Machine Learning for Ensemble Modelling in Systems Biology

Raoul Arianne Dias - 10588840

School of Computer Science

University of Manchester

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTERS OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2020

Declaration

I hereby declare that, except where specific reference is mode to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this or any other University. This dissertation is a result of my own work and includes nothing which is the outcome of work done in collaboration, except when specifically indicated in the text.

Raoul Arianne Dias
September 2020

Copyright

Abstract

This project explores the prediction of enzyme kinetic parameter distributions to be used in Ensemble Modelling in Systems Biology. Systems Biology is the computational analysis and modelling of complex biological reactions. Systems Biology uses a system of differential equations to model the behaviour of biological systems. These differential equations need precise kinetic parameters, that are experimentally obtained and often sparsely available. Ensemble modelling in systems biology aims to solve this by sampling from a distribution of possible parameter values and using a number of models and aggregating the results. The aim of this project is to use machine learning and an enzyme database to provide accurate prior distributions to be used for ensemble modelling. This project is implemented using a dataset obtained through querying the BRENDA database. The main focus of the project is on the Mixture Density Network, that is a neural network that produces a Gaussian Mixture Model as its output. The Mixture Density Model produced is able to model a unique prior distribution for every input to the system.

Acknowledgements

I would like to acknowledge several people for the roles that they have played in the successful completion of this dissertation.
First, I would like to thank my supervisor Dr. Rainer Breitling for all his support and guidance in the completion of this project. I would also like to thank Dr. Areti Tsigkinopoulou for her help and support with her expertise on the subject. Lastly, I would like to thank my parents for always supporting me in my endeavours.

Contents

Chapter 1

Introduction

Systems Biology is the computational analysis and modelling of complex biological reactions[1]. The field combines empirical experimental data, data analysis and computational modelling to model the behaviour of whole biological cells[2]. These complex biological reactions, such as mechanistic models of metabolism or cellular signalling pathways, are represented by a system of Differential Equations, which require accurate knowledge of all the kinetic parameters to generate predictive models [3]. The experimental data (kinetic parameters) required are often only partially known, noisy and uncertain[3]. Ensemble modelling [4] [5] has recently been used to attempt to explore the full range of biologically plausible parameter values. This is achieved by sampling parameter values from probability distributions of plausible values, that are derived from the experimental data[3]. This leaves us with an 'ensemble' of predictions that more accurately reflect the confidence limits for each model prediction [3].

A series of problems arise when trying to generate the distributions of parameters due to the scarcity of experimental data, such as a narrow distribution that is biased towards the experimental data or a distribution that is too focused in a particular area. These distributions also often don't account for the surrounding landscape of parameters in the system.

As a solution to these problems, Tsigkinopoulou et al created a protocol for converting experimentally derived kinetic parameter data into informative priors for systems biology in [3].

The focus of this project will be to enrich the ensemble modelling protocol set out by Tsigkinopoulou et al. in [3], by improving the quality of the probability distributions of the Kinetic parameters for a specific enzyme, organism and substrate, using an enzyme database and machine learning.

1.1 Motivation

Systems Biology has given us the opportunity to leverage our advances in computational power and data collection to better understand the biological world. The computational modelling of biological systems has allowed us to view biology through a new lens. In immunology, it has allowed us to understand the complex biochemical networks that regulate the interactions among the immune system's cells and between these cells and infectious organisms [1]. It has been instrumental in the development and testing of drugs and vaccines to fight diseases.
The accurate modelling of kinetic parameters would significantly aid in the accuracy of the ensemble models produced for systems biology. Accurate kinetic parameter modelling from enzyme databases could improve the quality of the informative priors generated by the protocol form [3].
The modelling of continuous data with uncertainty estimates is a relatively unexplored field with potentially large applications in science, technology and business. Extracting not just the most probable value, but a whole distribution of probable values is another motivating factor.

1.2 Aim and Objectives

1.2.1 Aim

The aim of this project is to develop a system to accurately model kinetic parameters for ensemble modelling in systems biology using machine learning and an enzyme database.

1.2.2 Objectives

The objectives of this research are presented in the following:

- Query BRENDA database to obtain dataset for prediction of kinetic parameters.
- Apply data pre-processing tools to prepare dataset for inference algorithms.
- Use feature selection techniques to provide biological insight into factors contributing to kinetic parameter values.
- Build machine learning models to generate prior distributions of kinetic parameters for ensemble modelling.
- Test and evaluate the quality of the priors generated.

- Comparatively review the machine learning approaches used

1.3 Structure

This document is organized into 6 chapters including the introduction. A brief description of the chapters are as follows:

1. **Chapter 2, Systems Biology**: The aim of this chapter is to provide the background of the biological domain that this project is based in. The Systems Biology section begins by introducing the field. This is followed by explanations of the mathematical modelling of metabolic networks and ensemble modelling in systems biology to give the project some context. Next, a brief summary of the protocol set out by Tsigkinopoulou et al. in [3] is presented along with the criticisms that led to the creation of this project. This is followed by some important definitions for the problem.

2. **Chapter 3, Machine Learning:** The machine learning section provides an overview of the concepts required to implement the project. It begins with an explanation of the general machine learning paradigm and the classification of machine learning algorithms based on the type of data and the problem to be solved. It then presents the machine learning models that we will be using in the project, namely, Linear Regression, Gaussian Mixture Models, Artificial Neural Networks and Mixture Density Networks. It then goes on to define the neural network concepts that are used to create the models and finally defines validation protocols and performance metrics.

3. **Chapter 4, Research Methodology and Experimental Design:** This chapter explains the research methodology that was used to implement this project and the design of the system. The software and hardware used for the project are listed here. This is followed by an intricate explanation of the various components of the system and the decisions made in implementing it. Then we define the procedure for model selection and the metrics used to evaluate the models.

4. **Chapter 5, Experimental Results and Discussion**: In this chapter the results of experimentation on the data and the evaluation of the machine learning models are presented.

5. **Chapter 6, Conclusion**: This chapter summaries the key achievements of the project and discusses future work.

Chapter 2

Systems Biology

This chapter will provide a brief background into the biological domain that this project is based in. It will explain the fundamental concepts that need to be understood to progress with the project.

Systems Biology is the computational and mathematical analysis and modelling of complex biological systems [1]. Systems Biology can be described as a combination of biological and computational modelling sciences, that together aim to explain the inner working of complex biological systems. It uses systems of differential equations to describe the dynamics of molecules in living organisms and predict their response to perturbations, such as drug treatments or mutations.

2.1 Mathematical Models for Metabolic Networks

Metabolic Networks are described as a system of differential equations in Systems Biology. Deterministic models of metabolic networks consist of systems of differential equations describing the kinetics of the metabolic process [2]. The mathematical expressions of kinetic laws require kinetic parameters. For example, in the case of a reversible enzyme catalysed reaction with one substrate and one product and assuming Michaelis-Menten kinetics, the net rate in the direction $A \longrightarrow B$ is:

$$\nu = \frac{\nu_{+}^{max}(A/K_A) - \nu_{-}^{max}(B/K_B)}{1 + A/K_A + B/K_B} \tag{2.1}$$

where K_A and K_B are the Michaelis-Manten constants for substrate A and B respectively, and ν_{+}^{max} and ν_{-}^{max} are the maximum velocities in either direction. Reactions with more substrates or products have more kinetic parameters.

2.2 Ensemble Modelling in Systems Biology

Ensemble Modelling(EM) is a computational modelling paradigm where multiple diverse models are created to model a system, either using different modelling algorithms or different training data. A voting or aggregation procedure is then used to generate a final prediction.

Ensemble Modelling in systems biology as described by Tran et al. in [6] is the process of generating a set of kinetic models where each model is described by different kinetic parameters but all models use the same mathematical structure and are anchored to the same steady state reference state. If each reaction in the model is represented as a non-linear ordinary differential equation:

$$\frac{dx_i}{dt} = \sum v_i(x,k) - \sum V_j(x,k) \tag{2.2}$$

where x_i represents the concentration of species/metabolites in the model, and v_i and v_j are the enzyme kinetic functions for the production and consumption of species x_i respectively, then the EM problem can be stated as sampling and obtaining all the different sets of kinetic parameters such that

$$v(x_{ss},k) = v_{\text{net}}^{\text{ref}} \tag{2.3}$$

where $v_{\text{net}}^{\text{ref}}$ is called the steady state flux, x_{ss} is the steady state concentration of the metabolites in the model and k is the kinetic rate constant [6].

2.3 Overview of the Protocol

The protocol proposed by Tsigkinopoulou et al. in [3] aims to translate biological knowledge to an informative probability distribution for all kinetic parameters. It assumes that a functioning kinetic model of the biological system is already in place.

The protocol consists of five stages. The first stage aims to find kinetic parameter values by collecting data from literature, databases(such as BRENDA) or experiments. After this, a plausibility weight is assigned to each of the values based on its quality and relevance, defined by a weighting table as shown in 2.1. Next, the Mode (most plausible value) and Spread (multiplicative standard deviation) are calculated using the log–transformed and normalized parameter values. The next stage involves the calculation of the location (μ) and scale (σ) parameters of the log-normal distribution based on the

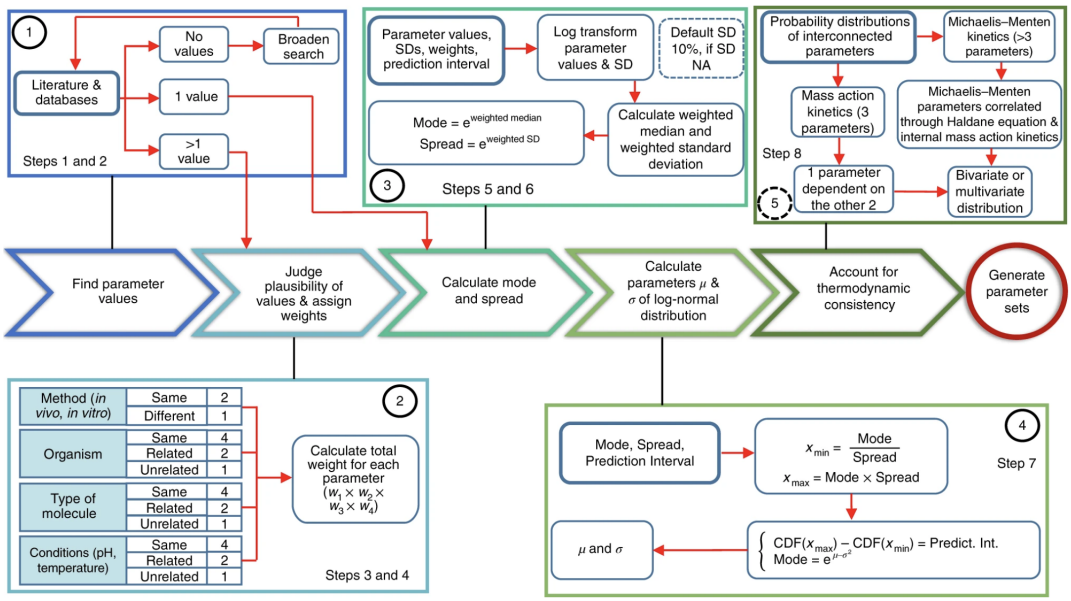

Figure 2.1: A graphical overview of the protocol from [3]

previously defined Mode and Spread. The last stage involves ensuring the thermodynamic consistency of the model by creating a multivariate distribution for independent parameters.

The current issues with the protocol stem from the arbitrary weighting of plausibility values, the general assumption of distribution shape and data that is both scarce in some regions and over-focused in others. The protocol doesn't exploit the possible specific relations between the hierarchies of, Organisms and enzymes, and kinetic parameter values. The prior distributions generated might be too generalised for parameters that haven't been specifically recorded before and too narrow for parameters that have been. This prompts the need for a system that can model the effects of the enzymes, organisms and substrates to produce more informative prior parameter distributions.

