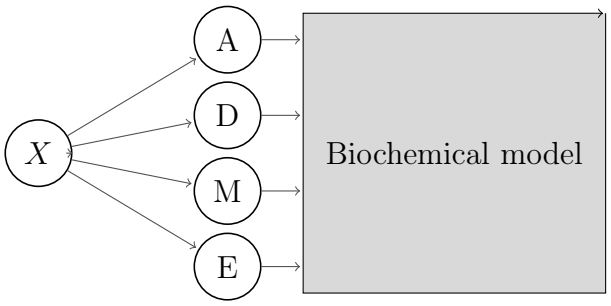

Figure 1.1: Graphical model for ADME predictions. The grey box, denoted as the *Biochemical model*, is to be interpreted in the sense that these ADME predictions will be used for further qualitative modelling, outside the scope of this work.

Moreover, as we will cover more extensively in Section 1.3.1, we only possess the target attribute of a single distribution parameter (corresponding to the letter D in the ADME abbreviation) for test purposes. Subsequently, we will also assume that the complexity of finding a univariate mapping $F : X \to D$, is representative for the complexity of the mapping for the original four-dimensional target value Y. Under this assumption, empirical results for our work could be seen representative for the real case of interest, with Y as output.

1.3.1 Data

The data set that reflects the mapping from X to D, consists of 606 observations with 6 909 explanatory variables with ordinal integer values which have been anonymised due to confidentiality reasons. Each of the 606 entries represents a particular pharmaceutical drug, the explanatory variables its different molecular properties and the response variable the estimated logarithmic *volume of distribution* (Hill, 2012, p.152) based on clinical trials on volunteering healthy humans.

The first thing we can notice about this dataset is its particularly high dimensionality and sparsity, as depicted in Table A.1 of Appendix A. Another

noteworthy observation is that the different variables are highly correlated with each other, indicated by the large condition number (see Table A.1 of Appendix A).

1.4 Outline

The structure of the thesis is as follows. Chapter 3 covers the central theory and background for what is later applied in Chapter 4 as empirical results for the research problem. However, as much of the work is centred around presenting a theory assumed to be new for the reader, we begin with Chapter 2. This chapter provides some intuition regarding network models and their Bayesian interpretation before laying out the theory more thoroughly. Then in Chapter 5, we continue with interpretation of the experiment, ideas of further research and shortcomings with our modelling. Finally, Chapter 6 summarises the paper by reviewing whether our objective was fulfilled.

Chapter 2

Network models for regression

Many regression models in the field of statistics can be described in simple terms of input and output. In the context of supervised problems, i.e. where the model gets to infer a mapping from previously observed data pairs, the regression models are usually constructed to perform some operation on its input data X such that its output $\hat{Y}$ is to be interpreted as the prediction of the true response value Y.

For network models however, we can abstract this view further by combining multiple standard regression models to form a network of elementwise operations in order to obtain the prediction. In this section, we will try to capture some intuition behind how these networks can be interpreted in terms of simpler regression models.

2.1 Graphical view on the regression model

Although not typically defined as networks, we can see that standard models such as the multiple linear regression model and the multiple logistic regression both fall under the category of single layer models. The purpose of this section is to make the reader familiar with a graphical representation of common regression models in order to later construct more complex and abstract models. While Section 3.1 will cover more precise definitions, this section and the following one aim to provide some intuition. In order to do this, we strive to orientate the reader with graphical representations of network models through examples to hopefully make the learning path smoother.

Example 2.1.1 (Multiple linear regression)**.** Consider a linear regression model with four explanatory variables and one (real valued) outcome. From

the theory of statistical linear models we know that given a vector of the effect parameters $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)^T$ and the intercept α are known, we get a prediction $\hat{y}$ of the outcome y, by computing

$$\hat{y} = \alpha + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 \,,$$

assuming that $y = \hat{y} + \epsilon$ for some $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

We can visualise our model's operations graphically as a network of element-wise operations, as shown in Figure 2.1. The nodes in the input layer denote the multiplications $\theta_i x_i$ for $i \in \{1, 2, 3, 4\}$. Although the additive operation of adding the intercept α is not explicitly shown in Figure 2.1, we could think of it as a separate input to the output layer if we would like expand our network graph. Lastly, we note that the node in the output layer represents the additive operation $\sum_i \theta_i x_i$ (plus the intercept) before finally returning $\hat{y}$. However, in the next example we will see that the values passed on from the nodes do not necessarily have to be the same as what is computed within the scope of the node's linear operations.

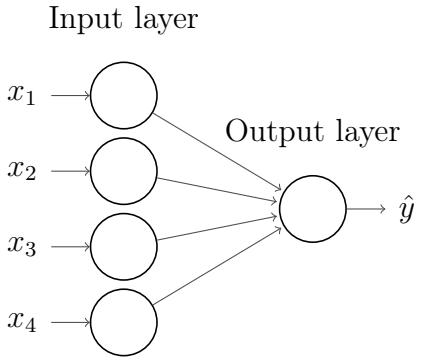

Figure 2.1: Graphical view of the multiple linear regression model.

Example 2.1.2 (Multiple logistic regression)**.** Consider a binary classification problem with 0 and 1 as outcomes, which we will approach with a logistic regression model with four explanatory variables. The idea is to extend the multiple linear regression model from Example 2.1.1 by applying the so called *Sigmoid function* that maps the real number line to the reals within the range of $[0, 1]$. The output could then be interpreted as probabilities for a specific event of interest in the context of classification problems.

The Sigmoid function is defined as

$$s(x) = \frac{1}{1 + e^{-x}} \,,$$

where if we substitute x above with $\hat{y}$ from Example 2.1.1 we can define the logistic regression model for 1 outcome variable and four explanatory variables as

$$\hat{y}_{\text{logistic}} = \frac{1}{1 + \exp\left(-\alpha - \sum_{i=1}^{4} \theta_i x_i\right)} \,.$$

However, visualising the logistic regression model in a similar fashion as in previous example, we note from Figure 2.2 that this model would be drawn in exactly the same way as for the linear regression model. The key point here is that we apply a function on the scalar value returned by node in the output layer to represent our prediction. Therefore, the logistic regression model can be thought of as a slight extension of the linear regression model where we simply apply the Sigmoid function on the output.

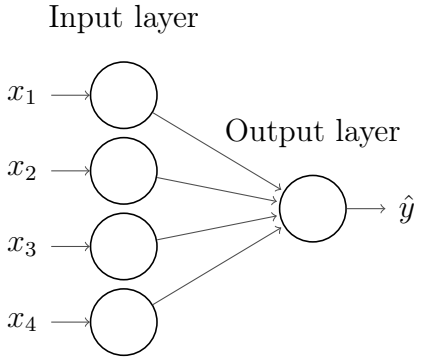

Figure 2.2: Graphical view of the multiple logistic regression model.

Example 2.1.3 (Multinomial logistic regression)**.** While regression problems typically contain a one-dimensional outcome, in a more general setting that does not always have to be the case. Consider a classification task with four outcomes and four explanatory variables. We could model this problem with a so called Multinomial logistic regression model, which is an extension of the standard logistic regression model to predict probabilities for more than one event. This translates into our prediction $\hat{y}$ being a vector, i.e. $\hat{y} = (\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4)^T$ in the case of four outcomes.

Since the linear operations from Example 2.1.1 and Example 2.1.2 (before applying the Sigmoid function) need to be applied for all four outcomes

individually, we need to define our effect parameters θ as a 4×4-matrix instead of a four-dimensional vector as before. Analogously, our intercept α needs to be a four-dimensional vector instead of a scalar. Therefore we may describe the model by first computing the affine transformation of input data $x = (x_1, x_2, x_3, x_4)^T$ as

$$\theta x + \alpha \,, \tag{2.1}$$

and then apply the Sigmoid function for each component of the output vector,

$$\hat{y} = s(\theta x + \alpha) \,,$$

where $s(\cdot)$ denotes the operation of applying the Sigmoid function element-wise on a given vector.

Graphically, we can visualise the Multinomial logistic regresion model as in Figure 2.1.3. We are once again excluding the intercept α here, but the reader should be able to visualise how each component of the intercept vector could be drawn into the output layer, if desired.

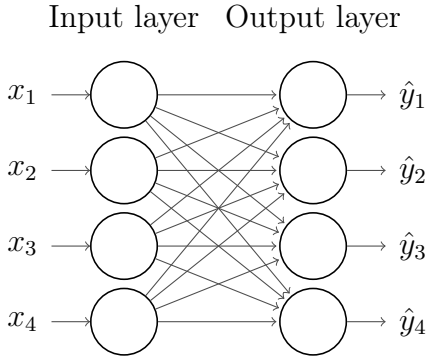

Figure 2.3: Graphical view of the multivariate logistic regression model.

Remark. *The affine transformation in Equation (2.1) is more widely known in statistics as the General Linear Model (GLM) and would have the same graphical representation as in Figure 2.3 (compare with Figure 2.1 and Figure 2.2).*

2.2 Introducing depth

Having liberated our view on regression models to only cover the input/output-view, we are now able to define more complex schemas. We will first briefly

describe the structure of an *Artificial Neural Network*, which given our previous examples, can simply be viewed as an iterative (multiple) regression model, in order to cover more arbitrary ways of constructing regression models.

Example 2.2.1 (A first encounter with an Artificial Neural Network)**.** Consider a multivariate regression model with four outcomes, for instance the Multinomial logistic regression model as in Example 2.1.3. Moreover, consider also a standard multivariate linear regression model with one-dimensional output (see Example 2.1.1). We can construct a new regression model by simply taking the output from the first multi-outcome model as input for the multivariate linear regression model, as shown in Figure 2.4.

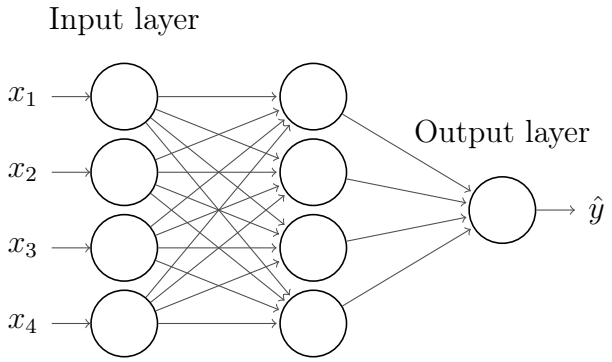

Figure 2.4: Graphical view of the regression model that in the second layer operates on the output of a multivariate regression model with a vector-valued outcome.

The takeaway from Example 2.2.1 is to demonstrate the ability to define a network of arbitrary regression models. While not defined rigorously yet, these classes of models are usually referred to as *Artificial Neural Networks* (ANNs). We will examine these models more thoroughly in Section 3.1.

2.3 Hyperparametric model choices

Having layed the framework of constructing network models, it would serve a good purpose to introduce the term *hyperparameters*. A hyperparameter in this context can be thought of something that defines the network, something we need before actually solving for the effect parameters in our model. For instance, if modelling a one-dimensional classification problem by four explanatory variables, we are in a way bounded by the dimensionality of our

input data and our outcome we want to find a mapping for. However, if we choose to model a network, we are free to bind an arbitrarily large number of componentwise regression models and their internal dimensionality. In Example 2.2.1, the first regression model was not necessarily required to have a four-dimensional output. This was instead a hyperparametric choice we made for our network model. We could instead have chosen to make this output (or input in regards to the second component) to be five-dimensional. Tuning these network models in this fashion, to be as good as possible for a given task, is known as a hyperparameter optimisation problem (Goodfellow et al., 2016).

2.4 A Bayesian perspective

In Bayesian statistics, beliefs about some unknown parameter value θ, when data has been collected and analysed, are represented as a probability density function known as the *posterior distribution*; unlike in frequentist statistics, where the true parameter is estimated by a single value. The probability mass in the form of the posterior distribution can easily be interpreted as how likely we think it is that θ is located within a specific range. Another important difference is that Bayesian methods update subjective beliefs about θ in terms of a *prior distribution*, which also is a probability density function that reflects our knowledge of the parameter, before data has been analysed. Therefore, the exact form of the posterior distribution is directly influenced by the prior beliefs.

In the context of network models for regression, the Bayesian viewpoint on the effect parameters would change the effect parameters θ to be a vector of random variables following some posterior distribution. This view implies that the prediction of the expected outcome is also of the form of a probability density function, what we shall refer to as a *predictive distribution*. By computing many predictions for the same input data, we can approximate this true predictive distribution for the given model by Monte Carlo methods through samples from θ.

While the standard non-Bayesian Neural Networks perform very well for some tasks, they tend to suffer from overfitting and not being able to depict predictive uncertainty (Lakshminarayanan et al., 2016) which in many problem domains is a crucial aspect of a predictive model. By instead using Bayesian Neural Networks, this predictive uncertainty is something that can modelled more easily, with the Bayesian viewpoint.

