Stockholms universitet

Boosting Regression Models With Neural Networks, Because We CANN

Luciano Egusquiza Castillo

Masteruppsats i försäkringsmatematik
Master Thesis in Actuarial Mathematics

Masteruppsats 2025:13
Försäkringsmatematik
Juni 2025

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2025:13**
http://www.math.su.se

Boosting Regression Models With Neural Networks, Because We CANN

Luciano Egusquiza Castillo*

June 2025

Abstract

The purpose of this thesis is to gain an understanding of neural networks and how they can be used to boost regression models. A theoretical foundation is presented, covering both generalized linear models (GLMs) and feed-forward neural networks (FNNs). This is followed by application in a non-life actuarial environment using the well–studied dataset `freMTPL2freq`, which contains French insurance data. GLMs are used as cornerstone models and are compared with feed-forward neural networks FNNs. Subsequently, the two are combined into a combined actuarial neural network (CANN) model in an attempt to boost the initial GLM model via skip connection, with successful results.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: Luciano.EC90@gmail.com. Supervisor: Filip Lindskog.

Sammanfattning

Syftet med denna uppsats är att skapa en förståelse för neuronnät och hur de kan användas för att förbättra regressionsmodeller. En teoretisk grund presenteras, som täcker både generaliserade linjära modeller (GLM:er) och neuronnät (FNN:er). Därefter följer en tillämpning inom aktuariell sakförsäkring på det välstuderade datasetet `freMTPL2freq`, som innehåller fransk försäkringsdata. GLM:er används som grundmodeller och jämförs med FNN:er i termer av prediktiv förmåga. Därefter kombineras de två till ett sammansatt aktuariellt neuronnät (CANN) i ett försök att förbättra den ursprungliga GLM-modellen via genvägskoppling med lyckosamma resultat.

Preface

This thesis ought to satisfy the degree project corresponding to 30 ECTS credits for a degree of Master of Science (*filosofie master*) in actuarial mathematics at the Department of Mathematics at Stockholm University.

I want to extend my deepest gratitude to my supervisor, Professor Filip Lindskog, for his support, insightful guidance, and exceptional mentorship throughout the writing of this thesis. For always being humble, yet giving the necessary push when needed. Thank you, Filip.

Amanuensis Leo G. Levenius, for your grammatical and mathematical support — and for persistently reminding me that patience is, indeed, a virtue I am yet to master. *Med all vördnad och tacksamhet, tillägnas detta dig, min kära vän.*

Amanuensis Jack Zhan, for your deep mathematical intuition, your patience when discussing topics, having to explain them once or twice too many times without ever complaining when I turned to you with a confused look on my face. 谢谢你，我的兄弟。

Contents

Chapter 1

Introduction

1.1 History of Insurance and Regression Modeling

What sparked the foundations of insurance as we know it today was the need to protect merchants from financial losses during risky trade expeditions. In a more general sense, insurance met the need to protect individuals or companies financially if certain unpredictable events were to occur. For instance, if one's house were to burn down, the financial repercussions of such an event would be devastating to a single individual, resulting in financial ruin.

However, having one's house burn down is a highly unlikely event. Therefore, if the risk of such an event is shared amongst a large group of individuals, the loss for each of the individuals will be tolerable. In fact, the need for such protection gave birth to one of the first fire insurance companies [5], dating back to 1705.

In terms of regression modeling as we know it today, the *Gaussian linear regression* was developed by Gauss and Legendre and published in *Nouvelles méthodes pour la détermination des orbites des comètes raditional* in 1805 [16].[1] Gaussian linear regression became a staple in the actuarial science until *generalized linear models* (GLMs) were developed, in the 1970s. They were introduced by Nedler and Wedderburn into the actuarial world [24] and were further explored upon by McCullagh and Nedler in their published book [20] Till this day, GLMs continue to be a staple tool for actuaries and statisticians in general.

1.2 Background

An insurance policy is a contract agreed upon between an insurance company, the insurer, and a policyholder, the customer. The insurance policy could be considered as a commitment from the insurer's behalf to offer the policyholder economic compensation in the case of certain unpredictable events occurring during a specific time period, usually one year. In compensation for this, the insurance company charges a fee, this fee is what is referred to as the *premium*.

The essential basis of non-life insurance pricing lies in the ability to determine the price

[1] For the historically curious reader, this is covered in more detail in *The history of statistics* [34].

of an insurance policy. This is done by taking into consideration various properties of the object in question in need of insurance, but also the policyholder. These properties are referred to as *covariates* or *features*, in a modeling context.

This is where an actuary comes in. An actuary's main task within non-life pricing is to assess data and apply the appropriate methods and models in order to achieve the best results. That is, to determine the premium of an insurance policy as accurately as possible in a market-competitive and risk-assessed manner.

