UAB
Universitat Autònoma de Barcelona

Facultat de ciències

Departament de Matemàtiques

Limiting Behaviour of Neural Networks and Feature Selection

Jennifer Fabà Moreno

Treball de Final de Grau

Supervised by: Dr. Ana Alejandra Cabaña Nigro

Call: September 2024

Acknowledgments

I would like to thank my supervisor, Dra. Ana Alejandra Cabaña Nigro, for her guidance and support during the writing of this work. Her valuable knowledge and experience in the field of Statistics has let me discover an entire range of methods for analysing data, not only for the development of this project but also for other applications to the real world. In particular, she has provided me with the necessary tools for writing the theoretical basis of this work, such as for example Section 2.3. She also invited me to take part in the Seminars she gave at UAB and I have really appreciated these opportunities.

On the other hand, I would like to thank my mum, my sister Mireia and my friends for their constant support and love. Especially, I would like to thank Adolfo Hilario for his contributions to the structure of the work and for his constant inspiration and motivation and Angel Lorenzo for his thorough help along the whole degree.

Abstract

Neural networks constitute a state of the art approach to tackle the big amount of available data that humans have. They are used in a wide range of fields, including computer vision, natural language processing, health care, finance and beyond. Neural networks can be considered as approximating function algorithms; thus scientists focus on their predictions. However, they are so close related with the concept “black box”, since we know little about why they choose such predictions.

In this work we are going to relate them with another statistical approach that is well known, Gaussian processes. They are characterised by their mean and covariance function. This way, we reduce a little the above mentioned black box concept and make the process of modelling our data more reliable. Moreover, model selection helps in understanding which are the most important features of the data, which let us reduce the required input information. We propose the use of the Shap tool and a test proposed by Morgan and Pittman in 1939 to compare nested models. After performing all these analyses in house prices of Boston, we conclude that all features in the original data are not essential for further experiments.

Contents

1 Introduction

Who has never wondered how our brain works? How the information flows between neurons (the fundamental units of the brain and nervous system) and how this information arrives to our muscles and organs? Neurons are the cells responsible for receiving sensory inputs from the external world, for sending motor commands to our muscles, and for transforming and relaying the electrical signals at every step in between. We can make an analogy between neurons and trees. A neuron has three main parts: dendrites, an axon, and a cell body or soma, which can be compared to the branches, roots and trunk of a tree, respectively. A dendrite is where a neuron receives input from other cells. The axon (tree roots) is the output structure of the neuron; when a neuron wants to communicate with another neuron, it sends an electrical message through the entire axon. The soma is where the nucleus lies, where the neuron's DNA is housed.

Having said that and considering that this is a mathematical research project, one can see the analogy between neural networks and brain cells. Neural networks (NN), inspired by the brain's neural architecture, are computational models designed to process information. They consist of layers of interconnected nodes, or neurons, which process input data through activation functions to produce outputs.

The field of neural networks has a rich history, dating back to the 1940's. The model for a neuron proposed by Warren McCulloch, a neuroscientist, and Walter Pitts, a logician, was the first attempt to mathematically formalise the behaviour of a neuron and to study its utilities in computation and information processing [1]. The McCulloch-Pitts neuron is a computation unit which tries to simulate the behaviour of a "natural" unit, that is, the kind that constitute our brains. Although their main purpose was to develop a theory explaining the inner workings of the human mind, McCulloch and Pitts laid the foundations of what is known today as Deep Learning.

Since their inception, neural networks have evolved significantly and their applications have spanned across various domains including computer vision, natural language processing, health care, finance and beyond. With ongoing technological advancements, neural networks are continuously being enhanced and tailored to address increasingly complex tasks and datasets. Moreover, different types of neural networks have been developed such as fully connected feed-forward neural networks (FCNN), convolutional neural networks (CNN) or recurrent neural networks (RNN). They are commonly used for tasks like classification, regression and pattern recognition (FCNN); image classification and object detection (CNN); and speech recognition (RNN).

Like every other tool in science, neural networks are in continuous progress. Although the basis of their architectures might be established, the main problem with current neural networks is the little explainability they provide about their predictions. In the wide range of fields where they can be used, a rational answer is always desirable either when concluding that a patient has a disease or whether there has been a new discovery in a planetary system.

A FCNN consists of a set of nodes (neurons) which are distributed in layers where each node of one layer is connected to all the nodes of the next one. Moreover, in this type of neural network, the information only flows in one direction, that is, it doesn't form cycles but it only moves forward. It is natural to consider which is the behaviour of a neural networks when the number of neurons of each layer becomes large. Does that it mean that our network is more reliable? Could the computational work required be greater than the benefit it provides? One of the pioneering scientists who wondered about the limiting behaviour of neural networks was Radford M. Neal, who in his doctoral thesis explored the properties of neural networks as they become infinitely wide. He demonstrated a certain convergence to Gaussian processes (GP), which was a significant contribution because it relates neural networks with established statistical methods.

One of the main objectives of this work is to address this relationship between random, wide FCNN (with more than one hidden layer) and Gaussian processes with a recursive kernel definition. As GP are sequence of random variables in which every finite linear combination of them is normally distributed, we can completely characterise it by giving its mean and covariance structure. Thus, by computing the covariance matrix associated with the network, we can obtain the GP linked to it.

On the other hand, we want to soften the "black box" concept related to neural networks regarding the little explainability of their predictions. By introducing the concept of Shapley values we can assess the importance of each feature of the input data in the predictions. This will help us to understand which are the most relevant characteristics of input data, thus providing a useful tool for future predictions.

To develop our strategy, we structure this work as follows. In section 2 we provide the reader with the necessary theoretical concepts to understand the objective of the experiments held. In subsection 2.1 we give a brief introduction to neural networks and in subsection 2.2, we introduce some valuable definitions of Gaussian processes to relate them with neural networks. In subsection 2.3 we state the necessary theorems for proving that neural network and GP approaches are equivalent in the limit of infinite nodes in a layer. In subsection 2.4 we compute the expression of the covariance function of the GP related to the NN. In section 3 we introduce the tools for performing the final model selection. A further line of research is suggested in section 4. Next, in section 5 we present our experimental set-up: introducing the input data and the tests applied to it. Finally, in section 6 we present the results and discuss them: comparing the models selected and their predictions with respect to the neural network ones, to decide whether the limiting behaviour can satisfactorily substitute the real and more complex one. section 7 is dedicated to a summary of the work.

2 Neural networks and Gaussian processes relationship

2.1 A general introduction to neural networks. Fully-connected neural networks.

The general idea of neural networks is to find a model that, based on a set of N samples $\{(x_1, ..., x_N), (y_1, ..., y_N)\}$, will approximate a unknown function f, with $f(x_i) = y_i$ as accurately as possible.

Schematically, neural networks can be seen as a finite set of neurons[1] which can be connected with its neighbours as well as with themselves. The information flows inside these connections, therefore the *input* (first) layer receives the information which will go through the network in the "hidden" layers until the *output* (final) layer. In this layer, we obtain the output predictions subtracted from our initial information. We see how the term *hidden* supports the "black box" concept related to neural networks and their predictions.

The dimension of the input layer is determined by the dimension of the input data, while the dimension of the output is related to the problem we want to solve using the model. For example, if we use the model for regression, the output layer will be of dimension one, because we want the model to assign one value to each input; for a classification task, we use an output layer of dimension k, where k represents the different categories we want to sort the data in.

The training of a neural network consists in optimising f_θ in the function space $\mathcal{F}$ with respect to a functional cost $C : \mathcal{F} \to \mathbb{R}$, such as regression cost. However, before training the neural network we should divide the initial dataset into 3 subsets corresponding to training, validation and test sets. In *supervised* learning methods, each set is divided in input and output data (labels to which the network will compare its predictions). On the other hand, in *unsupervised* methods, the sets are unlabelled data which allows the network to discover patterns and insights without any explicit guidance or instruction. In both cases, the test set is the only that will not be used in the training process but rather will serve to assess how well the network generalises to new data. The training and validation sets will be used as input data: the first one is used to train the model (adjusting the parameters of f_θ) and the second one aids in model selection and hyper-parameter tuning.

Note that the terms parameters and hyper-parameters do not refer to the same thing, whereas the former are random at initialisation, learned during the training process and modified in each iteration, the hyper ones are set prior to training and are not learned from the data. These hyper-parameters affect the behaviour and performance of the model, but they are not updated during the training process. Some of the hyper-parameters that we can modify are:

- Loss function: function that the network will try to minimise at each step of the training process, for example, the $\mathcal{L}^2$-norm or the euclidean norm.
- Learning rate: controls how much the model changes in response to the estimated error each time the model weights are updated.
- Batch size. A batch is a subset of the entire training dataset. The batch size, therefore, refers to the number of data points that the model is exposed to in one iteration of training.
- Number of epochs. An epoch defines the number of times that the learning algorithm will work through the entire training dataset.

Neural networks can be classified in feed-forward or recurrent, according to the kind of connections that exists between the nodes. Feed-forward neural networks make the information flow in one direction, while recurrent ones let the architecture to have loops and allows the output information from some nodes to affect nodes of previous layers.

[1] In this work, we will interchange the words neuron and node on equal footing

In this work, we are going to deepen in the feed-forward type, particularly, in fully-connected neural networks (Figure 2.1). Thus, let us introduce a formal definition.

Definition 1: Characterisation of fully connected feed-forward neural networks

Let the inputs of the network be $\mathbf{x} \in \mathbb{R}^M$. The initial step is described by:

$$f_i^{(1)}(\mathbf{x}) = \sum_{j=1}^{M} \omega_{i,j}^{(1)} x_j + b_i^{(1)}, \tag{2.1}$$

where the subscript i ranges from 1 to H_1 (H_μ denotes the number of nodes of the layer μ). For a network with D hidden layers ($\mu = 1, \dots, D$), hence $f^{(D+1)}(\mathbf{x}) \in \mathbb{R}^L$ corresponds to its output value, the recursion is:

$$g_i^{(\mu)}(\mathbf{x}) = \phi(f_i^{(\mu)}(\mathbf{x})), \tag{2.2}$$

$$f_i^{(\mu+1)} = \sum_{j=1}^{H_\mu} \omega_{i,j}^{(\mu+1)} g_j^{(\mu)}(\mathbf{x}) + b_i^{(\mu+1)}. \tag{2.3}$$

In this case, i ranges from 1 to H_μ in Equation 2.2 and from 1 to $H_{\mu+1}$ in Equation 2.3 (in the case of the final activation the top value is L).

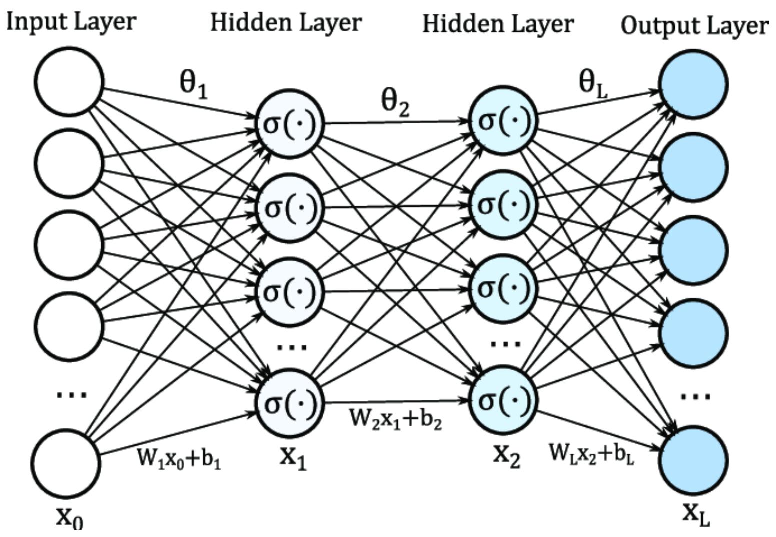

Figure 2.1: Fully connected neural network architecture. σ represents the activation function. Image extracted from: [2]

As one can see in Figure 2.1, each connection between nodes is associated with a weight and each node with a bias, so for each neuron in layer μ we have $H_{\mu-1} + 1$ parameters to be trained (by convention, we set $H_0 = M$).

