

**6CCS3PRJ Final Year**  
**Population Coded Inputs/Outputs in**  
**Neural Networks for Regression**

Final Project Report

Author: Mufid Alkhaddour

Programme of study: Computer Science BSc

Supervisor: Dr Michael Spratling

Student ID: 1710990

April 23, 2020

## **Abstract**

Regression models have been used for several years to make accurate predictions for many aspects of everyday world problems. To make those predictions, several different regression methods are used. This report aims to introduce a new approach called population coding. It is a data manipulation technique which transforms datasets before allowing a neural network to make predictions. The performance of this new technique will then be compared to that of other regression techniques being adopted. The motivation is that the population coding of inputs and outputs of neural networks will enable models to make more accurate predictions.

### **Originality Avowal**

I verify that I am the sole author of this report, except where explicitly stated to the contrary.  
I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Mufid Alkhaddour

April 23, 2020

### **Acknowledgements**

I would like to thank my supervisor, Dr Michael Spratling, for his support and guidance throughout this project. Without his thorough descriptions, the project's quality would not have been right up to where it is today.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Background and Motivation . . . . .	5
1.2	Report Structure . . . . .	8
<b>2</b>	<b>Literature Review</b>	<b>10</b>
2.1	Machine Learning . . . . .	10
2.1.1	Supervised Learning . . . . .	11
2.1.1.1	MLP Regression . . . . .	13
2.1.1.2	Linear Regression . . . . .	17
2.1.1.3	Kernel Ridge Regression . . . . .	17
2.1.1.4	Least-angle Regression with Lasso . . . . .	18
2.1.1.5	Linear Support Vector Regression . . . . .	19
2.1.1.6	Stochastic Gradient Descent Regression . . . . .	19
2.1.1.7	Random Forest Regression . . . . .	20
2.1.1.8	AdaBoost Regression . . . . .	20
2.1.1.9	Gradient Boosting & Extreme Gradient Boosting Regression . . . . .	20
2.1.2	Unsupervised Learning . . . . .	21
2.1.2.1	Gaussian Mixture Models . . . . .	21
2.1.2.2	Expectation Maximisation . . . . .	21
2.2	Related Works . . . . .	22
<b>3</b>	<b>Design</b>	<b>23</b>
3.1	Population Coding of Inputs . . . . .	23
3.1.1	Calculating Gaussian Centres and Standard Deviations . . . . .	23
3.1.2	Coding Function . . . . .	24
3.1.3	Coding the Dataset . . . . .	25
3.1.4	Training the Neural Network and Making Predictions . . . . .	25
3.2	Population Coding of Outputs . . . . .	26
3.2.1	Calculating Gaussian Centres and the Standard Deviation . . . . .	26
3.2.2	Coding Function & Coding the Dataset . . . . .	26
3.2.3	Training the Neural Network and Making Predictions . . . . .	26
3.2.4	Decoding Function . . . . .	27
3.3	Population Coding of Inputs and Outputs . . . . .	28

<b>4</b>	<b>Implementation</b>	<b>29</b>
4.1	Tools . . . . .	29
4.2	Data & Results . . . . .	29
4.3	Data manipulation . . . . .	30
4.4	Calculating Gaussian Centres and Standard Deviation for Input Coding . . . .	30
4.5	Calculating Gaussian Centres and Standard Deviation for Output Coding . . .	31
4.6	Population coding of Inputs and Outputs . . . . .	31
4.7	Decoding . . . . .	32
4.8	Choosing Hyperparameters . . . . .	32
4.9	Training and Predictions of MLP . . . . .	33
4.10	Accuracy Measure . . . . .	33
<b>5</b>	<b>Results/Evaluation</b>	<b>34</b>
5.1	Population Coding of Inputs . . . . .	35
5.1.1	Small Datasets ( $\leq 250$ samples $\leq 10$ features) . . . . .	35
5.1.2	Medium Datasets ( $\leq 500$ samples $\leq 25$ features) . . . . .	35
5.1.3	Large Datasets ( $\geq 500$ samples $> 25$ features) . . . . .	36
5.1.4	Other Datasets . . . . .	36
5.1.5	Effects of the Number of Features and Samples . . . . .	36
5.1.6	Number of Gaussian Centres . . . . .	37
5.1.7	Running Time . . . . .	38
5.2	Population Coding of Outputs . . . . .	39
5.2.1	Small Datasets ( $\leq 250$ samples $\leq 10$ features) . . . . .	39
5.2.2	Larger Datasets ( $\geq 500$ samples $\geq 25$ features) . . . . .	39
5.2.3	Other Datasets . . . . .	39
5.2.4	Effects of the Number of Features and Samples . . . . .	40
5.2.5	Number of Gaussian Centres . . . . .	40
5.2.6	Decoding . . . . .	41
5.2.7	Running Time . . . . .	42
5.3	Population Coding of Inputs and Outputs . . . . .	42
5.3.1	Other Datasets . . . . .	43
5.4	Dataset Analysis . . . . .	43
5.5	Evaluation & Other Algorithms . . . . .	43
<b>6</b>	<b>Legal, Social, Ethical and Professional Issues</b>	<b>46</b>
<b>7</b>	<b>Conclusion and Future Work</b>	<b>47</b>
7.1	Future Work . . . . .	47
7.2	Conclusion . . . . .	48
	Bibliography . . . . .	53
<b>A</b>	<b>Extra Information</b>	<b>54</b>
A.1	Results . . . . .	54

<b>B</b>	<b>User Guide</b>	<b>59</b>
B.1	Instructions . . . . .	59
B.1.1	Code Preparation & Prerequisites . . . . .	59
B.1.2	Parameters . . . . .	60
<b>C</b>	<b>Source Code</b>	<b>62</b>
C.1	Code for population coding . . . . .	62

# Acronyms

**AI** Artificial Intelligence. 6, 10

**ANN** Artificial neural network. 6–8, 11, 13, 25, 28, 29, 32, 38

**EM** Expectation maximisation. 21, 24

**GMM** Gaussian mixture model. 21, 23, 24, 28, 29, 41, 47, 48

**MLP** Multilayer perceptron. 11, 15, 19, 33

**SVR** Support-vector regression. 19



# Chapter 1

## Introduction

### 1.1 Background and Motivation

With the emergence of new computing power and computing technologies, machine learning today is not the same machine learning of the past. Its combination with the availability of big data meant that organisations would utilise machine learning, to gain insights, which would help in acquiring an edge over competitors.

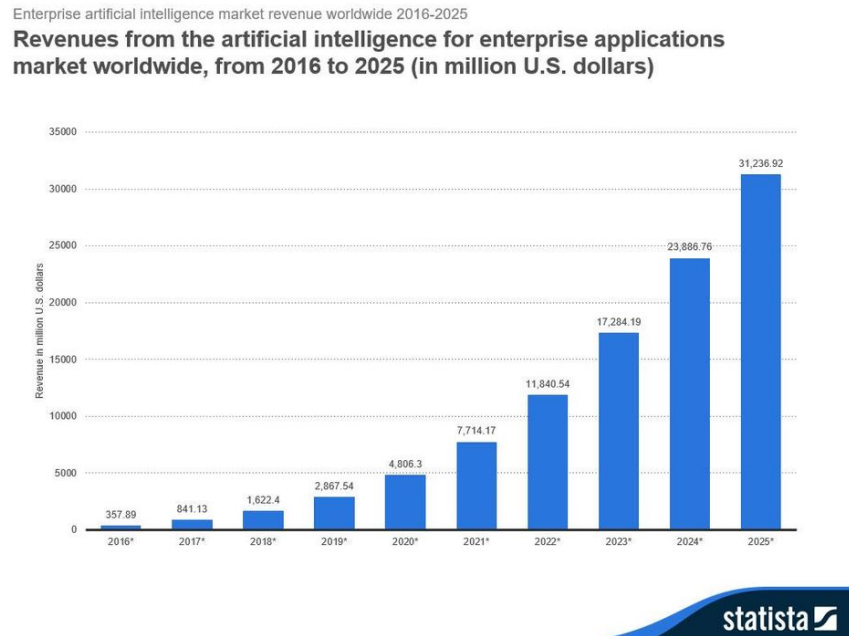


Figure 1.1: Previous and projected revenue from AI[1].

Today, machine learning is being involved across many industries, from financial services, where it is used to identify trends in data, helping investors know when to trade, or to prevent fraud, to transportation, where it is employed to analyse data, therefore, making routes more efficient and predicting potential problems. As can be seen from figure 1.1, Artificial Intelligence (AI) has been growing at an extreme pace, and this includes the many machine learning algorithms that are being developed regularly.

Supervised learning is the most common subbranch of practical machine learning. It involves making predictions on new unseen data depending on previous similar labelled training data. Supervised learning can be further divided into two subcategories, which are, classification and regression. This report mainly revolves around the regression subcategory. Practically, many different machine learning regression techniques are being adopted, where each of these techniques performs differently on varying datasets. An Artificial neural network (ANN) is often used as an important supervised learning technique to make accurate predictions.

ANNs are inspired by biological neural networks, where connections in the ANNs represent synapses in the brain, which transmit signals to other neurons. Another biological concept is population coding. “For many functions controlled by the brain or variables represented in the brain, the relevant unit is the neuronal population rather than the individual neuron”[2]. Population coding is a method that uses the combined behaviours of several neurons to interpret stimuli, where responses of multiple neurons to inputs can be joined to determine a value about the inputs.

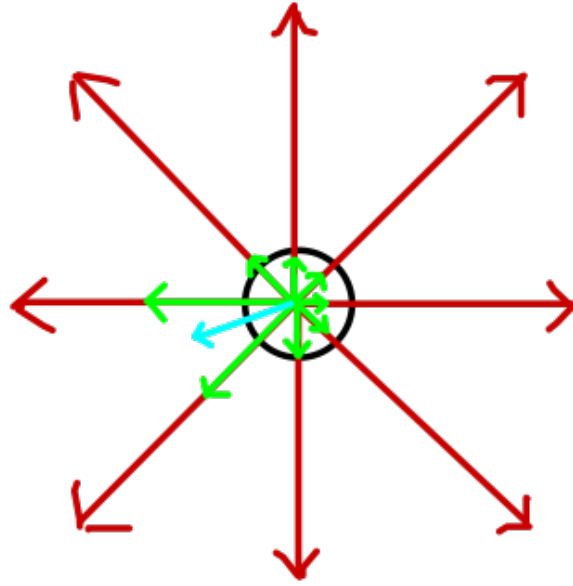


Figure 1.2: Neurons firing in the preferred directions[3].

Many tests of population coding are being conducted on animals. One test was carried out on Rhesus monkeys, where they were trained to press buttons that lit up[2]. Figure 1.2 demonstrates this test. The test aimed to predict the direction of arm movement using the neural responses of the populations of neurons. The red arrows represent the preferred directions of eight different neurons, while the green arrows portray the magnitudes in which each of the neurons are firing in response to a button lighting up. The blue line is the average of the entire population. The results suggest that the direction of movement is not dependant on single cells but rather by the neuronal ensemble. Population coding was shown to be one of the few well-formulated mathematical problems in neuroscience.