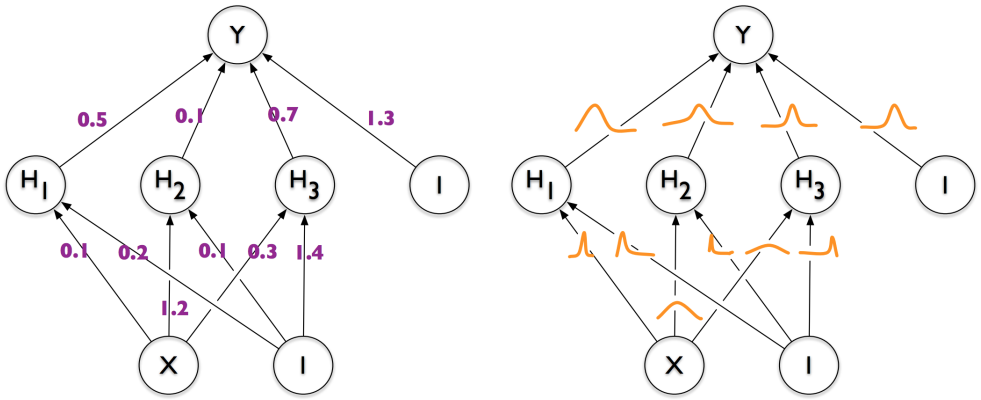

Figure 2.5: The difference between a frequentist and Bayesian viewpoint on a network model. To the left, a frequentist perspective of the model parameters, as constant values, is illustrated. To the right, the Bayesian view on the same parameters, with some posterior distribution, is depicted. The picture is taken from Blundell et al. (2015).

Chapter 3

Theory

Before presenting the theory of Artificial Neural Networks (ANN for short), it is worth pointing out that similar models were developed independently in the field of statistics to solve the same type of problems (Hastie et al., 2001, chap.11). These models strive to approximate some unknown function that maps our explanatory data X to some function space Y, by taking linear combinations of our variables and then adding a nonlinear function in order to improve our approximation.

3.1 Artificial Neural Networks

There has been a lot of excitement regarding the class of models named Artificial Neural Network during recent years (Hastie et al., 2001, p.392). One of the reasons potentially being the fact that they are modelled to emulate, in a simplified way, how our brain is believed to work, from a neuroscientific point of view (Hopfield, 1982). In addition, the ANNs are nowadays computationally feasible for quite large models. In this paper, we are not interested nor strive to answer any claims regarding the proposed connection with the human brain. Instead, we are interested in ANNs as function approximators and the mathematical theory behind this.

3.1.1 Deep architecture

(This section is based on Nielsen (2018) if not stated otherwise)

What differ the ANNs from other nonlinear (or linear) statistical models in particular is the fact that they (generally) use a deep architecture; often terminologically referred to as *multilayer perceptrons*. The term perceptron is

a synonym to the previously used term *layer*. A layer or perceptron in this context means a set of *neurons* that shares the same input. The reader might find our elaborations in this subsection easier to follow when simultaneously looking at Figure 3.1 that illustrates the deep architecture of an Artificial Neural Network.

These classes of models can be used both for classification and regression problems with different types of response variables. In a sense, even classification problems can be viewed as regression problems with bounded outcomes in terms of probabilities to predict. However, for this work we will restrain ourselves to study ANNs for regression problems only, with continuous and unbounded outcomes.

To begin with, we note that any node (called *neuron*) in Figure 3.1 corresponds to an operation performed on either the original data or the output from a previous neuron. The first layer, called the *input layer* will have input from the same number of neurons as there are variables available in the dataset, for which we wish to make predictions.

Now comes the interesting part, the *hidden layers*. However, as Nielsen (2018) mentions, these hidden layers might not be so mysterious as initially believed. *Hidden* in this context should rather be interpreted as something that is neither an input nor an output for the model, rather some intermediate step in order to find our true prediction output. The number of hidden layers and their respective dimension are not explicitly determined by the dataset, instead a model choice we do in order to strive for a better function approximation. In Figure 3.1 a Neural Network with two hidden layers of dimensions 4 and 3 respectively, are shown. Given this, the reader should be able to visualise other arbitrary forms of ANNs.

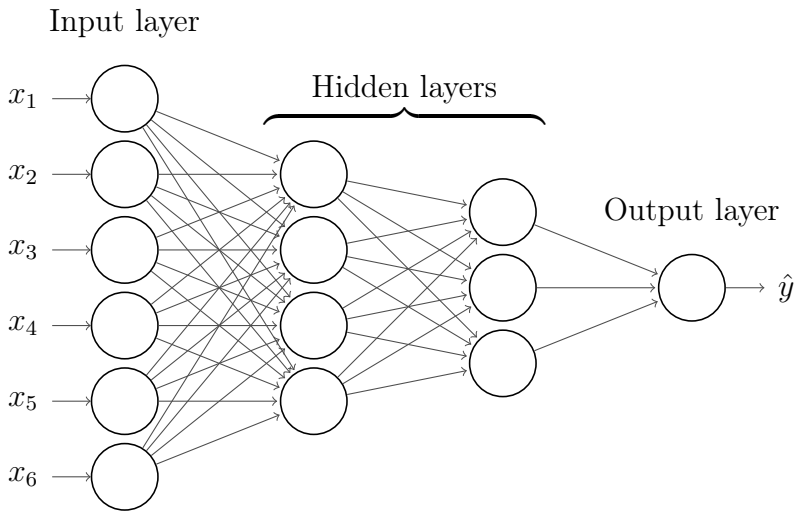

Figure 3.1: The deep architecture of an Artificial Neural Network. Here, a Neural Network with 6 input variables, 2 hidden layers (with dimensions 4 and 3 respectively) and one output variable is shown.

What we yet not have been elaborating on is the actual operations that our neurons are performing and how the mapping between the different layers work. In the simplest form, it all comes down to multiplying some weight matrix with some vector of input data, i.e. a linear transformation. We will from now on denote the weight matrix and the input vector at layer l as $W^{(l)}$ and $x^{(l-1)}$ respectively. With this notation established, the original input data corresponds to $x^{(0)}$ and the remaining $x^{(l-1)}$ with $l > 1$ are simply some transformation of the original data. The iterative scheme is described in Equation (3.1)

$$x^{(l)} = W^{(l)}x^{(l-1)} = \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \dots & w_{1k}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \dots & w_{2k}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{(l)} & w_{n2}^{(l)} & \dots & w_{nk}^{(l)} \end{pmatrix} \begin{pmatrix} x_1^{(l-1)} \\ x_2^{(l-1)} \\ \vdots \\ x_n^{(l-1)} \end{pmatrix}, \tag{3.1}$$

where input data $x^{(l-1)}$ for each layer l is a n-dimensional vector and weight matrix $W^{(l)}$ is of dimensions $n \times k$.

Remark. *Since the dimensions of each layer l can be arbitrary, n and k are not necessarily constant throughout the different layers of the model. Therefore an alternative notation n_l and k_l may be more appropriate, but we choose not to write this explicitly in Equation (3.1) in order to keep the notation simple.*

The type of Neural Network in Equation (3.1) is a so called *feedforward* Neural Network, which means that we take the output from the previous layer as input to the next, therefore always moving "forward" within the network. There exist other types of ANNs where this feedforward principle is extended. For instance, *Recurrent Neural Networks* feed its own output into itself and therefore constructs a sequence of its predictions (Goodfellow et al., 2016, p.387).

Concluding this subsection, it is worth mentioning that our output layer (alternatively our prediction) can be of arbitrary dimension, like the other layers. However, as with most regression problems in general, our paper will only be dealing with the response variable being one-dimensional.

3.1.2 Activation through bias

One can extend Equation (3.1) to contain a constant term $b^{(l)}$ being the *bias* of the layer l. It takes inspiration from the concept of perceptrons having different likelihoods of *firing* in order to be activated (Nielsen, 2018, chap.1). Larger values mean that the perceptron is more likely to fire, whereas the opposite is true for negative values. Another more mathematical interpretation would be that our bias is a constant or intercept in the matrix operation

$$x^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)}. \tag{3.2}$$

Remark. *We note from Equation (3.2) that computing $x^{(l)}$ will always be (before adding a nonlinearity) an affine transformation, i.e. on the form $Ax^{(l-1)} + b$. Another interesting interpretation would be to view the transformations between the layers as GLMs (General Linear Models).*

3.1.3 Adding nonlinearity

(This section is based on Nielsen (2018) if not stated otherwise)

Since we have previously been referring to the class of ANNs as nonlinear statistical models, the reader might not be too surprised hearing that Neural Networks usually are designed to use nonlinear functions applied to its output. The theoretical argument behind this is to make the Universal Approximation Theorem (covered in Section 3.1.5) applicable to our model. However, with a deep architecture, these nonlinear functions will be applied at all the outputs from different layers. In fact, we can even choose *if* we want to add a nonlinearity at each layer and which *activation function* (nonlinear

function) to use. Therefore, these choices (or hyperparameters) do not have to be consistent through all layers. This makes it possible for us to construct our ANN models in many different ways. Even though we might be able to find a rich variety of nonlinear functions applicable to our model, a smaller set of activation functions are usually used in practice (Goodfellow et al., 2016, p.196). In this paper, we will cover two of these common functions; the Sigmoid function and the Rectified Linear Unit (Goodfellow et al., 2016, p.174) (often referred to as ReLU). The reader might recognise the Sigmoid function as a special case of the logistic function

$$s(x) = \frac{1}{1 + e^{-x}} ,$$

which maps the real numbers to the interval between 0 and 1. This might especially be useful in cases where the output needs to be interpreted in terms of probabilities.

The other mentioned activation function, the Rectified Linear Unit, is arguable easier to compute due to its simple definition as

$$r(x) = \max(0, x) .$$

Noticeable here is that $r(x)$ maps the reals to the positive reals, hence it is not bounded, which will be discussed further in Section 3.1.5. To elaborate further on the meaning of computational ease, we will see later that in order to fit (or "train") our model, it is needed to compute the derivatives of each activation function used for our model and multiply them together. Therefore, since the derivative of the Rectified Linear Unit has the appealing property of being either 0 or 1, it will both be easier to compute and makes the network more sparse (because of the zeros), a feature that is believed to be more plausible biologically (Glorot et al., 2011).

Having introduced the activation functions, we can generalise our notation from Equation (3.2) slightly, to

$$x^{(l)} = \sigma^{(l)} \left(W^{(l)} x^{(l-1)} + b^{(l)} \right) . \tag{3.3}$$

Remark. *With $\sigma^{(l)}(\cdot)$ we mean applying the activation function elementwise to the vector $W^{(l)}x^{(l-1)} + b^{(l)}$.*

With Equation (3.3) in place, we are able to predict the outcome, given a new sample $x^{(0)}$.

Lastly, although we are implying with the name of this subsection that activation functions must be nonlinear, that does not need to be the case. However, the same effect of using a linear activation function could be achieved by only using the affine transformation in Equation (3.2). With proper changes of the weights and bias in a given layer, we would implicitly apply the linear activation function of interest. On the other hand, to *not* use an activation function in a layer could be viewed as a (hyper)-parametric choice. In fact, it could make sense to speak about an activation function that is the identity function, where its output simply equals the input, i.e. a function $F : X \to X$.

3.1.4 Objectives and training the model

(This section is based on Nielsen (2018) if not stated otherwise)

An obvious question arising is how we find our model's parameters. In order to find our weights and biases, we first need a function to optimise; something to quantify how well the current model performs in terms of predicting the correct results. This leads us to introducing the concept of an *objective function*, also called a loss function or an error function (Goodfellow et al., 2016, p.82). An example of such a function is the *Mean Squared Error* (MSE), whose definition is found in Definition A.1, Appendix A.

Once the objective function has been selected, our task is clear: find a model that minimises the Mean Squared Error, or any other chosen objective function, for our observed data. Our optimisation task can be decomposed into two parts: the first one regarding the choice of *hyperparameters* (such as the number of hidden layers, their dimensions, choice of activation functions etc.) and the second regarding optimising the parameters given a fixed set of hyperparameters. More specifically, we are referring to the weight matrices and the bias vectors as these parameters to optimise. Updating these parameters by means of our dataset is often referred to as the "learning" or "intelligence" part, which gives some intuition behind the names of the fields Machine Learning, Deep Learning and Artificial Intelligence. The idea is the same as with statistical inference: to learn some underlying structure by observing data.