To study the behaviour of this network as the widths of each layer becomes large, we will assume that, conditional on the inputs, the weights $\omega_{i,j}^{(\mu)}$ and biases $b_i^{(\mu)}$ are independent and identically distributed (i.i.d.) normal with mean 0 and variance $C_{\omega/b}^{(\mu)}$, respectively. Furthermore, we will impose the weight variances to be finite as $H_\mu \to \infty$ by rescalling them with a factor $1/H_{\mu-1}$: $C_\omega^{(\mu)} \to C_\omega^{(\mu)}/H_{\mu-1}$. However, a side effect of these factors is that they reduce greatly the influence of the connection weights during training when H_μ is large.

As the limit can be attained at different velocities, we are going to assign a function to the number of nodes for each layer.

Definition 2: Width function

For a given $n \in \mathbb{N}$, a width function $h_\mu : \mathbb{N} \to \mathbb{N}$ at depth μ specifies the number of hidden units H_μ at the layer μ.

We assume that for each $n \in \mathbb{N}$, $H_\mu(n) \to \infty$ as $n \to \infty$, *simultaneously.* This assumption permits for all the layers to have different widths.

Regarding Equation 2.2, notice that the information that arrives at a node is modified by a function ϕ before being sent to the following layer. This function must have to satisfy some property in order not to alter so sharply the information transmitted with respect to the received one. The property described is defined as follows:

Definition 3: Linear envelope property for non-linearities

A non-linearity $\pi : \mathbb{R} \to \mathbb{R}$ is said to obey the linear envelope property if there exists some constants $c, m \in \mathbb{R}^+$ such that: $|\phi(u)| \leq c + m|u|$, $\forall u \in \mathbb{R}$.

These are functions that do not differ much from the identity function, since the main objective of applying a non-linearity to data is not changing radically its behaviour but introducing a bit of complexity to the model.

The most popular functions for ϕ are the sigmoid and the positive part of the identity (generally denoted as ReLU). Intuitively, the linear bound on the non-linearity stops it from inducing heavy tail behaviour when a random variable is passed through it, which is of high interest since the Gaussian distribution has not heavy tails.

2.2 Gaussian processes

Having explained neural networks, let us introduce a formal definition of a Gaussian process.

Definition 4: Gaussian process

Let $\{X_t\}_{t\in T}$ be a family of real-valued random variables indexed by a set T, all defined on the same probability space. It is a Gaussian process (GP) if every finite linear combination $\sum_{t\in F} a_t X_t$ for any finite subset $F \in T$ is either identically zero or has a Gaussian distribution on R.

From Definition 4, we see that a GP is completely specified by its mean function and covariance function, which for a real process $f(\mathbf{x})$ we define as:

$$\mu \equiv m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{2.4}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right]. \tag{2.5}$$

It is usual to set the mean of the GP to be 0, hence, by Equation 2.5, we have: $k(x, x') = \mathbb{E}(f(x)f(x'))$.

A GP must satisfy the property known as *marginalisation* property. It states that if the GP specifies, e.g., $(y_1, y_2) \sim \mathcal{N}(\mu, \Sigma)$, then it must also specify $y_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$ (where the subscripts refer to the entry in the mean vector and in the matrix of covariance). That is, examination of a larger set of variables does not change the distribution of the smaller set.

2.2.1 Bayesian approach to regression

Consider the case of a linear regression, $\mathbf{y} = \mathbf{w}x + \varepsilon$ and specify a prior on $\mathbf{w}$, $p(\mathbf{w})$, and using Bayes' formula, update:

$$p(\mathbf{w}|\mathbf{y}, X) = \frac{p(\mathbf{y}|X, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|X)},$$

so that the posterior uses information from the prior and the dataset.
To get predictions at unseen data points x^*, obtain the *predictive distribution* by weighting all possible predictions by their posterior probabilities:

$$p(f^*|x^*, y, X) = \int p(f^*|x^*, w)p(w|y, X)dw.$$

The prior and the likelihood $p(y|X, w)$ are assumed to be Gaussian in order to ease computations, thus, the predictive distribution is also Gaussian and its mean provides a point prediction and the variance can be used to quantify uncertainty.

Gaussian Process Regression (GPR) is a non-parametric approach to regression. It calculates the probability distribution over all admissible functions that fit the data: specify a GP prior $\{f(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$ on the function space, and compute the posterior using the data (*training set* of the regression problem). We denote the Gaussian process as $f \sim GP(m, k)$.
Given a (training) set of observations $(X, \mathbf{y}) = \{(\mathbf{x}_i, y_i) : i = 1, \dots, n\}$, each $\mathbf{x}_i$ is d-dimensional input vector, y_i is the output. To find the relation between inputs and outputs of a Gaussian process regression, one assumes:

$$y_i = f(\mathbf{x}_i) + \epsilon_i \tag{2.6}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$ are i.i.d. random variables representing the noise in the data, since it is typical to not have access to functions values themselves, but only to noisy versions.
In this case, the explicit expression of $K(X, X)$, the covariance matrix from kernel $k(\mathbf{x}, \mathbf{x}')$, is:

$$K(X, X) = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}.$$

Let us start with a simpler case in which the observations are noise free, that is, we know $\{(\mathbf{x}_i, f_i) \mid i = 1, ..., n\}$. The joint distribution of the training outputs, $\mathbf{f}$, and the test outputs $\mathbf{f}_*$ according to the prior is:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

where $K(X, X_*)$ denotes the matrix of the covariances evaluated at all pairs of training and test point, and hence $K(X, X_*) = K(X_*, X)^T$, and $K_* \equiv K(X, X_*)$ (similarly for the rest of components). The introduced notation for the distribution as matrices means:

$$\mathbf{f} \sim \mathcal{N}(0, K) \qquad \mathbf{f}|\mathbf{f}_* \sim \mathcal{N}(K_* K_{**}^{-1}\mathbf{f}_*, K - K_* K_{**}^{-1} K_*^T)$$

In probabilistic terms, we have to condition the joint Gaussian prior distribution on the observations to obtain [3]:

$$\mathbf{f}_*|X_*, X, \mathbf{f} \sim \mathcal{N}(K_*^T K^{-1}\mathbf{f}, K_{**} - K_*^T K^{-1} K_*) \tag{2.7}$$

Now, let us return to our initial case and assume Equation 2.6. It follows that from the independence assumption about the noise, that a diagonal matrix is added, in comparison to the noise free case.

We can write the joint distribution of the observed target values and the function values at the test locations under the prior as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K + \sigma_n^2 I & K_* \\ K_*^T & K_{**} \end{bmatrix}\right)$$

where we have introduced the noise term.
Deriving the conditional distribution corresponding to Equation 2.7 we arrive at the key predictive equations for GP regression:

$$\mathbf{f}_* | X, \mathbf{y}, X_* \sim \mathcal{N}(K_*^T [K + \sigma_n^2 I]^{-1} \mathbf{y}, K_{**} - K_*^T [K + \sigma_n^2 I]^{-1} K_*)$$

Note that the mean prediction is a linear combination of observations $\mathbf{y}$ and that the variance does not depend on the observed targets, but only on the inputs. This is a property of the Gaussian distribution.
The variance is the difference between two terms: the first term K_{**} is simply the prior covariance; from it is subtracted a (positive) term, representing the information the observation gives us about the function.

2.3 Adapted central limit theorem for interchangeable random variables

To discuss neural networks with more than one hidden layer, it is useful to state the following lemma as it gives an idea of which is the behaviour of one layer given the information it has received from the previous one.

LEMMA 1. *If activations of previous layers are normally distributed with moments:*

$$\mathbb{E}(f_i^{(\mu-1)}(x)) = 0 \qquad \mathbb{E}(f_i^{(\mu-1)}(x) f_j^{(\mu-1)}(x')) = \delta_{ij} K(x, x')$$

Then, under the recursion of Equation 2.3 and considering the limit in the width function, the activation of the next layer converges in distribution to a Normal with 0 mean and second moment defined by the recursion:

$$\Sigma(x, x')^{(1)} = C_\omega x^T x' + C_b \tag{2.8}$$

$$\Sigma^{(\mu+1)}(x, x') = C_\omega^{(\mu)} \mathbb{E}_{(\epsilon_1, \epsilon_2) \sim \mathcal{N}(0, K)} [\phi(\epsilon_1)\phi(\epsilon_2)] + C_b^{(\mu)} \tag{2.9}$$

where $\epsilon_1, \epsilon_2 \sim N(0, K)$ with K an input matrix containing covariances between x and x', ϕ is the non-linearity defined in Definition 3 and C_ω, C_b are the variances of the weights and biases, respectively.

Lemma 1 assures that, under certain circumstances, the property of convergence goes from layer to layer. However, a finite set of input activations do not have a multivariate normal distribution, they may attain so in the limit. Thus, in order to address the non-sufficiency of Lemma 1, let us introduce some useful definitions that we will after relate.

Definition 5: Interchangeable random variables

Consider an infinite sequence of random variables $X = (X_1, X_2, ...)$. One says that it is interchangeable if for any finite collection of indexes $(i_1, i_2, ..., i_m)$ we have:

$$\mathcal{L}(X_{i_1}, X_{i_2}, ..., X_{i_m}) = \mathcal{L}(X_{\pi(i_1)}, X_{\pi(i_2)}, ..., X_{\pi(i_m)}),$$

where π is any permutation of indexes and $\mathcal{L}$ is the joint distribution function.

That is, the likelihood function $\mathcal{L}(\mathbf{X})$ is invariant under any permutation of indexes.

Definition 6: Conditionally i.i.d. random variables

One sequence of random variables ξ is conditionally i.i.d. if:

$$\mathbb{P}(\xi \in \cdot \,|\mathcal{F}) = \nu^{\infty} \text{ a.s.}$$

where a.s. stands for *almost surely*[a], $\mathcal{F}$ is some σ-field and ν a random probability measure defined in $\mathbb{R}$.

[a]A property is satisfied almost surely if the probability of it to happen is 1.

The notation ν^{∞} refers to the product of the probability measures of random variable of the sequence ξ. Explicitly:

$$\nu : (\Omega, \mathcal{A}, \mathbb{P}) \to (\mathcal{M}_1(\mathbb{R}), \mathcal{S}),$$

where $\mathcal{S}$ is the $\sigma-$algebra of sets of the space of probability measures on $\mathbb{R}$ generated by the projections $\pi_B : \mathcal{M}_1(\mathbb{R}) \to [0, 1]$, such that for all Borel set B, we have $\pi(B) = \mu(B)$. This random variable defines a probability γ on $\mathcal{M}_1$, defined for all measurable set of measures $\mathcal{K}$ as $\mathbb{P}(\nu(\omega, \cdot) \in \mathcal{K}) := \gamma(\mathcal{K})$.

Thus, let us present a theorem which links these two properties of random variables:

Theorem 1. *de Finetti. Let X be an infinite sequence of random variables. Then the following are equivalent:*

- *X is interchangeable*
- *X is conditionally iid*

See [4] for a complete proof of the theorem.

Our objective is to adapt the Central Limit Theorem (CLT) for sequence of interchangeable random variables. We cannot use the original version of it since one of its main hypotheseses is that the random variables are i.i.d., which is not the case for the network's neurons. Nevertheless, as the information that arrives to the different nodes is the same, they are, conditionally i.i.d. given that information. This way, the nodes of each layer can be considered as interchangeable random variables, given Theorem 1, hence its relevancy in this work.