In traditional ANNs, neurons work individually, on the other hand, in an artificial population code, each value is represented by a small population of neurons. Neurons within this population have different preferred values, such that the population as a whole, can represent the entire range of possible values. The motivation behind this representation is that an ANN will be able to train better due to each value being a representation of the entire range of values rather than a single arbitrary input.

One of the aims of this project is to introduce the idea of population coding into ANNs. This can be achieved by transforming inputs and outputs of ANNs by coding them to form populations, which represent the original values of the inputs and outputs. The objective of this transformation is to enable an ANN to make better predictions based on the values of the population. Testing the accuracy of the predictions made by this algorithm against the accuracy of other regression techniques, is another point this project aims to conclude.

To be able to test the accuracy of the predictions made by the different approaches, suitable datasets with a varying number of features and sample size need to be chosen. For this, datasets from the Penn Machine Learning Benchmarks repository are used in benchmarking the approaches[4]. These datasets are suitable to examine against, considering that they have been utilised to benchmark many other regression techniques, with the results available for comparison [5]. In this report, three versions of population coding will be tested. The first of these would be coding the inputs only, second would be coding the outputs, and finally, coding both inputs and outputs. After the coding is applied, the data would be passed to a model to train and then make predictions. The data from the predictions from all versions will be analysed, and an attempt to describe the results will be made.

## 1.2 Report Structure

Chapter 1 summarises the background, motivation, and aims of the project.

Chapter 2 illustrates multiple literature reviews carried out on various machine learning techniques and their uses. Related work on benchmarking regression algorithms is also discussed.

A description of the architecture of both population coding types as well as the machine learning techniques employed is seen in chapter 3. The motivation behind these decisions is also explored.

The implementation of the design outlined in Chapter 3 and the tools used during implementation are mentioned in Chapter 4.

In chapter 5, an evaluation and an analysis are carried out on the results produced by population coding of inputs, population coding of outputs, and population coding of inputs and outputs on multiple datasets. The performance is compared to that of other regression algorithms benchmarked on the same datasets.

Chapter 6 discusses future extensions to this report. It also concludes the work achieved by this project.

## Chapter 2

# Literature Review

### 2.1 Machine Learning

Machine learning is computer programming to refine a success criterion using examples or past experience[6]. It is usually identified as a subset of AI. Machine learning algorithms are used in a wide range of applications, such as email filtering and computer vision, where the creation of a traditional algorithm to successfully execute the task is impossible or unfeasible. In machine learning there are three major recognised categories:

- **Supervised learning:** Task-oriented algorithms that work by feeding a learning algorithm a set of labelled training data[7]. Over time, The algorithm will learn to estimate the exact nature of the relationship between inputs and their labels. After the completion of training, the supervised learning algorithm will be able to make good predictions on new, unseen before inputs. Applications of supervised learning include speech recognition, spam detection and object-recognition.
- **Unsupervised learning:** These algorithms involve no labels. They operate by finding similarities in the data fed, and then aim to organise the data into clusters[8]. The organised data then can be made sense of by a human or another algorithm. One major advantage of using unsupervised learning is the fact that the majority of the data in the world is unlabelled, and therefore making the algorithm extremely useful and a large source of profit for many industries. Unsupervised learning is used in recommender systems and finding fraudulent transactions.
- **Reinforcement learning:** Algorithms that progressively get better by adapting to their

environment[9]. This functions by pairing positive behaviours performed by the algorithm, with positive rewards and poor actions, with poor rewards. This way, the algorithm learns to make fewer mistakes over time. Reinforcement learning is used in video games, traffic light control and autonomous cars.

Supervised learning is used in this project to train a model on labelled coded data, and then make predictions about new unlabelled data. This can be done using many different techniques. The technique used to apply population coding in this project is a Multilayer perceptron (MLP)[10]. An MLP is a class of ANN. On the other hand, unsupervised learning plays a crucial role in coding the inputs of the MLP. Reinforcement learning is not utilised in the research conducted by this project.

### 2.1.1 Supervised Learning

Supervised learning can be further divided into two types which are[6]:

- **Classification:** A task to identify to which label a new instance belongs, based on a training set of data containing multiple instances. Labels in classification have discrete values. The goal is to measure the accuracy of label predictions. There are two forms of classification, which are, binary classification, where the model predicts 0 or 1, and multi-class classification, where the model predicts from more than two classes.
- **Regression:** A task that is very similar to classification, except in regression, a continuous value is predicted. The goal is to make a prediction that is close as possible to the actual value. The accuracy is measured by calculating an error value between the predicted output and the actual output.

An extremely useful feature, is the ability to learn a function which maps an input to an output based on training on labelled data. Given this, supervised learning has some issues :

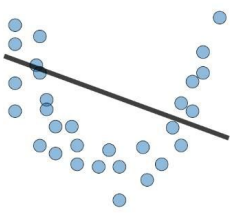
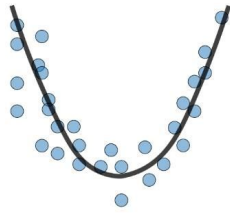
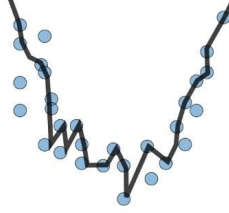
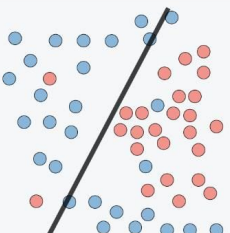
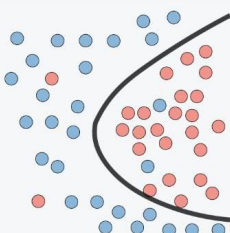
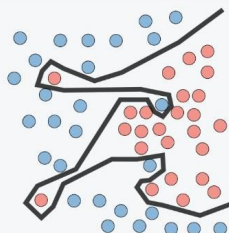
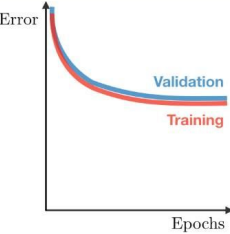
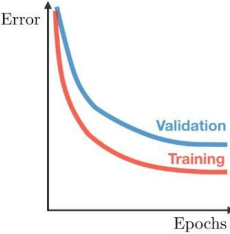
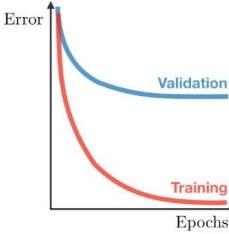
	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> <li>• High training error</li> <li>• Training error close to test error</li> <li>• High bias</li> </ul>	<ul style="list-style-type: none"> <li>• Training error slightly lower than test error</li> </ul>	<ul style="list-style-type: none"> <li>• Very low training error</li> <li>• Training error much lower than test error</li> <li>• High variance</li> </ul>
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"> <li>• Complexify model</li> <li>• Add more features</li> <li>• Train longer</li> </ul>		<ul style="list-style-type: none"> <li>• Perform regularization</li> <li>• Get more data</li> </ul>

Figure 2.1: Demonstration of the Bias-variance tradeoff.[11].

- Bias-variance trade-off[12]: Bias, is the assumptions made by a model to make the target function easier to learn, while variance is the amount the target function will change if new training data is used. A high bias would cause a high percentage error in prediction due to being less flexible. A high variance would cause a high percentage error, due to being highly flexible, causing the model to overfit the training data and therefore failing to generalise to new data. Usually, a trade-off between the two produces the best learning algorithm.
- Noise in the output values: Occasionally, errors occur in the training data, and if the learning algorithm has high variance, overfitting will occur and the target function will fit the noisy data, causing a higher percentage error when generalising. In practice, there are several ways of addressing this problem. One of them uses a complex algorithm to find and delete the noisy training instances[13].



In this report, classification is not discussed as population coding of inputs and outputs is expected to be useful on continuous values only. The reason behind this is that a population would usually take on multiple continuous values, but if it were to be applied in classification, then the population would consist of the classes of a feature or the outputs, and that is likely to be ineffective. This will be thoroughly explained in the next chapter. Regression techniques, on the other hand, will be further discussed below.

#### 2.1.1.1 MLP Regression

An ANN is an interconnected graph of nodes (neurons), that is roughly modelled on a biological neural network[14]. The goal of ANNs is to predict outputs based on previous training. Each neuron is connected to other nodes through links, where each of these links has a weight, which determines the influence of one neuron on another.

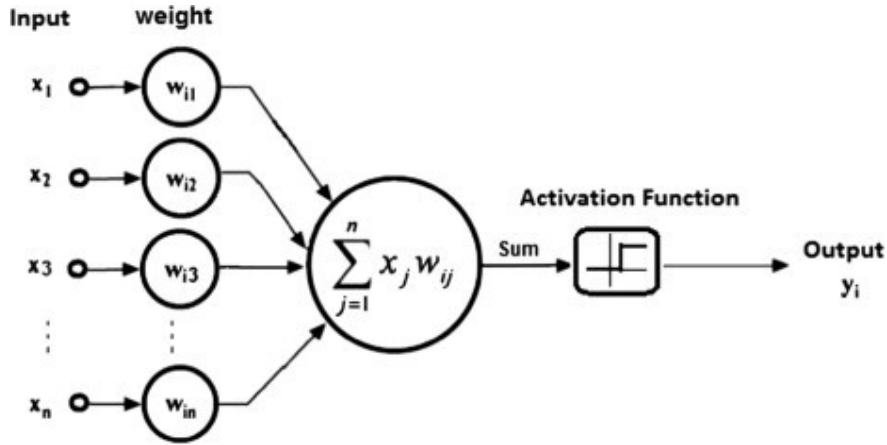


Figure 2.2: Perceptron[15].

Figure 2.2 shows the architecture of a perceptron (neuron), which is a single layer ANN. ANNs are usually composed of many perceptrons across multiple layers. Every perceptron has inputs that are connected to it through links called weights. These weights are adjusted through training to produce better predictions. In the neuron, the weighted sum of the multiplication of the weights by the inputs is calculated. The weighted sum is then passed to an activation function, which is a mathematical function, that takes an input or a set of inputs and produces a single output. Every perceptron has an input with a value of 1, which has a variable weight that is associated with it. This value is known as the bias. It is an essential part of the perceptron, as its value alone may change the output of the activation function. It is similar to the intercept/bias in the equation of a straight line.

Activation functions determine the outcome of a perceptron and its function. There are many different activation functions used in neural networks. Different activation functions are used in different layers depending on their functionality[16].

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

This equation is the sigmoid function[17], it produces outputs between 0 and 1. It reduces the computation capacity in training.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

The hyperbolic tangent function[18] is similar to the sigmoid function, though it outputs values between -1 and 1.

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

Rectified linear unit[19] outputs the input if it is greater than 0 and otherwise returns 0. It enables fast training of neural networks on large and complex datasets.