3.1.4.1 Gradient descent

With a loss function of a given model defined, we are now able to formulate an optimisation problem. Since our loss function L is a function of the

model's weights and biases, we want to find weights w and biases b such that L attains its global minimum. However, we know from multivariate calculus that finding the global optimum in a high dimensional vector space is often intractable to solve analytically. Therefore, our approach will be to use something called *gradient descent* (Nielsen, 2018, chap 1). It is a first order iterative scheme to update our variables by stepping along the negative gradient vector, since we know it will point to the direction that decreases our loss function the most. Phrased differently, in order to minimise our loss function, we want to choose ΔL as negative as possible by changing the parameter vector $p = (w, b)^T$, which contains all our weights and biases. From calculus of variations we know that

$$\Delta L \approx \nabla L \cdot \Delta p \,. \tag{3.4}$$

If we let $\Delta p = -\alpha \nabla L$ and substitute this relation into Equation (3.4) we get that

$$\Delta L \approx -\alpha ||\nabla L||^2, \tag{3.5}$$

where $\alpha > 0$ is called the *learning rate*, which translates to how far we are stepping along the negative gradient. Implicitly we make sure that $\Delta L \leq 0$ if Equation (3.5) is a good approximation since $||\nabla L||^2 \geq 0$. Therefore, we need to pick α small enough such that we achieve a good approximation, but not too small so that the model's convergence becomes too slow. To pick an optimal learning rate is a part of the hyperparameter optimisation problem, and it will be described in more detail later. To apply it to our own optimisation problem we phrase it as

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \alpha \frac{\partial L}{\partial w_{jk}^{(l)}} \,, \tag{3.6}$$

$$b_{j}^{(l)} \leftarrow b_{j}^{(l)} - \alpha \frac{\partial L}{\partial b_{j}^{(l)}} \,. \tag{3.7}$$

Remark. *With the notation above, we mean that we are updating our weights $w_{jk}^{(l)}$ and biases $b_j^{(l)}$ with their previous values after subtracting some change. To be more formal, one could denote the previous values with a new index indicating the iteration of the algorithm to emphasise that these terms are not in fact the same. However, to keep the notation simple, we choose to not include these indices.*

Since our models will be trained on multiple data points, typically huge datasets, our gradients need to be averaged over all our samples. This leads

us to one important assumption Nielsen (2018) mentions, that the loss can be expressed as an average. Therefore, we must assume that the objective function is of the form such that we are able to write the loss L as an average over multiple individual losses L_i corresponding to the i^{th} observation, that is, we assume that

$$L = \frac{1}{n} \sum_{i=1}^{n} L_i \, .$$

When we previously used MSE as an example of an objective function, we implicitly used the *squared error* as the individual loss whereby MSE is the total loss after averaging.

Moreover, since ANNs may contain thousands, in extreme cases even billions, of parameters, this approach will be extremely costly computationally. A more practical approach is to take subset samples of the dataset instead in order to approximate the true averaged gradient. This approach is called *Stochastic Gradient Descent*. The sample size to use for our stochastic approach will be a trade-off between computation complexity and accuracy of the gradient values; thereby also a hyperparameter for the user to choose beforehand. After choosing a batch-size m we are able to approximate the gradient ∇L by computing

$$\nabla L \approx \frac{1}{m} \sum_{i=1}^{m} \nabla L_i \, , \tag{3.8}$$

for some $m < n$.

Remark. *Moreover, the method of sampling the batch-size might differ between implementations of Stochastic Gradient Descent. One could at each iteration either sample the m number of observations individually or beforehand define a number of folds of size m and instead sample fold-wise. In other words, if choosing the fold method, each observation will always be paired with $m - 1$ other observations. Lastly, if the fold method of sampling is chosen, the number of observations n should be evenly divisible with the batch-size m, i.e. $n \equiv 0 \pmod{m}$, in order to not leave out any observations.*

3.1.4.2 Backpropagation

With the introduction of gradient descent, we have an idea of how to tackle our optimisation problem. From Equations (3.6) and (3.7) we note that computing $\partial L / \partial w_{jk}^{(l)}$ and $\partial L / \partial b_j^{(l)}$ is central for training a Neural Network (i.e. to

find an optimal model within the class of feedforward ANNs). To compute these values, a common approach is to use *backpropagation* (Nielsen, 2018, chap 2). The idea is to make a prediction with a given model, compare it against the true value and then update the weights and biases in the previous layer to achieve a more desirable result for the given example. Thereafter, we can recursively apply this algorithm backwards within our network to update all our parameters, a procedure that gives rise to the name backpropagation. In this section, we will look into the mathematics behind this popular algorithm.

To understand what values in a previous layer $l-1$ that affect the output of a neuron $x_j^{(l)}$ in the next layer, we note from Equation (3.3) that either $w_{jk}^{(l)}, x_k^{(l-1)}$ or $b_j^{(l)}$ could be changed accordingly to achieve a more satisfying result. Therefore, it will be in our interests to compute $\partial L/\partial w_{jk}^{(l)}, \partial L/\partial x_k^{(l-1)}$ and $\partial L/\partial b_j^{(l)}$, as mentioned in Equations (3.6) and (3.7). To follow the upcoming reasoning and calculations, it will be easier introducing an intermediate variable $z_j^{(l)}$ as

$$z_j^{(l)} := \left(W^{(l)}x^{(l-1)} + b^{(l)}\right)_j .$$

It now follows that the loss L is a function of $x^{(l)}$ that is a function of $z^{(l)}$ which is a function of $w_{jk}^{(l)}, x_k^{(l-1)}$ and $b_j^{(l)}$. So in order to compute $\partial L/\partial w_{jk}^{(l)}, \partial L/\partial x_k^{(l-1)}$ and $\partial L/\partial b_j^{(l)}$, we need to apply the chain rule which will give us that

$$\frac{\partial L}{\partial w_{jk}^{(l)}} = \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} \cdot \frac{\partial x_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial L}{\partial x_j^{(l)}} , \tag{3.9}$$

$$\frac{\partial L}{\partial x_k^{(l-1)}} = \sum_{j=0}^{n_l-1} \frac{\partial z_j^{(l)}}{\partial x_k^{(l-1)}} \cdot \frac{\partial x_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial L}{\partial x_j^{(l)}} , \tag{3.10}$$

$$\frac{\partial L}{\partial b_j^{(l)}} = \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} \cdot \frac{\partial x_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial L}{\partial x_j^{(l)}} . \tag{3.11}$$

Remark. *We get a sum in Equation (3.10), since multiple neurons in the previous layer can be connected to one neuron in the current layer. (See Figure 3.1).*

Since $x_j^{(l)} = \sigma\left(z_j^{(l)}\right)$ we get

$$\frac{\partial x_j^{(l)}}{\partial z_j^{(l)}} = \sigma'\left(z_j^{(l)}\right) . \tag{3.12}$$

Remark. *For ease of notation, we have dropped the layer index for our activation functions σ. But as elaborated on in Section 3.1.3, we are free to choose these functions differently for each layer.*

Moreover, by the way we defined $z_j^{(l)}$ earlier, we get that

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = x_k^{(l-1)} , \tag{3.13}$$

$$\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = 1 , \tag{3.14}$$

$$\frac{\partial z_j^{(l)}}{\partial x_k^{(l-1)}} = w_{jk}^{(l)} . \tag{3.15}$$

The factor $\partial L / \partial x_j^{(l)}$ occurring in all three equations (3.9) - (3.11) is however dependent on the choice of activation function and can not be solved for in the general case.

Substituting Equations (3.12) - (3.15) into (3.9) - (3.11) yields

$$\frac{\partial L}{\partial w_{jk}^{(l)}} = x_k^{(l-1)} \cdot \sigma'\left(z_j^{(l)}\right) \cdot \frac{\partial L}{\partial x_j^{(l)}} , \tag{3.16}$$

$$\frac{\partial L}{\partial x_k^{(l-1)}} = \sum_{j=0}^{n_l - 1} w_{jk}^{(l)} \cdot \sigma'\left(z_j^{(l)}\right) \cdot \frac{\partial L}{\partial x_j^{(l)}} , \tag{3.17}$$

$$\frac{\partial L}{\partial b_j^{(l)}} = \sigma'\left(z_j^{(l)}\right) \cdot \frac{\partial L}{\partial x_j^{(l)}} . \tag{3.18}$$

Given Equations (3.16) - (3.18), we have a framework for recursively updating our parameters backwards, as described in Equations (3.6) - (3.7).

Although we will only use Stochastic Gradient Descent as our optimising algorithm, it is worth mentioning that there exist a rich variety of similar

methods, often based on the same backpropagation principle. A few examples are *Adam*, *RMSprop* and *Adagrad* (Goodfellow et al., 2016, pp.307-309).

3.1.4.3 Initiating and stopping the training

So far, we have looked into the framework of updating the model parameters given a prediction by the model and the true label. However, in order for the model to make the very first prediction, it will need a way of initiating the parameters. A common strategy is to randomly assign the parameter values by some uniform or normal distribution centred around 0 (Goodfellow et al., 2016, pp.302-303).

Furthermore, since optimisation methods based upon the gradient descent principle are iterative, we need to define a stopping criterion that dictates when the training is finished. The simplest and most straight forward solution is to define a number of *epochs* that can be viewed as the number of iterations to perform our training algorithm. In other words, we perform Stochastic Gradient Descent, or any other gradient descent based optimising algorithm, for a fixed number of epochs. The number of epochs can also be viewed as part of the hyperparameter optimisation.

Finishing this section, we have the theoretical framework for explicitly writing the pseudo code for Stochastic Gradient Descent. We refer to it as Algorithm 1, and it is defined as follows:

Algorithm 1: Stochastic Gradient Descent

Data: Batch-size m, learning rate α and a pre-chosen way of initialising the parameters of the model.
Result: Optimised parameters for an ANN.
parameter initialisation;
while *not converged* **do**
  initiate zero valued gradient variable;
  sample m observations from training set;
  for i **from** 1 **to** m **by** 1 **do**
    increment gradient variable by computing the gradient for the i^{th} observation according to Equations (3.16) - (3.18);
  end
  update parameters according to Equations (3.6) - (3.7), where the derivatives is replaced by the proper element from the estimated gradient of Equation (3.8);
end

3.1.5 Universal approximation theorem

In essence, all forms of nonlinear statistical models aim to approximate some unknown function $f(X)$ with an approximation $\widetilde{f}(X)$ such that $|\widetilde{f}(X) - f(X)| < \epsilon$ for a reasonably small ϵ.

A major theoretical argument of choosing the class of ANNs as function approximators is their universal attributes in terms of function approximation, as stated by the *Universal Approximation Theorem* (Hornik, 1991). This result holds, under some conditions, that a Neural Network with as little as one single hidden layer can approximate any arbitrary continuous function arbitrarily well. In principle, this is a very powerful statement. In practice, however, a "good enough"-approximation is usually satisfactory since exact inference may require an unimaginable amount of model parameters and sample sizes.

While we will not include the precise formulation of this theorem here, and its accompanying proof, we note that some important condition for the theorem is that our nonlinear functions are bounded, i.e. they have a bounded range (image) and are continuously monotonously increasing. Although the Linear Rectifier function mentioned in Section 3.1.3 is not bounded, it has been proven that the Universal Approximation Theorem is still applicable

for models with the Rectified Linear Unit as activation function (Sonoda and Murata, 2015).

The identity function however (and other linear functions, as briefed upon in Section 3.1.3) is not a valid choice of activation function, in terms of this universality (Hornik, 1991).

3.1.6 Normalising the data

For a given regression problem of mapping data X to outcome Y, it is of central importance for our dataset to include variation in each variable x_i of the model, in order to infer the effect that a change in x_i has on Y. Therefore, we can transform a given variable x_i to x_i' as long as $x_i \propto x_i'$ to preserve its relative amount of variation.

In LeCun et al. (1998), it is argued that a Z-normalisation of each variable improves the convergence speed of backpropagation based optimisation algorithms for Neural Networks. In other words, we can transform each variable $x_i, i \in \{1, \ldots, n\}$ by

$$x_i' := \frac{x_i - \mu_{x_i}}{\sigma_{x_i}},$$

where μ_{x_i} and σ_{x_i} are the sample mean and sample standard deviation of x_i respectively, and consequently, x_i' is approximately $\mathcal{N}(0, 1)$-distributed. Since $x_i' \propto x_i$, our regression problem will remain invariant under the linear transformation of normalising the variables.

3.1.7 Model validation

Unquestionably, an important part of Machine Learning modelling is to validate the performance of a given model. While a model might perform well during training, it might also have been subject to overfitting, i.e. only learning local structure for the data it has been trained on, and not being able to generalise well for new unseen data. In order to investigate potential overfitting, the field of Machine learning differentiates the concept of minimising *training error* and *generalisation error*, sometimes called *test error* (Goodfellow et al., 2016, p.110). Therefore, we want to minimise both the training error and the generalisation error. This weighted objective is something that differentiates Machine learning from regular optimisation problems (Goodfellow et al., 2016, p.110).

3.1.7.1 Splitting the dataset

One common approach in order to estimate the generalised error is to split a given dataset X into two (or more, see below) disjoint sets (Goodfellow et al., 2016, p.121). The suggestion here is that our model should only be trained on the training dataset to minimise its training error first and then compute an estimate of the generalisation error, based on the unseen observations from the validation set. This split plays a crucial part in order to compare two models against each other, which will provide a necessary framework for hyperparameter optimisation, as will be elaborated further in Section 3.2.

While a typical 80/20-split (80% of the data for training, 20% for validation) is suggested as a rule of thumb (Goodfellow et al., 2016, p.121), a proper split for each problem is however very dependent of the dataset and type of problem to solve. Furthermore, our split will be a trade-off between statistical uncertainty in the estimate of the generalisation error and the model's performance as a function approximator for the training set. In examples of datasets with extensive amount of observations, this might not be a real concern. However, when sampling is expensive, for instance with problems that have many more variables than observations, it can be difficult to somewhat accurately compare different models (Goodfellow et al., 2016, p.122).