1.3 Models and Methods

As mentioned, an actuary's main task within insurance pricing is to determine the premium of an insurance policy. This is done by *regression*, which essentially means modeling a relationship between an outcome, or *response*, and some independent variables, the features. An actuary is more often interested in using regression for predictive purposes, i.e. to model future, unknown premiums for insurance contracts not yet to be signed. In other words, given some features $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, predict the unobserved response y by the estimation $\hat{y}$ using regression models.

The GLMs are nowadays considered a backbone within the actuarial field, mainly because they are well-studied, easy to use, and highly interpretable. There are, of course, downsides to these models. GLMs are bound by their linear limitations, and when working with GLMs, the treatment of interactive terms is often a tedious and time-consuming task. Interactivity and non-linear patterns may be important to capture in order to increase one's predictive power, which is why more modern methods may be suitable alternatives.

The *generalized additive models* (GAMs) are also commonly used. These models replace the features of the GLM with smooth functions [9], allowing for more non-linear patterns to be caught in data, which alleviates some of the concerns of the GLM.

The increased abundance and complexity of data have made models requiring less actuarial manual intervention popular, in particular machine learning. Methods such as gradient boosting machines (GBMs) [28] and *neural networks* (NNs) have risen in popularity. M. V. Wüthrich and M. Merz write about an array of methods in *Statistical Foundations of Actuarial Learning and its Applications* [37]. In this thesis, NNs, as well as, in particular *combined actuarial neural networks* (CANN), are studied and evaluated. The concept of CANN was initially proposed by Wüthrich and Merz in *Yes, we CANN!* [36]. The purpose of the CANN is to use neural networks to boost traditional models such as GLMs to achieve better predictive power while still maintaining a degree of interpretability.

1.4 Pure Premium, Claim Frequency, and Claim Severity

The following section is heavily inspired by F. Lindskog, who covers the subject in more detail in *Non-life pricing essentials* [19].

In a more mathematical and formal manner, we may represent an insurance contract by a triplet $(\boldsymbol{X}, Z, V)$, where $\boldsymbol{X} \in \mathcal{X}$ is the feature vector, $\mathcal{X}$ the feature space, Z is the claim cost, and V is the contract duration. The feature vector $\boldsymbol{X}$ is unknown until a contract

is agreed upon, at which point it collapses to some non-random vector $\boldsymbol{x}$ consisting of the observed features of the insured and the object in question. We let $\pi(\boldsymbol{x})$ denote the function mapping the observed features $\boldsymbol{x}$ to a one-year premium,

$$\pi : \mathcal{X} \to \mathbb{R}_{\geq 0}, \tag{1.1}$$

$$\boldsymbol{x} \mapsto \pi(\boldsymbol{x}). \tag{1.2}$$

Furthermore, if a contract with features $\boldsymbol{x}$ is assigned the one-year premium $\pi(\boldsymbol{x})$, we assume that the earned premium is $V\pi(\boldsymbol{x})$, where V is a observed contract duration.

If the expected value of the earned premium, $V\pi(\boldsymbol{x})$ is equivalent to the expected value of the claim cost, we say it is a *fair actuarial premium*, i.e. if

$$\mathbb{E}[V\pi(\boldsymbol{X}) \mid \boldsymbol{X}] = \mathbb{E}[Z \mid \boldsymbol{X}]. \tag{1.3}$$

What this states is that the expected earned premium should be equal to the expected claim cost given available information $\boldsymbol{X}$ at the time the contract is agreed upon. We note that, since $\pi(\boldsymbol{X})$ is conditioned on $\boldsymbol{X}$, $\pi(\boldsymbol{X})$ is considered non-stochastic. Consequently, Equation (1.3) may be re-express as

$$\mathbb{E}[V\pi(\boldsymbol{X}) \mid \boldsymbol{X}] = \mathbb{E}[V \mid \boldsymbol{X}]\pi(\boldsymbol{X}) = \mathbb{E}[Z \mid \boldsymbol{X}], \tag{1.4}$$

meaning the one-year premium can be expressed as

$$\pi(\boldsymbol{X}) = \frac{\mathbb{E}[Z \mid \boldsymbol{X}]}{\mathbb{E}[V \mid \boldsymbol{X}]}. \tag{1.5}$$