### **Feedforward Neural Networks**

A neural network where the connections between the nodes do not form a cycle is called a feedforward neural network[20]. In this network, there are multiple groups of perceptrons known as layers. The initial layer is known as the input layer, where the inputs of the problem enter the network. The final layer is known as the output layer, where the final prediction is made. The layers in between are known as hidden layers. With each extra layer, the model captures increasing complexity by discovering relationships between features in the input. Adding more layers might lead to overfitting or a decrease in the accuracy of predictions.

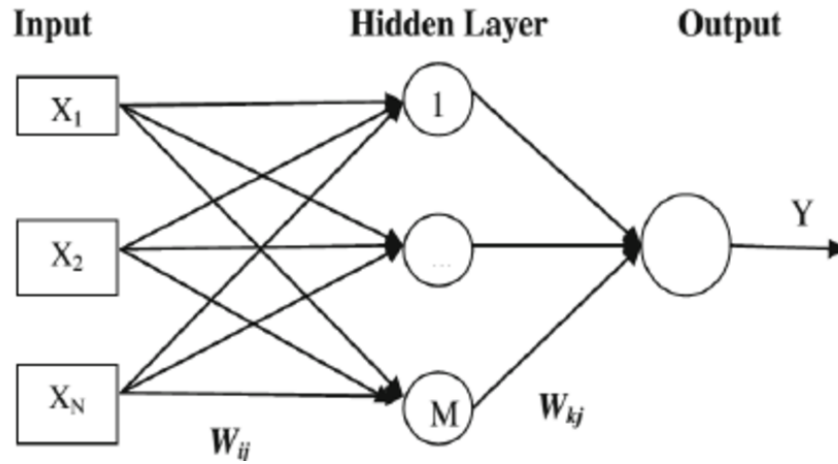


Figure 2.3: Multi-layer feed-forward neural network[21].

As can be seen from figure 2.3, every perceptron in each layer is connected to every perceptron in the consecutive layer. These connections represent the output from one perceptron becoming the input for the next.

### Multilayer Perceptron

An MLP is a class of feedforward neural networks. An MLP consists of at least three layers, an input layer, at least one hidden layer and an output layer. The advantage of using MLPs is that the hidden layers can be non-linear and therefore, the network can learn a non-linear function. An MLP regressor[22] implements a multilayer perceptron that uses backpropagation for training and has an output layer with no activation function.

### Learning

The adaptation of a network to produce better predictions is known as learning. It involves adjusting the weights to make predictions with higher accuracy. To measure the accuracy of predictions, the error between the predicted output of the network and the true output needs to be calculated. This is known as the loss function. There are multiple ways in which this error can be measured. In regression, one of the main loss functions used is the mean squared error function. After the errors are calculated, they can be utilised to train the network. An algorithm that is widely applied in training feedforward neural networks is known as backpropagation.

### Backpropagation

When training a neural network, backpropagation efficiently calculates the gradient of the loss function concerning network weights for a single input-output sample[23]. It does this by

calculating the gradient starting from the last layer and iterating to the first. This is known as the chain rule. It avoids the calculations of intermediate terms, making the entire algorithm more efficient. Backpropagation only refers to calculating the gradient and not how it is utilised. To update the weights, optimisers are used.

## Optimisers

The function of optimisers is to tweak the weights of a model depending on the loss function supplied. The most common optimisers are:

- Batch gradient descent: Commonly known as gradient descent, it updates weights by iteratively computing the gradient using the entire dataset[24]. These weights are modified to minimise the loss function. The extent to which the weights are updated is determined by using a learning rate. This will be discussed further below.
- Stochastic gradient descent: Updates the weights by selecting a random instance of the dataset to obtain the gradient from, rather than the entire dataset[25]. This results in a noisier gradient. The noisiness helps in avoiding the local minima of the loss function. Choosing one instance to obtain the gradient leads to faster iterations, but a slightly smaller convergence rate.
- Adaptive moment estimation: Adam uses gradients from the past to calculate the current gradients. It also applies the idea of momentum by adding fractions of prior gradients to the new one[26].

## Learning Rate

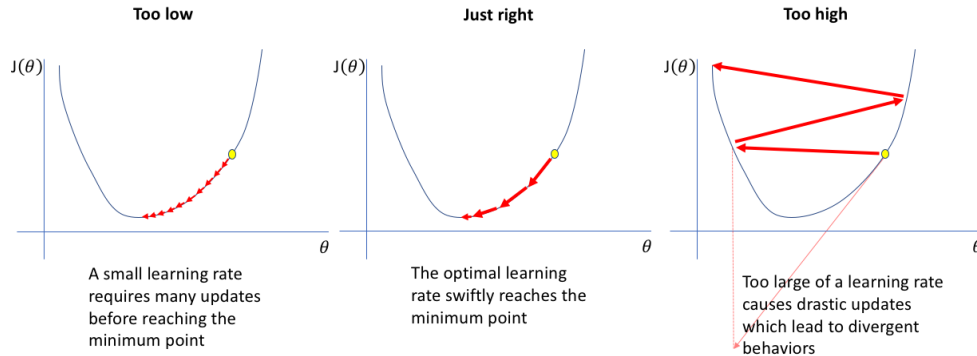


Figure 2.4: Effect of learning rate on loss function[27].

The learning rate is a parameter that is used in optimisers in many machine learning algorithms[28]. It is also applied in neural networks. It controls the rate at which a machine learning algorithm converges. A low learning rate usually leads to a slow convergence or getting stuck in a local minimum. A high learning rate could cause the optimiser to never reach a minimum due to overshooting it. Different regression algorithms use different forms of learning rate adjustment:

- Constant: The learning rate stays stable over time.
- Invscaling: The learning rate continually decreases over time.
- Adaptive: Depending on the model's results, the learning rate adapts. If the training loss continues to decrease, then the learning rate stays constant, otherwise, it decreases.

#### 2.1.1.2 Linear Regression

Linear regression is a popular statistics and machine learning algorithm[29]. It is a linear model that outputs a value  $y$  depending on a linear combination of the inputs  $x$ . The linear regression algorithm usually trains using the least squares technique. It is a technique that minimises the square of the difference between the predicted value and the actual output. There are multiple types of linear regression. Simple linear regression involves having one input variable to predict one output variable[30]. In this case, statistics can be used to calculate the coefficients (weights). This could include calculating the means, standard deviations, correlations and covariances. Then using these properties, a straight line is fitted through the points. Multiple linear regression involves having more than one input variable to predict one output variable[29]. Finally, multivariate linear regression predicts more than one output. In the case of the latter two forms of linear regression, gradient descent is used in training to minimise the error of the predictions.

#### 2.1.1.3 Kernel Ridge Regression

One problem with linear regression is it tends to overfit the training data. Regularisation is a technique to reduce overfitting. It penalises the weights in the loss function to be minimised. This leads to a simpler model, which does not overfit the data.

$$Loss = Error(h(x), y) + \lambda \sum_{i=1}^N w_i^2 \quad (2.1)$$

Equation 2.1 demonstrates the loss function with the  $\ell_2$  regularisation technique which is used in Ridge regression. The function is composed of two terms. The first is the error between the predictions made and the actual label. The second is the regularisation term[28].  $w$  represents the weights that are used when making predictions, and  $N$  is the number of existing weights. Machine learning algorithms aim to minimise this loss function which leads to more accurate predictions. To minimise the loss function, the value of the weights needs to be penalised due to the regularisation term being added to the error. This results in a simpler model that solves the problem of overfitting by not only depending on the error made in predictions.

$\lambda$  is a parameter that determines the degree to which the weights are penalised. A high value of  $\lambda$  results in underfitting due to the algorithm setting the weights' values close to 0 when minimising the loss function. A low value leads to overfitting again due to the weights not being penalised enough. In practice,  $\lambda$  is a hyper-parameter, and its value is determined by trying different values and selecting the one that produces the best outcome.

Kernel Ridge regression is Ridge regression with the addition of the kernel trick[31]. A kernel is a function that maps data points into a higher dimensional space to help the algorithm learn a better linear function.

#### 2.1.1.4 Least-angle Regression with Lasso

Similar to Ridge regression, Lasso adds a regularisation term to the loss function to reduce overfitting[32]. Lasso also applies feature selection to remove redundant features.

$$Loss = Error(h(x), y) + \lambda \sum_{i=1}^N |w_i| \quad (2.2)$$

Equation 2.2 is the loss function with  $\ell_1$  regularisation, which is used in Lasso. In  $\ell_1$  regularisation, the absolute values of the weights are added to the loss function instead of the squared values which are used in  $\ell_2$ . Both forms are meant to penalise the weights to make the model simpler, but when using the absolute values, the weights' are pushed towards 0 when updating them individually. This leads to redundant features having a coefficient close to 0, which is also known as feature selection.

Least-angle regression with Lasso is an algorithm made to fit linear regression models to high dimensional data[33]. It functions by providing an estimate of which features to include and their coefficients. It adds features one at a time starting with the feature that is most correlated to the target output. After a feature is added, its coefficient is increased in the direction of the sign of its correlation with the target until another feature has an equal correlation. From this point onward, both coefficients are increased in a direction which is equiangular between the features until another feature has an equal correlation. This is carried on until all features are added. This algorithm is numerically efficient when the number of features is considerably larger than the number of samples but is highly sensitive to noise in the dataset.

#### **2.1.1.5 Linear Support Vector Regression**

A support vector machine works by constructing a hyperplane in a high-dimensional space which can be used for classification or regression[34]. In simple regression, the aim is to minimise the error in the loss function, while in support-vector regression (SVR), a hyperplane is fit through the data after specifying a maximum margin for error,  $\epsilon$ .  $\epsilon$  can be tuned to improve the accuracy of the model. In some cases, the algorithm will find it difficult to learn a suitable linear function in a certain dimension of space, so mappings of data points to a higher dimension would be desirable. In SVR, the kernel trick is used to achieve this.

SVR is similar to kernel ridge regression, where the form of the model learnt by both is identical, and both use  $\ell_2$  regularisation[35]. However, they use different loss functions: kernel ridge regression uses squared error loss while SVR uses  $\epsilon$ -insensitive loss. On medium-sized datasets, training is much faster, using kernel ridge, while predictions are faster in SVR.

#### **2.1.1.6 Stochastic Gradient Descent Regression**

As with MLPs, stochastic gradient descent can be applied to linear regression. A stochastic gradient descent regressor can utilise  $\ell_2$  regularisation to avoid overfitting. The algorithm is particularly useful on large datasets with more than 10000 instances[36], and it remains very efficient due to the selection of one instance to calculate the gradient from when updating the weights.

### **2.1.1.7 Random Forest Regression**

Random forest regression is a learning algorithm that ensembles multiple decision trees at training time, and then produces a mean prediction of all the decision trees when producing outputs[37]. Ensembling is the use of multiple learning algorithms together to improve predictions[38]. A decision tree is a learning algorithm that functions by making splits of the data depending on the available features. Features that make the best homogeneous split, tend to be used earlier in the tree. In a decision tree regressor, the leaves of the tree are continuous values that represent the outcome of the prediction. Random forests use multiple trees and retrieve the mean value of all the predictions. The feature bagging method is used to render different trees. Feature bagging works by randomly selecting a subset of the original set of features available, to split the data. This subset is now used to make the decision tree. Due to this randomness, many different trees can be made, each producing different predictions. This results in better generalisation when making the predictions[37].