Lastly, in order for the model evaluation on the validation set to be an unbiased estimate of the generalisation error, we need to assume that the two different sampling distributions are homogeneous (Goodfellow et al., 2016, p.121).

3.1.7.2 K-fold Cross-Validation

Another approach that increases the accuracy of the generalisation error estimate while still providing a rich amount of observations for the model to train on is Cross-Validation. While there are different methods for doing this, *K-fold Cross-Validation* is the most common (Goodfellow et al., 2016, p.122). The method is based on the idea of splitting our original dataset into K disjoint subsets and let these subsets take turns on being the test set while the complement set is used for training. Therefore, we are training the model K times and averaging the observed test errors to better estimate the true generalisation error. A pseudo code for this method is shown in Algorithm 2 of Appendix A.

However, one problem with K-fold Cross-Validation, besides being computa-

tional very expensive, is that there is no unbiased estimate of the variance of Cross-Validation estimates for the generalisation error (Bengio and Grandvalet, 2004).

3.1.7.3 Final validation

Besides validating a model on one or several sets that are part of the original data, as described in Sections 3.1.7.1 - 3.1.7.2, one would ideally make sure that a returned model from the model selection actually has satisfying performance. In order to test this it is suggested in (Goodfellow et al., 2016, p.121) that a final validation set is introduced, referred to as the *test dataset*. It is important for the test data that these observations have not been subject to the hyperparameter optimisation in order to truly test the model's generalisation capability.

3.2 Hyperparameter optimisation

Since our ANN model's performance will be a function of the set of chosen hyperparameters, we would ideally like to have one or more algorithmic approaches towards finding somewhat optimal parameter values. In this section, we will introduce a few methods for hyperparameter optimisation.

To begin with, hyperparameter tuning can quickly become a time consuming task if the models themselves are expected to take a while to train. In order to evaluate different sets of hyperparameters and compare them with each other, we also need to train multiple ANNs and compare their output in terms of the loss value. This can be seen of finding $x^* \in \mathcal{X}$ such that we minimise the loss function $F : \mathcal{X} \to \mathbb{R}$, where each function evaluation $F(x)$ is expensive. Therefore, it would also be desirable to reach a quick convergence in terms of different hyperparameters to test for.

Although the scope of our work is not to cover the different hyperparameter optimisation algorithms in detail, we will try to outline how a few popular approaches work and capture some intuition behind them.

The perhaps most natural approach would be to use human intuition and heuristics to manually navigate through the hyperparameter space. However, this approach is not particularly scalable since we, as humans, do not tend to handle high dimensional data well in terms of spotting trends. Moreover, by using this manual search method, results are not very reproducible and

may even lead to faulty conclusions based on comparisons.

3.2.1 Grid search

Another straightforward approach towards finding optimal hyperparameters would be to use something called *grid search* (Goodfellow et al., 2016, pp.432-434). Basically, it is an approach based on testing every combination of hyperparameters to see which one results in the best model performance. In the case of continuous hyperparameters, a subset can be chosen beforehand, by specifying a stepping length, when changing a given parameter from its minimal value to its maximum value. However, as one might suspect, this method does typically not scale well as the number of hyperparameters and their discretised range increase.

3.2.2 Random search

Instead of testing every possible configuration of the set of, possibly high dimensional, hyperparameters one may evaluate a randomly chosen subset. It is shown that this random approach is just as good or even better than the grid search method, but still it has a much lower computational cost (Bergstra and Bengio, 2012). The main argument is that all hyperparameters are usually not equally important for model performance.

3.2.3 Sequential Model-based Global Optimisation

While both grid and random search can be seen as brute force approaches towards finding an optimal set of hyperparameters, methods like *Sequential Model-based Global Optimisation* (SMBO) (Bergstra et al., 2011) let us infer, under some model $\mathcal{M}$, the optimal next trial to perform. The idea with SMBO-approaches is to approximate the evaluation-expensive objective function F with some surrogate function S in order to propose a vector of hyperparameters x^* that maximise S as the next trial to evaluate for F.

In general, many functions could be used as the surrogate function. However, we will, as in Bergstra et al. (2011), focus on the so called *Expected Improvement* (EI) as our surrogate function S. The Expected Improvement criterion, denoted $EI_{y^*}(x)$, is defined as

$$EI_{y^*}(x) := \int_{-\infty}^{\infty} \max(y^* - y, 0) p_{\mathcal{M}}(y \,|\, x) dy \,,$$

where y^* is some threshold outcome value and x a vector of hyperparameters.

As the name suggests, we can consider the Expected Improvement criterion to evaluate, under some model, the expectation regarding how much $y = F(x)$ will decrease below some threshold y^* in an absolute sense. In the context of hyperparameter optimisation, y^* could intuitively be defined as $\min_x F(x)$, i.e. the best trial in terms of loss value. However, as we will see later, this does not have to be the case, even when regarding hyperparameter optimisation.

In Sections 3.2.3.1 and 3.2.3.2 we will cover the two approaches for picking the model $\mathcal{M}$ that Bergstra et al. (2011) propose.

3.2.3.1 Gaussian Processes

The first approach as the model in the surrogate is to use a method called *Gaussian Processes*. A Gaussian Process is essentially a generalisation of the multivariate normal distribution, where instead of being parametrised by its mean and covariance matrix, it is parametrised by a function of the mean and a covariance matrix given by some pre-defined *Kernel function* (Murphy, 2012, chap 14). Therefore, we are putting some prior beliefs upon the distribution by choosing our kernel function and the mean function, to later iteratively update the covariance matrix and the mean function. This allows us to sample points in the hyperparameter space, that upon our prior beliefs for a given iteration maximise the expected generalised error, while simultaneously model the uncertainty with this probabilistic approach. A more detailed elaboration on Gaussian Processes and Kernel functions can be found in Chapters 14-15 of Murphy (2012). Moreover, further explanations regarding using Gaussian Processes together with the Expected Improvement as the surrogate function, is found in (Bergstra et al., 2011).

A drawback of using the Gaussian Processes-approach in SMBO-models is an inability to handle categorical parameters, due to the continuity of the multivariate normal distribution. Another problem with the Gaussian Processes approach is that the kernel and the mean function also involve hyperparameters. Subsequently, we are in a sense simply replacing one optimisation problem with another.

3.2.3.2 Tree-structured Parzen Estimator

The second approach we will cover for the SMBO-algorithm, is the so called *Tree-structured Parzen Estimator* (TPE) (Bergstra et al., 2011). For this method, we obtain a model for $p(y \mid x)$ by first choosing models for $p(x \mid y)$ and $p(y)$. The first term, $p(x \mid y)$, is defined as

$$p(x \mid y) = \begin{cases} \ell(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \,, \end{cases} \tag{3.19}$$

where $\ell(x)$ is the interpolated probability density function by using the set of observations (hyperparameter configurations) whose objective function evaluations $F(x)$ fall below the threshold value y^*. Similarly, $g(x)$ is the probability density function interpolated from the rest of the observations.

Conceptually, we can by Equation (3.19) conclude, under the TPE model, that an optimal trial x^* to be evaluated by F should maximise $\ell(x)$ and minimise $g(x)$, or equivalently minimise the quotient $g(x)/\ell(x)$. However, to somewhat accurately interpolate $\ell(x)$, the threshold value y^* needs to be picked rather conservatively. By simply picking the lowest function evaluation of F as y^*, which we argued would be the most intuitive choice in Section 3.2.3, would make the TPE algorithm unable to interpolate $\ell(x)$. Therefore, it is required with this approach to pick y^* larger than the best observed $F(x)$. To make sure that $y^* > \min_x F(x)$, subsequently avoiding the interpolation problem, the TPE-algorithm chooses y^* such that $p(y < y^*) = \gamma$, where γ is some predetermined quantile. Moreover, we do not need to impose a specific model on $p(y)$ (Bergstra et al., 2011), but use the empirical distribution instead.

Applied as the model for the Expected Improvement surrogate, it is shown in Bergstra et al. (2011) that

$$EI_{y^*}(x) \propto \left(\gamma + \frac{g(x)}{\ell(x)}(1-\gamma)\right)^{-1} ,$$

which coincides with our previous reasoning that minimising $g(x)/\ell(x)$ is equivalent towards maximising the Expected Improvement under the TPE model.

Moreover, the TPE-algorithm is able to handle categorical parameters, unlike the Gaussian Processes approach. However, as covering this aspect of the

Tree-structured Parzen Estimator approach is outside the scope of this thesis, we refer to Bergstra et al. (2011) for further details.

3.3 Bayesian Artificial Neural Networks

Previously, in Section 3.1 we provided the framework of the *frequentist* viewpoint on Artificial Neural Network, though not mentioned explicitly. We viewed our model parameters w, being the weights and biases, as point estimates of their true parameter values. These estimates were implicitly derived as the maximum likelihood estimate (MLE). We will therefore denote it as

$$w^{MLE} = \arg\max_w \log P(\mathcal{D} \mid w) = \arg\max_w \sum_i \log P(y_i \mid x_i, w)\,, \tag{3.20}$$

and further assume that $\log P(y \mid x, w)$ is differentiable in w in order to solve for w^{MLE} with gradient descent based methods (Blundell et al., 2015).

In the context of regression problems with a continuous and normally distributed outcome, maximising the log likelihood for w is equivalent to minimising the squared loss, or the MSE for multiple observations, as shown in (Goodfellow et al., 2016, pp.131-134). This is due to the homogeneous and normality assumption about our residuals. However, if we would change our loss function to correspond to the MLE for another sampling distribution, we would impose different parametric assumptions.

Remark. *From now on we will denote the parameter vector as w to include both the weights and biases instead of p as in Section 3.1. We do this in order to not confuse parameters with probabilities. Furthermore, we denote a data pair as $\mathcal{D}_i = (x_i, y_i)$, whereby our notation $\mathcal{D}$ refers to a whole dataset of data pairs.*

With the Bayesian point of view however, we regard our different parameters w to follow some distribution we want to infer. From this *posterior distribution* $P(w \mid \mathcal{D})$ (see Apendix A, Definition A.3), we can get our predictive distribution of y by computing the expectation value of $P(y \mid x, w)$ with respect to this distribution, as

$$P(y \mid x) = E_{P(w \mid \mathcal{D})}\left[P(y \mid x, w)\right]\,. \tag{3.21}$$

In the context of working with ANNs, we quickly realise that evaluating the expectation in Equation (3.21) is often intractable since we would need to use an ensemble of infinitely many ANNs (Blundell et al., 2015) in order to calculate the evidence term $P(\mathcal{D})$ by Bayes' theorem.

3.3.1 Estimating the posterior distribution

Instead of calculating the posterior distribution analytically, our approach is to estimate it with a so called *variational posterior distribution*; an approach referred to as *variational inference* (Blei et al., 2016). In order to estimate a distribution, we need a metric to quantify the distance between an estimate $q(w) = q(w \mid \mathcal{D})$ and the true posterior distribution $p(w \mid \mathcal{D})$. A commonly used metric is the so called *Kullback Leibler divergence* (see Definition A.4 in Appendix A).

Remark. *The Kullback Leibler divergence is an example of a more general class of metrics called f-divergences or Csiszár functions that quantify the difference between two distributions (Csiszár and P.C. Shields, 2004, sec.4).*

By using the Kullback Leibler divergence (denoted $D_{\mathcal{KL}}$) we can write our optimisation task as

$$\begin{aligned} \hat{\theta} &= \arg\min_{\theta} D_{\mathcal{KL}}\left[q(w \mid \mathcal{D}) \,||\, P(w \mid \mathcal{D})\right] \\ &= \arg\min_{\theta} \left(D_{\mathcal{KL}}\left[q(w \mid \mathcal{D} \,||\, P(w))\right] - E_{q(w \mid \mathcal{D})}\left[\log P(\mathcal{D} \mid w)\right]\right) , \end{aligned} \tag{3.22}$$

where we have put the longer calculations in Calculation A.5 of Appendix A.

Following Equation (3.22), we define our objective function in the following way

$$\mathcal{F}(\mathcal{D}, \theta) = D_{\mathcal{KL}}\left[q(w \mid \mathcal{D} \,||\, P(w))\right] - E_{q(w \mid \mathcal{D})}\left[\log P(\mathcal{D} \mid w)\right] . \tag{3.23}$$

Remark. *As Blundell et al. (2015) point out, our objective function is also known as the (negative) Evidence Lower Bound (ELBO) (Blei et al., 2016).*

The vector θ contains the parameters of a given family of distributions, whereby our method for approximating of the true posterior will be parametric. Implicitly, we are using the uniqueness theorem (Gut, 2009, p.59) in probability theory which states that under certain conditions, a probability distribution is uniquely defined by its parameters. Even though Blundell et al. show by their method *Bayes by backprop*(agation), that the approximating class can be (under some conditions) any family of continuous distributions, due to ease of computation, we will only work with a diagonal multivariate normal distribution. By the definition of the covariance matrix, our variational posterior is therefore modelled to let each weight being independent of the others.