Theorem 2. *Central Limit Theorem (CLT). Let $\{X_n\}_n$ be a sequence of i.i.d. random variables with finite second order moment. We denote $\mu = \mathbb{E}(X_1)$ and $\sigma^2 = Var(X_1)$. Let $S_n = \sum_{j=1}^{n} X_j$, then:*

$$\frac{S_n - n\mu}{\sigma\sqrt{n}} \to Z \sim \mathcal{N}(0, 1) \text{ in distribution,}$$

or equivalently:

$$\frac{S_n - \mathbb{E}(S_n)}{\sqrt{Var(S_n)}} \to Z \sim \mathcal{N}(0, 1) \text{ in distribution.}$$

We are going to base the statement of the re-versioned CLT for interchangeable variables in [5]. We consider a set of interchangeable variables $\{X_n\}$ having finite first and second moments, thus, without loss of generality, we can assume that they have mean 0 and variance 1. For each integer n, define:

$$S_n = \sum_{i=1}^{n} X_i$$

the partial sum. We say that the CLT holds for $\{X_n\}$ if for every real number α:

$$\lim_{n\to\infty} \mathbb{P}\left(\frac{S_n}{\sqrt{n}} \leq \alpha\right) = \varphi(\alpha),$$

where $\varphi(x)$ is the probability density function of a normal variable with mean 0 and variance 1 evaluated at α.
To find a necessary and sufficient condition for the CLT to hold with a sequence of exchangeable variables, let us introduce another theorem, whose proof can be found in [5]:

Theorem 3. *Let $\epsilon = (\epsilon_1, \epsilon_2, \ldots)$ be a sequence of interchangeable random variables with zero mean and variance one. The CLT holds if and only if $\gamma(\mathcal{M}(0,1)) = 1$.*

In Theorem 3, γ refers to the already defined probability measure. The notation $\mathcal{M}(0,1)$ is introduced for our example. By de Fineti's Theorem:

$$P\left(\frac{S_n}{\sqrt{n}} \leq y\right) = \int_{\mathcal{M}(0,1)} \mathcal{P}\left(\frac{\epsilon_1(m) + \ldots + \epsilon_n(m)}{\sqrt{n}}\right) d\gamma(m)$$

where m is the marginal distribution of the i.i.d $\epsilon_i(m)$. This, if for m we define:

$$\mu(m) = \int_{\mathbb{R}} x dm(x) = \mathbb{E}\left(\epsilon_i(m)\right) \quad \text{and} \quad \sigma^2(m) = \int_{\mathbb{R}} (x - \mu(m))^2 dm(x) = Var\left(\epsilon_i(m)\right)$$

Thus, for all pair of number $(\mu, \sigma) \in \mathbb{R} \times \mathbb{R}^+$ we denote $\mathcal{M}(\mu, \sigma)$ the set of $\mathcal{M}_1$ such that $\mu(m) = \mu$ and $\sigma(m) = \sigma$.

Although the condition of Theorem 3 is more than enough for having the CLT, it is difficult to verify, since γ is rarely explicit. We might express the above condition using moments.

Theorem 4. *Let $\epsilon = (\epsilon_1, \epsilon_2, \ldots)$ be a sequence of interchangeable random variables with zero mean and variance one. Then the CLT holds if and only if for $i \neq j$:*

$$\mathbb{E}\left(\epsilon_i \epsilon_j\right) = 0 \qquad \text{and} \qquad \mathbb{E}\left([\epsilon_i^2 - 1][\epsilon_j^2 - 1]\right) = 0 \tag{2.10}$$

This condition is rather simpler than the one of Theorem 3. We can see it as the different random variables as well as their squares must be uncorrelated (recall the definition of covariance $Cov(X,Y) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)$ and the fact that we are considering variables with 0 mean and variance 1).

Proof. We are going to prove that this new condition is in fact equivalent to the previously mentioned.
Suppose first that $\gamma(\mathcal{M}(0,1)) = 1$. By de Finetti's theorem:

$$\mathbb{E}(\epsilon_i \epsilon_j) = \int_{\mathcal{M}(0,1)} \mu^2(m)\gamma(dm) = 0$$
$$\mathbb{E}(\epsilon_i^2 \epsilon_j^2) = \int_{\mathcal{M}(0,1)} \left(\int_{\mathbb{R}} x^2 dm(x)\right)^2 d\gamma(m) = 1$$

Thus, Equation 2.10 hold. On the other hand, assume these 2 conditions to be true, we use de Finetti theorem to prove $\gamma(\mathcal{M}(0,1)) = 1$:

$$\int_{\mathcal{M}_1} \mu^2(m)\gamma(dm) = \mathbb{E}(\epsilon_i \epsilon_j) = 0 \implies \mu(m) = 0$$
$$\int_{\mathcal{M}_1} \left(\int_{\mathbb{R}} (x-1)^2 dm(x)\right)^2 d\gamma(m) = \mathbb{E}((\epsilon_i^2 - 1)(\epsilon_j^2 - 1)) = 0 \implies \sigma^2(m) = 1$$

□

Theorem 5. *Consider a random deep neural network described by Equation 2.1, Equation 2.2 and Equation 2.3 with a continuous non-linearity satisfying the property of linear envelope. Let $h_\mu(n)$ be some set of strictly increasing width functions and $(x[i])_{i=1}^{\infty}$ any countable input set. Then the distribution of the output of the network converges in distribution* [2] *to a Gaussian process as $n \to \infty$. The limiting Gaussian process has zero mean and covariance function given by the recursion of Lemma 1.*

The scheme of the proof is based in upper bounding the difference between each layer and a multivariate normal distribution. It is a proof by induction over the layers of the network to finally bound the distance of the output layer for a collection of input points and a multivariate Gaussian distribution. The complete proof can be found in the Appendix of [6].

2.4 Covariance function of the Gaussian process associated to a neural network

For a deep fully connected neural networks of D layers, $D > 1$ and each labelled by μ, we assume that the weights of the networks at each layer form a matrix of independent random elements $W^{(\mu)} = (w_{i,j}^{(\mu)})_{H_\mu \times H_{\mu-1}}$ with $w_{i,j}^{(\mu)} \sim \mathcal{N}(0, C_m^{(\mu)})$ and the random bias $b_i^{(\mu)} \sim \mathcal{N}(0, C_b^{(\mu)})$.
Using this notation we can rewrite expression Equation 2.3 as:

$$f^{(\mu+1)}(x) = W^{(\mu+1)} g^{(\mu)} + b^{(\mu+1)}$$

From the distribution of the weight matrix, it holds:

$$g^{(\mu)}(x) = \phi(f^{(\mu)}(x))$$

$$f^{(\mu+1)}(x) = \sqrt{\frac{\hat{C}_W^{(\mu+1)}}{H_\mu(n)}} Z^{(\mu+1)} g^{(\mu)}(x) + b^{(\mu+1)},$$

where $Z \sim \mathcal{N}(0,1)$.
Furthermore, its conditional distribution given the vector $g^{(\mu)}(x)$ is a zero mean Gaussian vector with covariance matrix given by:

$$\left(\frac{\hat{C}_W^{(\mu+1)}}{H_\mu(n)} ||g^{(\mu)}(x)||^2 + C_b^{(\mu+1)} I_{H_\mu \times H_\mu} \right),$$

since the randomness of $g^{(\mu)}(x)$ has been deleted and now it is deterministic (known).

The Gaussian process associated to the neural network is fully described by a recursive covariance kernel determined by the architecture of the network, and it is expressed in terms of expectations. In [7], they introduce the sequence of random variables associated to an infinite real sequence $\chi = (x[i])_{i=1}^{\infty}$:

$$\gamma_j^{(\mu)}(\mathcal{L}, \alpha)(n) = \sum_{(\mathbf{x}_{J_M}, i) \in \mathcal{L}} \alpha^{(\mathbf{x}_{J_M}, i)} g_j^{(\mu-1)}(\mathbf{x}_{j_m}) \epsilon i, j,$$

where:

$$g_j^{(\mu-1)}(\mathbf{x}_{J_M}) = \phi \left(\sqrt{\frac{\hat{C}_W^{(\mu-1)}}{H_{\mu-2}(n)}} \sum_{k=1}^{H_{\mu-2}(n)} \epsilon_{j,k} g_k^{(\mu-2)}(\mathbf{x}_{J_M}) + b_j^{(\mu-1)} \right),$$

$\mathcal{L}$ is a set of tuples of indexes $\mathcal{L} \in \chi \times \mathbb{N}$ and $\alpha^{(\mathbf{x}_{J_M}, i)}$ are real parameters.

Conditioning with respect to the random variables $g_k^{(\mu-2)}(\mathbf{x}_{J_M})$ for $k = 1, ..., H_{\mu-2}(n)$, we have that $\gamma_j^{(\mu)}$ are interchangeable by de Finetti Theorem 1. Furthermore, they satisty the hypothesis of

[2]A sequence $X_1, X_2, \ldots$ of real-valued random variables, with cumulative distribution function (cdf) $F_1, F_2, \ldots$, is said to converge in distribution to a random variable X with cdf F if: $\lim_{n\to\infty} F_n(x) = F(x)$.

the Central Limit Theorem for interchangeable random variables 1 [6].

In order to characterise the Gaussian process, we state the expression to by recurrence compute the limit variance:

$$\mathbb{E}\left[(\gamma_j^{(\mu)}(\mathcal{L},\alpha)(n))\right] = 0$$

$$\mathbb{E}\left[(\gamma_j^{(\mu)}(\mathcal{L},\alpha)(n))^2\right] = \sum_{l=1}^{|\mathcal{L}|} \alpha_{(l)}^2 \mathbb{E}\left[\phi^2\left(\sqrt{\frac{\hat{C}_W^{(\mu-1)}}{H_{\mu-2}(n)}} \sum_{k=1}^{H_{\mu-2}(n)} \epsilon_{j,k} g_k^{(\mu-2)}(\mathbf{x}_l) + b_j^{(\mu-1)}\right)\right]$$

2.4.1 Hermite polinomials

In order to alleviate the computational time required in the above computation, we use its expansion series with Hermite polynomials:

$$H_n(x) = (-1)^n \left[\frac{d^n}{dx^n}(e^{-\frac{1}{2}x^2})\right] e^{\frac{1}{2}x^2}$$

It is well known that these polynomials conform a complete and orthogonal set for $\mathbb{L}^2(\mathbb{R}, \varphi(x)dx)$, where φ is the density of the standard Gaussian. Some properties of the Hermite polynomials are:

- $||H_n||^2 = n!$
- $H_0(x) = 1$ and $H_n(x) = n! \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} = \frac{(-1)^j}{2^j j!(n-2j)!} x^{n-2j}$
- $H_{2k-1}(0) = 0$ for $k > 0$
- $H_{2k}(0) = (-1)^k (2k-1)!!$ for $k > 0$

Suppose that F and G are two functions belonging to $\mathbb{L}^2(\mathbb{R}, \varphi(x)dx)$ with variance 1 and covariance ρ. Then, the expansion and coefficients have the following form:

$$F(x) = \sum_{n=0}^{\infty} \hat{F}_n H_n(x), \text{ where } \hat{F}_n = \frac{1}{n!}\int_{\mathbb{R}} F(x) H_n(x) \varphi(x) dx$$

and similarly for G.