### **2.1.1.8 AdaBoost Regression**

Converting multiple weak learners into strong learners is a process known as boosting[39]. Adaboost improves on areas where the base learner fails. The base learner is a weak learner that a boosting method is applied on to modify it to a strong learner. In the case of Adaboost, multiple small decision trees or stumps (decision trees composed of only one level) are used as weak learners. The main differences between Adaboost and random forest are:

- In Adaboost, not all stumps are valued equally when making the final decision on the output, while in a random forest, the mean prediction of all the trees is used.
- The errors made by each stump influence the next stump to be made when using Adaboost, while each tree is independant of the others in a random forest.

Stumps that produce the best predictions are the ones that influence the final prediction the most. New stumps that are generated, focus on fixing the errors of previous stumps that made good predictions. This leads to an incremental improvement in performance.

### **2.1.1.9 Gradient Boosting & Extreme Gradient Boosting Regression**

Similar to AdaBoost, gradient boosting uses an ensemble of weak prediction models to make a stronger one[40]. Typically the prediction models would be decision trees. Unlike Adaboost, gradient boosting starts with a single leaf as its tree, which would be equal to the average of all



the training labels. Each tree after the first one has a fixed size, which is usually larger than a stump, normally having between 8 to 32 leaves. Before generating every new tree, the residuals between the previous tree's prediction and the actual labels are calculated. These residuals are then utilised in future trees to generate their leaves. To control overfitting, each tree is scaled equally using a learning rate, so that its contribution to the next tree is a small step in the right direction of the actual label. Training stops when the number of trees reaches the maximum specified, or the predictions stop varying by a certain threshold. Predictions then can be made by adding the original average to the scaled output of each tree. Extreme gradient boosting adds the concept of regularization to control overfitting[41]. It is considered a state-of-the-art method in machine learning.

### **2.1.2 Unsupervised Learning**

Clustering is a useful task that is used in many fields[8]. There are two main varieties of clustering, which are soft clustering and hard clustering. In hard clustering, elements can either belong to a cluster or not, while in soft clustering, each element belongs to each cluster to a certain degree. There are many algorithms that solve the clustering task. This project is particularly interested in achieving soft clustering by using a Gaussian mixture model (GMM).

#### **2.1.2.1 Gaussian Mixture Models**

In a GMM, each cluster represents a probability distribution which is a Gaussian. Each probability distribution has its own parameters which are the mean and the covariance. In order to find the clusters, GMMs use the Expectation maximisation (EM) algorithm[42].

#### **2.1.2.2 Expectation Maximisation**

The EM algorithm starts by creating  $K$  (specified by the user) clusters, by assigning a random mean and covariance to each cluster. It then measures the probabilities of each point (sample) belonging to each of the clusters. After these probabilities are calculated, the EM algorithm readjusts the mean and covariance of each cluster to fit the points more adequately. This is done over several iterations. By the end of the last iteration, there will be  $K$  clusters with each point being assigned to the cluster it most likely belongs to.

## 2.2 Related Works

The idea of implementing or testing population coding has not been explored before, but the benchmarking of regression algorithms on datasets has been discussed in many papers. Orzechowski et al. have benchmarked recent symbolic regression approaches as well as machine learning approaches from scikit-learn on multiple datasets from open-source repositories[5][43]. These datasets come from the Penn Machine Learning Benchmark (PMLB) repository[4]. The repository contains a collection of standardised datasets for classification and regression problems. The datasets used to benchmark the regression approaches are of various sizes, therefore they are very useful to test the performance of different approaches.

In this project, population coding will be benchmarked on the same datasets and the accuracy of the predictions will be compared to that of the scikit-learn machine learning approaches[44].

# Chapter 3

## Design

Population coding involves the use of both supervised and unsupervised learning techniques, and in the following section these techniques will be discussed in further detail.

### 3.1 Population Coding of Inputs

Population coding of inputs revolves around the idea of generating more useful inputs (features) to a neural network from the original set. This is achieved by creating new features from every original feature. Therefore, every scalar (single number) value in the dataset will be transformed to be represented by multiple new values, which are known as the population.

To be able to code an input, the first step is to calculate two essential parameters, which are, the Gaussian centres of the feature, of which the input belongs to, and the standard deviation of each Gaussian mixture of that feature. This entails that these two parameters should be calculated for each feature in the dataset. The next step is to apply the coding function to the entire set of inputs. This causes a growth in the number of features in the dataset, which further leads to a change in the architecture of the neural network to be used. Finally, predictions are made by the neural network and the mean squared error is calculated.

#### 3.1.1 Calculating Gaussian Centres and Standard Deviations

To determine the values of these parameters, a univariate GMM is fitted through each of the features in the dataset. This will result in the generation of multiple Gaussian mixtures (clusters) spread throughout each feature, which together, include all the points of the dataset.

The Gaussian centres of each feature will be the mean of each cluster produced by the EM algorithm. The standard deviation will be the square root of the clusters' variances.

This technique was chosen because a GMM will be able to locate areas with a high concentration of points automatically. This is important as these points will be a good representation of the entire range of values occupied in each feature. One problem with this approach is determining the number of required clusters for each feature before fitting the GMM. This is an issue, as an excessive amount of clusters results in clusters with a few data points. These are redundant because they give little information about the entire range of values. A sparse amount of clusters results in a population that is not complex enough to be a good representation of the range of values of the feature. This will become more evident in the results chapter later in this report. One approach attempted to solve this problem is to remove a centre if it is close to another. The intuition is that this should eliminate redundant centres.

### 3.1.2 Coding Function

The coding function takes in three parameters. The first is the scalar value to be coded by the function. This value represents a data point belonging to a feature from the dataset. The second and third parameters are the Gaussian centres and standard deviations of that feature.

The generation of the new values from the scalar input is based on the similarity between the input and each of the Gaussian centres. If the value is close to a Gaussian centre's value, then the output will be closer to 1, otherwise, if there is a big difference in value, then the output will be closer to 0. The number of values generated from the scalar input is equal to the number of centres passed to the function.

$$e^{-\frac{(x-gc)^2}{2\sigma^2}} \quad (3.1)$$

The equation 3.1 is a Gaussian function that demonstrates how this is achieved[45]. The parameter  $x$  represents the scalar value to be coded.  $gc$  and  $\sigma$  represent the vector of Gaussian centres and the vector of standard deviations respectively. Since  $gc$  is a vector of values, the subtraction of  $x$  from the vector yields a new vector with the same number of values as  $gc$ . The function outputs the new set of values which replace the original input.

### 3.1.3 Coding the Dataset

Before inputs can be coded by the function, the dataset is split into training data and testing data (to measure performance). The Gaussian centres and standard deviations of the features are calculated using the training data only because while training, the neural network should not know anything about the data to be tested on. After calculating the two parameters, all scalar inputs are coded by the function, which causes the manipulation of the entire dataset. The number of samples will remain unchanged, but the number of features will increase by a factor of the number of Gaussian centres.

### 3.1.4 Training the Neural Network and Making Predictions

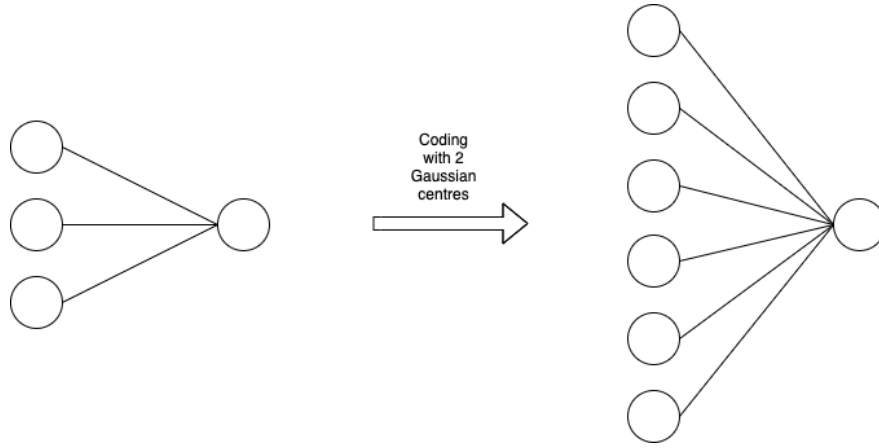


Figure 3.1: Result of coding inputs of a sample.

Figure 3.1 demonstrates the change to a simple neural network with two layers after coding the inputs with two Gaussian centres. A typical ANN used in this project would contain hidden layers, but for demonstration purposes, only two layers were used. Each original input gets replaced with two new ones. The idea behind this approach is that the neural network will have a better idea of which centre each scalar value is close to, and therefore, if close to one centre, then far away from the other centres. The intuition is that the neural network will be able to adjust the weights more adequately by having this extra information.

After the network is trained, the coded testing data is passed to the network to make predictions. The mean squared error between the predictions and the actual labels are measured. The ambition is for the algorithm implemented to have a smaller error than that if no coding was employed.

## 3.2 Population Coding of Outputs

Population coding of outputs is very similar to the idea of input coding. The only major difference is that the coding occurs on the labels instead of the features. It increases the number of labels for each sample in the training dataset, therefore forming a population of outputs. This population will be used to train the network. When making predictions, the network will predict a population of outputs. These outputs will then be decoded by a decoding function to then be compared against the labels of the testing dataset.

To code an output, the same two parameters for coding an input are required. However, the Gaussian centres and the standard deviation is calculated differently for coding an output.

### 3.2.1 Calculating Gaussian Centres and the Standard Deviation

A separate approach is taken from that of input coding to calculate the Gaussian centres for output coding. The first step is to find the range of the values for the labels in the training data. The next step is to decide on the number of required centres. Finally, Gaussian centres are placed at intervals starting from the label with the minimum value and ending with the label with the maximum value. The centres are placed at equal intervals and the number of intervals is equal to the number of centres chosen. Only one standard deviation is calculated, and that is of all the labels in the training set. The reason for the change in calculating the parameters is the decoding function. This will be further explained in the decoding function section.

### 3.2.2 Coding Function & Coding the Dataset

The coding function is the same coding function used for input coding in equation 3.1. The only difference is that the standard deviation passed to the function is a scalar value instead of a vector. Only the training labels are passed to the function to be coded. Every label is replaced with  $K$  new labels, where  $K$  is the number of Gaussian centres used. These new labels are then passed to the neural network for training.

### 3.2.3 Training the Neural Network and Making Predictions

Output coding leads to an increase in the number of outputs to be predicted by the network. The network is trained to predict a value close to 1 for an output if its Gaussian centre was

close in value to the original label's value and a value close to 0 otherwise.