2.4 BRENDA

BRENDA (BRaunschweig ENzyme DAtabase) is a comprehensive protein function database, containing enzymatic and metabolic information extracted from the primary literature[7]. The database currently consists of 4.3 million data for 84,000 enzymes[8]. Amongst other enzyme information, the BRENDA database contains measured K_M values for specific enzymes (based on EC number) and organisms, that have been obtained experimentally[7]. BRENDA supports several querying methods including Test-based, Structure-based and GUI-based querying[8]. BRENDA will be used as the primary data

source for the project. All data sets used will be extracted from BRENDA

2.5 EC numbers

The EC number (Enzyme Commission number) is a classification scheme for enzymes, based on the chemical reactions they catalyze [9]. It is based on a system of enzyme nomenclature produced and maintained by the International Union of Biochemistry and Molecular Biology. Every enzyme code consists of the letters ”EC” followed by four numbers separated by periods. The numbers represent a top-down a hierarchical classification of the enzymes [10].

2.6 Classification of Organisms

Living organisms are classified using a hierarchical classification based on Taxonomic Rank. The rank used is in accordance with the nomenclature codes [11]. These ranks, top-down, are Domain, Super Kingdom, Kingdom, Phylum, Class, Order, Family, Genus and Species. Each rank is represented by a name, in the form of a string.

Chapter 3

Machine Learning

This chapter will provide a brief background into the machine learning concepts that were required to implement the project.
Machine Learning is a multi-disciplinary intersection of computer science, statistics and artificial intelligence [12]. The field allows systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine Learning algorithms aim to build mathematical models based on 'training data', and then use the information learnt to make predictions or decisions on previously unseen data, without being explicitly programmed to do so [13].
Machine Learning algorithms are being increasingly used in a wide array of domains to solve problems that have been thus far limited to human beings. Today machine learning is used in Text Mining, Computer Vision, Targeted Advertising, Self-Driving Cars, Automated Trading and a whole ethos of diverse domains [12].

3.1 General Machine Learning Paradigm

The general machine learning paradigm consist of three main components; *data*, a *learning algorithm* and a *model*. A learning algorithm is a procedure that is run on the *data* to create a *model*. The *model* is a data structure such as a tree, graph, algebraic equation or probability distribution, that stores the information gained by using the *learning algorithm* on the *data*.
The general formulation of a linear model is shown in the equation below

$$W^T X - b = \sum_{j=1}^{d} w_j x_j - b \tag{3.1}$$

where W and X are $d-$dimensional vectors representing the wights and inputs vectors

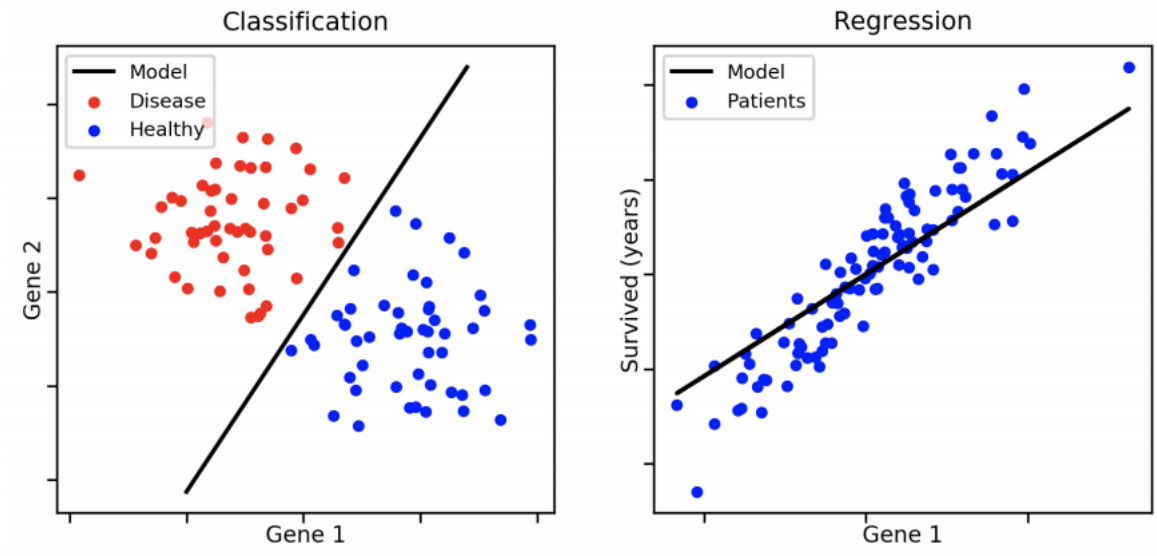

Figure 3.1: The two types of supervised learning tasks

respectively, w_j and x_j represent the jth elements of their respective vectors, and b represents the bias. This equation is a general case of a linear classification problem, in which the learning algorithm is used to find the weight and bias combination that best divides the data being classified.

Machine Learning is divided into three groups based on the type of *data* provided namely *supervised*, *unsupervised* and *reinforcement learning* [14].

3.2 Supervised Learning

Supervised Learning is a machine learning paradigm where the training data is labelled [14]. Each training example consists of some *input data* and a *label*. A supervised learning algorithm attempts to *learn* a function that maps the *input data* to the *label*. The model can *learn* this functional mapping by comparing the current predictions of the model with the *true labels*.

Supervised Learning can be further subdivided into two general tasks depending on the nature of the *labels*. If the labels consist of continuous variables, such as integers, the task is called regression, and if the labels consist of categorical variables, the task is called classification. Figure 3.1 highlights the difference between classification and regression. Linear regression is an example of a regression model, whereas logistic regression is an example of a classification model.

3.3 Unsupervised Learning

Unsupervised learning is a machine learning paradigm where the training data is unlabelled [14]. The aim of this paradigm is to find emergent patterns or irregularities within the data. *Clustering* is an example of an unsupervised learning task, where the data is split into several *clusters* of similar data. The instances in the same cluster are expected to be similar to each other and different from instances in other clusters. The K-means algorithm is an example of a clustering algorithm [15].

3.4 Reinforcement Learning

Reinforcement learning is a machine learning paradigm concerned with how software agents ought to take actions in an environment to maximize the notion of cumulative award [16]. The paradigm consists of evaluating the existing policies and rewards, and learn from the past series of actions to generate an optimal policy. Game playing and decision making are examples of reinforcement learning tasks. Q-learning is an example of a reinforcement learning algorithm for game playing [17].

3.5 Loss function

A loss function or cost function is an objective function that measures the performance of the machine learning algorithm on the data [12]. A machine learning algorithm *learns* by trying to minimize the loss function. The summation of the loss function over the whole set is called the error function and it represents the overall error of the model. Loss functions can also be broadly divided into two categories depending on the machine learning task that is being dealt with - Regression losses and Classification losses.

3.5.1 L2 loss

First, we will start with an example of a Regression loss function called L2 loss [18]. It's associated error function is called *Mean Square Error* (MSE). The are defined as:

$$L = (f(x) - y)^2 \tag{3.2}$$

$$C = \frac{1}{2m} \sum_{i=1}^{m} (f(x_i) - y_i)^2 \tag{3.3}$$

Equation 3.2 represents the loss function where, $f(x)$ and y are the predicted label and true label respectively. Equation 3.3 represents the error function where m is the total number of samples in the dataset and $f(x_i)$ and y_i are the predicted label and true label of the i^{th} sample from the dataset.

3.5.2 Cross-entropy loss

Next, we will discuss an example of Classification loss called cross-entropy loss [19]. It is also called log loss. It is defined as:

$$L = -\sum_{c=1}^{M} y_c log(f(x)_c) \tag{3.4}$$

where M is the number of classes, $f(x)_c$ is the probability of x belonging to class c and y_c is the true label. The cross-entropy loss measures the similarity between the predicted and real distributions of the dataset. The error function for cross-entropy loss is called the cross-entropy error or the negative log-likelihood error.

3.6 Linear Regression

Linear Regression is a machine learning technique that models the relationship between a scalar dependant variable and one or more independent variables [12]. The relationship is modelled using linear predictor functions whose unknown model parameters are estimated from the data. This function is called a linear model, shown in the equation below, where X is the matrix of independent variables, β is a matrix of learned coefficients and ε is the bias[20]. Linear Regression minimizes the sum of squared errors to fit the data(least squares)

$$Y = X\beta + \varepsilon \tag{3.5}$$

Linear Regression has many practical applications. Most of them could be classified as one of the following:

1. Prediction problems, where linear regression can be used to fit a predictive model to a dataset of explanatory and target variables. After which the model can predict the target variable when it is shown unlabeled explanatory variables.

2. Problems that are trying to associate variation in the targets with variation in the explanatory variables. Linear Regression can also be used to quantify the strength of the relationship between a target variable and the explanatory variable, to check

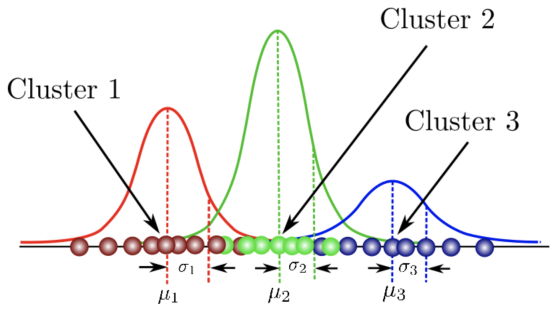

Figure 3.2: Graphical representation of GMM from [21]

if there is a linear relationship between target and explanatory variables or to identify a subset of the explanatory variable that may contain redundant information about the target [12].

3.7 Gaussian Mixture Model

Gaussian Mixture Models are unsupervised clustering machine learning models. A Gaussian Mixture is a function that is comprised of several Gaussians, each identified by $k \in 1,...,K$, where K is the number of clusters in the dataset. Each Gaussian k has three parameters; a mean μ that defines its center, a covariance σ that defines its width and a mixing coefficient π that defines the size of the Gaussian [21]. Figure 3.2 graphically represents a Mixture of Gaussians with $k = 3$. The mixing coefficients π can be treated as probabilities and sum up to 1.

In general, the Gaussain density function can be expressed as:

$$\mathcal{N}(X|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} exp\big(-1/2(X-\mu)^T\Sigma^{-1}(X-\mu)\big) \tag{3.6}$$

where X is the data, D is the dimensions of each point, μ and σ are the mean and covaraince, respectively.

The Expectation-Maximization [21] algorithm is used to learn the appropriate parameters of the Gaussian Mixture Model. The number of Gaussians is a hyper-parameter for the model. A Mixture of Gaussians can model any distribution with enough Gaussians.

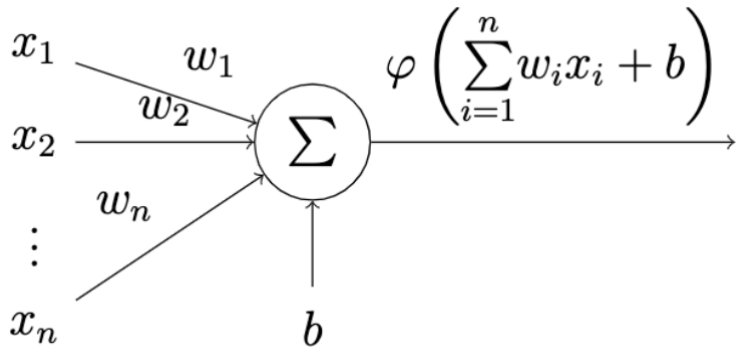

Figure 3.3: Representation of a single neuron in a Neural Network from[22]

3.8 Artificial Neural Networks

Artificial Neural Networks are machine learning algorithms that are inspired by the neural structure of the biological human brain [12]. An artificial neuron is a mathematical function which constitutes the basic computing unit for artificial neural networks. Figure 3.3 represents a single neuron, which receives n weighted inputs $(w_i, ..., w_n)$ and a bias b, and sums it up and applies a non-linear function known as an activation function φ to produce an output. Therefore, Each node gets information from its input variable, forms a linear combination with the weights and then is passed through an activation function to produce an output variable [12].

Figure 3.4 represents a Multi-Layer Neural Network. Each *layer* consists of a column of neurons from figure 3.3. The first layer of the neural network is called the input layer, and the last layer is called the output layer. All intermediate layers are called hidden layers.

The mathematical formulation of a Multi-layer neural network with a single hidden layer with an element-wise non-linear activation function φ_i to the input vector x and the hidden output can be represented as:

$$y = \varphi_2(W_2\varphi_1(W_1 x + b_1) + b_2) \tag{3.7}$$

where W_i, b_i are the weights and bias vectors of the i^{th} layer.