By using Mehler's formula for Hermite polynomials [8]:

$$\mathbb{E}(x,y) \sum_{n=0}^{\infty} \frac{\rho^n}{n!} H_n(x) H_n(y),$$

we obtain the following expression for the expectation of the product:

$$\mathbb{E}(F(X)G(X)) = \sum_{n=0}^{\infty} \hat{F}_n \hat{G}_n n! \rho^n,$$

where:

$$\rho(x(l), x(l')) = \frac{\frac{\hat{C}_W^{(0)}}{M} x(l) \cdot x(l') + C_b}{\sigma(x(l))\sigma(x(l'))}. \tag{2.11}$$

To simplify notation, we have defined:

$$\sigma(x(l))^2 = \frac{\hat{C}_W^{(0)}}{M} \sum_{k=1}^{M} \varepsilon_{1,k} x_m(l) + C_b \qquad (X(x(l)), Y(x(l'))) \sim \mathcal{N}\left(0, \begin{bmatrix} 1 & \rho(x(l), x(l')) \\ \rho(x(l), x(l')) & 1 \end{bmatrix}\right).$$

We have then:

$$\mathbb{E}\left[\phi\left(\sqrt{\frac{\hat{C}_W^{(0)}}{M}}\sum_{k=1}^{M}\varepsilon_{1,k}x_m(I)+b_1\right)\phi\left(\sqrt{\frac{\hat{C}_W^{(0)}}{M}}\sum_{k=1}^{M}\varepsilon_{1,k}x_m(I')+b_1\right)\right]= \tag{2.12}$$

$$=\mathbb{E}\left[\phi(\sigma(x(I))X(x(I)))\phi(\sigma(x(I'))Y(x(I')))\right]= \tag{2.13}$$

$$=\sum_{n=0}^{\infty}\hat{\phi}_n(\sigma(x(I)))\hat{\phi}_n(\sigma(x(I')))n!(\rho(x(I)),\rho(x(I')))^n, \tag{2.14}$$

where the coefficients are defined:

$$\hat{\phi}_n(\sigma)=\frac{1}{n!}\int_{\mathbb{R}}\phi(\sigma(x))H_n(x)\varphi(x)dx.$$

The Equation 2.14 can be naturally extended to multiple layer networks ($\mu=1,\dots,D$) as follows:

$$K^{(\mu)}(x(I),x(I'))= \tag{2.15}$$

$$=\mathbb{E}\left[\phi((K^{(\mu-1)}(x(I),x(I)))^{\frac{1}{2}}X^{(\mu-1)}(x(I)))\phi((K^{(\mu-1)}(x(I'),x(I')))^{\frac{1}{2}}Y^{(\mu-1)}(x(I')))\right]= \tag{2.16}$$

$$=\sum_{n=0}^{\infty}\hat{\phi}_n((K^{(\mu-1)}(x(I),x(I)))^{\frac{1}{2}})\hat{\phi}_n((K^{(\mu-1)}(x(I'),x(I')))^{\frac{1}{2}})n!(\rho_{x,x'}^{(\mu-1)})^n, \tag{2.17}$$

where:

$$\rho_{x,x'}^{(\mu)}=\frac{K^{(\mu)}(x(I),x(I'))}{\left[K^{(\mu)}(x(I),x(I))\right]^{\frac{1}{2}}\left[K^{(\mu)}(x(I'),x(I'))\right]^{\frac{1}{2}}}$$

As might be expected, the initial step coincides with the obtained expression for the case of a single layer (Equation 2.11).

2.4.2 Activation funtion ReLU

We use the ReLU (Rectified Linear Unit) as activation function for all of the hidden layers since it is the most efficient one during the training process [9] and it requires less computational work than other common activation functions such as tanh or sigmoid.
As ReLU is a positively homogeneous function of degree 1 ($\phi(\lambda x)=\lambda\phi(x)$, for $\lambda>0$), we can reexpress it as $\phi^{(1)}(x)=\Theta(x)x$ where Θ is the Heaviside function.
We first work with the Heaviside example to compute the expression of the Hermite coefficients associated to it (for $n\geq 1$):

$$\hat{\phi}_n^{(0)}=\frac{1}{n!}\int_{\mathbb{R}}\Theta(x)H_n(x)\varphi(x)dx=\frac{1}{n!}\frac{(-1)^n}{\sqrt{2\pi}}\int_0^{\infty}\frac{d^n}{dx^n}\left(e^{-\frac{1}{2}x^2}\right)dx=\frac{1}{\sqrt{2\pi}}\frac{H_{n-1}(0)}{n!} \tag{2.18}$$

Now, we compute the coefficients for the ReLU function, which we will later relate with the previous expression. For n=0:

$$\hat{\phi}_0^{(1)}(\sigma)=\int_{\mathbb{R}}\sigma(\mathbf{X})x\Theta(x)\varphi(x)dx=\frac{1}{\sqrt{2\pi}}\sigma(\mathbf{X})\int_0^{\infty}xe^{-\frac{x^2}{2}}dx=\frac{1}{\sqrt{2\pi}}\sigma(\mathbf{X})$$

where $\mathbf{X}$ stands for the set of features, not to be confused with the integration variable. For $n=1$:

$$\hat{\phi}_1^{(1)}=\int_0^{\infty}\sigma(\mathbf{X})xH_1(x)\varphi(x)dx=\frac{\sigma(\mathbf{X})(-1)^1}{\sqrt{2\pi}}\int_0^{\infty}x\frac{d}{dx}\left(e^{-\frac{1}{2}x^2}\right)dx=\frac{\sigma(\mathbf{X})}{\sqrt{2\pi}}\int_0^{\infty}x^2e^{-\frac{1}{2}x^2}=$$

$$=\frac{\sigma(\mathbf{X})}{\sqrt{2\pi}}\sqrt{\frac{\pi}{2}}=\frac{\sigma(\mathbf{X})}{2}$$

and for $n \leq 2$:

$$\hat{\phi}_n^{(1)}(\sigma) = \frac{1}{n!}\int_0^\infty \sigma(\mathbf{X}) x H_n(x)\varphi(x)dx = \frac{\sigma(\mathbf{X})(-1)^n}{n!\sqrt{2\pi}}\int_0^\infty x\frac{d^n}{dx^n}\left(e^{-\frac{1}{2}x^2}\right)dx \iff$$
$$(-1)^n\sqrt{2\pi}n!\hat{\phi}_n^{(1)} = \int_0^\infty x\frac{d^n}{dx^n}\left(e^{-\frac{x^2}{2}}\right)dx = x\frac{d^{n-1}}{dx^{n-1}}\left(e^{-\frac{x^2}{2}}\right)\Bigg|_0^\infty - \int_0^\infty \frac{d^{n-1}}{dx^{n-1}}\left(e^{-\frac{x^2}{2}}\right)dx$$

where in the last equality we have used integration by parts. The first summand vanishes due to the fact that the behaviour of the exponential at infinity outgrows the polynomial one and at 0 the x term vanishes. On the other hand, the second term looks like Equation 2.18 but with $n-1$. Thus:

$$(-1)^n\sqrt{2\pi}n!\hat{\phi}_n^{(1)} = -(n-1)!\hat{\phi}_{n-1}^{(0)}\sqrt{2\pi}(-1)^{n-1} \iff$$
$$\hat{\phi}_n^{(1)} = \frac{\hat{\phi}_{n-1}^{(0)}}{n} = \frac{1}{\sqrt{2\pi}n}\frac{H_{n-2}(0)}{(n-1)!} = \frac{H_{n-2}(0)}{\sqrt{2\pi}n!}$$

Taking into account that $H_m(0)$ are non-zero only for odd m, we might consider taking $n = 2k$ for $k \geq 0$. Thus, by Equation 2.17:

$$K^{(\mu)}(\mathbf{x}(l), \mathbf{x}(l')) = \tag{2.19}$$
$$\sum_{n=0}^\infty (\hat{\phi}_n^{(1)})^2 n! (\rho_{\mathbf{x},\mathbf{x}'}^{(\mu-1)})^n = \sigma(\mathbf{x}(l), \mathbf{x}(l'))\left(\frac{1}{2\pi} + \frac{1}{4}\rho_{\mathbf{x},\mathbf{x}'}^{(\mu-1)} + \frac{1}{2\pi}\sum_{n=2}^\infty \frac{H_{n-2}^2(0)}{n!}(\rho_{\mathbf{x},\mathbf{x}'}^{(\mu-1)})^n\right) = \tag{2.20}$$
$$\sigma(\mathbf{x}(l), \mathbf{x}(l'))\left(\frac{1}{2\pi} + \frac{1}{4}\rho_{\mathbf{x},\mathbf{x}'}^{(\mu-1)} + \frac{1}{2\pi}\sum_{k=0}^\infty \frac{H_{2k}^2(0)}{(2k+2)!}(\rho_{\mathbf{x},\mathbf{x}'}^{(\mu-1)})^{2k+2}\right) \tag{2.21}$$

3 Methods for selection and comparison of models

3.1 Selection of model. Shapley values

The ability to correctly interpret a prediction model's output engenders appropriate user trust, provides insight into how a model may be improved and supports understanding of the process being modelled. In some applications, simple models such as linear models are often preferred for their ease of interpretation, even if they may be less accurate than complex ones: the best explanation of a simple model is the model itself. On the other hand, for complex models such as deep networks, we might not use the original model as its own best explanation because it is not easy to understand. Instead, we must use a simpler explanation model, which we define as any explainable approximation of the original model. In order to partly address this "black box" characteristic, we introduce the concept of Shapley regression, a well-grounded statistical inference tool.

We introduce the concept of additive feature attributions models, which will provide us a tool for interpretability.

DEFINITION 1. *Additive feature attribution methods have an explanation model that is a linear function of binary variables:*

$$g(x') = \phi_0 + \sum_{i=1}^{m} \phi_i x'_i,$$

where $x' = \{0,1\}^m$ *is the number of inputs, and* $\phi_i \in \mathbb{R}$.

These models are known as additive since they consider the model output to be the addition of every feature contribution.

Considering x to be the inputs of the model and x' the simplified ones (which are related to the original ones by a function h such that $h_x(x') = x$), let us state the properties of these models:

1. Local precision. The attribution model g matches the original model f at x_i:

$$f(x) = g(x') = \phi_0 + \sum_{k=1}^{m} \phi_k(x'_i).$$

2. Missingness. If a variable is missing from the model, no attribution is given to it: $x'_i = 0$ then $\phi_i = 0$.

3. Consistency. For any two models f and f':

$$f(x') - f(x' \backslash k) \geq f'(x') - f'(x' \backslash k)$$

for all possible x', then, $\phi_i(f, x) \geq \phi_i(f', x)$. This means that variable attributions do not decrease if an input's contribution to a model increases or stays the same regardless of other variables in the model.

Now that we are familiar with this kind of models, we introduce a particular class of them given by *Shapley values* Φ^S. Shapley values have an interpretation derived from their game theoretic origin (as introduced by the mathematician Lloyd Shapley in 1953 [10]); making the analogy between players of a multi-player game cooperating to generate a pay-off and variables x_k within a model to generate predictions $\hat{f}(x)$. The marginal contribution from variable k is defined in the form of its Shapley value:

$$\phi_k = \sum_{S \subseteq |M| \backslash \{k\}} \frac{|S|!(M - |S| - 1)!}{M!} \left[\hat{f}(x_S \cup \{k\}) - \hat{f}(x_S) \right], \qquad k = 1, ..., m \tag{3.1}$$

where $M\backslash\{k\}$ is the set of all possible coalitions of $m-1$ model variables when excluding the k^{th} variable (M is the number of input features). The Equation 3.1 is the weighted sum of marginal contributions of variable k accounting for the number of possible coalitions for a certain x'.

Intuitively, the definition of a Shapley value is similar to the regression anatomy of a coefficient $\hat{\beta}_k$. In regression analysis, the coefficient (β_k) measures the change in the dependent variable for a one-unit change in the independent variable, while holding other variables constant. In the context of feature importance in machine learning models, Shapley values assign each feature a value that represents its contribution to the prediction. This contribution is measured by comparing predictions with and without the feature, considering all possible combinations of features. This method requires retraining the model on all feature subsets $S \subseteq M$. It assigns an importance value to each feature that represents the effect on the model prediction including that feature. To compute this effect, a model $\hat{f}(x_S \cup \{k\})$ is trained with that feature presents, and another model $\hat{f}(x_S)$ is trained with the feature withheld. Then, predictions from the two models are compared. Since the effect of withholding a feature depends on other features in the model, the differences are computed for all possible subsets $S \subseteq M\backslash\{k\}$.

Regarding the computation of Shapley values, most models cannot handle missing variables in any of the observations. If missing from a coalition, the contribution of a variable is integrated out via conditional expectations relative to a representative background sample.