3.3.1.1 Learning the variational posterior

Since we have chosen our variational posterior to be within the family of diagonal multivariate normals, we know from the uniqueness theorem that our distribution of interest is uniquely defined by its parameters. Therefore, we wish to infer the vector of means and the vector of variances (since the covariances are assumed to be 0) such that the variational posterior distribution they define minimises our objective function.

Analogous with Section 3.1.4, learning the parameters of the variational posterior, i.e. solving for $\hat{\theta}$ in Equation (3.22), can be shown to be solved by the gradient based method Bayes by backprop (Blundell et al., 2015); a small alteration of the regular backpropagation algorithm. As shown explicitly in Calculation A.6 of Appendix A, our objective function in Equation (3.23) can be rewritten as

$$\mathcal{F}(\mathcal{D}, \theta) = E_{q(w \mid \mathcal{D})} \left[\log q(w \mid \mathcal{D}) - \log P(w) - \log P(\mathcal{D} \mid w) \right] .$$

However, in order to use gradient descent, we need to be able to compute

$$\frac{\partial}{\partial \theta} E_{q(w \mid \mathcal{D})} \left[\log q(w \mid \mathcal{D}) - \log P(w) - \log P(\mathcal{D} \mid w) \right] .$$

Proposition 3.3.1 (Gaussian reparameterisation trick)**.** *Let ϵ be a standard normally distributed random variable, $q(\epsilon)$ its probability density function and let $w = t(\theta, \epsilon)$ where $t(\theta, \epsilon)$ is a deterministic function. Suppose further that the variational posterior $q(w \mid \theta)$ is a diagonal multivariate normal distribution, then*

$$\frac{\partial}{\partial \theta} E_{q(w \mid \theta)} \left[f(w, \theta) \right] = E_{q(\epsilon)} \left[\frac{\partial f(w, \theta)}{\partial w} \cdot \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta} \right]$$

Proof. Proposition A.7 of Appendix A.

Remark. *Proposition 3.3.1 is a corollary of the proposition presented in Blundell et al. (2015) and it is known as the Gaussian reparameterisation trick (Opper and Archambeau, 2009).*

We apply Proposition 3.3.1 to our optimisation problem and let

$$f(w, \theta) = \log q(w \mid \theta) - \log P(w) - \log P(\mathcal{D} \mid w) .$$

Moreover, to make sure that the standard deviations σ are always positive, we need to find a transformation that maps the real numbers to the positive

reals in order to parametrise the standard deviation as a function of a trainable parameter defined for all reals. The transformation $f(x) = \log(1 + e^x)$ is an example of such an transformation, which is also used in Blundell et al. (2015). Therefore, we parametrise the standard deviation as $\sigma = \log(1 + e^\rho)$ and hence we are able to train ρ for any range.

Lastly, since $w \sim \mathcal{N}(\mu, \sigma^2 I)$, $\theta = (\mu, \rho)^T$ and $\sigma = \log(1 + e^\rho)$ we get by the chain rule that

$$\begin{aligned}\frac{\partial w}{\partial \mu} &= 1\,,\\ \frac{\partial w}{\partial \rho} &= \frac{\epsilon}{1 + \exp(-\rho)}\,,\end{aligned}$$

where ϵ occurs in the second formula due to the fact that $w = t(\theta, \epsilon) = \mu + \log(1 + e^\rho) \circ \epsilon$ according to Proposition 3.3.1.

Remark. *With the notation $\circ$ we mean the pointwise multiplication. In other words, since $\epsilon \sim \mathcal{N}(0, I)$ it is implied by the notation*

$$w = \mu + \log(1 + e^\rho) \circ \epsilon\,,$$

that $w \sim \mathcal{N}\left(\mu, I \cdot \log(1 + e^\rho)^2\right)$ due to the properties of a (multivariate) normal distribution.

Concluding this section, we can write the algorithm for updating the Gaussian variational posterior parameters in the following very concise way (as in Blundell et al. (2015)):

1. Sample $\epsilon \sim \mathcal{N}(0, I)$.
2. Let
$$\begin{aligned}w &= \mu + \log(1 + \exp(\rho)) \circ \epsilon\,,\\ \theta &= (\mu, \rho)^T\,,\\ f(w, \theta) &= \log q(w \,|\, \theta) - \log P(w) - \log P(\mathcal{D} \,|\, w)\,.\end{aligned}$$
3. Calculate the gradients
$$\begin{aligned}\Delta_\mu &= \frac{\partial f(w, \theta)}{\partial w} + \frac{\partial f(w, \theta)}{\partial \mu}\,,\\ \Delta_\rho &= \frac{\partial f(w, \theta)}{\partial w} \cdot \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(w, \theta)}{\partial \rho}\,.\end{aligned}$$

4. Update the parameters with respect to the learning rate α

$$\mu \leftarrow \mu - \alpha\Delta\mu\,,$$
$$\rho \leftarrow \rho - \alpha\Delta\rho\,,$$

where this scheme is iterated subject to the number of epochs or any other stopping criterion, as briefed upon in Section 3.1.4.3.

Interestingly, the term $\partial f(w, \theta)/\partial w$ can be solved by the standard form of backpropagation (see Section 3.1.4.2) (Blundell et al., 2015), whereby a smaller alteration[1] of the backpropagation algorithm allows for modelling the parameters w as random variables.

3.4 Dimensionality reduction

In its essence, the phrase *curse of dimensionality* was originally coined by Richard Bellman and it generally refers to that many common algorithmic approaches in optimisation, function approximation, numerical integration among others becomes exponentially more difficult as the number of dimensions increases (Donoho, 2000). This trend of dealing with high dimensional data has increased rapidly in recent years (Jolliffe and Cadima, 2016), probably due to easier access. This emphasises the need to excessively reduce the number of variables in order to draw meaningful inference, i.e. picking the most valuable insight. Manual dimensionality reduction for very high dimensional data quickly scales up and almost becomes an impossible mission. For this reason, the need of algorithmic approaches arises naturally.

Central to all theory covering dimensionality reduction is the so called Manifold Hypothesis (Fefferman et al., 2013; Goodfellow et al., 2016, pp.162-165) which is an assumption that the important structure of a high dimensional dataset resides in a lower dimensional topological space (manifold).

3.4.1 Principal Component Analysis

One of the most common approach in modern data analysis is to use a method called *Principal Component Analysis (PCA)* (Jolliffe and Cadima, 2016). At its core, the idea is to remove multicollinearity in a given dataset X by proper dimensionality reduction. To do this, the PCA algorithm's objective

[1] However, this alteration might be noticeably more complex for other families of distributions than diagonal multivariate normals.

is to find linear combinations aX of X such that the new data aX maximises the variance $\text{Var}(aX)$. Implicitly, we rely on the assumption that a variable with a large variance contains important structure for our dataset (Shlens, 2014). This assumption may however not always be applicable for noisy data.

Proposition 3.4.1 (Eigendecomposition for PCA)**.** *The eigenvectors a with the highest eigenvalue, under the assumption that these vectors also are unit-norm vectors*[2] *($a^T a = 1$), are the linear combinations aX of our original data matrix X that maximise the variance.*

Proof. See Proposition A.2 of Appendix A.

Using Proposition 3.4.1, we can compute the number d of so called *principal components* by performing the matrix multiplication

$$T = XA\,, \tag{3.24}$$

with A being the eigendecomposition of $X^T X$, a matrix whose column vectors (the eigenvectors) are sorted by the magnitude of their corresponding eigenvalues (vectors with highest eigenvalues are placed to the left in A). Therefore, if X is of dimension $n \times m$, by truncating A to contain $d < m$ column vectors, our transformed data matrix T has dimensions $n \times d$ and hence we have reduced the dimensionality of our data. Obviously, the more principal components we choose to include in T, by picking a bigger d, the more variance will be explained endogenously (by the model).

Remark. *In this paper, when the term principal component is used, we refer to it only as the column vectors in T from Equation (3.24) and* ***not*** *the column vectors in A, as some literature also uses the same term for (Jolliffe and Cadima, 2016).*

Since the covariance matrix $X^T X$ has the well-known property of being symmetric, it can be proven that its eigenvectors will be orthogonal to each other (Goodfellow et al., 2016, p.149). This provides a convenient geometrical interpretation of PCA where each principal component corresponds to a direction and where its eigenvalue answers how much variance lies in that specific direction. Under the assumption that larger variance implies a clearer structural difference, our first principal components will pick the most influential directions.

[2]Demanding that $a^T a = 1$ is common in order to solve the problem analytically (Jolliffe and Cadima, 2016).

Moreover, PCA has a close relationship with *Singular Value Decomposition* that provides a more generalised view on PCA. We will stick to the definition through the eigendecomposition, but the interested reader can find further generalisations in (Goodfellow et al., 2016, sec.5.8.1), Jolliffe and Cadima (2016) and Shlens (2014).

Chapter 4

Experiments

4.1 Outline

In this section, the methodology for our empirical results presented in Section 4.2, is outlined. The technical aspects in terms of software versions and references to code snippets can be found in Appendix B. As an overview, our approach of applying the theory of Bayesian Artificial Neural Networks to our research problem of interest is listed below. A more detailed explanation of each step can be found in Section 4.1.1 - 4.1.4.

1. Pre-process the input data
2. Define a prior distribution over the model parameters and define parametric assumptions.
3. Specify hyperparameters and their range. Thereafter, perform hyperparameter optimisation and compare models by their loss score on the validation set.
4. Criticise the best performing model by computing the posterior predictive distribution with Monte Carlo methods and compare it with true response values from the validation dataset.

4.1.1 Data pre-processing

The very first part of the experiments was to process the data in order to prepare it for further analysis. We began by performing simple manual dimensionality reduction; removing 59 variables that did not have any variance at all and therefore did not add any structural information about the mapping. Furthermore, we noted a strong pairwise correlation in between our

variables. Out of 6849 variables, 3147 variables were manually removed on the basis that its correlation coefficient with another variable in the dataset was exactly 1, in other words they conveyed exactly the same information. Thereafter, we reduced for multicollinearity by choosing the number d of principal components with the highest variance (see Section 3.4.1), where the integer d was subject to hyperparameter optimisation, elaborated further on in Section 4.1.3.

To prevent overfitting, we split our dataset of 606 observations roughly according to the 80/20-rule referred to in Section 3.1.7.1, by assigning 484 observations for model training and the remaining 122 observations for validation (described in Section 4.1.4). Ideally, we would also analyse a final test dataset in order to check that the model found from our hyperparameter optimisation simply was not subject to randomness. However, as we do not possess too many data observations nor intend to use the final model for production, we will not include this step of the validation.

Lastly, in order to increase convergence speed, our data was normalised as described in Section 3.1.6 with the standard score

$$\frac{x_i - \mu_i}{\sigma_i}\,.$$

Assuming homogeneity between the training and validation datasets (see Section 3.1.7.1); the latter was transformed by the sample mean and standard deviation from the training dataset.

4.1.2 Prior beliefs and model assumptions

We modelled our prior beliefs over our effect parameters (i.e. the weights and biases, denoted by w) with a multivariate diagonal normal distribution, where each expected value was set to 0. To use a zero expectation vector is to be interpreted as we did not want to state beforehand whether each weight would have a positive or negative effect on its output.

Moreover, since we lack the domain knowledge, we impose no string prior beliefs in terms of the variances. Instead, we used a relatively wide variance of 1 as suggested in Blundell et al. (2015), in order to obtain more conservative subjective beliefs. Subsequently, the prior of our weights (and biases) was

$$w \sim \mathcal{N}(0, I)\,.$$

Lastly, as we used a squared loss, we implicitly assumed homoscedasticity and normality for the residuals, i.e. $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ for all $i = 1, \ldots, n$ where n is the number of samples in the validation set.

4.1.3 Hyperparameters and trials

Our approach to find a proper model was composed by letting the set of parameters, as specified in Table 4.1, be subject to hyperparameter optimisation, with corresponding ranges and sampling distributions. Specifying ranges and sampling distributions for our hyperparamaters can, from the Bayesian point of view, be interpreted as our prior beliefs about the topology of the objective function.

To minimise the dimensionality of our hyperparameter optimisation problem, we held the batch size fixed to 44 (such that the training set of 484 observations is evenly divisible by the batch size) and the number of epochs to 10 000. Moreover, we made an assumption about symmetry in between the hidden layers. In other words, we assumed that the number of neurons and activation functions were the same through the hidden layers in order to drastically decrease the dimensionality of our hyperparametric space.

Hyperparameter	**Range**	**Sampling distribution**
Learning rate	$[10^{-3}, 0.1]$	Uniform
Hidden layers	$\{1, \ldots, 7\}$	Uniform
Activation function	{Sigmoid, ReLU}	Uniform
Neurons/layer	$\{3, \ldots, 200\}$	Uniform
Principal components	$\{20, \ldots, 1\,000\}$	Uniform
Batch size	44	Constant
Epochs	10 000	Constant

Table 4.1: Initial hyperparameters, their ranges and sampling distributions, and with symmetry between the hidden layers assumed. Continuous ranges are denoted with $[\cdot, \cdot]$ and discrete ranges with $\{\cdot, \ldots, \cdot\}$, the latter having increments of 1 (if not categorical elements).