Neural Networks have been a theoretical construct in Artificial Intelligence for decades, but have only recently been practically used due to the increase in computational power and superior techniques. Today, neural networks are used in a wide variety of domains that we can see in products and services that we use daily. The applications of neural

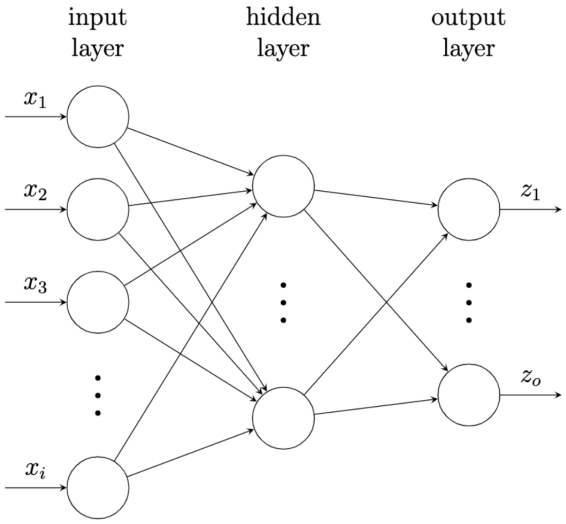

Figure 3.4: Graphical representation of a multi-layer neural network from [22]

networks stretch from chat-bots [23], image manipulation and enhancement [24] to generating art [25] and music [26].
Deep learning is the use of large neural networks with multiple hidden layers. A hidden layer is a layer that is located between the input and output of the network, that is some non-linear transformation of the previous layer and is learned through training the model [12].

3.9 Mixture Density Networks

Mixture Density Networks are a formulation of neural networks, that were first proposed by Bishop in 1994 [27]. In the paper, Bishop begins by re-framing the conventional least squares loss (Equation 3.2) as the conditional average of the target data, conditioned on the input vector as illustrated in Figure 3.5, where the solid curve represents the optimal least squares function, which is given by the conditional average of the target data, such that for a given value of x, such as x_0, the least-squares function $(t|x)$ is given by the average of t with respect to the probability density $p(t|x)$ at the given value of x.
In the case of neural network outputs, this implies that we are approximating the conditional average of the target data. As Bishop explains [27], most applications of neural networks only make use of the prediction of the mean, which represents the optimal solution for classification problems, but only represents a very limited statistic when predicting continuous variables. Bishop realised that there were many applications that would benefit from the complete description of the probability distribution of the target

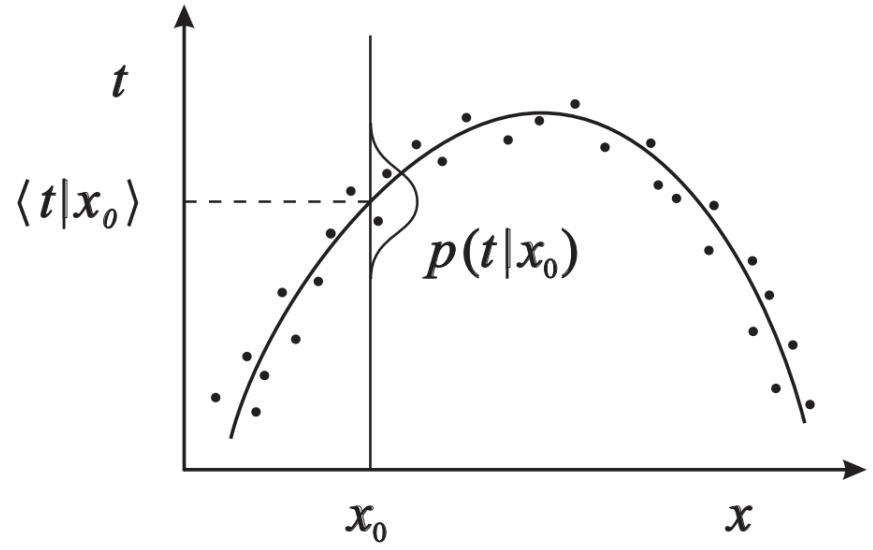

Figure 3.5: Illustration of Bishop's least squares formalism from [27]

data, which motivated him to create this new formalism [27].

Bishop stated that if we assume the conditional distribution of the target data is Gaussian, then we can obtain the least-squares formalism using maximum likelihood. This motivates replacing the Gaussian distribution in the conditional density of the complete target vector with a mixture model, which has the flexibility to completely model general distribution functions. The probability density of the target data is then represented as a linear combination of kernel function of the form

$$p(y|x) = \sum_{i=1}^{m} \alpha_i(x)\phi_i(y|x) \tag{3.8}$$

where ϕ is a kernel function, m is the number of components in the mixture and $\alpha_i(x)$ are the mixing coefficients. Bishop chose Gaussian kernel functions of the form:

$$\phi_i(y|x) = \frac{1}{(2\pi)^{c/2}\sigma_i(x)^c} exp\left\{\frac{||y-\mu_i(x)||^2}{2\sigma_i(x)^2}\right\} \tag{3.9}$$

where μ_i represents the center of the i^{th} kernel. Bishop assumed that the components of the output vector are statistically independent within each component of the distribution, and it can be described by a common variance $\sigma_i(x)$. According to [28] and [29], a Gaussian Mixture Model with this simplified kernel can approximate any given density function to arbitrary accuracy, provided the mixing coefficients and the Gaussian parameters are chosen correctly.

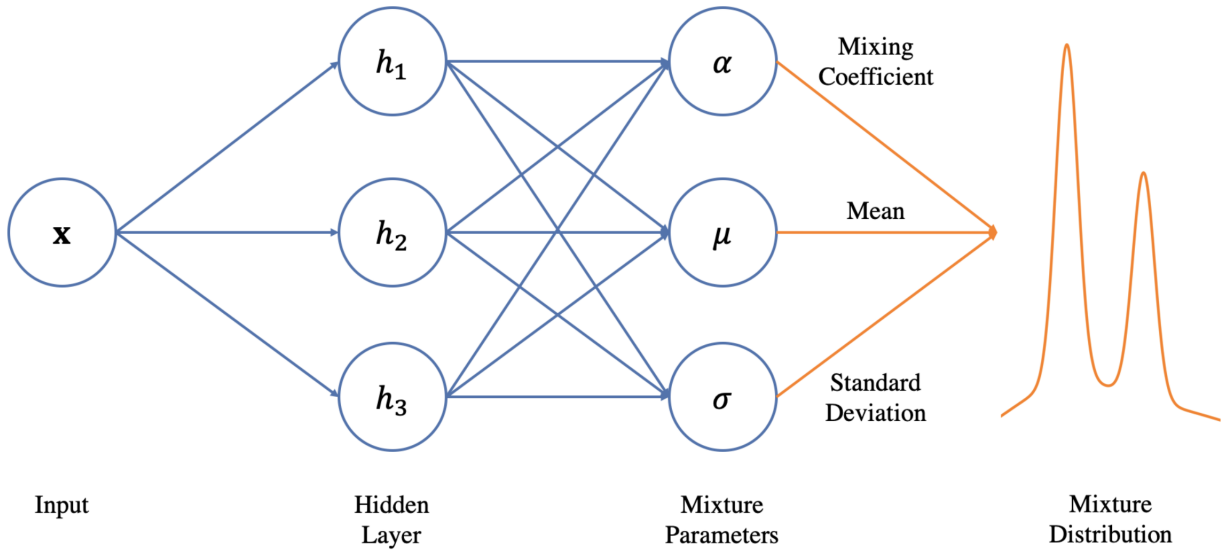

Figure 3.6: Representation of a Mixture Density Network from [30]

Figure 3.6 represents a Mixture Density Network, where the output of the feed-forward neural network determines the parameters of a mixture model. The Mixture Density Network provides a general formalism for modelling an arbitrary conditional density function $p(y|x)$. This combination of the traditional neural network and the mixture model is achieved by using the log-likelihood of the linear combination of kernel functions as the loss function of the neural network. The error function can be defines as the maximisation of the log-likelihood function or, equivalently, minimisation of the negative log-likelihood. Therefore the error function can be defined as :

$$logL(y|x) = -log(p(y|x) = -log\left(\sum_{i=1}^{m} \alpha_i(x)\phi_i(y|x)\right) \tag{3.10}$$

Where $p(y|x)$ is substituted from 3.8. Therefore, the Mixture Density Network models the complete conditional probability density of the output variables.

Mixture Density Networks have been used for numeral applications that require a measure of uncertainty. Apple's Siri in iOS 10 uses MDNs for speech generation [31]. In [32] MDNs were used in combination with Recurrent Neural Networks to generate artificial handwriting.

3.10 Activation Functions

An activation function in a neural network is a function that defines the output of a node given an input or set of inputs [33]. An activation function allows the output to be framed in the desired format for use later in the networks. Some activation functions such as the Sigmoid function allow for the output to be perceived as probabilities. A neural network with two or more layers and activation functions can approximate any linear or non-linear function [33].

3.10.1 Sigmoid Function

The Sigmoid function, also called the Logistic curve, is expressed as

$$s(x) = \frac{1}{1+e^{-x}} \tag{3.11}$$

where x is the input with the domain of all real numbers and the function produces an output ranging between 0 and 1. It has a characteristic S-shaped curve. The sigmoid function is monotonic, bounded and differentiable [12], which enables it to act as an appropriate activation function.

The sigmoid function has a drawback called the vanishing gradients problem, when the absolute value of the input increases, the gradient of the sigmoid function reduces to a small value. Therefore, for large values, the gradients get very small.

3.10.2 Rectified Linear Unit

Rectified Linear Unit(ReLU) is a common activation function. It is expressed as

$$f(x) = max(0,x) \tag{3.12}$$

where x is a real number. If the input of a ReLU function is less than zero, the output will be zero or the output will be the input. The gradient of ReLU with respect to its input is a constant value, which allows it to avoid the vanishing gradients problem. Figure 3.7 showcases the curve produced by ReLU.

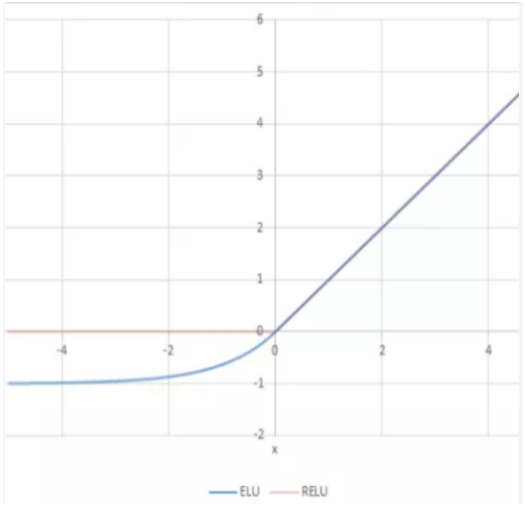

Figure 3.7: ReLU and ELU activation functions.

3.10.3 Exponential Linear Unit

Exponential Linear Unit(ELU) is an activation function expressed as

$$f(x) = \begin{cases} \alpha(exp(x)-1) & \text{for} \quad x \leq 0 \\ x & \text{for} \quad x > 0 \end{cases} \tag{3.13}$$

where α is a tuneable hyper-parameter. ELU is an activation function based on ReLu that also has the benefit of avoiding the vanishing gradients problem. ELU tends to converge cost to zero quicker and produce more accurate results when compared to ReLU. Unlike ReLU, ELU can produce negative outputs. Figure 3.7 showcases the curves of ELU and ReLU.

3.10.4 Softmax function

The Softmax function is used to normalize an arbitrary real-valued K-dimensional vector into a K-dimensional vector of values ranging from 0 to 1, and the sum of all elements is 1 [12]. The output of a softmax function is often used as a categorical distribution representing the probability of occurrence of each element in the vector. The softmax function can be expressed as:

$$\text{Softmax}(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \text{ for} j = 1, ..., K \tag{3.14}$$

where x is a K dimensional vectors representing the input, x_j is the jth element of the input vector and $\text{Softmax}(x)_j$ is the jth element of the output vector. For a classification

task, the vector represents class membership, with the value being the probability.