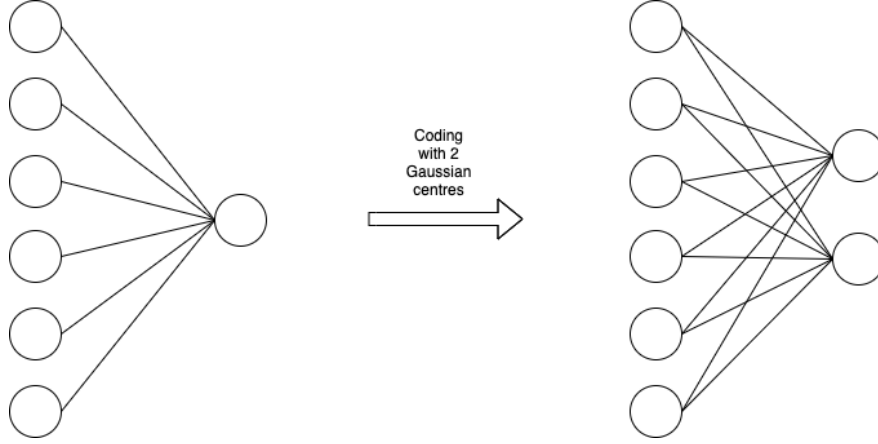


Figure 3.2: Result of coding labels of a sample.

Figure 3.2 demonstrates coding the output of a simple network using two Gaussian centres. The result shows that the network now makes two predictions instead of one. The reasoning behind this is that making predictions of how close a value is to a Gaussian centre can be more accurate than just making one prediction with the help of decoding.

The next step would be to decode the predictions made by the network and calculate the mean squared error between the decoded predictions and the labels.

### 3.2.4 Decoding Function

The function takes in two parameters, the Gaussian centres used to code the outputs of the training set and the population to be decoded by the function.

$$\frac{\sum_{i=1}^n P_i * gc_i}{\sum_{i=1}^n P_i} \quad (3.2)$$

Equation 3.2 illustrates how the decoding function operates.  $P$  represents the population to be decoded.  $gc$  denotes the Gaussian centres used to code the labels of the training set and  $n$  is the number of elements in the population.  $P_i$  refers to the  $i$ th element in the population vector. The function decodes the population into a single value using the Gaussian centres. The dot product between the centres and the population is calculated and then divided by the sum of the population. This leads to the population being decoded into one value that represents a prediction. The expectation is that this prediction would be more accurate than the prediction

that would have been made with no population coding.

The reason why a GMM is unsuitable when locating Gaussian centres for output coding is that it calculates multiple standard deviations to be used in the coding function. This causes a large amount of error when values are decoded. Unfortunately, the alternative is to use this interval-based approach to calculate the Gaussian centres, even though it may lead to worse predictions.

### **3.3 Population Coding of Inputs and Outputs**

Population coding of inputs and outputs combines the ideas of the two techniques. Both inputs and outputs are coded using separate parameters. The data is then passed to the ANN and the predictions made are decoded. Finally, the mean squared error between the prediction and the test label is calculated. The aim is that the combination of the two techniques will lead to predictions with higher accuracy.



## Chapter 4

# Implementation

### 4.1 Tools

Several tools can be used to carry out this experiment. The first selection to be made is the programming language. Python was chosen because of the availability of a range of libraries and frameworks that are built to make the task of data processing and machine learning easier.

To build and train ANNs, GMMs and other machine learning techniques the scikit-learn library is used[44]. The reason behind this is that the scikit-learn library already has implementations of these approaches. Another reason is that previous benchmarks have used this library, so this will maintain fair testing. Pandas and NumPy are used for data manipulation.

The experiments were ran using a Jupyter notebook on Google Colab[46]. Google’s cloud TPU was also used to boost performance.

### 4.2 Data & Results

One of the essential parts of this experiment is to choose suitable datasets to test the performance of the different approaches. The Penn machine learning benchmark repository supplies datasets of different sizes[4]. Some of these datasets have few features with many samples, while others have many features with fewer samples. The largest dataset has 1000 samples with 100 features. For some experiments, these datasets could be considered small, but for this one, they are sufficient to test any improvements on predictions. Another reason these datasets

are suitable is the limited computing power, as larger datasets require more computation to produce predictions.

Results for non-population coding regression approaches were obtained from another benchmark study of regression approaches, which was made available on the paper’s Github repository[5]. These approaches were also tested while conducting this experiment to confirm that the results were accurate.

### 4.3 Data manipulation

Some necessary steps need to be completed before population coding can be applied. The first of them is to standardise the data. This causes each column in the data (feature or label) to have a mean of 0 and a standard deviation of 1. Standardisation is a general requirement for many machine learning algorithms, especially the algorithms that use gradient descent for updating weights, as standardising results in faster convergence[47]. This is mainly due to the data being scaled to have similar ranges of values. The StandardScaler scikit-learn library is used to accomplish this.

### 4.4 Calculating Gaussian Centres and Standard Deviation for Input Coding

The inputs are split for training and testing, and only the training set is used to obtain the two parameters for both sets of data. Calculations are made on each feature one at a time. This means that all the data points for each feature are obtained, which leads to multiple one-dimensional vectors being collected. A Gaussian mixture from scikit-learn is fitted through each of these vectors with the number of clusters being specified beforehand. If the number of unique values in a vector is less than the number of chosen clusters, then the number of clusters used is equal to the number of unique values in the vector. The means and variances of each of these vectors are variables of the Gaussian mixture. The means represent the Gaussian centres, and the square root of the variances represent the standard deviations. There is one change left to complete before the parameters can be passed to the algorithm, and that is to remove centres that are close in value. If any two centres are closer than a tenth of the range, one of the centres is removed. A tenth was chosen because it is small enough to not remove

important centres and big enough to remove redundant ones. The parameters are then passed to be coded.

## 4.5 Calculating Gaussian Centres and Standard Deviation for Output Coding

This is implemented as explained in the design chapter. The number of Gaussian centres is chosen. The range of the values of the outputs is calculated using NumPy. The minimum value is also calculated using NumPy. The Gaussian centres are then placed at equal intervals starting from the minimum value and ending at the maximum value with as many intervals as centres. The standard deviation is calculated using the NumPy std function, which takes all the labels in the training set as a parameter and then returns their standard deviation.

As discussed in the design chapter, this interval-based approach was chosen to calculate the parameters because of the amount of error produced when decoding values coded with multiple standard deviations. The main intent behind placing Gaussian centres at intervals is that some of these intervals might be placed at positions where the concentrations of the data points are, and therefore after coding a value, the model would have a good intuition of which centres the original value was closer to. Similar to input coding, outputs are split for training and testing, and the parameters are calculated using the training set only.

## 4.6 Population coding of Inputs and Outputs

The coding function from equation 3.1 is implemented using NumPy functions. One extra thing checked in implementation is the value of the standard deviation. If it is smaller than  $10^{-5}$ , it is replaced with  $10^{-5}$ . This is a safety measure to avoid dividing by 0. Another reason is that a standard deviation smaller than this value will make little to no change to the results: the output will most definitely be a 0 for every centre unless the value to be coded lies exactly on one of the centres then the output for that centre will be 1. Both the training and testing sets are then coded using this coding function. Every scalar value in both sets is coded using the Gaussian centres and standard deviations of the feature it belongs to or using the ones of the output values depending on where the scalar value comes from in the dataset.

## 4.7 Decoding

This is conducted before the mean squared error is calculated. The Gaussian centres of the outputs are passed alongside the predictions to a function. Each prediction made by the model is then decoded using the equation 3.2. This is achieved by using functions from NumPy.

## 4.8 Choosing Hyperparameters

Before an ANN can start learning, the hyperparameters need to be chosen. These include the activation function, optimiser and learning rate. To achieve this, GridSearchCV is used from scikit-learn. It takes an estimator as a parameter, which is the regression algorithm to be used. In the case of this project, that is the MLPRegressor from scikit-learn. It then takes a list of hyperparameters. It starts an exhaustive search by trying all the possible combinations of the hyperparameters. The best combination is then used to test the accuracy of the regression approach.

Each hyperparameter has multiple options. The options tested in this project are:

- Activation function: Logistic, tanh, relu.
- Optimiser: Lbfgs, adam, stochastic gradient descent.
- Learning rate: Constant, invscaling and adaptive.

One way to have a variable number of Gaussian centres depending on the dataset used is to set it as a hyperparameter to be optimised. The reason this option was not chosen in this project is that this would cause a massive increase in the training time. Population coding of inputs approximately increases the size of the dataset by the number of Gaussian centres chosen, which results in the MLPRegressor taking a significantly larger time to train, therefore having it optimised for each dataset would be unreasonable considering the amount of running time it requires.

Only one hidden layer was used in this experiment. This was chosen to maintain fair testing between the results obtained using population coding and the results obtained from the benchmarking paper[5]. The maximum number of epochs was set to 200, but training would be stopped if there was a loss smaller than  $10^{-4}$  in 10 consecutive epochs. Upon testing on the larger datasets, convergence was always achieved.

## 4.9 Training and Predictions of MLP

The MLPRegressor from scikit-learn is trained on the data and then makes predictions. The results from the grid search are passed as parameters to the model. The MLP has 1 hidden layer with 100 neurons by default. After training is complete, predictions on the testing dataset are made, and the mean squared error is calculated. Other less significant tests are also taken, such as testing the predictions on the training set and calculating the mean absolute error. In the case of output coding, decoding needs to be performed before the mean squared error can be calculated.

### 4.10 Accuracy Measure

For each dataset, the whole process is executed 10 times, and the median mean squared error for each dataset is determined. The median is the chosen measurement taken by the benchmark of other regression algorithms[5]. These tests were measured on population coding of inputs, population coding of outputs and population coding of inputs and outputs.

## Chapter 5

# Results/Evaluation

In this chapter, the results of running the three different variations of the population coding algorithm on many datasets are evaluated. The main focus is to compare the accuracy of the predictions made by a scikit-learn MLPRegressor with and without population coding. Most of the tests are carried out on the same 94 datasets from the Penn Machine Learning Benchmark repository[4]. These datasets have at most 1000 samples and 100 features. Larger datasets would take a large amount of time with the computing power available. Testing predictions for all 94 datasets once takes approximately 12 hours using Google Colab. For this reason, extensive testing is not possible.

## 5.1 Population Coding of Inputs

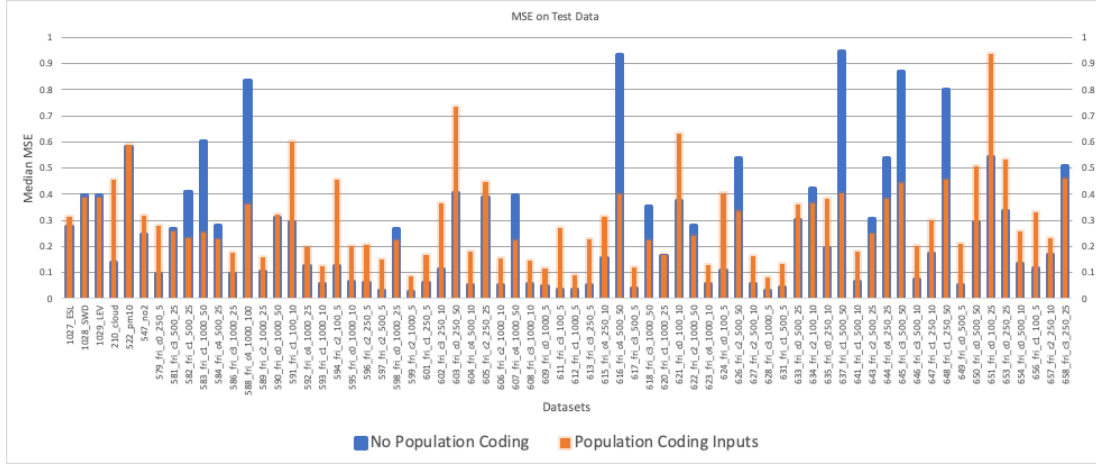


Figure 5.1: Results of population coding of inputs compared with no population coding. Note: zooming in might be required to view the dataset names. Most datasets end with two sets of numbers representing the number of samples and features. For example, ‘612\_fri\_c1\_1000\_5’ has 1000 samples and 5 features. Datasets that had a mean squared error higher than 2 were removed from this figure for scaling purposes.