In [11], it is stated that there is only one possible additive feature attribution method with the above-mentioned properties for an additive attributions model. While these properties are familiar to the classical Shapley value estimation methods, they were previously unknown for other additive feature attribution methods:

Theorem 6. *Only one possible explanation model follows definition 1 and satisfies Properties 1,2 and 3. It is given by:*

$$\phi_k(\hat{f}, x) = \sum_{z' \subset x'} \frac{|z'|!(M - |z'| - 1)!}{M!} \left[\hat{f}_x(z') - \hat{f}_x(z' \backslash k) \right],$$

where $|z'|$ is the number of non-zero entries in z.

3.2 Comparison of nested models

Once we have selected a reduced model basing the decision in the Shapley values of the model features, we are going to compare both versions considering them to be nested models. Thus, we can determine whether the prediction obtained with the reduced model have a comparable error with the one associated to the general. Although the predictions of the Gaussian process do not give rise to a linear function of the features, we are going to base our procedure in some of the theory underlying these models.

We can reduce a general linear model to be:

DEFINITION 2. *Consider a variable Y in $\mathbb{R}^n$ obtained as:*

$$Y = X\beta + e$$

where X is a $p \times n$ matrix whose columns (that are assumed to be independent) span $\mathcal{R}$ and β is a vector in $\mathbb{R}^p$ and e is a variable of errors whose distribution is assumed to be isotropic Normal with variance σ^2 in $\mathbb{R}^n$.

The fitting of the linear model relies on some basic assumptions: the relationship between Y and $\mathbf{X}$ is linear, the variance of the errors is constant and the errors are normally distributed and independent.

The (random) error can be estimated by:

$$\hat{e} = Y - \hat{Y} = Y - X\hat{\beta} = Y - X(X^tX)^{-1}X^tY := (I - H)Y$$

Then, the concept of sum of squared errors (SSE) happens to be: $SSE = ||\hat{e}||^2 = \hat{e}^t\hat{e} = e^t(I - H)e$. Observe that $SSE = X^tY - \hat{\beta}X^tY$.
If we want to compare models:

$$\begin{cases} (1) y_i = \beta_0 + \beta_1 x_{i,1} + \cdots + \beta_l X l, i + \epsilon_i \\ (2) y_i = \beta_0 + \beta_1 x_{i,1} + \cdots + \beta_l X l, i + \beta_{l+1} x_{l+1,i} + \cdots + \beta_k X k, i + \epsilon_i \end{cases}$$

All explanation features of model (1) are in model (2): model (1) is nested in model (2). In vector form:

$$\begin{cases} (1) \mathbf{Y} = X_1\beta + \epsilon \\ (2) \mathbf{Y} = X\beta + \epsilon \end{cases}$$

All columns of X_1 are in X, that is, $\langle X_1 \rangle$ is a subspace of $\langle X \rangle$. So, we see, that comparing both models is equivalent to compare these hypothesis:

$$\begin{cases} H_0 : \beta_{l+1} = \cdots = \beta_k = 0 \text{ vs.} \\ H_1 : \beta_j \neq 0, \text{ for some} j = l + 1, ..., k \end{cases}$$

Under the null hypothesis, that is, the true model is (1), then the difference between e_1 and e_2 should be due to chance.

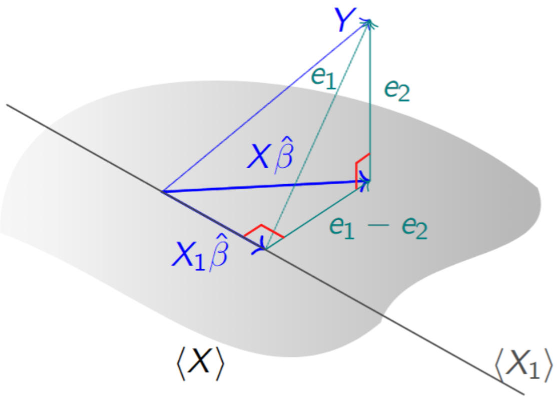

Figure 3.1: Geometry of the problem.

Since

$$\frac{1}{\sigma^2}||\mathbf{e_1} - \mathbf{e_2}||^2 \sim \chi^2_{k-l}$$
$$\frac{1}{\sigma^2}||\mathbf{e_2}||^2 \sim \chi^2_{n-(k+1)},$$

we have that considering the statistic F [12]:

$$F = \frac{\frac{||\mathbf{e_1} - \mathbf{e_2}||^2}{k-l}}{\frac{||\mathbf{e_2}||^2}{n-(k+1)}} \sim F_{k-l,n-(k+1)}.$$

We reject H_0 if $F > F^{\alpha}_{k-l,n-(k+1)}$, for a fixed level of confidence α. By looking at Figure 3.1, we see that a large value of F means that the difference between errors is large or that the error of the general model is small. In both cases, as we have a right-angled triangle, we have that the error when using the reduced model for predicting **Y** is large, so it is not efficient to use it.

However, we do not know the real distribution of the variance of the errors. For this purpose, we introduce the following method to work with their quotient.

3.2.1 Approach: Morgan-Pitman test

When we test the hypothesis that two dependent variables have equal variances, there exists an exact statistical test for significance whose distribution is known, although the population correlation coefficient is unknown: the Morgan-Pitman test [13], [14].

Let a sample of n pairs of variables (x, y) be bi-variate Normal with probability law:

$$\rho(x,y) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho_{12}}} exp\left[-\frac{1}{2(1-\rho_{12}^2)}\left\{\left(\frac{x-\epsilon_1}{\sigma_1}\right)^2 - 2\rho_{12}\frac{(x-\epsilon_1)(y-\epsilon_2)}{\sigma_1\sigma_2} + \left(\frac{x-\epsilon_1}{\sigma_1}\right)^2\right\}\right].$$

W. A. Morgan based the creation of the test on the likelihood ratio method of Neyman and Pearson [15]. He stated that by making the transformation:

$$x = X + Y \text{ and } y = X - Y,$$

so that: $X = \frac{1}{2}(x+y)$ and $Y = \frac{1}{2}(x-y)$, then the population variances of x and y may be expressed as functions of the variances and correlation for X and Y, as follows:

$$\begin{cases} \sigma_1^2 = \sigma_X^2 + \sigma_Y^2 + 2\rho_{XY}\sigma_X\sigma_Y \\ \sigma_2^2 = \sigma_X^2 + \sigma_Y^2 - 2\rho_{XY}\sigma_X\sigma_Y \end{cases}.$$

Thus, the necessary and sufficient condition that $\sigma_1 = \sigma_2$ (the null hypothesis of his test) is that: $\rho_{XY} = 0$, where ρ_{XY} is the Pearson's correlation coefficient between X and Y:

$$\rho_{XY} = \frac{\text{Cov}(X,Y)}{\text{sd}(X)\text{sd}(Y)}$$

The statistic associated to this test is:

$$T_{xy} = r_{xy}\sqrt{\frac{n-2}{1-r_{xy}^2}}$$

where r_{xy} is the usual estimate of ρ_{xy} and T_{xy} is assumed to have a Student's T distribution with $n-2$ degrees of freedom, where n is the sample size.

In case that the sample that data come from heavy-tailed distributions, there has been shown that the Type I error probability exceeds the nominal level and suggest replacing Pearson's coefficient with Spearmans' correlation [16], which is defined as the Pearson correlation coefficient between the rank variables.

4 Further lines of research

4.1 Bootstrap

Although we already presented a test whose distribution is well known, there also exists a vaguer approach to tackle the problem of comparing variances, which we firstly considered.

Let $\mathbf{x} = (x_1, ..., x_n)$ be an observed random sample from an unknown probability distribution F. The bootstrap method depends on the notion of a *bootstrap sample* [17]. A bootstrap sample $\mathbf{x}^* = (x_1^*, x_2^*, ..., x_n^*)$ is obtained by randomly sampling m times, with replacement, from the original data points. The star notation indicates that $\mathbf{x}^*$ is not the actual data set $\mathbf{x}$ but rather a resampled version of it.

On the other hand, the *empirical distribution* function $\hat{F}$ (EDF) is defined to be the discrete distribution that puts probability $1/n$ on each value $x_i, i = 1, ..., n$, that is, $\hat{F}$ assigns to a set A in the sample space of x its empirical probability:

$$\mathcal{P}_{\hat{F}}\{A\} = \frac{\#\{x_i \in A\}}{n},$$

which represents the proportion of the observed sample $\mathbf{x}$ occurring in A.

As the vector of observed frequencies $\hat{F} = (\hat{f}_1, \hat{f}_2, ..., \hat{f}_n)$ is a sufficient statistic for the true distribution $F = (f_1, f_2, ..., f_n)$, the process of going from the full data set to the reduced representation in terms of frequencies does not represent a loss of information. This means, that all of the information about F contained in $\mathbf{x}$ is also contained in $\hat{F}$. Because the EDF puts equal probabilities on the original data values, each bootstrap sample is independently sampled at random from those data values.

However, in general, the non-parametric resampling method does not work for dependent data, where the dependence might be too complex. This leads to the idea of doing bootstrap by taking blocks of consecutive observations rather than resampling at the level of individual observations [18]. The simplest version of this idea divides the data into b non-overlapping blocks of length l, where we suppose that $n = bl$. We set $z_1 = (y_1, ..., y_l)$, $z_2 = (y_{l+1}, ..., y_{2l})$ and so forth, giving blocks $z_1, ..., z_b$. The procedure is to take a bootstrap sample with equal probabilities b^{-1} from the z_j and then to paste these end-to-end to form a new series. To illustrate this method, let us provide an example. Imagine that the time series is $y_1, ..., y_{15}$ and we take $b = 3$, $l = 5$, then: $z_1 = (y_1, ..., y_5), z_2 = (y_6, ..., y_{10})$ and $z_e = (y_{10}, ..., y_{15})$. If the resampled blocks are: $z_1^* = z_2, z_2^* = z_1, z_3^* = z_1$, the new series of length 15 is: $\{y_j^*\} = z_1^* z_2^* z_3^* = y_6, ..., y_{10}, y_1, ..., y_5, y_1, ..., y_5$.

This approximation will be best if the dependence is weak and the blocks are as long as possible, thereby preserving the dependence more faithfully. Note that with this method we have a minor problem since the last block is shorter than the rest if $\frac{n}{l}$ is not an integer. We can use the overlapping alternative and wrapping the data around a circle, in order to ensure that each of the original observations has an equal chance of appearing in a simulated series.

A further but less important difficulty with these block schemes is that the artificial series generated by them are not stationary, because the joint distribution of resampled observations close to a join between blocks differs from that in the centre of a block. This can be overcome by taking blocks of random length. The stationary bootstrap takes blocks whose lengths L are geometrically distributed, with density:

$$P(L = j) = (1p)^{j-1}p \text{ where } j = 1, 2, ...$$

This yields resampled series that are stationary with mean block length $l = \frac{1}{p}$. The results are similar to those for the block bootstrap, except that the varying block length preserves slightly more of the original correlation structure.

The problem with this approach is that errors of both models are not independent between them. Moreover, we do not know the exact dependence between both sets of errors, so it would be difficult to create a test for testing the equality of variances of errors.

4.2 Neural tangent kernel

This section is intended to introduce a future line of research that we have found to be interesting but could not properly develop. However, let us introduce the theoretical foundation to do so.

At initialization, neural networks are equivalent to Gaussian processes in the infinite width limit (as already stated in Theorem 5). Nevertheless, the evolution of a neural network during training can also be described by a kernel function, as first introduced in [19].
During the training process, f_θ (as defined in subsection 2.1), evolves along the negative kernel gradient:

$$\partial_t f_{\theta(t)} = -\Delta_{\Theta^{(L)}} C\big|_{f_{\theta(t)}}$$

with respect to the neural tangent kernel (NTK):

$$\Theta^{(L)}(\theta) = \sum_{i=1}^{P} \partial_{\theta_p} F^{(L)}(\theta) \otimes \partial_{\theta_p} F^{(L)}(\theta),$$

where C is the functional cost.