3.11 Optimization Algorithm

Optimization in the context of neural networks refers to the method of updating parameters of the neural network. Optimization involves updating the weights of a model using the loss function. It is the method by which neural networks **learn**. These optimization algorithms act as the learning algorithm for the neural network.

3.11.1 Gradient Descent

Gradient Descent(GD) is an optimization method that updates the weights towards the direction of the negative gradient of the cost function, which causes the error in the model to decrease. Specifically, GD starts by calculating the derivatives(gradient) for each of the parameters w.r.t to the cost function, then uses the gradients to calculate the adjustments that need to be made to the weights to minimize the cost function.
It is expressed as:

$$\theta = \theta - \eta.\nabla_{\theta} J(\theta) \tag{3.15}$$

where $J(\theta)$ is the objective function to be minimised, θ is the parameters and η is the learning rate.
Batch Gradient Descent involves computing the cost function w.r.t the model parameters θ for the entire dataset. As all the gradients need to be calculated for each update, batch gradient descent can be very slow.
Stochastic Gradient Descent(SGD) performs a parameter update for each training example [34]. Batch GD performs redundant computations on large databases, which is mitigated by using SGD.
A common problem with Gradient Descent methods is that they tend to get stuck in local minima [35]. Momentum [36] is a method that helps accelerate SGD in the relevant direction, by adding a fraction of the update vector of the previous time-step to the current time-step [34].

3.11.2 Learning Rate

In a neural network, the learning rate(η) is a hyper-parameter that defines how much the parameters of the model change with each iteration. A small learning rate can take very many iterations to converge, while a large learning rate can overshoot the minimum.

3.11.3 AdaGrad

AdaGrad [36] is an algorithm for gradient-based optimization, that adapts the learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features and larger updates for parameters associated with infrequent features [34]. It is therefore good at dealing with sparse data. AdaGrad uses a different learning rate for every parameter θ_i at every time step t. AdaGrad's per-parameter learning rate is

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \tag{3.16}$$

where $g_{t,i}$ is the partial derivative of the objective function w.r.t the parameter θ_i at time step t. In its update rule, AdaGrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i [34]:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} \cdot g_{t,i} \tag{3.17}$$

where G_t is a diagonal matrix where each diagonal element i,i is the sum of squares of the gradients w.r.t θ_i up to the time step t [34] and ε is a smoothing term that avoids division by zero.

AdaGrad eliminates the need to manually tune the learning rate [34]. AdaGrad's main weakness is the accumulation of squared gradients in the denominator, this causes the learning rate to shrink to an infinitesimally small value, which causes the algorithm to stop learning [34].

3.11.4 Adam

Adaptive Moment Estimation(Adam) [37] is an optimization method that computes adaptive learning rates for each parameter. It deals with the diminishing learning rate problem from AdaGrad by introducing an exponentially decaying average of past squared gradients v_t, it also keeps an exponetially decaying average of past gradients m_t, simlar to momentum [34]. The decaying averages of past and past squared gradients m_t and v_t respectively are as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{3.18}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3.19}$$

where β_1 and β_2 are the decay rates and g_t is the past gradients. They are then bias corrected, due to their initial configuration as vector's of 0s.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3.20}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3.21}$$

The update rule for the parameters is then expressed as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\varepsilon}\hat{m}_t \tag{3.22}$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2 and $1e-8$ for ε [37]. They show empirically that Adam outperforms other adaptive learning-method algorithms [37].

3.12 Regularization

Regularization is a technique used to reduce the generalisation error of a machine learning model [28]. Overfitting [28] is an error that occurs when an ML algorithm models the data that it is trained on too well. Overfitting occurs when the model learns the details and noise in the training data to the extent that it negatively impacts the performance of the model on new data. Underfitting [28] occurs when a model can neither model the training data nor generalise to new data. Both overfitting and underfitting result in poor generalisation of the model.

3.12.1 L_2 Regularization

L_2 regularization, also known as weight decay, is a method to improve the generalization of a model. This is achieved by adding a regularization term to the cost function [12]. The total cost function with an L_2 regularization term and update rule can be expressed as:

$$C_{total} = C + \frac{\lambda}{2m}||W||_2^2 \tag{3.23}$$

$$w_i^{t+1} = w_i^t - \alpha\left(\frac{\partial C_{total}}{\partial w_i^t}\right) = w_i^t - \alpha\left(\frac{\partial C}{\partial w_i^t} + \frac{\lambda}{2m} + \frac{\partial||W||_2^2}{\partial w_i^t}\right) \tag{3.24}$$

Equation 3.23 represents the new cost function C_{total} with the L_2 regularization term added to it, where λ is a hyper-parameter that controls the effect of the regularization

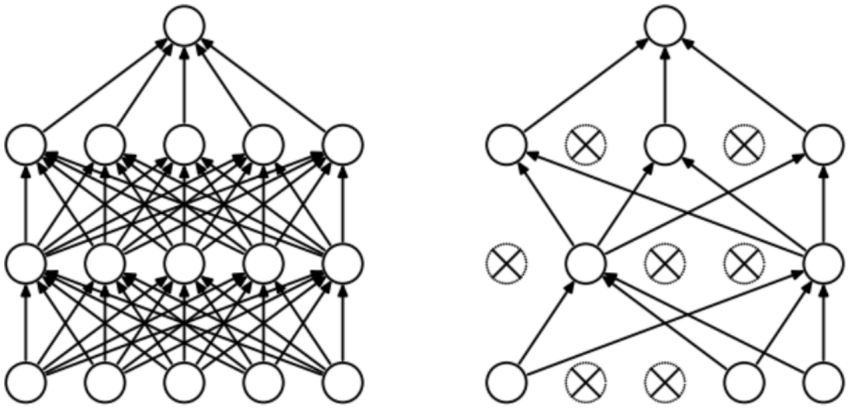

Figure 3.8: Representation of Dropout form [38]

term, W is the weight vector with m weights and $||W||_2^2$ is the L_2 norm of W. Equation 3.24 represents the new update rule, as a result of the new cost function, where w_i^t represents the the i^{th} weight at time step t and α is the learning rate. The value of λ is used to control the effect of the regularisation term and lies between 0 and 1, this decides the value of the penalty imposed on the cost function, a large value will lead to underfitting.

3.12.2 Dropout

Dropout is a regularization method that approximates training a large number of neural networks in parallel. The term 'dropout' refers to dropping out units in a neural network [38]. Dropping out refers to the process of temporarily removing a unit form the network, along with all its incoming and outgoing connections. Dropout has the effect of making the training process noisy, forcing nodes within a layer to take on more or less responsibility for the input. This has the effect of training many sub-networks with different structures [38]. This causes the network to be more robust to avoid generalization error. Figure 3.8 represents dropout.

3.13 Normalization

Normalization is a technique that is used in the data preparation step of machine learning. It involves centering and standardising the values of an input variable. Standardization refers to the process of rescaling the data to have a mean of zero and a standard deviation of one. It helps to bring the dataset to a common scale, without distorting differences in the ranges of values. Normalization helps mitigate the effects of variables

that have larger values, that can bias the performance of the model.

3.13.1 Batch Normalization

Batch Normalization involves bringing the idea of normalization to the internal layers of the neural network. It is a technique for training deep neural networks that standardizes the inputs to a layer for each mini-batch. It achieves this by standardizing the activations of each input variable per mini-batch. Standardizing the activation of the prior layer means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update rule will not change dramatically. This causes the network to stabilize and speeds up the training process of the network.

3.14 Feature Selection

Feature Selection is a dimensionality reduction technique that selects the features that contribute most significantly to the prediction to be generated. As the dimensionality of data increases, the number of samples required to accurately model the data increases exponentially, this is called the curse of dimensionality. Feature selection can remove redundant data and intern prevent the model from overfitting, it can improve the modelling accuracy by only using the most relevant features and can reduce training time due to the reduced dimensionality.

For supervised learning problems, feature selection can be split into two categories, namely, Filter and Wrapper methods. Filer feature selection methods use statistical tests to score the correlation or dependence between input variables and target variables. Wrapper feature selection methods evaluate multiple models using procedures that add or remove input variables to find the optimal combination that maximizes model performance [39]

3.14.1 Analysis of Variance

Analysis of variance(ANOVA) [40] is a parametric statistical hypothesis test for determining the differences among group means in a sample. ANOVA first finds the correlation between each input variable and the target variable. It then uses the F-statistic to compute an F score and a p-value. ANOVA is a filter-based method

3.14.2 F-statistic

F-statistic or F-test is a class of statistical tests to calculate the ratio between variances, such as the variance between two samples or the explained or unexplained variance by a statistical test. F-test has an F-distribution under the null hypothesis. It is used to compare statistical models that have been fitted on a dataset, to identify the model that best fits the population from which the data was sampled [41].

3.14.3 Mutual Information

Mutual Information(MI) is a quantity that measures the information one random variable tells us about another. MI between two random variables is a non-negative value that measures the dependency between the variables. It is equal to zero if the two random variables are independent, a higher value of MI means higher dependency. It can be formally defined as:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) log \frac{P(x,y)}{P(x)P(y)} \tag{3.25}$$

where X and Y are two random variables and P(X) and P(Y) are the marginal distributions of X and Y. MI is a filter-based method.

3.14.4 Recursive Feature Elimination

Recursive Feature Elimination(RFE) is a wrapper based feature selection method. Recursive Feature Elimination works to select features by recursively considering smaller and smaller sets of features. First, the model is trained on the initial set of features and the impotence of each feature is obtained through a coefficient or feature importance attribute. The least important features are pruned from the current set of features. The procedure is recursively repeated until the desired number of features is obtained.

3.15 Categorical Input Handling for Neural Networks

Neural Networks are an extremely flexible tool for modelling data, but can only accept numeric data as inputs. Therefore, if the dataset to be used contains categorical data, it has to be transformed into a form that can be digested by the neural network [42].

3.15.1 Ordinal Encoding

Ordinal Encoding involves assigning each possible category an integer value. The downside to this approach is that the quantitative values of the integers do not correspond to information about the categories. This means that all the high-level information about the categories is misinterpreted and skewed unless the labels possess some ordinal relationship [42].

3.15.2 One-Hot Encoding

One-Hot Encoding(OHE) involves splitting a single categorical variable into k binary variables, where k is the number of categories in the categorical variable. If the original categorical variable 'colour' had three categories 'Red', 'Blue' and 'Green', an OHE would convert the categorical variable into three binary variables called 'Red', 'Blue' and 'Green'. If a sample initially had the 'colour' equal to 'Red', it would now consist of the 'Red' variable having the value 1 and the 'Blue' and 'Green' variables having the value 0.

One-Hot Encoding avoids the need for an ordinal relationship between the categories, although it assumes independence among the categories. It is not suitable for a variable with a large number of categories as the dimensionality of the data would balloon, adversely affecting the performance of the neural network [42].

3.15.3 Embeddings

Word Embeddings [43], the idea of using a neural networks to convert words or phrases to real-valued low-dimensional representations, are a staple in the Natural Language Processing(NLP) paradigm. The learned vector embeddings provide intrinsic information about the data by mapping similar entities together in the vector space. The embeddings essentially defined a distance measure for the entities. Word2Vec [44] is a common tool for creating word embeddings.

In [45], Guo et al. proposed using entity embeddings for categorical variables as inputs to neural networks. The paper [45] showed that the method was not only more memory efficient but also sped up neural networks when compared to One-Hot Encoding. The embeddings were also able to model complex relationships between entities. The authors [45] also went on to demonstrate that the embeddings helped to generalize better when the data is sparse.

3.16 Cross-validation

In Machine Learning, cross-validation(CV) is a technique used to test the effectiveness of a machine learning model on data that is previously unseen [46]. cross-validation is implemented through setting aside a portion of the data, which is not used to train the model, for testing or validating the model.

3.16.1 Train Test Split

Train test split is the most simple form of cross-validation. It is implemented by randomly splitting the dataset into two sets: a training and a testing set. The model is first trained on the training set and then validated of the testing set. The validation on the testing set provides an estimate of the generalization error of the model. The train test split has the possibility of bias towards the training set, especially in the event of limited data.