The regression algorithm is executed on each dataset 10 times. The median mean squared error is then calculated. With population coding of inputs on the 94 datasets, the MLPRegressor made better predictions on 27 of the datasets, while performing worse on the rest of the datasets.

### 5.1.1 Small Datasets ( $\leq 250$ samples $\leq 10$ features)

On these datasets, population coding of inputs performed the worst. The predictions made were sometimes twice as worse compared to running the regressor without coding. Probable explanations for this could include the lack of a large amount of data, therefore the mixture model choosing Gaussian centres that are unsuitable for coding. Another explanation could be the large margin for error due to the availability of a smaller set of data for testing.

### 5.1.2 Medium Datasets ( $\leq 500$ samples $\leq 25$ features)

The performance was varied across these datasets. In occasions, population coding led to much more accurate predictions, while in others, it caused worse predictions. Due to the variability in performance across similar datasets, it is difficult to analyse the reason behind the worse predictions.

### 5.1.3 Large Datasets ( $\geq 500$ samples $> 25$ features)

In almost all of these datasets, population coding led to better predictions. In some cases, the mean squared error halved when using population coding compared to not using it. It seems like the increase in the number of features and samples leads to better results when coding. That might be due to the increased training set size.

### 5.1.4 Other Datasets

Practically, the datasets that were considered large during this benchmark are not regarded as large today. So to measure the performance of input coding on larger datasets, further tests were conducted on two relatively larger datasets. These datasets come from the same repository that was used to get the previous datasets[4]. The first dataset had 6435 samples and 37 features. The mean squared error scored with no coding was 0.5399. After coding, it dropped to 0.5090. On the other hand, the second dataset had 1054 samples with 118 features. The mean squared error with no coding was 0.2225, while it increased to 0.4552 with it. This is difficult to analyse as different datasets with the same sizes are producing different results. One of the differences between these two datasets was the values. The data in the second dataset was more continuous, and that might have played a role in the results.

### 5.1.5 Effects of the Number of Features and Samples

These two properties play a big role in the accuracy of predictions when using input coding. General trends that were observed include:

- When looking at the results of different datasets with the same number of features but a different number of samples, there was a significant improvement in performance when increasing the number of samples. This improvement also occurs when no population coding is used but to a much smaller extent. This can be accredited to having more data points, therefore having more clusters with a high concentration of data points.
- On the other hand, datasets with the same number of samples but a varying number of features had a lower mean squared error when having fewer features with and without population coding, but when a high number of features was available, population coding led to considerably better predictions.



### 5.1.6 Number of Gaussian Centres

Studying how the number of Gaussian centres selected affects the accuracy of predictions might be one way to understand the effect of coding. Unfortunately, testing this on all 94 datasets is not feasible due to the limitations in time and computing power. 5 datasets of various sizes are selected to test the difference caused to predictions by changing the number of clusters.

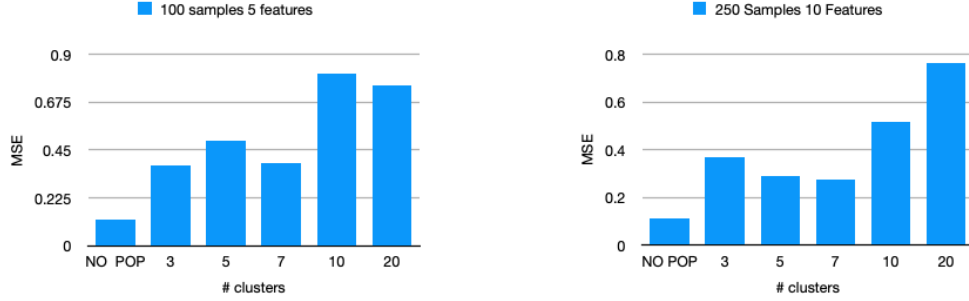


Figure 5.2: Results of varying the number of clusters for input coding on small datasets.

Figure 5.2 demonstrates the effect of varying the number of clusters used by the Gaussian mixture on small datasets. From the results, having a smaller number of Gaussian centres leads to better predictions (smaller mean squared error). A reasonable explanation for this could be the lack of many samples, therefore the existence of only a few areas where there is a concentration of data points within each feature. With this reasoning, the use of many clusters leads to the creation of redundant centres with few points in them. The many centres might lead the model to make faulty predictions on new data because of the similarity of the new data point to one of the redundant centres.

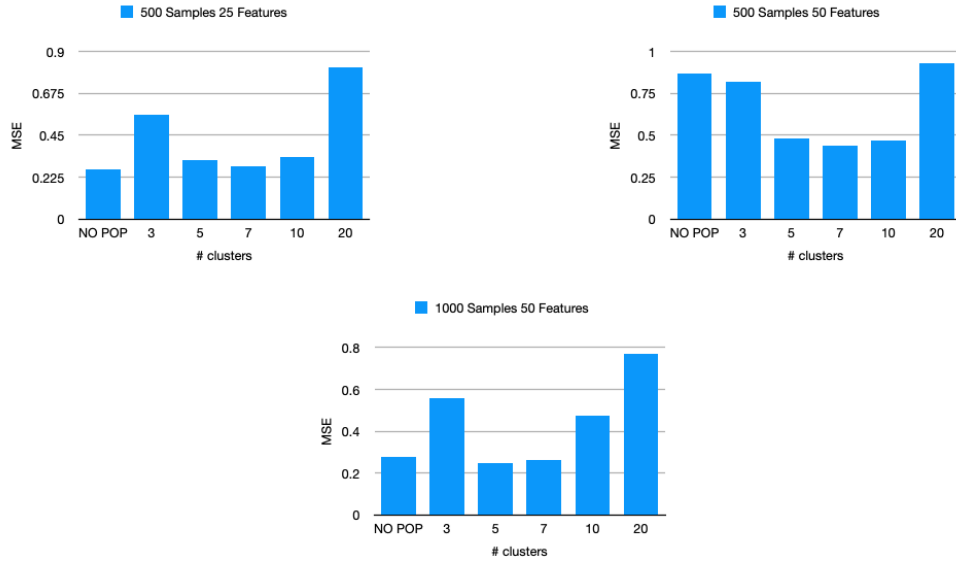


Figure 5.3: Results of varying the number of clusters for input coding on larger datasets.

From figure 5.3 it can be seen that in larger datasets too few and too many Gaussian centres, point to worse performance. Using a small number of clusters might not be enough to find all the regions in which the data points are concentrated within each feature. Having too many centres would have the same effect as explained above for smaller datasets.

These results are conducted on 5 out of the 94 datasets only and may not be a good representative of how changing the number of Gaussian centres may affect the rest of the datasets. However, these datasets were chosen to represent the rest of the datasets based on their sizes, as datasets with similar sizes produced similar results when the input coding test was conducted on all 94 datasets.

### 5.1.7 Running Time

Population coding is reasonably slower than running an MLPRegressor with no coding. Coding the dataset is not a slow process, but its growth in size (number of features) causes a much larger neural network leading to slower processing. The growth of the dataset is proportionate to the number of Gaussian centres used to code it. For example, if two Gaussian centres were used in input coding, then the dataset will double in size (excluding the labels), leading to the input layer of the ANN to have double the number of neurons. This is an important factor in choosing the number of Gaussian centres to use for input coding.

## 5.2 Population Coding of Outputs

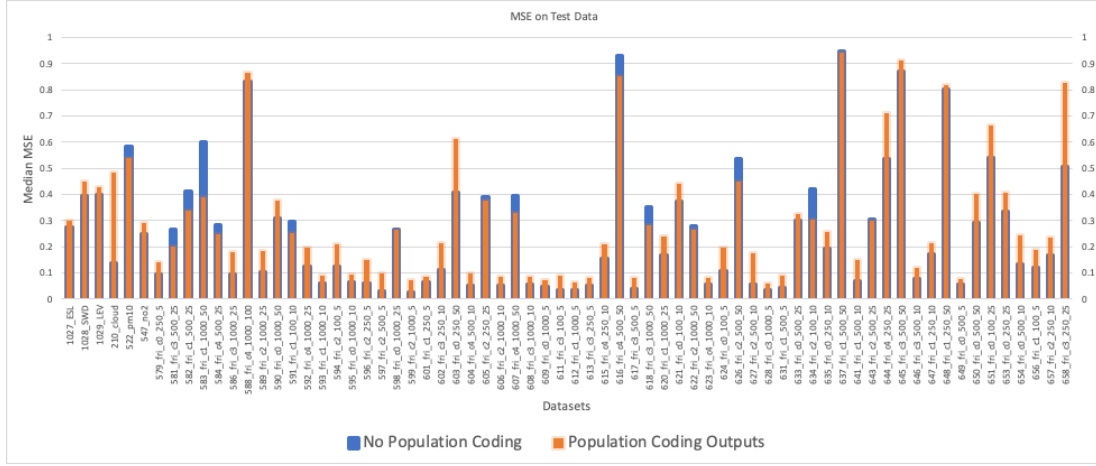


Figure 5.4: Results of population coding of outputs compared with no population coding.

Output coding resulted in improved predictions on 20 of the datasets. Compared to input coding, output coding produced some interesting results. When input coding improves on a prediction, the difference in the mean squared error is much more noticeable compared to output coding. Similarly, when input coding results in worse predictions, the difference in the error is also bigger than the difference caused by output coding. Another interesting behaviour is that the 20 datasets are of various sizes, however the majority are from the larger ones.

### 5.2.1 Small Datasets ( $\leq 250$ samples $\leq 10$ features)

Again population coding suffered dealing with the smaller datasets. However, population coding of outputs achieved more success with these datasets compared to input coding. Similar to the case of input coding, this can be attributed to not having a large amount of data, therefore clusters not having a high concentration of data points (reundant clusters).

### 5.2.2 Larger Datasets ( $\geq 500$ samples $\geq 25$ features)

These datasets have again shown to be the ones that received the best results from population coding. This could again be due to the increased training set size.