For the hyperparameter trials, we chose the Tree-Structured Parzen Estimator[1] (see Section 3.2.3.2) as our hyperparameter optimisation algorithm and we performed 1 000 trials. Each trial was performed by training a model

[1]With $\gamma = 0.25$ to determine y^*.

with a given set of hyperparameters from Table 4.1 and evaluating the loss value for each configuration on the validation set to be used as the criterion of model selection.

4.1.4 Model criticism

While a numeric loss value enables us to compare different models under the assumption that lower losses yield better performance in terms of predictive accuracy, these numeric values themselves are not interpretable in an absolute sense. Instead, we are interested in criticising the predictive distribution that can be interpolated for a given model.

However, evaluating a predictive distribution for a given model is not as straightforward as for the frequentist approach of fixed point estimates for the true value of the expected outcome. In the frequentist setting, one can easily evaluate the model accuracy with the MSE metric, for instance, by comparing the predicted value with the true value for the whole validation dataset. With the Bayesian viewpoint however, evaluating the accuracy will arguably not be as straightforward since our view of the prediction is in terms of a probability density function.

One approach would be to compute the mode of the distribution to represent a single point estimate similar to frequentist methods and compare the accuracy accordingly. This is commonly referred to as the maximum-a-posteriori (MAP) estimate (Goodfellow et al., 2016, pp.138-139). However, using the MAP estimate is arguably not reflecting the aims of the Bayesian viewpoint. While it can provide some insights about the model accuracy, we are truncating a lot of information inherent in the predictive distribution if only the MAP estimate is used.

Furthermore, if the model outputs a predictive distribution with a wide variance, we can interpret the output as a high degree of uncertainty about the predicted outcome, which in the context of drug predictions is very valuable information and not something we would discard as an undesirable output. Instead, another appealing property of our model would be to correctly assign a credibility interval, within which we could feel confident about the true value being located. To evaluate such a credible interval, one could approximate the predictive distribution by Monte Carlo methods and interpolate the density function to obtain percentiles that define a satisfactory credibility interval. In order to evaluate the model's ability of accurately depict predictive uncertainty, we could compute the number of times the true label

is within the constructed credibility interval. The corresponding frequency would ideally be approximately the same as the nominal coverage probability we assign to the interval.

In order to evaluate the final model's performance, we chose to use the frequency of how often the true target values in the validation dataset was within a 95% credible interval constructed by the empirical distribution based on 10 000 Monte Carlo simulations. Furthermore, we used the MAP estimate, also based on 10 000 Monte Carlo simulations, in order to compute the MSE as for frequentist methods.

4.2 Results

After performing our hyperparametric trials as described in Section 4.1, we found the model presented in Table 4.2 which we will refer to as the final model.

Hyperparameter	**Final model**
Learning rate	0.09
Hidden layers	7
Activation function	Sigmoid
Neurons/layer	3
Principal components	369
Loss value	0.476
MSE	0.353
Frequency for $y \in \mathbf{CI}_{95\%}$	0.426

Table 4.2: The characteristics for the final model in terms of its hyperparameters we optimised for. Note that the accuracy metrics are subject to randomness and may alter slightly between different runs. The notation $y \in \mathrm{CI}_{95\%}$ means the percentage of times the true value y, in the validation dataset, was inside the empirical 95% credibility interval.

Chapter 5

Discussion

5.1 Interpreting the results

As the reader might have realised by this point, modelling our problem of interest can be done in countless different ways in terms of the hyperparametric configuration and subjective assumptions such as the prior distribution. Therefore, we want to emphasise that our approach in terms of the empirical experiments in Chapter 4 is merely to be seen as one of many ways of constructing these type of models. While the abstractions of the class of Neural Networks allow for generalisations of the more well-known single layer regression models, it comes with a great cost of the complexity for model fitting due to a largely increased hyperparameter space. Furthermore, since the assumption about symmetry in between the hidden layers, we naively disregard the effect of each layer's hyperparametric configuration. For instance, the number of optimal neurons and activation functions might alter noticeably throughout the hidden layers of the network. This is one immediate suggestion for potential model improvement.

However, if we consider the final model returned by our hyperparameter optimisation in Chapter 4, our evaluation metrics suggest that the accuracy of our model is arguably very poor. This is indicated both in terms of the one point predictors based on the MAP estimates (see Section 4.1.4) and accurately approximating the probability that the credibility interval covers the true value.

Moreover, it is noticeable that two of our hyperparameters for the final model fell within the endpoints of our specified range for the hyperparameter space; our prior beliefs about the hyperparameters. The number of neurons per layer

was chosen as 3 and the number of hidden layers as 7; the lower and upper bounds from our ranges depicted in Table 4.1, respectively. This suggests that we perhaps should revise our prior beliefs to increase the range of our search space, but the inferred hyperparameters could also be a consequence of the symmetry assumption. Another noteworthy aspect of the result is the somewhat drastic dimensionality reduction from the input layer of 369 principal components towards 3 latent variables in the first (and upcoming) hidden layers. While it is hard to directly reason about the feasibility of such a transform, we see this aspect as something to track for further work when liberating the model from the symmetry assumption.

Another aspect to mention is the parametric assumption about homoscedastic and normality for our residuals. Since we did not find our proof-of-concept model from Chapter 4 to perform satisfactory in terms of its predictive ability, we did not see the purpose of evaluating the model further. In a real setting however, testing for normality and homogeneity of variance would be a crucial step before accepting the model as satisfactory for production use.

For production usage, we would also like to extend our validation phase of the model to cross validate the our chosen metric with the K-Fold Cross-Validation method for instance (see Section 3.1.7.2 and Algorithm 2 of Appendix A) in order to feel more confident about our model evaluation. As mentioned earlier, this is clearly a trade-off between time complexity for the computations and evaluation accuracy.

Recall the assumption that all ADME values can be inferred with good accuracy from the distribution parameter value (the letter D in the abbreviation ADME). It is worth mentioning that we do still not know how this assumption relates in practice in terms of fitting the full four-dimensional[1] problem of interest. Furthermore, one should also be careful about extrapolating any empirical results based on (parts of) the ADME prediction problem to qualitative questions regarding drug predictions. We want to emphasise that we do not have the domain knowledge to reason about the more precise impact of ADME predictions in terms of qualitative inference for drug predictions or drug discovery.

Lastly, an important trade-off to mention when interpreting the results of models within the class of Artificial Neural Networks is the ability of inter-

[1]Recall from Section 1.3 that each element in the $(A, D, M, E)^T$-vector could be multi-dimensional in the general setting.

preting the effect parameters. For more complex models as for ANNs, it is much harder, arguably even intractable, to directly interpret the most important explanatory variables for the response variable compared to single layer models. In the multiple regression setting for instance, the model definition allows for much clearer interpretation of the effect parameters as how much each explanatory variable impacts the response. However, when propagating the output from the first layer in an ANN model into a similar affine operation in the upcoming layer, we lose much of the ability for easy interpretation of the effects. Especially since other rather complex operations such as PCA, Z-normalisation and non-linear activation functions are added. Therefore, *if* the introduction of hidden layers, i.e. what usually characterises the ANN models, would perform noticeably better than single layer models; it comes with the trade-off of not being able to relatively easy interpret the model parameters. Although explanatory models as single layer regression models would never account for causal inference, ANN models would still be a more questionable choice in problem domains where interpretation is important.

5.2 Improving our model

As the scope of our work was heavily weighted towards providing the theoretical framework for using the class of Bayesian Artificial Neural Networks for predictive modelling, we see a lot of interesting aspects for model improvement purposes as suggestions for further work. In the previous section, we talked about the symmetrical hyperparametric assumption and briefed upon testing its validity. One approach would be to hold the optimal number of hidden layers fixed throughout a second hyperparameter optimisation phase where we allow the number of neurons and activation functions differ between layers. However, this would presuppose that the optimal number of hidden layers would be the same for the two different optimisation phases. In the light of computationally expensive trials, there exists a strong incentive to reduce the dimensionality of our hyperparameter vector by imposing some assumptions based on practical experience.

Moreover, the results from our hyperparameter optimisation algorithm are by no means any guarantee that the returned model is near optimal in the specified hyperparametric space. For further studies, it would be interesting to compare the performance of different hyperparameter optimisation algorithms, to potentially identify better models. It is perhaps also obvious that further trials with any algorithm would yield better chances of finding better models.

Other immediate points of interest we target as potential improvement areas are the choice of training algorithm (Goodfellow et al., 2016, pp.306-317), other types of activation functions (Goodfellow et al., 2016, pp.195-197) and *Batch Normalisation* (Ioffe and Szegedy, 2015). Especially, we find Batch Normalisation interesting since Ioffe and Szegedy (2015) argue that it improves the capability of fitting deeper models, in other words models with a higher number of hidden layers. Due to the deep structure (many hidden layers) for our final model, we find the idea of layerwise normalisation interesting as a subject for further studies. However, we feel modest towards the rich variety of other extensions of our model that could potentially be proven to increase performance.

Furthermore, as briefed upon in Section 3.3.1, the variational posterior $q(w \mid \mathcal{D})$ does not necessarily have to be picked within the family of diagonal multivariate normals, as have been used for our work. However, other types of distributions may drastically change the complexity of the learning scheme that was covered in Section 3.3.1.

5.2.1 Prior beliefs

As mentioned before, Bayesian inference is dependent on the specified prior distribution that is supposed to reflect our subjective beliefs. In terms of putting prior beliefs upon the weights, this can be a tricky quest even for a person with extensive domain knowledge. Our approach of defining a standard normal distribution for each weight, therefore also specifying a-priori independence between our weights (and biases), is questionable regarding its accuracy. As described in Section 4.1.2, we chose a somewhat simple prior distribution with a relatively large variance to reflect our lack of domain knowledge. However, we believe this is an important area for discussion whether our prior can be assumed to be accurate and if modelling a prior distribution over the weights is valid. Regarding the validity, it is worth mentioning that a somewhat analogous problem also regards the frequenstist viewpoint due to the problem of initialising the parameters (see Section 3.1.4.3), although arguably not a typical issue in maximum likelihood estimation. This problem is due to the fact that even frequentist ANNs are sensitive towards their initial parameter values, specifically when using Stochastic Gradient Descent as the learning algorithm (Ioffe and Szegedy, 2015).

Furthermore, Blundell et al. (2015) suggest a so called Scale Mixture Prior

defined as

$$P(w) = \prod_{i=1}^{n} \left[\pi \mathcal{N}(w_i \,|\, 0, \sigma_1^2) + (1 - \pi) \mathcal{N}(w_i \,|\, 0, \sigma_2^2) \right] , \tag{5.1}$$

where $\mathcal{N}(x \,|\, \mu, \sigma^2)$ denotes a normal density function evaluated at x. Therefore, we can interpret this Scale Mixture Prior as two separate normal distributions mixed together in accordance with the mixing probability parameter $\pi \in [0, 1]$. Moreover, the authors also advise that $\sigma_1 > \sigma_2$ and $\sigma_2 \ll 1$ in order to put much of the density around the mean value of 0, but still account for prior uncertainty by adding a second component with a larger variance. By still assuming independence between the different model parameters in w, we can define the multivariate Scale Mixture Prior as the product of the marginal density functions as in Equation (5.1).

One way of solving for the parameters π, σ_1 and σ_2 that define the Scale Mixture Prior distribution would be through hyperparameter optimisation. However, approaches where the distribution parameters in the prior distribution is determined empirically is known as empirical Bayes which validity is sometimes questionable (Gelman, 2008). Furthermore, it was found empirically that tuning prior parameters under training yielded worse results for the Bayes by backprop algorithm (Blundell et al., 2015). Despite the problem of choosing the prior parameters, as the algorithm is free to pick a prior distribution in any family of distributions, the Scale Mixture Prior would account for another interesting approach of modelling our prior beliefs.

5.2.2 Weighted objective

As we saw in Equation (3.23) from Section 3.3.1, our objective function that we would like to minimise contains two terms; the likelihood cost and the Kullback Leibler divergence between a given variational posterior to optimise and the prior distribution. The likelihood cost is, besides viewing our model parameters as random variables, the one used for standard forms of non-Bayesian ANNs. The Kullback Leibler divergence term has moreover a very interesting interpretation of punishing models that differ too much from our prior beliefs. Given that our subjective beliefs are somewhat accurate, this term could be viewed as an implementation to account for model complexity in the optimisation phase, whereby we implicitly apply the principle of Occam's razor. However, since our objective is a sum of two terms, we propose for further work to look into parameterisations of these two terms in order to weight their importance against each other. In Blundell et al. (2015), the

authors suggest using a parameterisation that is dependent on which phase of the learning we are within. They suggest multiplying the Kullback Leibler term of Equation (3.23) with a parameter

$$\pi_i = \frac{2^{M-i}}{2^M - 1}\,,$$

where $i \in \{1, \ldots, M\}$ is the current batch (see Section 3.1.4.1). This scheme allows the model to be heavily influenced by the Kullback Leibler term (or the complexity cost) initially for the first batches, but then later on be influenced by the likelihood cost, i.e. the data dependent term.