3.16.2 K-folds cross-validation

K-Folds cross-validation is a popular form of CV that generally results in a less biased estimate of generalisation error. This is due to the fact that in this method every data point appears in both the training and the testing set at some point. It is implemented using the following process. First, the dataset is randomly split into K folds. Next, the model is trained on $K-1$ folds and validated on the remaining fold. Then the process is repeated until every fold is used as the validation fold. The error from testing every fold is then averaged out to produce an estimate of the generalisation error. The value of K should be chosen such that each fold should be large enough to be statistically representative of the broader dataset. A value of $K = 10$ is generally used as a starting point for this.

3.16.3 Leave One Out cross-validation

Leave One Out cross-validation(LOOCV) is another commonly used CV method. This CV method is implemented by training the model on $n-1$ data points, where n is the total number of data points, and test the model on the remaining data point. This process is repeated n times, until every single data point has been used as the testing point. The error is then averaged across all points and an estimate of generalisation error is produced. LOOCV is generally used when no splits of the dataset could accurately statistically represent the dataset.

3.17 Performance Metrics

Performance metrics are the methods by which the performance of a model on a dataset is measured. For Classification problems a base metric often used is accuracy i.e., the number of samples correctly predicted. Other classification metrics such as Precision, Recall and F-1 score are based on it. Regression has several different performance metrics.

3.17.1 Mean Squared Error

Mean Squared Error(MSE) is one of the most popular regression metrics. It is the average of the squared difference between the target value and the value predicted by the regression model, as shown:

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2 \tag{3.26}$$

where y is the target value and $\hat{y}$ is the predicted value. As MSE squares differences, it tends to penalize even small errors which could lead to over-estimation of error. Root Mean Squared Error is the square root of the mean squared error, which is used as now larger errors are penalized.

3.17.2 Mean Absolute Error

Mean Absolute Error(MAE) is the absolute difference between the target value and predicted value of the model, as shown:

$$MAE = \frac{1}{n}\sum|y - \hat{y}| \tag{3.27}$$

MAE is more robust to outliers and does not penalize errors as extremely as MSE.

3.18 Current Approaches for Predicting Kinetic Parameters using Machine Learning

In [2], Broger et al. used a linear regression model to predict K_M value using Enzyme Number and Organism. In the experiment, all organisms and EC values were extracted for a single metabolite from the BRENDA database. A data matrix of K_M values for various Enzymes numbers and organisms was created. There were several missing K_M values for certain Enzyme-Organisms pairs. The proposed regression model used the

existing data to predict the missing K_M values. The model proposed was expressed as:

$$x_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij} \tag{3.28}$$

where μ is the general mean, α_i denotes the effect of enzyme i, β_j is the effect of organism j and ε_{ij} are independent identically distributed Gaussian random numbers.
The paper used Mean Square Error(MSE) to calculate the prediction error and used Leave One Out cross-validation(LOOCV) to validate the data. For the full dataset, they achieved a prediction error of 1.01. The paper concluded that the Michaelis-Menten constant K_M values may be conserved between related species and have also been shaped by functional requirements. The paper also suggested that these findings could be applied to other kinetic parameters such as maximal velocities or catalytic constants.

Chapter 4

Research Methodology and Experimental Design

This chapter will discuss the research methodology and experimental design of the systems used in the project. First, we will go through the research methodology that has been used to facilitate the project. Next, we will go through the software environment used to create the project. Then, we will look at the design and implementation of the machine learning models and finally we will look at the methods for evaluating them.

4.1 Research Methodology

The methodology of research in this project was designed to be that of a machine learning project, it was staggered into a few steps as described by Dietterich in [47]. The first stage involves exploratory research, which in the case of this project consisted of researching the domain, understanding and breaking down the problem. The next step involved proposing and testing initial solutions. After this, the initial solutions proposed are refined and evaluated. The competing solutions are then compared and evaluated. This is followed by a mapping of the solution space. Then finally the knowledge gained is transferred into working technology.

4.2 Software Environment

This section will describe the software environment used to facilitate the project. This includes the choice of programming languages, libraries, version control and programming environment.

4.2.1 Python

Most of the project is developed in the Python(Version 3.7) programming language. Python is an interpreted, high-level, general-purpose programming language with dynamic semantics. It is maintained by the Python Software Foundation [48]. Python is one of the most widely used programming languages today[49]. Python has a large online community of developers with excellent support. Python has also been at the forefront of machine learning research and development. It has several high-quality machine learning libraries such as TensorFlow[50], Theano [51] and PyTorch[52], that make it the ideal choice for a machine learning project.

The Jupyter Notebook

The models are developed as Jupyter Notebooks. The Jupyter Notebook is an open-source web application that facilitates the creation of documents that contain live code blocks, equations, visualizations and narrative text. Jupyter Notebooks are often used for the rapid prototyping of machine learning models. It allows for in-depth explanations of the models using markup and latex, as well as the ability to contain small executable blocks of code which makes it ideal for carrying out machine learning tasks.

4.2.2 TensorFlow

The main Machine Learning library used in this project is TensorFlow TensorFlow(TF) [50] is an open-source python library for numerical computation using data flow graphs. The graph structures consist of nodes and edges, where the nodes represent mathematical operations and the edges represent multidimensional data arrays called tensors. TensorFlow has dedicated Deep Learning modules that simplify the process of constructing neural networks.

Tensorflow 2.0

TensorFlow 2.0 is the new version of TensorFlow that was released in late 2019. It features several upgrades from the original system that make it an attractive choice, even though the majority of TensorFlow resources available online are in TensorFlow 1. TF 2.0 features a more streamlined API, eager execution, a reduced reliance on global variables and a new intuitive functional structure, as opposed to the old sessions structure.

Keras API

Keras is a high-level neural network library, written in Python and built as a wrapper on top of Tensorflow. TensorFlow 2.0 absorbed and integrated the Keras wrapper. The new tf.keras wrapper is extremely intuitive to use. The new Sequential API allows rapid and intuitive prototyping of models with a linear stack of layers. The Functional API allows intuitive creation of models with nonlinear topologies. The eager debugging mode allows you to debug custom graphs, loss and activation functions quite easily, a task that was challenging in TF 1.x.

TensorBoard

TensorBoard is used to track the progress of the models, generate visualizations and produce structural graphs.
TensorBoard [53] is a tool for providing the measurements and visualizations needed during the machine learning workflow. It allows for the tracking of experimental metrics such as loss and accuracy, visualizing model graphs, projecting embeddings onto lower-dimensional spacers and profiling TF programs.

4.2.3 Pandas

Pandas dataframes are used for the loading, pre-process and manipulation of the datasets. Pandas[54] is a python library for data manipulation and analysis. It provides fast, flexible and expressive data structures, that make it easy to deal with structured data [54]. Pandas is built on top of NumPy[55] and integrated naturally with the data structures from TensorFlow.

4.2.4 scikit-learn

Scikit-learn[56] is used for preprocessing, to generate models, produce performance metrics and model selection.
Scikit-learn is a general-purpose machine learning library that is built on NumPy. It has several modules focusing on classification, regression, clustering, dimensionality reduction, model selection and preprocessing.

4.2.5 GitHub

GitHub was used for version control, file management and backup. GitHub is a Git repository hosting service. It offers the distributed version control software of git along

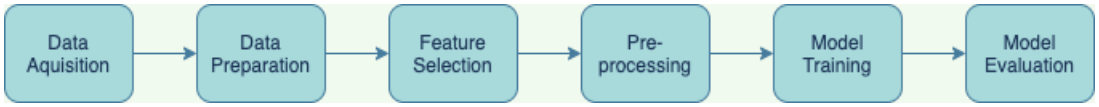

Figure 4.1: Structure of system

with a graphical interface and several collaborative features.

4.3 Hardware

All the computation for the models were run on Google Colab. Colaboratory is a research project by Google to help make high-quality machine learning hardware available to the public. It is a cloud environment for Jupyter Notebooks with hardware that is built for running machine learning algorithms. Colab offers compute resources in the form of CPU, GPU and TPU(Tensor Processing Unit). The deep learning models are run on the CPU, GPU and TPU to test for performance and training time.

The CPU compute resources consist of an Intel Xeon Processor with two cores, clocking in at 2.3 GHz with 13 GB of RAM. The GPU accelerated compute nodes offer the same configuration along with an Nvidia Tesla K80 with 12 GB of GDDR5 VRAM. The TPU resources, that were experimentally constructed by google, specifically to be used for machine learning, contains 8 TPU compute cores.

4.4 System Design

The design of the system is showcased in figure 4.1. The system consists of a data acquisition step that involves querying the BRENDA database to obtain the appropriate dataset for the objective. The next step in the pipeline is data preparation, in which the data required for the project is extracted from the data acquired in the previous step. The next step involves extracting the appropriate features for inference using feature selection techniques. After that, we have the pre-processing step, which prepares the data to be ingested by the inference algorithms. We then train the machine learning models. The final step is the evaluation and visualization of the performance of the model and the comparison between models.

4.5 Data Acquisition

The dataset for the project is obtained through querying the BRENDA database. BRENDA supports a variety of querying methods including Test-based, Structure-based and GUI-based querying[8].
The required dataset needs to contain all the K_M values recorded on the database, along with 'EC Number', 'Substrate' and 'Organism'.
I initially tried to use the GUI-based querying on the BRENDA website, using the 'Search KM value Mm' portal. This query constantly timed out due to the large nature of the query and the need to cross-reference the database for Organism information.
After this, I tried using the SOAP API client maintained by BRENDA to access the information. The data was acquired using the 'getKmVlaue()' API and was facilitated using Python. The data obtained was stored as a CSV file.

4.6 Data Preparation

The Dataset obtained from the data acquisition step contained five columns, namely, 'EC Number', 'Organsim', 'Substrate', 'KM value' and 'Comments'. The CSV file was read into a pandas dataframe.

EC Number

The 'EC Number' is a string of four number(and a few letters) that indicate the top-down hierarchical structure of enzymes. Some of the models we are using are trying to test whether this hierarchical structure affects the values of the kinetic parameters. Therefore we are going to split this field into four separate fields named 'EC1', 'EC2', 'EC3' and 'EC4'. 'EC1' is going to contain just the first character of 'EC Number', 'EC2' contains the first two characters of 'EC Number', 'EC3' contains the first three characters of 'EC Number' and 'EC4' contains the whole 'EC number'.
This process was implemented through a python function using pandas.

Organism

The 'Organism' column contains the biological species name of the organism. As we are also trying to test the hierarchical relationship between taxonomic structure and kinetic parameter value, we need to obtain the whole taxonomic tree of the organism. This was achieved by querying the National Center for Biotechnology Information's(NCIB) taxonomy database with the species name and extracting the full taxonomic tree. Out

of this, seven new columns were created, namely, 'superkingdom', 'phylum', 'class', 'order', 'family', 'genus' and 'species'.
The taxonomy database was downloaded as a CSV file. A python function was created to search for the organism name in the taxonomy database, retrieve the relevant taxonomic tree columns and add it to the pandas dataframe.

Remaining Columns

The 'Comments' column contained miscellaneous information about the experimental conditions in which the K_M value was recorded. This information though useful is not consistently present or evenly formatted, so had to be removed from the dataset.
The 'Substrate' column contains the chemical name of the substrates used in the reactions, that were represented as strings. This was left unchanged.
The 'KM value' column contains the K_M value for each reaction and is represented as a floating point. This was also left unchanged.

4.7 Pre-processing

The Pre-processing step is used to remove formatting irregularities and incorrect values in the dataset. The 'KM Value' field had several outputs with values '-999' to represent an error during measurement. All rows with these values were removed from the dataset. There were also several 'NaN' values that were present in the dataset after the Data Preparation step in the taxonomic and ec number columns. These were also removed. These procedures were implemented using the pandas 'drop()' and 'dropna()' functions.

4.8 Feature Selection

The 'f_regression()' function was used to perform an ANOVA f-test on the dataset. It was formed of multiple univariate linear regression tests. The function created a linear model for testing the individual effect of each feature.
The 'mutual_info_regression is used to find the Mutual Information between the input variables and the target variables. The MI was also included to test for any nonlinear association that might have been missed by the ANOVA test.

4.9 Mixture Density Network Model

The Mixture Density Network(MDN) model was created using the Keras Functional API from TF 2.0. The Mixture Density Model created was based on the formalism described by Bishop in [27]. The base structure of the neural network is presented in Figure 4.2. The base structure consists of an input layer, an embedding layer, several hidden layers, a parameter layer and an output layer. The optimal architecture for the problem is achieved through experimental changes to this base network. The number of distribution in the mixture is a hyper-parameter that is experimented with.