The NTK is random at initialization and varies during training, since it depends upon the parameters θ. However, in the infinite-width limit, it becomes deterministic at initialization and stays constant during training, as stated in the following 7.

Theorem 7. *For a network of depth L, with a non-linearity ϕ, and in the limit as the layers width $n_1, ..., n_{L-1} \to \infty$, the NTK $\Theta^{(L)}$ converges in probability[3] to a deterministic limiting kernel:*

$$\Theta^{(L)} \xrightarrow{\mathcal{P}} \Theta_\infty^{(L)} \otimes Id_{n_L}.$$

The scalar kernel $\Theta_\infty^{(L)} : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ is defined recursively by:

$$\begin{aligned} \Theta_\infty^{(1)}(x,x') &= \Sigma^{(1)}(x,x') \\ \Theta_\infty^{(L+1)}(x,x') &= \Theta_\infty^{(L)}(x,x')\dot{\Sigma}^{(L+1)}(x,x') + \Sigma^{(L+1)}(x,x'), \end{aligned}$$

where $\Sigma^{(\mu)}$ is defined in Equation 2.9 and

$$\dot{\Sigma}^{(L+1)}(x,x') = \mathbb{E}_{f\sim\mathcal{N}(0,\Sigma^{(L)})}[\dot{\phi}(f(x))\dot{\phi}(f(x'))], \tag{4.1}$$

taking the expectation with respect to a centered Gaussian process f of covariance $\Sigma^{(L)}$ and where $\dot{\phi}$ denotes the derivative of ϕ.

From here we see, that the kernel has the same form as the one obtained in (1), but adding a term related to derivatives of the activation function. Note that the limiting $\Theta_\infty^{(L)}$ only depends on the choice of ϕ, the depth of the network and the variance of the parameters at initialization. The proof of Theorem 7 can be found in [19].

[3] A sequence $\{X_n\}$ of random variables converges in probability towards the random variable X if for $\forall\epsilon > 0$ $\lim_{n\to\infty} \mathbb{P}(|X_n - X| > \epsilon) = 0$.

5 Methodology

5.1 Data

The data we are going to use for the experiments can be found in GitHub_TfgMatematiques, file: *boston.csv*. I have downloaded from Kaggle datasets and it consists in a set of features to predict the Median Value (MEDV) of owner-occupied homes in $\$1000's[k\$]$. The explanation features are:

- CRIM: crime rate by town. Since CRIM gauges the threat, to well-being that households perceive in various neighbourhoods of the Boston metropolitan area (assuming that crime rates are generally proportional to people's perceptions of danger) it should have a negative effect, on housing values.

- ZN: proportion of a town's residential land zoned for lots greater than 25,000 square feet. Such zoning restricts construction of small lot houses.

- INDUS: proportion non-retail business acres per town. INDUS is associated with industry-noise, heavy traffic, and unpleasant visual effects, and thus should affect housing values negatively.

- CHAS: Charles River dummy variable. It is $= 1$ if tract bounds the Charles River; $= 0$ if otherwise. CHAS captures the amenities of a riverside location.

- Air Pollution NOX: nitrogen oxide concentrations in pphm (annual average concentration in parts per hundred million).

- RM: average number of rooms in owner units. RM represents spaciousness and, in a certain sense, quantity of housing. It, should be positively related to housing value. The RM^2 form was found to provide a better fit than either the linear or logarithmic forms.

- AGE: proportion of owner units built prior to 1940.

- DIS: Weighted distances to five employment centres in the Boston region. According to traditional theories of urban land rent gradients, housing values should be higher near employment renters. DIS is entered in logarithm form.

- RAD: index of accessibility to radial highways. The highway access index was calculated on a town basis. RAD captures other sorts of location advantages besides nearness to workplace. It is entered in logarithmic form.

- TAX: Full value property tax rate $(\$/\$10000)$. Measures the cost of public services in each community.

- PTRATIO: Pupil-teacher ratio by town school district. Measures public sector benefits in each town. The relation of the pupil and teacher ratio to school quality is not entirely clear, although a low ratio should imply each student receives more individual attention.

- B: black proportion of population. At low to moderate levels of B, an increase in B should have a negative influence on housing value if Blacks are regarded as undesirable neighbours by Whites.

- LSTAT: proportion of population that is lower status $= \frac{1}{2}$ (proportion of adults without, some high school education and proportion of male workers classified as labourers). The logarithmic specification implies that socioeconomic status distinctions mean more in the upper brackets of society than in the lower classes.

Following the specifications of the authors [20], we consider the logarithm of some of the features and the squared value of RM. We also normalise the target values by using min-max feature scaling:

$$X' = \frac{X - min(X)}{max(X) - min(X)}.$$

which sets the value between 0 and 1. This way, as in the network we use the activation function `'relu'`, having the output values to be positive will help the network during the training process.

The dataset consists in 209 samples which we divide into training and testing set with a test size of 0.1. As we have an output label $MEDV$, we work with supervised learning methods. At first glance, there are some features whose values are mostly 0, so we decide to subtract these from the dataset, in order to reduce the complexity of the training process. These variables are CHAS and ZN.

We see in Figure 5.1 that the Charles river does not cross many cities of Boston, so this factor might not affect much the target value.

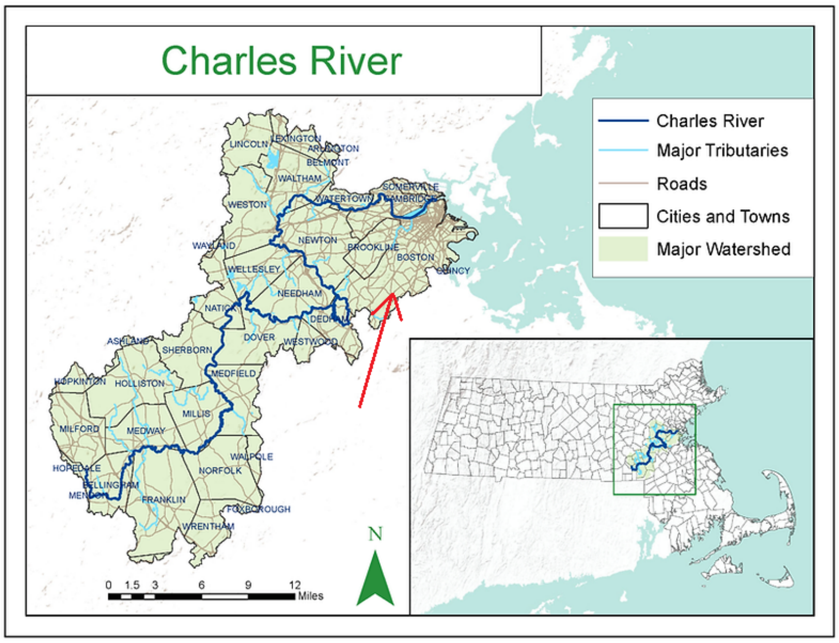

Figure 5.1: Map of Massachusetts and the Charles river. The arrow points to Boston. Source: [21].

On the other hand, the fact that ZN is 0 almost in every sample of data, means that there is no place for large house in most of the locations. This feature itself might have an important contribution to the house prices but as in the training data it is almost always 0, the network might consider the ones which are not outliers, which will imply a reduction of the accuracy. Thus, we have a total of 11 features.

5.2 Experiments

You can find the used codes for all the experiments in Appendix A.
First, we plot both the data and the logarithm of it, to check which is more stable. We use this version to train the network and the Gaussian process.

The next step is to train the neural network. We choose to build a fully-connected one by looking at the examples found on the literature, such as in [22]. We use `Python` language for this step. Regarding the particular neural network architecture, we choose it to have 3 layers and 150 neurons each. Given the number of samples that we have, we expect that 3 layers would be enough for the network to attain a validation loss of relative low order (with respect to the value of the data) and 150 neurons to be more than enough to attain the desired limiting behaviour.

```
model = keras.Sequential([
  layers.InputLayer(input_shape=(11,)),
  layers.Dense(150, activation='relu'),
  layers.Dense(150, activation='relu'),
  layers.Dense(150, activation='relu'),
  layers.Dense(1, activation='linear')
])

reduce_lr = ReduceLROnPlateau(monitor='loss', factor=0.1, patience=20, min_lr=1e
    -15)

initial_learning_rate = 0.001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=
    initial_learning_rate),
              loss='mse')
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=50,
    restore_best_weights=True)
history = model.fit(X_train,y_train,
          epochs=300,
          shuffle=True,
          batch_size=16, callbacks=[reduce_lr, early_stopping])
```

Listing 1: Neural network architecture.

Regarding the parameters of the network, we implement a function `reduce_lr` to make the network reduce the learning rate when there are signals of getting stuck. As optimizer we use a commonly used function `Adam`, which is already implemented in `keras`. When fitting (training) the model, we must specify our input data, the expected output, the number of epochs that the network will be working at maximum, the batch size and some complementary functions such as `shuffle`, which shuffle the sets in order to not be training with the same subset of the training set on every epoch and `callbacks` parameter, where we specify the extra conditions for the training process.

Once we have the model trained with the complete training set, we compute the Shapley values associated to it in order to select the features that will represent the reduced model. Each point on the `beeswarm` plot is a Shapley value for a feature and an instance (that is an element of our sample). The position on the y-axis is determined by the feature and on the x-axis by the Shapley value. The colour represents the value of the feature from low (blue) to high (red). Overlapping points are spread over in y-axis direction, so we get a sense of the distribution of the Shapley values per feature. The features are ordered according to their importance. In order to do so, SHAP calculates the average absolute Shapley value for each feature across all instances in the dataset. This value represents the average impact of the feature on the model's predictions, regardless of the direction (positive or negative).

We stress that these both concepts (Shapley value and feature value) are different given that we can have, for example, blue points with large absolute Shapley value. These points represent instances where the corresponding feature has a significant impact on the model's predictions. The fact that they are coloured blue suggests that these instances have low feature values. In other words, for these instances, the feature has a low value, but it still has a considerable impact on the model's output. See subsection 6.1, Figure 6.2 in order to visualise in an image this naive explanation.

Choosing the most important features basing our election on their Shapley values, we have the 2 nested models to compare, the complete and the reduced one.

Now, let us describe the process of constructing the covariance kernel for the GP. From Equation 2.21 we define $H0_k = \frac{H_{2k}(0)^2}{(2k+2)!}$ taking into account the definition of the Hermite numbers:

$$H_n(0) = H_n = \begin{cases} 0, & \text{if } n \text{ is odd} \\ (-1)^{n/2}(n-1)!!, & \text{if } n \text{ is even} \end{cases}$$

where $(n-1)!! = 1 \cdot 3 \cdot \ldots \cdot (n-1)$.

To construct the kernel we have to choose the variance of the weights and biases for the first step of the recursion. We set them to be: $C_\omega = 2.5$ and $C_b = 1$. We also need to choose how many terms of the infinite sum of Equation 2.21 we are going to use: it is a question of computational time and accuracy. We find an equilibrium by choosing the first 30 terms ($t0 = 30$, where the notation $t0$ is introduced in Appendix A), that is, the relative error of the GP predictions and the NN ones is considerably low with respect to the time it takes the training process of the GP.

On the other hand, as we are going to train the GP in `R`, we work on a `C` file for the definition of the GP, since it requires less computational work than compiling the recursion directly in `R`. To train the GP we must specify the number of hidden layers that the associated network has: $l = 3$, as we have already discussed.

```r
gpfit = gausspr(TARGET~., data=training,
                        type="regression",
                        kernel=K, #the kernel stored in C (KHC11)
                        kpar = list(l=3),
                        var = 0.003 # the initial noise variance
```

Listing 2: GP training.