5.2.3 Heteroscedasticity

As briefed upon both in Section 3.3 and Section 5.1, we made an assumption about homoscedasticity for our sampled sequence of random variables, in other words the sampling distribution of the outcome. While the homoscedastic assumption allows for an easier objective function in terms of calculating the likelihood cost, it is a pretty strong assumption which we would need to test the validity of upon further modelling. One less strong assumption we particularly find interesting is the heteroscedastic one where the residual variance is allowed to alter between different areas of the data domain. We believe this area would be particularly interesting to investigate further, not only due to arguably more reasonable model assumptions in many cases, but due to the nature of how Bayesian methods really shine in terms of production value. If we consider the regression setting with an unbounded outcome variable, a heteroscedastic model viewpoint would allow for less confident extrapolations in areas where observed data is sparse, but converge towards tighter predictive beliefs in data rich parts of the domain. In order to implement the heteroscedastic gaussian model, we would need to change or (negative) loglikelihood towards

$$-\log P(\mathcal{D} \mid w) = \log \sigma(x) + \frac{(y - \mu(x))^2}{2\sigma^2(x)} + C\,,$$

where C is some constant. This is according to the paper by Lakshminarayanan et al. (2016). A similar approach can also be found in (Gal, 2016, sec.4.6).

Concluding the discussion regarding heteroscedasticity, we believe similarly to Gal, that heteroscedastic models can be very useful for problems where the residual error variance change throughout the explanatory domain. This

could arguably be useful in the context of drug discovery when it is of particular interest not to be too confident about extrapolating empirical results over less explored parts of the domain.

5.3 Working with dimensionality

While Donoho provides greater theoretical arguments on the topic regarding the *blessing of dimensionality* in Chapter 8 of Donoho (2000), we will brief upon this topic a bit more heuristically. Although the curse of dimensionality (see Section 3.4) certainly applies to a lot of real-world situations and hence makes data analysis exponentially harder, arguably more often in times with increased access to data, it also provides a greater potential for efficient variable selection. Previously, a more common scenario for statisticians was to work with datasets characterised by a smaller number of variables, but still potentially many observations (Donoho, 2000, chap.3). However, in the 21st century, a more common scenario is to encounter datasets with a high-dimensional variable space. Subsequently, problems regarding dimensionality reduction and the Manifold Hypothesis (see Section 3.4) is arguably more important than ever. Nonetheless, with the topological perspective of the Manifold Hypothesis, an increased dimensionality could possibly allow statisticians and analysts to more efficiently target manifolds with greater explanatory capacity for a given task. Therefore, it illustrates the potential of the development of modern data analysis.

5.3.1 Instabilities of PCA and more robust alternatives

By using a standard form of PCA, covered in Section 3.4.1, we are implicitly assuming that the lower dimensional manifold according to the Manifold Hypothesis is linear, since PCA only computes linear combinations of our original data. While this approach might work for many applications and simplifies the theory, it can still be a questionable assumption topologically.

Another approach for investigating other types of non-linear manifolds is to use Kernel PCA, which applies a Kernel (Murphy, 2012, chap.14) (briefly described in Section 3.2.3.1) that can be viewed as a parametrisation of the operation of matrix multiplications. However, as we have previously encountered, any generalisation comes with the trade-off of having more alternatives to cover. In the context of Kernel PCA, we would also need to optimise for the best Kernel-method and its parameters; increasing our hyperparametric space further.

5.4 Generative models

5.4.1 The reversed regression problem

For this paper, we have solely been dealing with the regression problem of finding a function $F : X \to Y$, where X would denote a given drug and Y its ADME attributes, or a subset of them. However, as mentioned in Section 5.1, interpreting the results of this high dimensional regression problem with a fairly complicated model can be challenging even for the most experienced analyst. In the context of drug discovery, one would beyond accurately be predicting the effects of a given drug in terms of pharmacokinetics, want to infer the properties of a new drug for a given disease or set of symptoms. Assuming that the ADME attributes explain the qualitative pharmacological (focused solely on pharmacokinetics) effects of a drug, then the process of finding new better drugs for a given area of treatment corresponds to the regression problem of finding $F' : Y \to X$, i.e. mapping the ADME values to a potential drug.

5.4.2 Variational Autoencoders

One method to solve for this reversed problem is called *Variational Autoencoders* and is an unsupervised approach (i.e. not having true labels or outputs to compare its results with). This method tries to find a latent space Z, that as compactly and well as possible describes the properties of its input X. Then this input is reconstructed again into X' where the task is to minimise a norm $||X - X'||$. Formally, we can view the task as first finding a function $F : X \to Z$ and then another function $G : Z \to X'$, where X' ideally reflects the key attributes of X. These functions F and G are approximated with ANNs, which by Section 3.1.5 are universal function approximators, if tuned correctly. Moreover, since in our application (and other common applications), we want to sample many candidates, Variational Autoencoders model the latent space to be a distribution instead of fixed values. Interestingly, the idea of estimating this true distribution over our latent space is also based on variational inference, as we described in Section 3.3.1. The interested reader will find more elaborate explanations in (Goodfellow et al., 2016, pp.696-699).

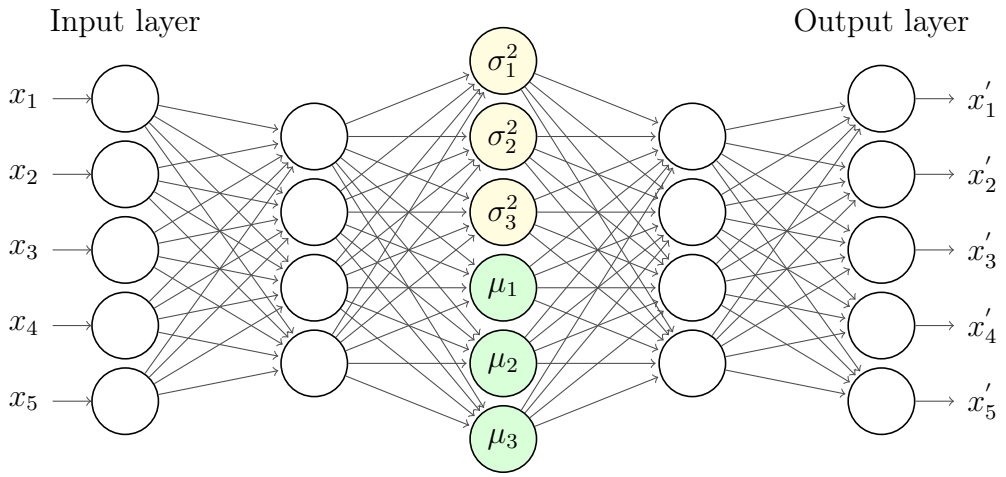

Figure 5.1: The architecture of a Variational Autoencoder with normally distributed latent variables. Note that the dimensionality of each layer could be arbitrary in the general case.

Assuming we could successfully construct a Variational Autoencoder for our problem of interest, we would then have solved the first partial problem of the reversed regression problem. At this stage, we would like to find a mapping between our latent space Z and the true ADME values, denoted as Y. This amounts to finding a function $H : Y \to Z$, which is then composed with G. If successful, we would have solved the inverted regression problem $F' : Y \to X$ by means of $G \circ H : Y \to X'$.

Naturally, a central problem would be to test for biological feasibility of the generated features for the output drug. However, if we assume this task could be tackled with some, not too computationally expensive, form of brute force approach, many samples could be drawn from the distribution over the latent variables to later be tested for feasibility.

Similar approaches with using variants of the Variational Autoencoder, extended with a latent space mapping from the labelled outcome, have been made by Gómez-Bombarelli et al. (2016), Kadurin et al. (2017) and Rampasek et al. (2017).

Moreover, if choosing to model the reversed regression problem with this suggested approach, it is important to realise that an implicit model assumption would be that minimising the norm of $||X - X'||$ implies that similar molecular properties are shared by X and X'. As we do not have the domain

knowledge nor empirical results to rely on, this approach should rather be interpreted as merely a suggestion for further analysis, given that the reversed regression problem is of interest.

Lastly, besides being used for solving for the reversed regression problem, generative models like the Variational Autoencoder could also be used for increasing the already limited training set in order to improve our model's performance of generalising its function approximation to unseen data. It would be important here, however, not to include these augmented or bootstrapped observations in our validation or test dataset in order to avoid biased inference.

5.5 Other methods for measuring uncertainty

Concluding the discussion chapter, we want to point out that although the variational inference approach of depicting a Neural Network's predictive uncertainty is interesting, it is by no means the *only* one; there exist other methods, based on both frequentist and Bayesian statistics, that could potentially also be used for our problem of depicting predictive uncertainty.

Another approach would be *Markov Chain Monte Carlo* (MCMC) methods such as the Metropolis Hastings Algorithm (Held and Bov, 2013, sec.8.4) to approximate the true posterior distribution by sampling. One problem however with MCMC methods however is that determining convergence and checking for model assumptions will in many cases be problematic (Gelman, 2008).

As a final note, in Khosravi et al. (2011), the authors describe and compare empirically other alternatives based on the Delta method, Bayesian bootstrap and Mean-Variance Estimation. Similar approaches of comparing variational inference based BANNs with other methods for modelling predictive uncertainty, could be an interesting approach for future research.

Chapter 6

Conclusion

The purpose of this work was to provide the theoretical framework of using Bayesian Artificial Neural Networks (BANNs) for predictive modelling in domains like drug discovery where it is crucial to accurately account for predictive uncertainty. We did this by assuming that the specific problem of predicting drug effects was representative to this larger class of problems and briefed upon its usability on empirical grounds under some simplifications. Although the scope of the thesis was never to find a state-of-the-art model, our results did not directly prove the applicability of this class of models to drug predictions. Nonetheless, we feel very humble towards the complexity of fitting near-optimal BANN models where our experimental work is simply to be seen as one out of many ways to constructing these type of models. Furthermore, it is with great excitement we look forward to see future work in the field of probabilistic modelling within this type of domain.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Y. Bengio and Y. Grandvalet. No Unbiased Estimator of the Variance of K-Fold Cross-Validation. *Journal of Machine Learning Research*, 5:1089–1105, 2004. URL `http://dblp.uni-trier.de/db/journals/jmlr/jmlr5.html#BengioG04`.

J. Bergstra and Y. Bengio. Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.*, 13:281–305, Feb. 2012. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2188385.2188395`.

J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox. Hyperopt: a Python library for model selection and hyperparameter optimization. *Computational Science Discovery*, 8(1):014008, 2015. URL `http://stacks.iop.org/1749-4699/8/i=1/a=014008`.

J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011. URL `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational Inference: A Review for Statisticians. *ArXiv e-prints*, Jan. 2016.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Networks. *ArXiv e-prints*, May 2015.

Csiszár and P.C. Shields. Information Theory and Statistics: A Tutorial. *Foundations and Trends in Communications and Information Theory*, 1 (4):417–528, 2004.

A. Czarnik and H.-Y. Mei. 2.12 - How and Why to Apply the Latest Technology*. In J. B. Taylor and D. J. Triggle, editors, *Comprehensive Medicinal Chemistry {II}*, pages 289 – 557. Elsevier, Oxford, 2007. ISBN 978-0-08-045044-5. doi: https://doi.org/10.1016/B0-08-045044-X/00048-1. URL https://www.sciencedirect.com/science/article/pii/B008045044X000481.

D. L. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. In *AMS CONFERENCE ON MATH CHALLENGES OF THE 21ST CENTURY*, 2000.

C. Fefferman, S. Mitter, and H. Narayanan. Testing the Manifold Hypothesis. *ArXiv e-prints*, Oct. 2013.

Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

A. Gelman. Objections to Bayesian statistics. *Bayesian Anal.*, 3(3): 445–449, 09 2008. doi: 10.1214/08-BA318. URL https://doi.org/10.1214/08-BA318.

F. Giordano, W. Fox, and S. Horton. *A First Course in Mathematical Modeling*. Cengage Learning, 2013. ISBN 9781285531762.

X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL http://proceedings.mlr.press/v15/glorot11a.html.

R. Gómez-Bombarelli, D. K. Duvenaud, J. M. Hernández-Lobato, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *CoRR*, abs/1610.02415, 2016. URL http://arxiv.org/abs/1610.02415.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

A. Gut. *An Intermediate Course in Probability*. Springer Publishing Company, Incorporated, 2nd edition, 2009. ISBN 1441901612, 9781441901613.

I. Guyon, J. Li, T. Mader, P. A. Pletscher, G. Schneider, and M. Uhr. Competitive Baseline Methods Set New Standards for the NIPS 2003 Feature Selection Benchmark. *Pattern Recogn. Lett.*, 28(12):1438–1444, Sept. 2007. ISSN 0167-8655. doi: 10.1016/j.patrec.2007.02.014. URL http://dx.doi.org/10.1016/j.patrec.2007.02.014.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

L. Held and D. S. Bov. *Applied Statistical Inference: Likelihood and Bayes*. Springer Publishing Company, Incorporated, 2013. ISBN 3642378862, 9783642378867.