4.9.1 Embedding layer

Since all of the inputs to the model are categorical variables, embedding layers are used to convert the data into condensed vectors that can be digested by the neural network. The categorical variables need to be encoded into numeric types to be inputted into the embedding layers. This encoding is achieved through a python function that outputs the new encoded feature along with a dictionary of labels.
The size of each embedding layer is a hyper-parameter. The size of each embedding layer is calculated using the formula:

$$Embeddingsize = Min(600, round(1.6k^{0.56})) \tag{4.1}$$

where k is the number of unique labels. This formula is more an in-practice rule of thumb rather than an optimal theoretical value and was proposed by Jeremy Howard in [57]. Each embedding layer is represented using the 'tf.keras.layers.Embedding' function and is of the shape (x,None), where x is the custom embedding size. All the embeddings layers are concatenated into one large vector in a concatenation layer. This vector is then fed into the hidden layers. As the embeddings are trained, the intrinsic underlying hierarchical structure of the data should be emergent, if the hierarchy truly affects the kinetic parameters.

4.9.2 Hidden Layers

The base network contains two hidden layers. The hidden layers are located in between the embedding layers and output layer. They perform nonlinear transformations of the concatenated embedding vector. The hidden layers are used to break down the neural network into specific transformations of the data. They combine to produce the inference formed by the neural network.

Each hidden layer is represented by a 'tf.keras.layers.Dense' layer.

Activation function

Activation functions are used to introduce the nonlinear features to the layer. Several choices for activation function were presented in Section 3.10 such as the Sigmoid or ReLU functions. The choice of activation function in neural network computation has a significant effect on performance, therefore it is important to choose that correct activation function. The Sigmoid function often results in vanishing gradients, in which the gradient of the function reduces to an infinitesimally small number as the value of the input increases. The ReLU function produces a gradient that is a constant value with respect to its input, which allows it to avoid the vanishing gradient problem. ReLU also seems to outperform the Sigmoid function in practice. It is for these reasons that the ReLU function was chosen as the activation function for the hidden layers.

4.9.3 Mixture Layer

The Mixture Layer consists of three outputs, namely, the mixing coefficient: alpha, the mean: mu and the variance: sigma. These outputs are meant to represent the mixing coefficient(α), mean(μ) and variance(σ) of a Gaussian Mixture Model as explained in Section 3.9. Each output has a unique activation function and output. The output of the three layers is then concatenated into a singular output vector which is the final output of the network.
The mixture layer is implemented as three separate 'tf.keras.layers.Dense' layers each initialised with an output shape of (c,None), where c is the number of distributions in the mixture.

Mixing Coefficient

The Mixing Coefficients as explained in Section 3.9 must sum to unity.

$$\sum \alpha(x) = 1 \tag{4.2}$$

Therefore we need to select an activation function that outputs a vector of values that sum up to 1. The Output of this node essentially decides how strongly each distribution is expressed in the mixture distribution and can be thought of as a probability value. As discussed in Section 3.10, the Softmax function constrains the output to the form of a K-dimensional vector with values ranging from 0 to 1 and the elements all sum up to

1. We also discussed how the output of the Softmax function is used as a categorical distribution representing the probability of occurrence of each element. This formalism, therefore, matches what we were looking for in an activation function, therefore we have chosen the Softmax function as an activation.

Mean

The Mean (μ) parameter represent the centre of each of the distributions in the mixture. As suggested by Bishop in [27] the output of this node is expressed directly by the network output, without a nonlinear activation function.

Variance

The σ parameter represents the scale of the distribution. The constraint is imposed on it is that it has to be greater than zero. For this reason, Bishop [27] recommends using the exponential activation function to represent them. The exponential, however, can lead to numerical instability. Alternatively, we can use a variant of the ELU activation function from Section 3.10 with an offset of one, to make it non-negative, as suggested by [22] and [30].
The ELU + 1 function is implemented by defining a custom activation function in TensorFlow.

4.9.4 Loss Function

The loss function with respect to the weights can be represented as

$$log\mathcal{L}(y|x) = \frac{-1}{N}\sum_{n=1}^{N} log\{\sum_{k}\alpha_k(x_n,w)\mathcal{N}(y_n|\mu_k(x_n,w),\sigma_k^2(x_n,w))\} \tag{4.3}$$

where α, μ and σ are the parameters of the mixture distribution. This is the loss function from Section 3.9 expressed as a Gaussian Mixture.
The loss function is implemented by creating a custom loss function and directly translating the equation 4.3 into code.

4.9.5 Optimization Algorithm

The Optimization Algorithm for the MDN decides how the model learns. There are several candidate Optimization Algorithms presented in Section 3.11. Out of these algorithms, the stochastic and mini-batch gradient descent methods use a fixed learning rate

for all parameters, whereas the AdaGrad and Adam use a dynamic learning rule to update parameters. We are going to choose an adaptive learning rate method as the method computes individual learning rate for each parameter and eliminates the need to tune a learning rate parameter. Adam is chosen over AdaGrad because of AdaGrad's accumulation of squared gradients. This is overcome in Adam by using a decaying average of past and past-squared gradients. Adam is used as the optimization method for the MDN. The Optimization is implemented using the 'tf.keras.optimizers.Adam()' function.

4.9.6 Regularization

Regularization is used in the model to reduce the generalization error on previously unseen data. In Section 3.12, two approaches were presented, namely, L_2 Regularization and Dropout. Out of the two approaches, dropout was chosen, as according to [58], it is more effective at dealing with generalization error that L_2 for networks with a large number of neurons, like the MDN model.
Dropout is implemented using the 'tf.keras.layers.Dropout()' function within the functional structure of the neural network.

4.9.7 Persistent NaN Problem

The loss function occasionally outputted NaN(Not-a-Number) values after several epochs of running the algorithm. NaNs are often generated due to underflow by the computer. Underflow is caused when we try to represent a very small number and it can not be represented by the CPU or memory or the datatype being used, this often results in NaN values. We will investigate the cause of these NaNs and propose possible solutions. There are three typical situations in which NaNs appear in machine learning problems:

1. The logarithm of a value is close to zero
2. The denominator of a fraction is very close to zero
3. Exponential of a big enough value to NaN

If we take a look at the loss function:

$$log\mathcal{L}(y|x) = -log\left(\sum_{i=0}^{m} \frac{\alpha_i(x)}{2(\pi)^D/2\sigma_i(x)^D} exp\left\{-\frac{||y-\mu_i(x)||^2}{2\sigma_i(x)^2}\right\}\right) \tag{4.4}$$

We are doing an exponential and then a logarithm of values that tend to be very small, this is potentially the source of the underflow problem.

Using Tensorflow Probability

An alternative to natively implementing the loss function would be to implement it using some existing TensorFlow constructs. TensorFlow Probability is a TensorFlow library for probabilistic reasoning and statistical analysis. It contains a wide selection of implemented probability distributions that could help solve the underflow problem. In [30], an alternative implementation of the loss function was presented using TensorFlow Probability.
The alternative representation from [30] is implemented using the 'tensorflow_probability.distributions' module. The 'MixtureSameFamily' function from the module can represent a mixture of distributions that are of the same type of distribution. The function is used to represent the Gaussian Mixture Model that is outputted by the MDN, a categorical distribution is used to represent the mixing coefficients and the component distributions are represented by normal distributions with the 'loc' and 'scale' parameters from the MDN representing each Gaussian in the mixture. The 'MixtureSameFamily' function returns a Gaussian Mixture Model. Once this GMM is constructed the log-likelihood of the target variable y is calculated using the 'log.prob()' function and the mean value returned.

log-sum-exp trick

A solution to the underflow problem when we have a logarithm of summed exponential values is to use the log-sum-exp trick that helps to eliminate the numerical instability of the logarithm of a sum of exponents:

$$log \sum_{i=1}^{n} e^{x_i} = \max_i x_i + log \sum_{i=1}^{n} e^{x_i - \max_i x_i} \tag{4.5}$$

If we apply this to the loss function, we obtain:

$$log\mathcal{L}(y|x) = -log\left(\sum_{i=0}^{m} exp\left\{ log(\alpha_i(x)) - d/2log(2\pi\sigma_i(x) - \frac{||y-\mu_i(x)||^2}{2\sigma_i(x)^2} \right\} \right) \tag{4.6}$$

We can apply the log-sum-exp trick to combat the NaN problem as suggested by [22]

Gradient Clipping

This is a solution to situation 3. Applying a Gradient Clipping technique while training the model to try to keep the value of the gradients reasonable, if a gradient gets too

large, it is clipped off. This can be applied to the α and σ parameters. Excessive Gradient Clipping could lead to a poorly generalised solution.

Too Many Distributions in the Mixture

If there are too many distributions in the mixture m, then it is possible for the $\alpha(x_i)$ of a distribution that isn't very represented. It is possible that one of the distributions are not used at all for the i^{th} point. In this case, the distribution will have a very small $\alpha(x_i)$, and when we apply the softmax function on it this value will be even further reduced. A possible solution would be to clip the $\alpha(x)$ value by defining a minimum value for it, as suggested in [22].
All of the methods mentioned above were experimented with to see which remedy or combination of remedies effectively reduced the loss function going to NaN. The method that worked best in achieving this was the switch to the TensorFlow Probability implementation of the loss function, as described in [30]. Choosing the appropriate number of distributions also proved to be an important factor in avoiding NaNs in the loss function.

4.10 Deep Neural Network Model

A Deep Neural Network(DNN) model was developed to test the efficacy of the Mixture Density Model on the dataset. This DNN is meant to mimic the performance of the corresponding MDN developed to test The DNN has the same input, embedding and hidden layers as the MDN. The DNN has a single output node that generates a continuous output that represents the 'KM Value' target variable. The DNN uses Mean Squared Error(MSE) as defined in Section 3.5.1 as the error function.

4.11 Linear Regression Model

A Linear Regression Model as proposed by Borger et al. in [2] was developed for comparison with the Mixture Density Network Model. Borger proposed fashioning the linear model with a mean μ and effects α for the enzyme and β for the organism, we have added another parameter γ for the effect of the substrate.
The Linear Regression model was also developed to test whether the hierarchical expansions of the Enzyme number and taxonomic expansion of the Organisms had any effects on the prediction of the kinetic parameters. The formalism of the regression model is an Ordinary Least Squares Regression. It is implemented using the 'sklearn.linear_model' module in Python.