Now, we can make the test to compare the variance of the errors (that is, the difference between the GP predictions and the actual values of the $MEDV$) associated to each GP (complete and reduced one). Substantially, what we want to compare are the Gaussian processes associated to each model. Thus, given the fact that both are centred, there is only left to compare their variances. We will use the test introduced in subsubsection 3.2.1 on the errors of the Gaussian processes. Thus, we check the normality of the errors of the Gaussian processes. In order to do so, we visualize the QQ-plot for a graphical answer and we perform a `Shapiro-Wilk` normality test, for a quantitative answer. The first approach shows the quantiles of the studied data plotted versus the quantiles of the standard normal distribution and the second approach is a test which has as a null hypothesis that the sample follows a normal distribution.

6 Results and discussion

Before training the neural network, we observe the behaviour of the target on the different observations. As it presents high variability, we decide to take the logarithm which is better-behaved (refer to Figure 6.1). We choose the logarithmic function to soften the behaviour since it is a smooth and monotonous function (we have taken into account that the target is strictly positive).

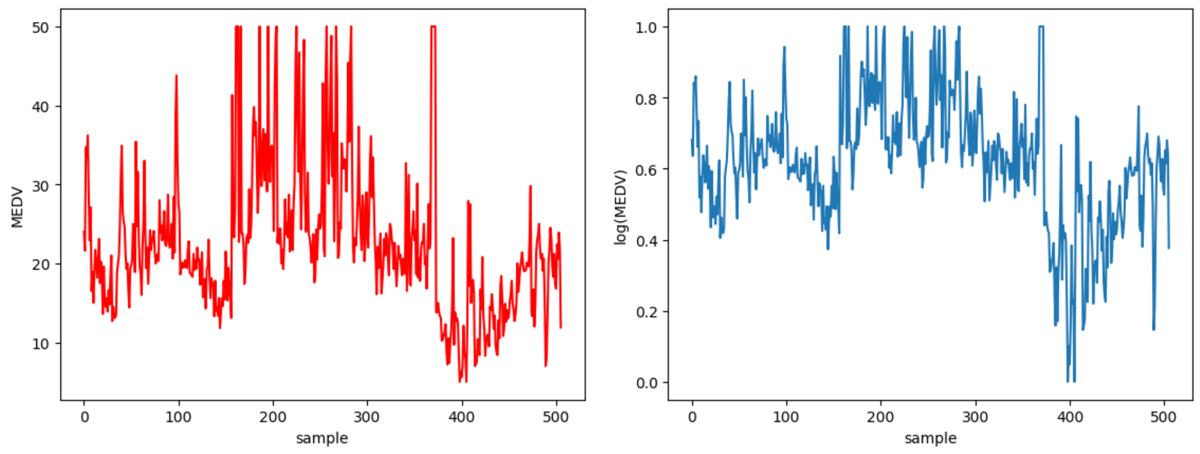

Figure 6.1: Target (MEDV) (left) and log(target) (right).

6.1 Neural network and feature selection

We train multiple times the network until it has converged, that is, until the loss function gets stabilised around a minimum. We obtain a training loss value: 0.0036 and a testing MSE of 0.445541.

With this network, we compute the Shapley values and the ranking of features by this criterion is the following (Figure 6.2):

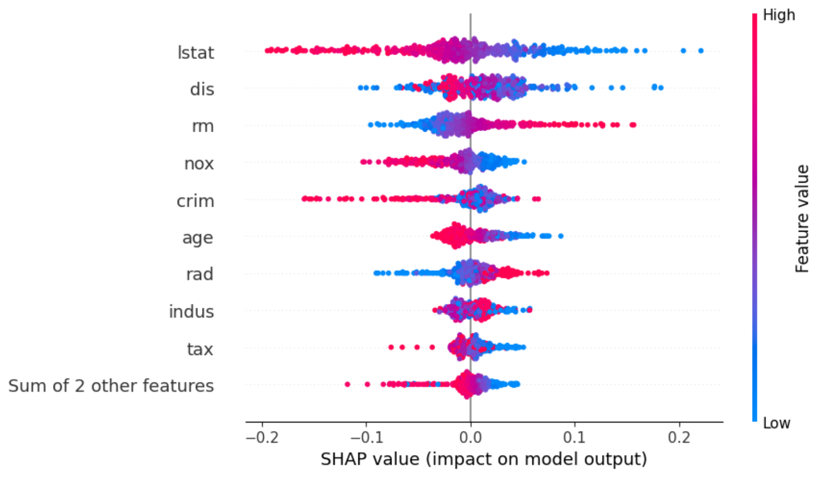

Figure 6.2: Shapley values distribution for each feature.

We see that "LSTAT" is by difference the most influential feature. Taking into account its definition, we might have data related to high socioeconomic environments where the status distinction is more marked. On second place, we can place "DIS" and "RM" features. This is in agreement with the previous argument in which we stated the expected high standard of living. "RM" is related to spacious houses which is something that rich people tend to search for. On the other hand, when someone is concerned about the distance to their workplace is because they do have a job.

The next 2 features that we will consider to be in the reduced model are "NOX" and "CRIM". The fact that the crime rate by town is not the most influential feature suggests that people might have private security guards or maybe that US people have normalized criminality in towns. Also, the fact that the quantity of nitrogen oxide concentrations is a main concern in people minds, make us believe that they might not have other common worries such as taxes ("TAX"), which happens to be on the tail of the classification.

6.2 Gaussian process and Morgan-Pitman test

We train both the GP for the complete model and for the reduced one with the features: "LSTAT", "DIS", "RM", "NOX" and "CRIM". We test the GPs with the test data so we can compare the predictions of the neural network with the error of the GP. From Table 6.1 we see that the errors of

Approach	Complete Model	Reduced Model
NN	0.435541	0.497316
GP	0.409429	0.563343

Table 6.1: Comparison of errors between NN and GP predictions.

both approaches are substantially the same. We stress the fact that the GP represents the limiting behaviour of the network, which means that one should expect it to have lower error than the NN predictions. For the complete model, we obtain this result and we see that with 150 neurons in each layer we have obtained a relative error of only 6.3%. Regarding the reduced model although we also obtain errors of similar values, we see that the GP has greater error than its associated neural network. It might seem somehow strange, however, we should also consider that for the construction of the covariance kernel matrix of the GP we have also made an approximation of the Hermite expansion into a finite sum. We retrain the GP for this model with more terms of the Hermite expansion ($t0 = 50$ instead of $t0 = 30$) and we obtain an error of 0.563242, which does not represent a significant improvement. Maybe, with even more terms of this expansion we could have obtained more accurate results, but this will imply a noticeable increasing of the computational time required and considering that the training process of the network is already faster than the one for the GP, it might not be worthy. We can conclude that using the GP for further analysis rather than its associated neural network is a ground-based decision.

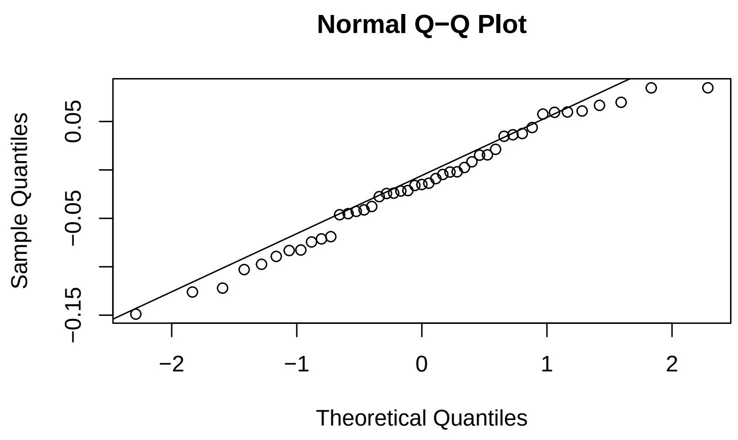

Figure 6.3: Q-Q plot of errors of the complete model.

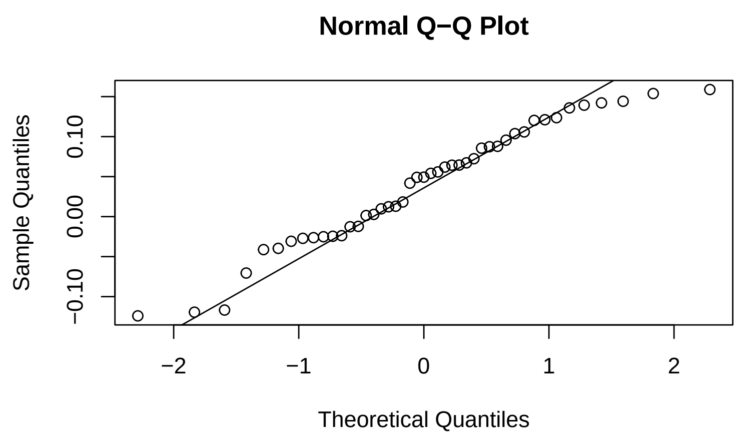

Figure 6.4: Q-Q plot of errors of the reduced model.

This way, we check the normality of both GP errors by visualizing the QQ-plot. From Figure 6.3 and Figure 6.4 we see that the quantiles of the errors adapt satisfactory upon the quantiles of a standard normal distribution, thus, indicating that they might be distributed following one.

By choosing a level of significance of $\alpha = 0.05$, we cannot reject the null hypothesis of the Shapiro-Wilk test, that is, we can consider that both errors are normally distributed, since we obtain p-values greater than α:

$$e1 : \text{p-value} = 0.4518 \qquad e2 : \text{p-value} = 0.1362 \tag{6.1}$$

Thus, we can perform the Morgan-Pitman test.

```
Paired Pitman-Morgan
test data:  e1 and e2: p-value = 0.1135
alternative hypothesis: true ratio of variances is not equal to 1
```

Listing 3: p-value of the Morgan-Pitman test.

By choosing the same α, we can not reject the null hypothesis that the variance of both errors is equal.

Thus, we conclude that we are not losing information when working with the reduced model for regression task, since both GP are equivalent (at a confidence level of 95%). Furthermore, this also mean that we can work with the already trained neural network for the reduced models, which happens to be the one with smaller error of all 4 approaches.

7 Conclusion

We have analyzed the behavior of neural networks in the limit where the nodes in every layer tend to infinity. We have delved into the neural network performance using the Shapley value tool, in order to search for the most relevant features in its predictions. Additionally, we introduced the Morgan-Pitman test to perform model selection, by choosing as a reduced model the one selected by the features with the highest Shapley value. After completing the theoretical research, we applied this knowledge to a particular example where we evaluated whether it is suitable for our data to work with fewer input parameters or if each one of them is useful for further analysis. Feature reduction is a great tool to interpret more easily what deep learning techniques, such as neural networks, decide to learn during the training process and, thus, provide them with more appropriate input data for further predictions.

Theoretically, we introduce some theorems that let us conclude that the limiting behavior of a neural network corresponds to that of a Gaussian process with zero mean and covariance kernel given by a recursion based on the variances of the weights and biases of the neural network (prior to training) as well as on the activation function. We explore the description of both statistical tools and finally, we relate them. Moreover, we have coded both approaches and compared their errors on the predictions of test datasets. We obtained a relative error of 6% and 15% for the complete model and the reduced one, respectively. We consider the differences to be caused by the fact that we are neither studying the real limit $h_\mu \to \infty$ nor considering the infinite terms of the Hermite expansion for the elements of the covariance kernel matrix. However, the errors of both models do not differ significantly, so we can conclude that the expected limiting behavior of the neural network is attained. From these results, we derive another conclusion: the main cause for the theoretical performance and the experimental results to differ is the required computational time to achieve the theoretical limit.

Additionally, we compared the two nested models for checking whether less information is enough to reproduce similar predictions for the GP. By setting a level of confidence of 95%, we conclude that errors from both models have equal variances, which leads to the consideration that both models follow the same GP. This way, we can work with the reduced model instead of the complete one, which implies a reduction of the needed input information as well as of the computational time required.