R. Hill. *Drug Discovery and Development - E-Book: Technology in Transition*. Elsevier Health Sciences, 2012. ISBN 9780702053160. URL https://books.google.se/books?id=jarRAQAAQBAJ.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. ISSN 0027-8424. URL http://www.pnas.org/content/79/8/2554.

K. Hornik. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Netw.*, 4(2):251–257, Mar. 1991. ISSN 0893-6080. doi: 10.1016/0893-6080(91)90009-T. URL http://dx.doi.org/10.1016/0893-6080(91)90009-T.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*, Feb. 2015.

I. T. Jolliffe and J. Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 374(2065), 2016. ISSN 1364-503X. doi: 10.1098/rsta.2015.0202. URL http://rsta.royalsocietypublishing.org/content/374/2065/20150202.

A. Kadurin, S. Nikolenko, K. Khrabrov, A. Aliper, and A. Zhavoronkov. druGAN: An Advanced Generative Adversarial Autoencoder Model for de Novo Generation of New Molecules with Desired Molecular Properties in Silico. *Molecular Pharmaceutics*, 14(9):3098–3104, 2017. doi: 10.1021/acs.molpharmaceut.7b00346. URL https://doi.org/10.1021/acs.molpharmaceut.7b00346. PMID: 28703000.

A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya. Comprehensive Review of Neural Network-Based Prediction Intervals and New Advances. *Trans. Neur. Netw.*, 22(9):1341–1356, Sept. 2011. ISSN 1045-9227. doi: 10.1109/TNN.2011.2162110. URL http://dx.doi.org/10.1109/TNN.2011.2162110.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *ArXiv e-prints*, Dec. 2016.

Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65311-2. URL http://dl.acm.org/citation.cfm?id=645754.668382.

R. McFadden. *Introducing Pharmacology for Nursing and Healthcare*. Pearson, 2nd edition, 2013. ISBN 9781447927754, 9781447927785.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

M. A. Nielsen. Neural networks and deep learning, 2018. URL http://neuralnetworksanddeeplearning.com/.

M. Opper and C. Archambeau. The Variational Gaussian Approximation Revisited. *Neural Computation*, 21(3):786–792, 2009. doi: 10.1162/neco.2008.08-07-592. URL https://doi.org/10.1162/neco.2008.08-07-592. PMID: 18785854.

L. Rampasek, D. Hidru, P. Smirnov, B. Haibe-Kains, and A. Goldenberg. Dr.VAE: Drug Response Variational Autoencoder. *ArXiv e-prints*, June 2017.

C. Samuelsson. Bayesian Neural Networks with TensorFlow. https://github.com/csamuelsson/bayesianNN, 2018.

J. Shlens. A Tutorial on Principal Component Analysis. *CoRR*, abs/1404.1100, 2014. URL http://arxiv.org/abs/1404.1100.

S. Sonoda and N. Murata. Neural Network with Unbounded Activations is Universal Approximator. *CoRR*, abs/1505.03654, 2015. URL http://arxiv.org/abs/1505.03654.

Chapter

Appendices

Appendix A

This section tables, pseudo code, contains definitions, theorems, theoretical arguments and calculations referred to in the main chapters of this paper.

Key characteristics of data	
Number of variables	6909
Number of observations	606
Minimum value for all explanatory variables	0
Maximal value for all explanatory variables	12
Minimum value for the response variable	-1.15
Maximal value for the response variable	2.15
Average value for the response variable	0.004
Percentage of zero elements	99.2 %
Condition number	$2.078 \cdot 10^{16}$

Table A.1: Key characteristics of the molecular dataset used for our empirical experiments.

Definition A.1 (Mean Squared Error)**.** Let N be the number of observations in a given dataset, y_i the i^{th} true value and $\hat{y}_i$ the i^{th} predicted value. Then the Mean Squared Error (MSE) is defined as

$$\text{MSE} := \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \,.$$

Algorithm 2: K-fold Cross-Validation.

Data: Pre-chosen number of folds K, dataset $\mathcal{X}$ where x_i is the i^{th} element, a learning algorithm $\mathcal{A}$ (outputs a model, i.e. a learned function) with fixed hyperparameters, and a loss function $\mathcal{L}$ that returns an error given a function evaluation and a single observation.

Result: Estimates the generalisation error by averaging test errors for K iterations according to the K-fold Cross-Validation scheme.

initiate empty error array;
initiate empty function array;
for i **from** 1 **to** K **by** 1 **do**
    function$_i$:= $\mathcal{A}(\mathcal{X} \setminus \mathcal{X}_i)$;
    for x_j **in** $\mathcal{X}_i$ **do**
        error$_j$:= $\mathcal{L}(function_i, x_j)$;
    end
end
return sum(error)/K

Proposition A.2 (Eigendecomposition for PCA)**.** The eigenvectors a of X^TX with largest eigenvalue, under the assumption that these vectors also are unit vectors, are the linear combinations aX of our original data matrix X that maximise the variance.

Proof. Firstly, we note that $\text{Var}(aX) = a^TX^TXa$ by using basic variance properties. Therefore, we can equivalently rewrite our optimisation problem to find vectors a such that we maximise the quadratic form a^TX^TXa. Using the Lagrange multiplier method (Giordano et al., 2013, p.588) we get, after differentiating the objective function of the Lagrange method, that

$$X^TXa - \lambda a = 0 \Leftrightarrow X^TX = \lambda a\,,$$

from which we recognise the eigendecomposition of X^TX, where a is the eigenvector (that from our restriction also is a unit vector) and λ its eigenvalue. Since $\text{Var}(aX) = a^TX^TXa = \lambda a^Ta = \lambda$ by our restriction; maximising the variance for our linear combination implies that the eigenvector a with the largest eigenvalue will have the largest variance. Here we are using the fact that X^TX is a semidefinite matrix (since the variance and covariance is non-negative), hence all its eigenvalues are non-negative.

□

Definition A.3 (Posterior distribution)**.** Let $p(\theta)$ be the prior probability density function, $p(x \mid \theta)$ the density for the likelihood of the sampled data x. Then the density for the *posterior distribution* is given by

$$p(\theta \mid x) := \frac{p(x \mid \theta)p(\theta)}{\int_{-\infty}^{\infty} p(x \mid \theta')p(\theta')d\theta'} ,$$

where θ' is a dummy (placeholder) variable. The discrete case is analogous, simply requiring us to replace the integral by a sum.

Definition A.4 (Kullback Leibler distance)**.** The Kullback Leibler distance or divergence between two distributions $P_1(X)$ and $P_2(X)$ (denoted $D_{\mathcal{KL}}(P_1(X) \,||\, P_2(X))$) is defined as

$$D_{\mathcal{KL}}(P_1(X) \,||\, P_2(X)) := \int_{-\infty}^{\infty} p_1(x) \log \frac{p_1(x)}{p_2(x)} dx ,$$

where $p_i(x)$ is the probability density function of $P_i(X)$. The discrete case is analogous; replacing the integral by a sum.

Calculation A.5 (Deriving the optimisation task)**.** We start of using the definition for the Kullback Leibler distance (see Definition A.4)

$$\hat{\theta} = \arg\min_{\theta} D_{\mathcal{KL}} \left[q(w \mid \theta) \,||\, P(w \mid \mathcal{D}) \right] = \arg\min_{\theta} \int_{-\infty}^{\infty} q(w \mid \theta) \log \frac{q(w \mid \theta)}{P(w \mid \mathcal{D})} dw ,$$

thereafter we use the proportional argument following from Bayes' theorem to substitute $P(w \mid \mathcal{D})$ with $P(\mathcal{D} \mid w)P(w)$ (this follows since we are minimising in respect to θ and the denominator $P(\mathcal{D})$ does not depend on θ). Given this substitution, we then use the logarithmic laws to conclude that

$$\hat{\theta} = \arg\min_{\theta} \int_{-\infty}^{\infty} \left(q(w \mid \theta) \log \frac{q(w \mid \theta)}{P(w)} - q(w \mid \theta) \log P(\mathcal{D} \mid w) \right) dw .$$

From here, the trick is to recognise the first term in our integrand as the Kullback Leibler divergence $D_{\mathcal{KL}}(q(w \mid \theta) \,||\, P(w))$ and the second term as the expected value of $\log(P(\mathcal{D} \mid w))$ with respect to $q(w \mid \mathcal{D})$. Therefore, we can write

$$\hat{\theta} = \arg\min_{\theta} \left(D_{\mathcal{KL}} \left[q(w \mid \mathcal{D} \,||\, P(w)) \right] - E_{q(w \mid \mathcal{D})} \left[\log P(\mathcal{D} \mid w) \right] \right) ,$$

whereby we have shown the equivalence.

Calculation A.6 (Rewriting the objective function as an expectation)**.** By the definition of the optimisation task (Equation (3.22)) and the Kullback Leibler distance (Definition A.4) to get that

$$\hat{\theta} = \arg\min_{\theta} \left[\mathcal{D}_{KL}\left(q(w \mid \mathcal{D}) \,||\, P(w \mid \mathcal{D})\right)\right] = \arg\min_{\theta} \left(\int_{-\infty}^{\infty} q(w \mid \mathcal{D}) \log \frac{q(w \mid \mathcal{D})}{P(w \mid \mathcal{D})}\right) .$$

Thereafter, by the definition of an expected value it holds that

$$\hat{\theta} = \arg\min_{\theta} E_{q(w \mid \mathcal{D})} \left[\log q(w \mid \mathcal{D}) - \log P(w) - \log P(\mathcal{D} \mid w)\right] ,$$

where we used that $P(w \mid \mathcal{D}) \propto P(w)P(\mathcal{D} \mid w)$ by Bayes' theorem and substituted accordingly in our minimisation task. Subsequently, it follows that

$$\mathcal{F}(\mathcal{D}, \theta) = E_{q(w \mid \mathcal{D})} \left[\log q(w \mid \mathcal{D}) - \log P(w) - \log P(\mathcal{D} \mid w)\right] ,$$

since $\hat{\theta} = \arg\min_{\theta} \mathcal{F}(\mathcal{D}, \theta)$.

Proposition A.7 (Gaussian reparameterisation trick)**.** Let ϵ be a standard normally distributed random variable, $q(\epsilon)$ its probability density function and let $w = t(\theta, \epsilon)$ where $t(\theta, \epsilon)$ is a deterministic function. Suppose further that the variational posterior $q(w \mid \theta)$ is a diagonal multivariate normal distribution, then

$$\frac{\partial}{\partial \theta} E_{q(w \mid \theta)} \left[f(w, \theta)\right] = E_{q(\epsilon)} \left[\frac{\partial f(w, \theta)}{\partial w} \cdot \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta}\right]$$

Proof. Starting with the definition of an expected value we obtain

$$\begin{aligned} \frac{\partial}{\partial \theta} E_{q(w \mid \theta)} \left[f(w, \theta)\right] &= \frac{\partial}{\partial \theta} \int_{-\infty}^{\infty} q(w \mid \theta) f(w, \theta) dw \\ &= \int_{-\infty}^{\infty} \frac{\partial}{\partial \theta} \left[q(w \mid \theta) f(w, \theta)\right] dw . \end{aligned} \tag{A.1}$$

Since $q(w \mid \mathcal{D})$ is assumed to be diagonally Gaussian, each weight will be marginally Gaussian under the assumption of the variational distribution. Subsequently, we can substitute $q(w \mid \theta)\, d\theta$ with $q(\epsilon)\, d\epsilon$ in Equation (A.1), whereby we get that

$$\begin{aligned} \int_{-\infty}^{\infty} q(\epsilon) \frac{\partial f(w, \theta)}{\partial \theta} d\epsilon &= E_{q(\epsilon)} \left[\frac{\partial f(w(\theta, \epsilon), \theta)}{\partial \theta}\right] \\ &= E_{q(\epsilon)} \left[\frac{\partial f(w, \theta)}{\partial w} \cdot \frac{\partial w}{\partial \theta} + \frac{\partial f(w, \theta)}{\partial \theta}\right] , \end{aligned}$$

and in the very last step we used the chain rule.

□

Appendix B

This section is devoted to the technical implementation of our work. The software configurations we used are displayed in Table B.2. Lastly, all the used programs for our work can be found in Samuelsson (2018), where the dataset is not open for the public due to confidentiality reasons.

Software	**Version**	**Nightly version**
Python	3.5.2	
TensorFlow (Abadi et al., 2015)	1.9.0	*20180424*
TensorFlow Probability	0.0.1	*20180426*
HyperOpt (Bergstra et al., 2015)	0.1	
Numpy	1.14.2	
Scikit-learn	0.19.1	
Abseil Common Libraries	0.2.0	

Table B.2: Software configurations used for this paper. Notice that the TensorFlow Probability module (extending TensorFlow), central for the technical implementation, is still under development and released only as a nightly version at the time of writing.