### 5.2.3 Other Datasets

The same two larger datasets tested for input coding are tested again for output coding. The mean squared error for output coding on the first dataset was 1.7239, while before coding it

was 0.5399. On the second dataset, it was 0.3035, and with no coding, it was 0.2225. While testing on these datasets 10 Gaussian centres were used as this number of centres produced good results on the other datasets. This could have been one of the reasons behind the worse predictions.

#### 5.2.4 Effects of the Number of Features and Samples

- Similar to population coding of inputs, a larger number of samples while having the same number of features results in predictions with a lower mean squared error. However, the error in output coding is considerably smaller than that of input coding when there were fewer samples. Output coding involves coding only one value, while input coding codes every input, and this might be the reason behind the difference in results.
- Fewer features led to smaller mean squared errors in population coding of outputs compared to input coding. However, for most of the cases, the lack of population coding had better results when fewer features were available. When there were more features, output coding had worse predictions than input coding, but better predictions than if population coding was not used.

#### 5.2.5 Number of Gaussian Centres

Once more, to be able to determine why population coding of outputs behaves this way, tests of a different number of Gaussian centres are completed. The same 5 datasets tested in input coding are tested again in hope to see how changing the number of centres affects predictions in different datasets.

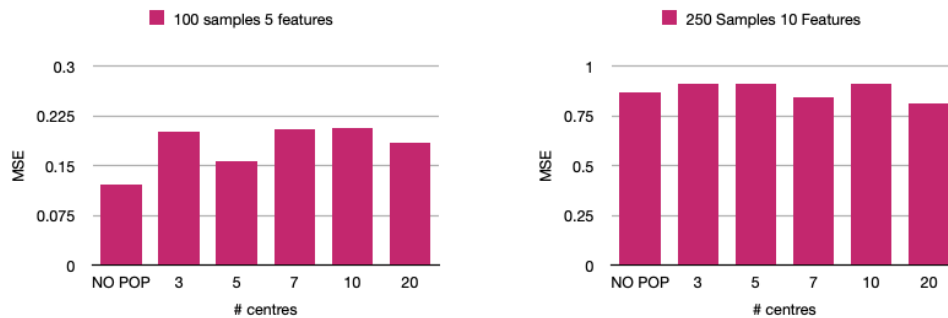


Figure 5.5: Results of varying the number of centres for output coding on small datasets.

Figure 5.5 depicts how the number of Gaussian centres in output coding affects predictions on smaller datasets. It seems like the number of Gaussian centres, in this case, plays a little

effect in predictions. The decoding function can be one of the reasons behind this behaviour.

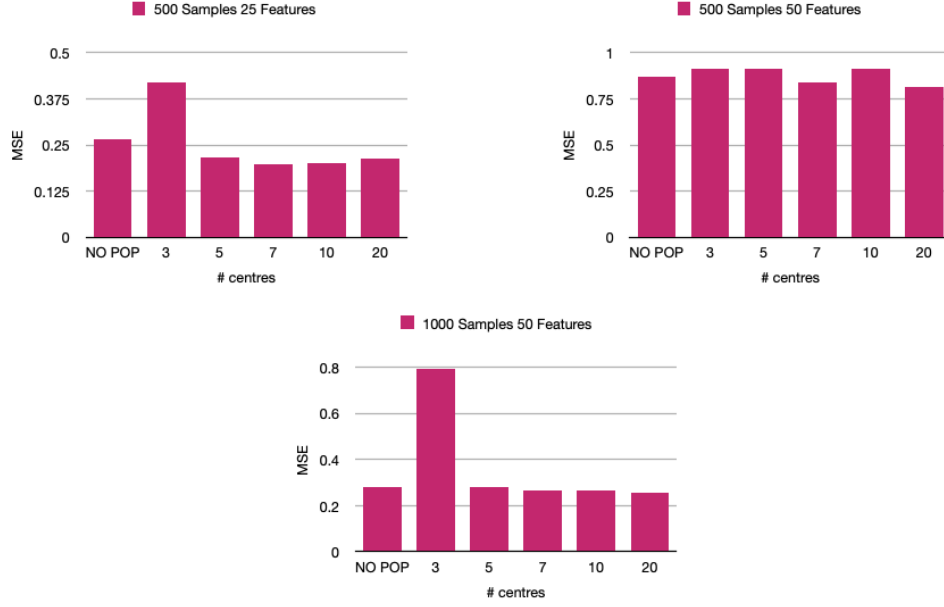


Figure 5.6: Results of varying the number of centres for output coding on larger datasets.

In the case of larger datasets, tests from figure 5.6 show that the predictions drop in accuracy only when the number of centres chosen is too small for the dataset. In output coding, the Gaussian centres are placed at equal intervals rather than using a GMM to find the centres, so if there are only a few, these may be a bad representation of where the concentrations of labels are in the training set. This will lead to faulty training due to most points being far away from these centres, and therefore be the cause of poor predictions on the testing set. The results indicate similar accuracies in the case of using more centres, this can again be attributed to the use of the decoding function.

### 5.2.6 Decoding

Decoding could be one of the reasons why predictions are not as desirable as they are supposed to be. Upon testing, if a value is coded with a standard deviation that is too high, then the value will be represented by new values (population) close to 1 for many of the centres. So if the decoding function was to decode the new values back to the original value, there would be a large margin for error due to the decoding function not knowing to which centres the original value was closest, as it predicts that most centres are close. On the other hand, if the standard deviation was too low, then the new values will be close to 0 for most of the centres. So when decoding, the function will again not know which centres were closest, because all seem far

away. Due to this complication, when decoding predictions, the value obtained might be far away from the actual label.

### 5.2.7 Running Time

Compared to input coding, output coding hardly increases the total amount of running time. The explanation behind this is that only the outputs are coded, while all the features are coded in input coding, and typically there are far more features in data sets than outputs. Nevertheless, the higher the number of Gaussian centres chosen, the higher the amount of running time, as more predictions are required.

## 5.3 Population Coding of Inputs and Outputs

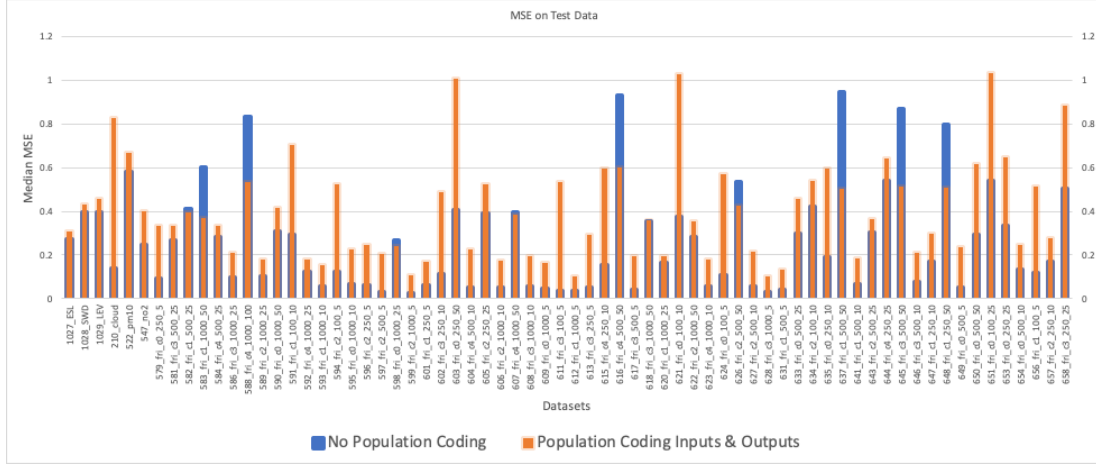


Figure 5.7: Results of population coding of inputs and outputs compared with no population coding.

The combination of the two forms of population coding yielded unexpected results. The results were worse than the results of either form used alone. Only in 11 of the 94 datasets, there was an improvement in performance compared to not using coding with an MLPRegressor. All 11 datasets were a subset of the 27 datasets that input coding showed better results in. Another interesting observation that can be seen is when input coding was employed by itself, the mean squared error achieved for these 11 datasets was lower than when output coding was added.

When benchmarking the smaller datasets using this approach, tests showed that no population coding resulted in a lower mean squared error for each of the datasets. In the case of the

larger datasets, results were more promising as all 11 of the datasets that saw improvements were part of these larger datasets. The reasons for the worse performance can be due to the flaws of either input coding or output coding or both approaches.

### 5.3.1 Other Datasets

The two larger datasets were used again to test the performance of this approach. On the first dataset, the mean squared error achieved was 1.708 compared to 0.5399 with no coding. On the second, 0.4907, while without coding 0.2225.

## 5.4 Dataset Analysis

Before coding, all the columns in the datasets would have a mean of approximately 0 and a standard deviation of 1. This is the effect of standardising the datasets. In some cases, this affects decoding. Some manual tests of decoding were carried out on values obtained from the dataset, and in certain instances, the percentage error made by decoding reached up to 30%. This may play a major factor behind the reason for worse predictions when using output coding.

## 5.5 Evaluation & Other Algorithms

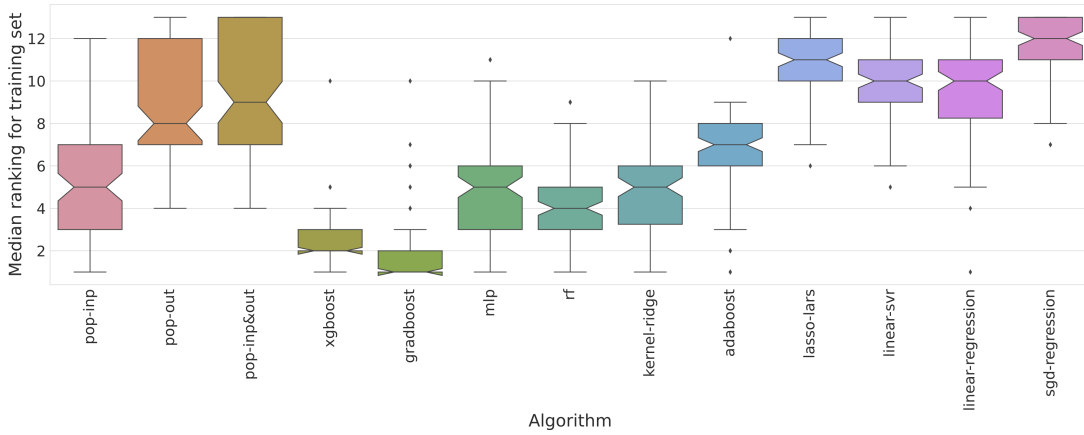


Figure 5.8: Ranking of the performance of the algorithms based on the MSE score on training datasets[5].

Figure 5.8 demonstrates the median ranking of the three types of population coding and ten other machine learning regression algorithms on the performance on training data from the 94 datasets[5]. Population coding of inputs results in performance that is very similar to that if

no coding was used (MLP). On the other hand, population coding of outputs and population coding of both, inputs and outputs, fit the training data worse. This could be one reason why predictions are less accurate when using these two forms of population coding.