As a further line of research, we plan to investigate the tangent kernel approach thoroughly. From the already trained GP, we want to derive the covariance kernel and add the necessary terms to the definition of the recursion. Let us note that making some modifications to the bootstrap method could have been useful, but as we have found another approach whose distribution is well-known, we have decided to implement the latter.

Bibliography

[1] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133 (cit. on p. 5).

[2] Andrea M Tonello et al. “Machine learning tips and tricks for power line communications”. In: *IEEE Access* 7 (2019), pp. 82434–82452 (cit. on p. 8).

[3] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006 (cit. on p. 10).

[4] Olav Kallenberg et al. *Probabilistic symmetries and invariance principles*. Vol. 9. Springer, 2005 (cit. on p. 12).

[5] Julius R Blum et al. “Central limit theorems for interchangeable processes”. In: *Canadian Journal of Mathematics* 10 (1958), pp. 222–229 (cit. on pp. 12, 13).

[6] Alexander G de G Matthews et al. “Gaussian process behaviour in wide deep neural networks”. In: *arXiv preprint arXiv:1804.11271* (2018) (cit. on pp. 14, 15).

[7] Argimiro Arratia, Alejandra Cabaña, and José Rafael León. “Deep and Wide Neural Networks Covariance Estimation”. In: *International Conference on Artificial Neural Networks*. Springer. 2020, pp. 195–206 (cit. on p. 14).

[8] George Neville Watson. “Notes on generating functions of polynomials:(2) Hermite polynomials”. In: *Journal of the London Mathematical Society* 1.3 (1933), pp. 194–199 (cit. on p. 15).

[9] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. “Activation functions in neural networks”. In: *Towards Data Sci* 6.12 (2017), pp. 310–316 (cit. on p. 16).

[10] Lloyd S Shapley et al. “A value for n-person games”. In: (1953) (cit. on p. 18).

[11] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 19).

[12] Enrique M. Cabaña. “F Distribution”. In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Springer, 2011, pp. 499–501. DOI: 10.1007/978-3-642-04898-2_268 (cit. on p. 20).

[13] WA Morgan. “A test for the significance of the difference between the two variances in a sample from a normal bivariate population”. In: *Biometrika* 31.1/2 (1939), pp. 13–19 (cit. on p. 21).

[14] EJG Pitman. “A note on normal correlation”. In: *Biometrika* 31.1/2 (1939), pp. 9–12 (cit. on p. 21).

[15] Jerzy Neyman and Egon Sharpe Pearson. “Contributions to the theory of testing statistical hypotheses.” In: *Statistical research memoirs* (1936) (cit. on p. 21).

[16] Charles E McCulloch. “Tests for equality of variances with paired data”. In: *Communications in statistics-theory and methods* 16.5 (1987), pp. 1377–1391 (cit. on p. 21).

[17] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. Chapman and Hall/CRC, 1994 (cit. on p. 22).

[18] Anthony Christopher Davison and David Victor Hinkley. *Bootstrap methods and their application*. 1. Cambridge university press, 1997 (cit. on p. 22).

[19] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural tangent kernel: Convergence and generalization in neural networks". In: *Advances in neural information processing systems* 31 (2018) (cit. on p. 23).

[20] David Harrison Jr and Daniel L Rubinfeld. "Hedonic housing prices and the demand for clean air". In: *Journal of environmental economics and management* 5.1 (1978), pp. 81–102 (cit. on p. 25).

[21] Anna Renkert. *Charles River*. `https://www.massriversalliance.org/charles-river`. Accessed: May 2024 (cit. on p. 25).

[22] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021 (cit. on p. 25).

A Codes

```
from google.colab import files
import csv
import math
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
import pickle
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
from tensorflow.keras import layers, losses
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import ReduceLROnPlateau

#data
data = pd.read_csv('boston.csv', parse_dates=True)
feat = pd.concat([np.log(data['CRIM']), np.log(data['NOX']), np.square(data['RM'])
    ,  data['DIS'], np.log(data['LSTAT'])],axis=1)
feat.columns = ['crim','nox', 'rm', 'dis', 'lstat']
names = feat.columns

target = data['MEDV']
target = np.log(target)
target = (target-np.min(target))/(np.max(target)-np.min(target))
feat = feat[0:len(target)]
T = len(target)
p=0.9
T_trn = int(p*T)

X_train = feat[0:T_trn]
X_test = feat[T_trn:]
y_train = target[0:T_trn]
y_test = target[T_trn:]


#NN architecture
model = keras.Sequential([
  layers.InputLayer(input_shape=(11,)),
  layers.Dense(150, activation='relu'),
  layers.Dense(150, activation='relu'),
  layers.Dense(150, activation='relu'),
  layers.Dense(1, activation='linear')
])

reduce_lr = ReduceLROnPlateau(monitor='loss', factor=0.01, patience=20, min_lr=1e
    -15)

initial_learning_rate = 0.001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=
    initial_learning_rate),
              loss='mse')
optimizer = tf.keras.optimizers.Adam()
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=50,
    restore_best_weights=True)
history = model.fit(X_train,y_train,
          epochs=300,
          shuffle=True,
          batch_size=16, callbacks=[reduce_lr, early_stopping])

#Shapley values
X = shap.utils.sample(X_train, 10)
explainer_ebm = shap.Explainer(model.predict, X)
shap_values_ebm = explainer_ebm(X_train)
```

Listing 4: Python code. Neural network architecture and Shapley value plot.

```
to= 50
def fact_odd(n):
  fact = 1
  for k in range(n+1):
    if (k==0):
      fact = 1
    else:
      fact *= (2*k-1)
  return fact
H0=np.zeros(to)
for i in range(to):
  H0[i] = (fact_odd(i))**2/(math.factorial(2*i+2))
```

Listing 5: Python code. Definition of H0.

```
// [[Rcpp::plugins(cpp11)]]
#include <Rcpp.h>
#include <numeric>
#include<math.h>
#include<iostream>
using namespace Rcpp;
using namespace std;


// [[Rcpp::export]]
double meanC(NumericVector x) {
  int n = x.size();
  double total = 0;

  for(int i = 0; i < n; ++i) {
    total += x[i];
  }
  return total / n;
}

// [[Rcpp::export]]
double prodC(NumericVector x){
  double total=1;
  NumericVector::iterator it;
  for(it = x.begin(); it != x.end(); ++it) {
    total *= *it;
  }
  return total;
}

// [[Rcpp::export]]
double prodC11(NumericVector xp){
  double total=1;
  for(const auto &x:xp) {
    total *= x;
  }
  return total;
}

// [[Rcpp::export]]
double innerprod(NumericVector& x, NumericVector& y){
  double sum=0;
  for(int i=0; i<x.size();i++){
    sum +=x[i]*y[i];
  }
  return sum;
}

#created from the above-mentioned code
const double H0[] ={
    5.00000000e-01, 4.16666667e-02, 1.25000000e-02, 5.58035714e-03,
       3.03819444e-03, 1.86434659e-03, 1.23948317e-03, 8.72802734e-04,
       6.41766716e-04, 4.88080476e-04, 3.81378900e-04, 3.04688578e-04,
       2.47969627e-04, 2.05001345e-04, 1.71776989e-04, 1.45629484e-04,
       1.24732562e-04, 1.07804571e-04, 9.39264577e-05, 8.24264876e-05,
       7.28052774e-05, 6.46858728e-05, 5.77797965e-05, 5.18635141e-05,
       4.67618378e-05, 4.23360380e-05, 3.84752043e-05, 3.50898860e-05,
       3.21073518e-05, 2.94680187e-05, 2.71227322e-05, 2.50306762e-05,
       2.31577515e-05, 2.14753073e-05, 1.99591417e-05, 1.85887077e-05,
       1.73464782e-05, 1.62174355e-05, 1.51886571e-05, 1.42489791e-05,
       1.33887201e-05, 1.25994539e-05, 1.18738219e-05, 1.12053766e-05,
       1.05884513e-05, 1.00180510e-05, 9.48975982e-06, 8.99966367e-06,
       8.54428410e-06, 8.12052213e-06
};


// [[Rcpp::export]]
double KHC11(NumericVector& a, NumericVector& b,int l=3)
{
  const double  Cw=2.5, Cb=1.0;
  const int to=30;
  double sigma1, sigma2;
  int i;
  NumericVector vro(to),vherm(to);
  if (l == 0)
  {
    return innerprod(a,b)*Cw/a.size() + Cb;
  }
  else
  {
    double ro = KHC11(a,b,l-1)/sqrt(KHC11(a,a,l-1)*KHC11(b,b,l-1));
    sigma1 = pow(KHC11(a,a,l-1),1/2.);
    sigma2 = pow(KHC11(b,b,l-1),1/2.);
    for(int i=0;i<to;i++)
    {
        vro[i]=pow(ro,2*i+2);
    }
    for(int i=0;i<to;i++)
    {
        vherm[i]=H0[i];
    }
    return sigma1*sigma2*(1/(4.0*M_PI) + 1/4*ro + 1/(2*M_PI)*innerprod(vherm, vro)
   );
  }
}
```

Listing 6: C code. GP Kernel.

```
fileloc <- dirname(rstudioapi::getSourceEditorContext()$path)
setwd(fileloc)
rm(fileloc)
library(kernlab)
library(MASS)
library(plyr)
library(reshape2)
library(ggplot2)
```

```r
library(Rcpp)
library(bench)
library(fBasics)
install.packages("PairedData")
library("PairedData")



sourceCpp("C-codes/kernelsC.cpp")
K<-KHC11

class(K) <- 'kernel'

# Cov matrix for kernel:
SigmaK <- function(X1,X2) {
  Sigma <- matrix(0, length(X1),length(X2))

  for (i in 1:length(X1)) {
    for (j in 1:length(X2)) {
      Sigma[i,j] <- K(as.vector(X1[i]),as.vector(X2[j]),3)
    }
  }
  return(Sigma)
}


#Data: BOSTON
boston = read.csv2(file='data/Boston.csv', sep=',', header=T)

#Target
target <- log(as.numeric(boston$MEDV))
target = (target-min(target))/(max(target)-min(target))

#Complete model
feat= cbind(log(as.numeric(boston$CRIM)), log(as.numeric(boston$INDUS)), log(as.
    numeric(boston$NOX)), (as.numeric(boston$RM))^2, log(as.numeric(boston$AGE)),
    as.numeric(boston$DIS), as.numeric(boston$RAD), log(as.numeric(boston$TAX)),
    log(as.numeric(boston$PTRATIO)), log(as.numeric(boston$B)), log(as.numeric(
    boston$LSTAT)))
dataset = cbind(feat,target)
colnames(dataset) = c("CRIM","INDUS","NOX","RM", "AGE", "DIS", "RAD", "TAX", "
    PTRATIO", "B", "LSTAT", "TARGET")

T<-nrow(dataset)
p=0.9
T_trn <- round(p*T)
trainindex <- 1:T_trn

## data frames
training = as.data.frame(dataset[trainindex,])
testing = as.data.frame(dataset[-trainindex,])
rownames(training) = NULL
rownames(testing) = NULL

system.time({
  gpfitfull = gausspr(TARGET~., data=training,
                      type="regression",
                      kernel=K,
                      kpar = list(l=3),
                      var = 0.003)
})

#Reduced model
system.time({
  gpfitred = gausspr(TARGET~., data=training[, c(1,3,4,6,11,12)],
                      type="regression",
                      kernel=K,
                      kpar = list(l=3),
                      var = 0.003)
})


#errors
e1=testing[,"TARGET"]-predict(gpfitfull, testing)
qqnorm(e1)
qqline(e1)
shapiro.test(e1)

e2=testing[,"TARGET"]-predict(gpfitred, testing[, c(1,3,4,6,11,12)])
qqnorm(e2)
qqline(e2)
shapiro.test(e2)

(sum(e1^2))^(1/2)
(sum(e2^2))^(1/2)


pitman.morgan.test.default(e1,e2, alternative = 'two.sided', ratio=1)
```

Listing 7: R Code. GP and Morgan-Pitman test.