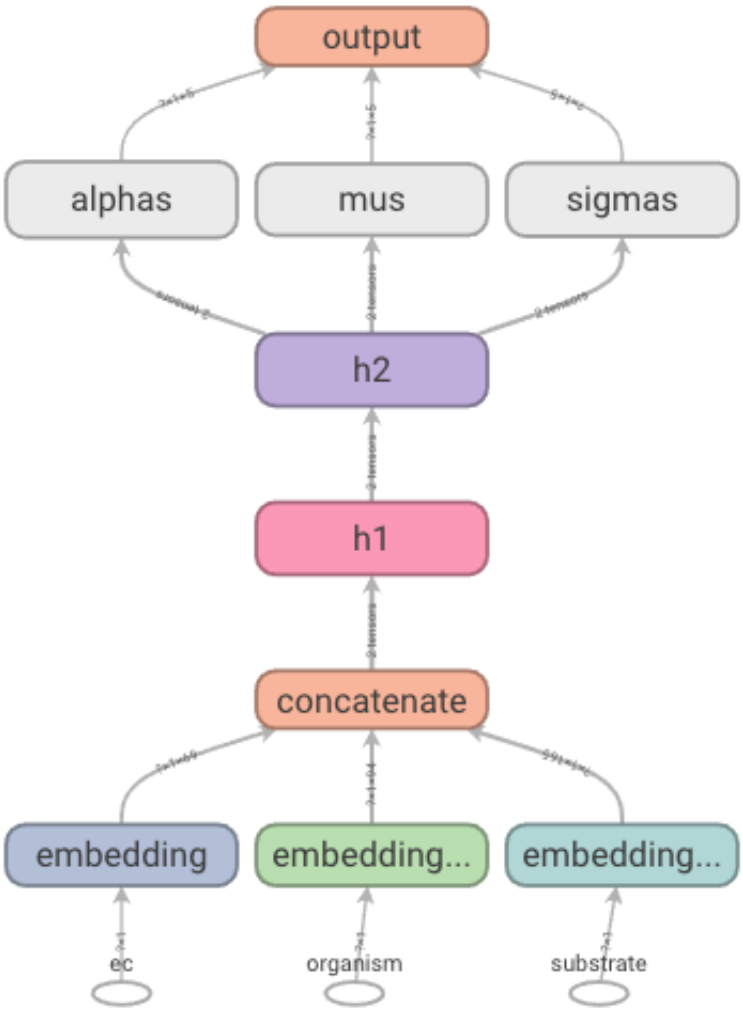

Figure 4.2: Graph for Mixture Density Network from TensorBoard

Number of Mixtures	1, 2, 3, 4, 5, 6
Number of hidden layers	2, 3, 4
Size of hidden layers	50, 100, 200, 300
Dropout	0.1, 0.2, 0.3

Table 4.1: MDN: Values for grid search

Number of hidden layers	2, 3, 4
Size of hidden layers	50, 100, 200, 300
Dropout	0.1, 0.2, 0.3

Table 4.2: DNN:Values for grid search

4.12 Model Selection

A Grid search is used to find the optimal hyper-parameters for the MDN and DNN models. Grid Search is a Model Selection method designed to find the optimal hyper-parameters of a model by checking all combinations of values for a set of specified hyper-parameters. The models generated through the grid search need to be evaluated to determine which one had the best performance. The MDN model usually converged after about 250 epochs, whereas the DNN converged after 200 epochs. The parameters for the MDN are: the number of mixtures in the distribution, the number of hidden layers, the size of the hidden layers, the percentage of nodes that are dropped out of the hidden layers. The values inputted into the grid search are in 4.1. The parameters for the DNN are: the number of hidden layers, the size of the hidden layers and the percentage of nodes dropped out the percentage. The values used for the DNN grid search are presented in Table 4.2 .The grid search was implemented using the HParams() function from TensorFlow.
Models are usually evaluated using a form of model validation, like cross-validation, as discussed in Section 3.16. Each model is evaluated using 10-fold cross-validation, as it ensure that the full dataset is trained an tested on. It is for these reasons that it generally produces a less biased or less optimistic estimate of testing error.

4.13 Evaluation Metrics

Model Validation requires an evaluation metric to generate a testing error value. An evaluation metric is a measure that is used to test the fit of the prediction of a machine learning model against the label data. Due to the nature of the predictions from the Mixture Density Network Model, i.e., the Gaussian Mixture Model output rather than a continuous regression output, we have to devise some appropriate evaluation metrics.

4.13.1 Negative Log Loss as an Evaluation Metric

The Negative Log Loss(NLL), as it is defined for the Mixture Density Model in Section 4.9. This allows us to test the log-likelihood of the target variable being present in the predicted distribution. The Linear Model and DNN can also use the metric by simulating a continuous uniform distribution around the means. Using this method we can use the NLL as an evaluation metric for the models.

4.13.2 Mean Square Error as an Evaluation Metric

The Mean Square Error, as it is defined in Section 3.17 could also be used as a metric for evaluation. It is the natural choice for a regression problem and there are a few ways that it can be adapted to be used with the Mixture Density Network. The MDN produces a Mixture of Gaussian's as its output. The first strategy would be to sample from the distribution produced for the input and using that as the point output. Another possible strategy would be to calculate the mean for the whole mixture of Gaussians and using that as the point output. We are going to pick the first option in the spirit of testing the model's ability to generate a probability distribution function(pdf). The MSE can be naturally used with the DNN and Linear Regression Models.

Chapter 5

Experimental Results and Discussion

This chapter will contain the experimental results that were obtained from the evaluation of the system and the resulting implications. We will first analyze the dataset to explore the biological relationship between the hierarchical nature of enzymes and organisms, and their effect on K_M parameter values. We will then test the performance of the proposed mixture density network against the deep neural network and linear model.

5.1 Preliminary Analysis of Dataset

In this section, we will analyze the dataset to explore which contributing factors most effect the prediction of the kinetic parameters. We will first be looking at the relationship between the biological hierarchies of enzymes and organisms, and the values of the Michaelis-Menten constant K_M.

As described in Section 4.6, the EC number is split into its constituent sections: 'ec1', 'ec2', 'ec3', 'ec4'. The Organism is also split into its taxonomic tree, that is represented by seven constituents: , 'superkingdom', 'phylum', 'class', 'order', 'family', 'genus' and 'species'.

An Analysis of Variance Explained for Regression is then performed on the dataset to test for linear associations, as described in Section 4.8, to test variance explained by each of the explanatory variables to the output variable. The Mutual Information of each feature and the K_M value is also calculated to test for non-linear associations. The results of this analysis is presented in Table 5.1. The table contains the F-score obtained from the ANOVA test, the corresponding p-value obtained through the F-statistic and the value of Mutual Information.

A high F-score corresponds to high linear association between the feature and the target.

Feature	F-score	p-value	MI
ec2	6.0184	0.0141	0.0642
ec3	2.9314	0.0868	0.0683
ec4	2.2389	0.1345	0.0564
substrate	17.8044	2.4646e-05	0.0237
superkingdom	1.5921	0.2070	0.0316
phylum	1.5003	0.2206	0.0305
class	1.7350	0.1877	0.0357
order	0.5668	0.4515	0.0362
family	0.4835	0.4868	0.0186
genus	0.1012	0.7503	0.0237
species	0.0756	0.7832	0.0322

Table 5.1: Table representing the significance of each feature to K_M value

The corresponding p-value obtained from the F-statistic corresponds to a measure of statistical significance of the result. The lower the p-value, the more significant the result. A p-value < 0.05 corresponds to 95% confidence of statistical significance.

According to the values obtained from this analysis we see that only 'substrate' and 'ec2' can be considered statistically significant.

The Mutual Information scores also have the various splits of the EC number as having the highest Mutual Information This is an interesting result, as this seems to point to the fact that the K_M value owe a majority of their variability to the Enzyme type and the substrate in the reaction, but not as much to the Organism. As we can see from the table 5.1, the top 3 F-scores after the 'substrate', all belong to various splits of the EC number.

5.2 Comparison of Feature Subsets on Model Performance

Following on from the previous section, this section will explore the performance of the various models on a variety of different feature sets. All the models will first be tested using the original features('ec_number', 'organism' and 'substrate') to predict the Michaelis-Menten constant K_M. We will then test the models on the modified feature set as described in Section 4.8, these will include the split EC number ('ec1', 'ec2', 'ec3', 'ec4') and the taxonomic expansion of the Organism ('superkingdom', 'phylum', 'class', 'order', 'family', 'genus', 'species). After this, we will use feature selection procedures to select the best subset of features for the models.

Feature	CV Error
ec2 + substrate	-0.00255
substrate	-0.00278
ec2 + ec4 + substrate	-0.00366
ec2 + ec4 + substrate + phylum	-0.00387
ec2 + ec4 + substrate + phylum + class	-0.00415

Table 5.2: Feature sets and 10-fold cross-validation error(R^2) for Linear Regression Model

5.2.1 Linear Model

First, we will test the performance of the various subsets on a Linear Model. A Linear Regression Model was implemented to test the various subsets of the data. The Linear Regression model was exposed to the fully transformed dataset with new features. An exhaustive search of all the features is performed to obtain the best combination of features that can predict the K_M value. Each model is evaluated using 10-fold cross-validation. The best feature sets along with the cross-validated error are presented in table 5.2.

The results of this analysis show that the best performing model contained the feature set of 'ec2' and 'substrate'. Every top performer contained the features 'substrate', 'ec2', 'ec4', 'phylum' and 'class'. If we compare these results to the ANOVA f-test that was performed in Section 5.1, we can see that all the top-performing individual regressors are present in the top combinations, except for 'ec3' and 'superkingdom'. These could possibly be overshadowed by more discriminatory elements of the hierarchies in Enzyme and Organism.

The negative R^2 value of all the linear regression models show that the difference between the true label and the predicted label is larger than the difference between the true label and the true mean. This tells us that the models have all probably overfit the data. This could indicate that the linear decision function formed by the Linear Regression Model as conceived by [2] is probably not an ideal regression for this problem.

5.2.2 Mixture Density Model

First we will test the different subsets on the Mixture Density Network Model. The structure of the MDN is as described in Section 4.9. Each model has a specific separate structure due to the embeddings used to express each input. Each model has n input nodes, where n is the number of features, and therefore also n embedding layers. The size of each embedding layer is calculated by the method described in 4.9.1. After this

Features	Training Error	Testing Error
All	1.504	1.585
substrate + ec4 + species	1.5878	1.664
substrate + ec2	1.477	1.596
substrate + ec2 + ec3	1.4167	1.542
substrate + ec2 + ec3 + ec4	1.6728	1.735
substrate +ec2 +ec3 + ec4 + class	1.4768	1.683
substrate +ec2 +ec3 + ec4 + class + phylum	1.4798	1.49

Table 5.3: MDN model tested on different subsets of the data

the embeddings are concatenated into a large embedding layer. After the concatenation layer, there are two hidden layers with 200 units in each. This is followed by the output as it is described in Section 4.9. Each model uses a mixture of 5 Gaussians to represent each output. The MDN loss function as described in section 4.9 is used to calculate training error. Each model is run for 250 epochs, which was seen to be around the time when most of the models converged.
As we can see from the results in Table 5.3, the distributions formed by the MDN model are not significantly improved as steps in the hierarchy are added. The model with all of the hierarchical features of the EC number and the split of the taxonomic tree only performs slightly better than the model with only the initial configuration, that is the full EC number('ec4') and the Organism('species'). This might be because the hierarchies are learnt by the trained embeddings or maybe that the hierarchies do not translate to the kinetic parameter K_M. The combination of the model with only the 'substrate' and 'ec2' was able to predict the distribution to a reasonable degree. The best distribution was produced by the combination of 'substrate', all sections of the EC number and the 'class' and 'phylum' from the organism.

5.3 Optimal Parameters

In this section we will discuss the optimal parameters that were found for the Mixture Density Network and the Deep Neural Network models.

5.3.1 Mixture Desnity Network

A grid search was performed to find the optimal parameters form the Mixture Density Network Model. The full set of features(Split EC number and taxonomic tree of Organism) are used for the parameter tuning. The parameters being tuned are the number of mixtures in the distribution c, the number of hidden layers, the size of the hidden layers

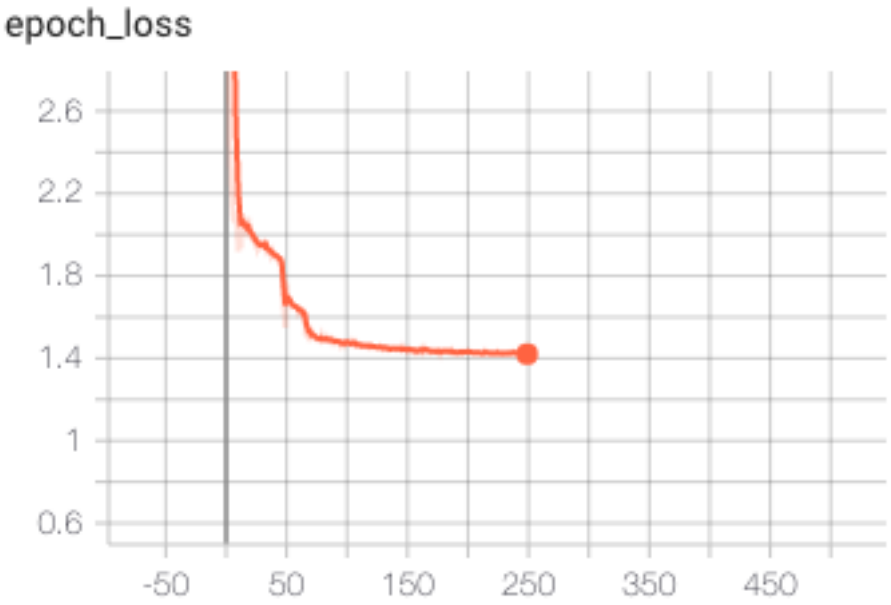

Figure 5.1: MDN training loss from tensorboard

c	Hidden Layers	Dropout	Training Error	Testing error
4	(200,200,200,200)	0.2	1.417	1.453
5	(200,200,200,200)	0.2	1.447	1.476
5	(200,200,200,200)	0.2	1.476	1.54
5	(100,100,100,100)	0.2	1.504	1.585

Table 5.4: Top results from grid search of MDN model.

and the rate of dropout. Each model in this search was run for 250 epochs to determine the best combination of parameters. The best performing models are presented in Table 4.1. The training loss of the best model is displayed in Figure 5.1.