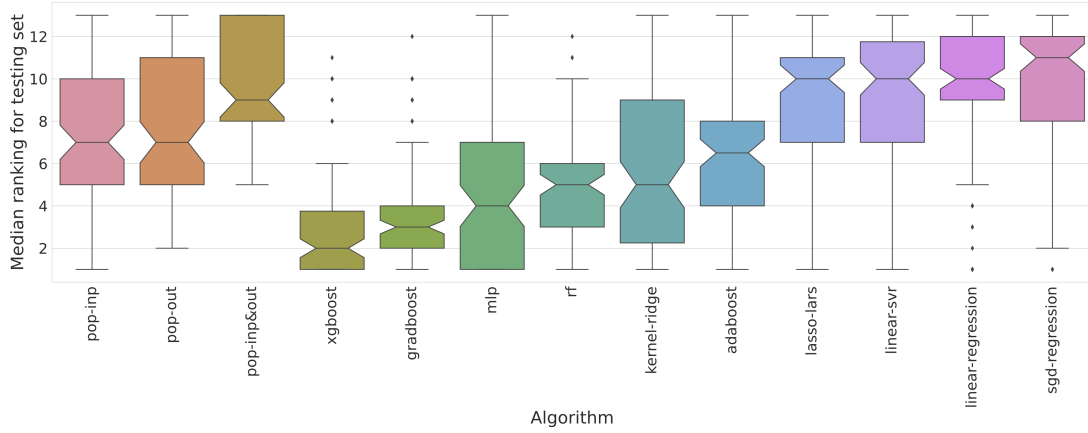


Figure 5.9: Ranking of the performance of the algorithms based on the MSE score on testing datasets[5].

Figure 5.9 shows the performance of population coding and the other regression algorithms on the testing data. The median ranking of the algorithms depending on the aggregated results achieved on the test sets is demonstrated by figure 5.9. Out of all the algorithms, XGBoost ranked first followed by gradient boosting and then MLP. Population coding of inputs produced results that were very desirable on larger datasets, but on the smaller datasets it performed significantly worse than the regular MLP and therefore would have a worse ranking. On the other hand, population coding of outputs performed better than the MLP on most larger datasets, but performed worse on the smaller datasets. However, output coding performed better than input coding on these smaller datasets. From figure 5.9 it can be seen that both input and output coding have a median ranking of seventh. Unfortunately, when both techniques were combined, the performance dropped considerably, which led to the ranking being worse than either technique used individually.



Algorithm	Total Running Time (Hours)	Median Running Time (Seconds)
No Population Coding	6.2	16.8
Population Coding of Inputs	17.8	52.3
Population Coding of Outputs	9.0	26.7
Population Coding of Inputs & Outputs	18.3	52.6

Table 5.1: Running time of the population coding approaches on the 94 datasets. Note: the running time for no population coding was obtained from [5] but results remain consistent.

Another major factor to consider when evaluating machine learning algorithms is the running times of the algorithms. XGBoost, gradient boosting, and MLP are all considered as slow for machine learning algorithms. From table 5.1 it can be seen that input coding causes a massive increase in the running time of the algorithm due to the large increase in the size of the dataset. The increase in running time is relative to the number of Gaussian centres used in coding. The more centres used, the larger the dataset becomes and the more time is needed for training the model. When producing the results for this project, 7 Gaussian centres were used for input coding. In some cases, the running time was three times slower compared to not coding when producing results. Output coding also causes an increase in the running time, but one that is not as significant as input coding. This is due to the increase in the number of outputs to be predicted by the model. The running time in this case also varies with the number of Gaussian centres chosen. Applying both techniques together would certainly cause a further increase in the running time compared to using the techniques individually. If population coding were to be applied in a real-world application, the decision of the number of Gaussian centres to be used would be dependant on the importance of efficiency versus accuracy.

Overall, the performances of both forms of population coding are very promising. If the performance of input coding is consistent on even larger datasets, then that can be utilised in many machine learning regression applications in various industries. That also applies to output coding, as even though the results it produces on larger datasets is a smaller improvement than that of input coding, it is considerably faster.

## Chapter 6

# Legal, Social, Ethical and Professional Issues

While completing the project, great care was taken to comply with the Code of Conduct & Code of Good Practice provided by the British Computer Society[48]. All libraries and code used from third parties are open-source and allowed to be used commercially. Google Colab[46] was the chosen IDE to obtain the results from. It provides a free membership with limited computing power which was utilised throughout the implementation of the project. The datasets used in benchmarking the population coding algorithms are publicly available online, are allowed to be used commercially, and do not contain private information about any particular individual[4]. Third parties were given full credit for all code and work that was used in this project that was not my own.

## Chapter 7

# Conclusion and Future Work

This project uses a wide range of techniques including, using a clustering algorithm to find Gaussian centres, applying a Gaussian function in population coding to find the degree of similarity between a value and a centre, and employing a neural network to train a model and then make predictions, therefore it has a wide range of topics to choose from to be further explored. Some of these possibilities will be discussed in the next section.

### 7.1 Future Work

One part of this report that could have been explored further was testing on datasets of different sizes, especially much larger datasets. This was not possible due to the limitation of computing power and time resources. With these extra tests, the performance of both forms of population coding could have been further analysed. This would have also given a better indication of whether using more Gaussian centres on larger datasets would be beneficial or just costly. This is one of the main areas that can be expanded on beyond this project.

Further research can be conducted to find other coding and decoding functions that can be used to replace the current functions utilised. One problem that arose with applying the decoding function discussed in this project was the margin of error when decoding. Another problem with using the current decoding function was the enforcement of interval-based Gaussian centres rather than centres calculated using a clustering algorithm while output coding. In all likelihood, output coding using a GMM would have led to more accurate predictions. If these issues could be improved by applying different functions, then the results from output

coding can notably improve. The performance of these other functions can be compared against each other and then analysed.

In this project, unsupervised learning was used in the form of GMMs, but many different clustering algorithms can be used to obtain the Gaussian centres. Scikit-learn provides the implementations of many different clustering algorithms, such as K-means[49], Mean-shift[50], and DBScan[51], to name a few. These algorithms can be used to find Gaussian centres, and subsequently, the results of the predictions can be compared and analysed to better understand the relationship between the Gaussian centres used for coding and the predictions made by the model.

One interesting area to be studied in population coding is the number of Gaussian centres used in coding a value. The current approach practised is to set a fixed number of centres to be used to code the entire set of data. The first point that could be further analysed is the relationship between the number of Gaussian centres chosen and the characteristics of the datasets they are to be used on. A deeper understanding of this will lead to better predictions. Another interesting solution that can be tested after improving this understanding is employing a variable number of Gaussian centres that would be optimal when used in coding each feature. The current solution used is to remove centres that are close in value, but having a variable amount of centres depending on the feature would be a much more welcomed solution as that will most definitely lead to better predictions.

## 7.2 Conclusion

Similar to neural networks, artificial population coding was inspired by population codes which exist in the human brain. Biologically, it is suggested that populations of neurons determine certain actions rather than single cells. Fortunately, when applying this concept in machine learning, this inspiration yielded great results. In today's world, innovation is demanded by all industries, especially in the rapidly growing fields of AI and Data Science. Population coding is a fresh and original approach of data manipulation designed to improve the performance of traditional artificial neural networks. One fascinating aspect of population coding that can be found while looking at the tests is the significant difference in predictions caused by feature engineering. Input and output coding both result in a unique set of model predictions. It will be very interesting to see how this can be further expanded on to optimise the results from

population coding.

# References

- [1] Louis Columbus. Charts that will change your perspective on artificial intelligence’s growth, 10.
- [2] Apostolos P Georgopoulos, Andrew B Schwartz, and Ronald E Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.
- [3] Charles Frye. *What is population coding? Describe the population coding model proposed by Georgopoulos in the 1980s for M1 control of arm direction.*
- [4] Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):36, 2017.
- [5] Patryk Orzechowski, William La Cava, and Jason H Moore. Where are we now?: a large benchmark study of recent symbolic regression methods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1183–1190. ACM, 2018.
- [6] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.
- [7] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 2002.
- [8] Victor Roman. *Unsupervised machine learning: Clustering analysis*, 2019.
- [9] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.
- [11] Amide, Shervine and Amidi, Afshine. *Bias/variance Tradeoff*. <https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricksclassification-metrics>.

- [12] Ron Kohavi, David H Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–83, 1996.
- [13] Carla E Brodley, Mark A Friedl, et al. Identifying and eliminating mislabeled training instances. In *Proceedings of the National Conference on Artificial Intelligence*, pages 799–805, 1996.
- [14] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.
- [15] Medium. Artificial neuron networks(basics) — introduction to neural networks, 2017. URL `\url{https://becominghuman.ai/artificial-neuron-networks-basics-introduction-to-neural-networks-3082f1dcca8c}`.
- [16] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.
- [17] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.
- [18] *Hyberbolic Tangent Function*. [https://en.wikipedia.org/wiki/Hyperbolic\\_functions](https://en.wikipedia.org/wiki/Hyperbolic_functions).
- [19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [20] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [21] Ali Can Yilmaz, Cigdem Inan Aci, and Kadir Aydin. Mffnn and grnn models for prediction of energy equivalent speed values of involvements in traffic accidents.
- [22] *Scikit MLPRegressor*. [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPRegressor.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html).
- [23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [24] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251:254, 2012.

- [25] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes*, 91(8):12, 1991.
- [26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [27] Jeremy Jordan. *Setting the learning rate of your neural network*.
- [28] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [29] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.
- [30] Howard J Seltman. Experimental design and analysis. *Online at: <http://www.stat.cmu.edu/~hseltman/309/Book/Book.pdf>*, 2012.
- [31] Max Welling. Kernel ridge regression. *Max Welling’s Classnotes in Machine Learning*, pages 1–3, 2013.
- [32] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- [33] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.
- [34] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [35] *Scikit Kernel ridge regression*. [https://scikit-learn.org/stable/modules/kernel\\_ridge.html](https://scikit-learn.org/stable/modules/kernel_ridge.html).
- [36] *Scikit learn SGD*. <https://scikit-learn.org/stable/modules/sgd.html>.
- [37] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.
- [38] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.
- [39] hackernoon.com. *Boosting Algorithms: AdaBoost, Gradient Boosting and XGBoost*. <https://hackernoon.com/boosting-algorithms-adaboost-gradient-boosting-and-xgboost-f74991cad38c>.



- [40] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [41] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- [42] Guoshen Yu, Guillermo Sapiro, and Stéphane Mallat. Solving inverse problems with piecewise linear estimators: From gaussian mixture models to structured sparsity. *IEEE Transactions on Image Processing*, 21(5):2481–2499, 2011.
- [43] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12 (Oct):2825–2830, 2011.
- [44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [45] *Wikipedia Gaussian function*. [https://en.wikipedia.org/wiki/Gaussian\\_function](https://en.wikipedia.org/wiki/Gaussian_function).
- [46] Ekaba Bisong. Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 59–64. Springer, 2019.
- [47] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [48] *Code of Conduct & Code of Good Practice*. <https://cdn.bcs.org/bcs-org-media/2211/bcs-code-of-conduct.pdf>.
- [49] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28 (2):129–137, 1982.
- [50] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.
- [51] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.