The grid search revealed that choosing a value of c that was below 4 often resulted in a NaN in the loss function. For this reason, much of the results had to be disregarded due to instability. Similarly, $c = 6$ most of the models generated produced errors of NaN. This tells us that the number of mixtures is key to the stability of the model. The model was most stable when using 4 mixtures in the model, using 5 mixtures was generally stable, but occasionally produced NaNs. Adding hidden layers to the model generally resulted in better training error and generalization. The optimal number of nodes in the hidden layer was seen to be 200 nodes.

Hidden Layers	Dropout	Training error	Testing Error
(300,300,300,300)	0.2	253.228	374.786
(300,300,300,300)	0.1	255.388	394.364
(200,200,200,200)	0.3	300.3281	418.7301
(200,200,200,200)	0.2	305.8443	486.4447

Table 5.5: Top results from grid search of MDN model

Model	MDN Loss	MSE
Mixture Density Network	1.453	871.9803
Deep Neural Network	7.541	374.786

Table 5.6: Model Comparison

5.3.2 Deep Neural Network

A gird search was also performed on the Deep Neural Network Model to find it's optimal parameters. The full set of features(Split EC number and taxonomic tree of Organism) are used for the parameter tuning. The hyper-parameters to be tuned are the number of hidden layers, the size of the hidden layers. Each model in the search was run for 200 epochs, which is when they converged. The training and testing errors are measured in terms of Mean Squared Error(MSE). The grid search revealed that the deeper neural networks were better at modelling the point predictions than shallower one, all of the top 4 models had 4 hidden layers. It was also found that models with 300 hidden layers produced the models with the best performance.

5.4 Comparison of MDN and DNN models

In this section, we will compare the performance of the Mixture Density Network and Deep Neural Network Models. This will be a little challenging, as the Mixture Density Network outputs a Mixture of Gaussian's and the Deep Neural Network produces a continuous numeric output. We discussed a few strategies for bridging this gap in Section 4.13. We will now attempt to provide a Mean Square Error estimate for the MDN, and use the MDN loss function on the DNN. The training error was produced through 10-fold cross-validation of the data. The results, as presented in 5.6, show us that the Mixture Density Network performs better using the MDN loss function and the Deep Neural Network performs better using the Mean Squared Error. This is probably more indicative of both metrics being poor in capturing the performance of either model. The MDN model performed significantly better than the DNN model using the MDN-loss metric, the uniform distribution around the mean proved to be a poor method of representing

the continuous output. Sampling from the MDN and measuring that against the target variable provided poor results. This is indicative of poor generalisation of the model.

Chapter 6

Conclusion

In this chapter, we will conclude the research for this dissertation and provide a few possible future avenues for this work.

6.1 Conclusion

Throughout this project, the aim has been to developed a system to accurately model kinetic parameters for ensemble modelling in systems biology. The project set out to enrich the protocol set out by Tsigkinopoulou et al in [3]. This was accomplished through first, acquiring the dataset through querying the enzyme database BRENDA to find extract the enzyme, organism and chemical information for all Michaelis-Menten constant (K_M) values recorded. Prepossessing the database and converting it into a usable form. The original dataset was then enriched by splitting up the predictor variables into their biological and chemical hierarchies. The dataset was then tested to see if the biological information translated into a tangible difference in kinetic parameter values. As we saw through the analysis of the database, the substrate value had the most individual effect on the K_M value. We then proposed a novel solution to the problem, by appropriating and implementing a neural network model that predicts a learned distribution of parameter values. We then compared this model to a Deep Neural Network model to test the performance of the generated probability distribution functions.

Although not precisely producing the optimal priors for the kinetic parameter K_M, this project explored an interesting possible formalism for uncertainty estimation for scientific problems that require specific modelling of the distribution of quantities.

6.2 Shortcomings

The shortcomings of the mixture density model have to do with its predictive power. The prior distributions generated did not generalise very well. This could be improved by more experimentation with the architecture of the network, using an alternative formalism. The project also fell short when coming up with an appropriate method to accurately compare the distribution generated with the continuous output of other models. A less precise, but possibly as effective method of estimating this could be a visual representation. Not exploring this avenue is another shortcoming of this project.

6.3 Future Work

The next step would be to truly refine the network architecture, for a more precise model that can better leverage the data. After this, I would like to integrate fully with the protocol set out by Tsigkinopoulou et al. and extend the model to include sampling that ensures thermodynamic consistency through Monte Carlo sampling.

Bibliography

[1] I. Tavassoly, J. Goldfarb, and R. Iyengar, "Systems biology primer: The basic methods and approaches," *Essays in biochemistry*, vol. 62, no. 4, pp. 487–500, 2018.

[2] S. Borger, W. Liebermeister, and E. Klipp, "Prediction of enzyme kinetic parameters based on statistical learning," *Genome Informatics*, vol. 17, no. 1, pp. 80–87, 2006.

[3] A. Tsigkinopoulou, A. Hawari, M. Uttley, and R. Breitling, "Defining informative priors for ensemble modeling in systems biology," *Nature protocols*, vol. 13, no. 11, pp. 2643–2663, 2018.

[4] M. A. H. Samee, B. Lim, N. Samper, H. Lu, C. A. Rushlow, G. Jiménez, S. Y. Shvartsman, and S. Sinha, "A systematic ensemble approach to thermodynamic modeling of gene expression from sequence data," *Cell systems*, vol. 1, no. 6, pp. 396–407, 2015.

[5] T. Khazaei, A. P. McGuigan, and R. Mahadevan, "Ensemble modeling of cancer metabolism," *Frontiers in physiology*, vol. 3, p. 135, 2012.

[6] L. M. Tran, M. L. Rizk, and J. C. Liao, "Ensemble modeling of metabolic networks," *Biophysical journal*, vol. 95, no. 12, pp. 5606–5617, 2008.

[7] I. Schomburg, A. Chang, O. Hofmann, C. Ebeling, F. Ehrentreich, and D. Schomburg, *Brenda: A resource for enzyme data and metabolic information*, 2002.

[8] L. Jeske, S. Placzek, I. Schomburg, A. Chang, and D. Schomburg, "Brenda in 2019: A european elixir core data resource," *Nucleic acids research*, vol. 47, no. D1, pp. D542–D549, 2019.

[9] E. C. Webb *et al.*, *Enzyme nomenclature 1992. Recommendations of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology on the Nomenclature and Classification of Enzymes.* Ed. 6. Academic Press, 1992.

[10] A. Lesk, *Introduction to bioinformatics*. Oxford university press, 2019.

[11] C. Jeffrey *et al.*, *Biological nomenclature.* Ed. 2. Edward Arnold (Publishers) Ltd., 1977.

[12] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, 10. Springer series in statistics New York, 2001, vol. 1.

[13] J. R. Koza, F. H. Bennett, D. Andre, M. A. Keane, and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Transactions on evolutionary computation*, vol. 1, no. 2, pp. 109–128, 1997.

[14] D. Michie, D. J. Spiegelhalter, C. Taylor, *et al.*, "Machine learning," *Neural and Statistical Classification*, vol. 13, no. 1994, pp. 1–298, 1994.

[15] K. Alsabti, S. Ranka, and V. Singh, "An efficient k-means clustering algorithm," 1997.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[17] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[18] L. Birgé, "Model selection for density estimation with l2-loss," *arXiv preprint arXiv:0808.1416*, 2008.

[19] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in neural information processing systems*, 2018, pp. 8778–8788.

[20] H. L. Seal, "Studies in the history of probability and statistics. xv the historical development of the gauss linear model," *Biometrika*, vol. 54, no. 1-2, pp. 1–24, 1967.

[21] *Gaussian mixture models explained*, `https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95`, Accessed: 2019-06-03.

[22] A. Brando, *Mixture density networks (mdn) for distribution and uncertainty estimation*, GitHub repository with a collection of Jupyter notebooks intended to solve a lot of problems related to MDN., 2017. [Online]. Available: `https://github.com/axelbrando/Mixture-Density-Networks-for-distribution-and-uncertainty-estimation/`.

[23] R. Socher, Y. Bengio, and C. D. Manning, "Deep learning for nlp (without magic)," in *Tutorial Abstracts of ACL 2012*, 2012, pp. 5–5.

[24] R. Tolosana, R. Vera-Rodriguez, J. Fierrez, A. Morales, and J. Ortega-Garcia, "Deepfakes and beyond: A survey of face manipulation and fake detection," *arXiv preprint arXiv:2001.00179*, 2020.

[25] L. Crnkovic-Friis and L. Crnkovic-Friis, "Generative choreography using deep learning," *arXiv preprint arXiv:1605.06921*, 2016.

[26] R. Manzelli, V. Thakkar, A. Siahkamari, and B. Kulis, "Conditioning deep generative raw audio models for structured automatic music," *arXiv preprint arXiv:1806.09905*, 2018.

[27] C. M. Bishop, "Mixture density networks," 1994.

[28] ——, *Pattern recognition and machine learning*. springer, 2006.

[29] D. Geary, "Mixture models: Inference and applications to clustering," *Journal of the Royal Statistical Society Series A*, vol. 152, no. 1, pp. 126–127, 1989.

[30] *A hitchhiker's guide to mixture density networks*, `https://towardsdatascience.com/a-hitchhikers-guide-to-mixture-density-networks-76b435826cca`, Accessed: 2019-02-15.

[31] S. Team, "Deep learning for siri's voice: On-device deep mixture density networks for hybrid unit selection synthesis," *Apple Machine Learning J*, vol. 1, no. 4, 2017.

[32] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[33] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, 2015, vol. 2018.

[34] S. Ruder, *An overview of gradient descent optimization algorithms*, 2016. arXiv: `1609.04747 [cs.LG]`.

[35] R. Sutton, "Two problems with back propagation and other steepest descent learning procedures for networks," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, 1986, pp. 823–832.

[36] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[39] M. Kuhn, K. Johnson, *et al.*, *Applied predictive modeling*. Springer, 2013, vol. 26.

[40] E. R. Girden, *ANOVA: Repeated measures*, 84. Sage, 1992.

[41] K. A. Fox and T. K. Kaul, "Intermediate economic statistics," Wiley New York, Tech. Rep., 1968.

[42] *An overview of categorical input handling for neural networks*, `https://towardsdatascience.com/an-overview-of-categorical-input-handling-for-neural-networks-c172ba552dee`, Accessed: 2019-01-16.

[43] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[44] X. Rong, "Word2vec parameter learning explained," *arXiv preprint arXiv:1411.2738*, 2014.

[45] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*, 2016.

[46] M. W. Browne, "Cross-validation methods," *Journal of mathematical psychology*, vol. 44, no. 1, pp. 108–132, 2000.

[47] *Research methods in machine learning*, `http://web.engr.oregonstate.edu/~tgd/talks/new-in-ml-2019.pdf`, Accessed: 2020-04-04.

[48] G. Van Rossum *et al.*, "Python programming language.," in *USENIX annual technical conference*, vol. 41, 2007, p. 36.

[49] T. Group *et al.*, *Tiobe index for ranking the popularity of programming languages*.

[50] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[51] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, 2016.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[53] D. Mané *et al.*, *Tensorboard: Tensorflow's visualization toolkit, 2015*.

[54] W. McKinney *et al.*, "Pandas: A foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, vol. 14, no. 9, 2011.

[55] E. Bressert, *SciPy and NumPy: an overview for developers*. " O'Reilly Media, Inc.", 2012.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[57] *Introduction to machine learning for coders*, `http://course18.fast.ai/ml`, Accessed: 2020-04-04.

[58] E. Phaisangittisagul, "An analysis of the regularization between l2 and dropout in single hidden layer neural network," in *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, IEEE, 2016, pp. 174–179.