Suppose Z can be expressed as a sum of individual claims, i.e., $Z = \sum_{j=1}^{N} C_j$, where C_j represents claim j. Also, suppose that the number of claims N is, conditioned on $\boldsymbol{X}$ and V, independent of the sequence of claim sizes (C_j). Lastly, suppose that (C_j) is a sequence of independent and identically distributed random variables when conditioned on $\boldsymbol{X}$, independent of V. Then

$$\begin{aligned}
\mathbb{E}[Z \mid \boldsymbol{X}] &= \mathbb{E}\Big[\mathbb{E}[Z \mid V, \boldsymbol{X}] \mid \boldsymbol{X}\Big] \\
&= \mathbb{E}\Big[\mathbb{E}[N \mid V, \boldsymbol{X}]\mathbb{E}[C \mid V, \boldsymbol{X}] \mid \boldsymbol{X}\Big] \\
&= \mathbb{E}\Big[\mathbb{E}[N \mid V, \boldsymbol{X}]\mathbb{E}[C \mid \boldsymbol{X}] \mid \boldsymbol{X}\Big] \\
&= \mathbb{E}\Big[\mathbb{E}[N \mid V, \boldsymbol{X}] \mid \boldsymbol{X}\Big]\mathbb{E}[C \mid \boldsymbol{X}\Big] \\
&= \mathbb{E}[N \mid \boldsymbol{X}]\mathbb{E}[C \mid \boldsymbol{X}] = \pi_N(\boldsymbol{X})\pi_C(\boldsymbol{X}).
\end{aligned}$$

This factorization is important. It means we can estimate two expectations separately, since otherwise, large variability in claim size would introduce noise, leading to less accurate estimates. This is why, when modeling for pure premium, actuaries often model claim frequency and claim severity separately. That is

$$\text{Pure premium} = \text{claim frequency} \times \text{claim severity}.$$

This implies that π_N is the theoretical function that solves the minimization problem

$$f^* = \arg\min_f \mathbb{E}[V \cdot L(\frac{N}{V}, f(\boldsymbol{X}))], \tag{1.6}$$

where L is a loss function, which is a function used to measure and penalize deviances between actual observations and approximations made by a function f. Empirically, we estimate the right-hand side of Equation (1.6) by the mean of the loss over some sample of data

$$\hat{f}^* = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_i^n v_i L(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}_i)), \tag{1.7}$$

where $\boldsymbol{\theta}$ is the parameter vector. In other words, we find a function $f \in \mathcal{F}$ that approximates f^*, where $\mathcal{F}$ is a set of sufficiently many functions.

In Lindholm *et al.* [18] this is explored upon further, as well as an alternative, weight-adjusted probability measure, $\mathbb{P}_v$, albeit it will not be covered in this thesis.

1.5 Key Ratios and Tariffs

The pure premium is called a *key ratio* [27, p. 6]. A key ratio, $\tilde{Y}$, is a ratio between the outcome of a random variable, Y, and a volume measure, V, commonly referred to as an *exposure* such that $\tilde{Y} = \frac{Y}{V}$. In the case of pure premium, the duration is the exposure. In the case of claim frequency, Y denotes the number of claims, V the duration, and in the case of claim severity, Y is the total claim cost and V is the number of claims.

When modeling, as previously mentioned, an actuary observes several relevant properties of the policyholder as well as the insured object, which are referred to as *features*. Using these features, an actuary computes relevant key ratios and presents them in a table, the *tariff*. Features may be categorical by nature, or they may be formed by dividing variables into classes, e.g., age into age intervals. The main goal of tariff analysis is to gain an understanding of how a key ratio, $\tilde{Y}$, behaves under variations of the different features.

Table 1.1: An example of a simple tariff using two features.

Class	Age	Duration	No. Claims	Claim Frequency	Claim Severity	Pure Premium	Actual Premium
1	1	62.9	17	270	18,256	4,936	2,049
1	2	112.9	14	124	13,632	845	1,230
1	3	133.1	9	68	20,877	1,411	762
1	4	376.6	7	19	13,045	242	396
2	1	4.4	1	228	8,018	1,829	396
2	2	352.1	52	148	8,232	1,216	1,229
2	3	840.1	69	82	7,418	609	738
2	4	1378.3	75	54	7,318	398	457

An example of how a tariff could look is shown in Table 1.1. This tariff is a simplified, non-accurate tariff inspired by one based on real data [27, p. 5]. The feature Class has two different groups, 1 or 2. Age is divided into 4 sub-intervals, each represented by a group.

Chapter 2

Generalized Linear Models

While Gaussian linear regression has long served as a foundational tool in statistical modeling, its applicability is often limited due to a set of restrictive assumptions. Generalized Linear Models (GLMs) were introduced as a more flexible alternative that relaxes many of these constraints. Specifically, Gaussian linear regression relies on the following assumptions:

- Linear regression models assume homoscedasticity, i.e., the residuals ϵ_i, have constant variance and for all i.
- Normality and zero mean of residuals is also assumed for all i

$$\epsilon_i \sim N(0, \sigma^2).$$

- The expected value is a linear function of the features $\boldsymbol{x}$. This means the identity link is assumed to be the appropriate link-function, see Section 2.5 for details.

2.1 Exponential Dispersion Models

GLMs relax the assumption of a normal distribution; rather, GLMs work with a general class of distributions depending on what the presumed response distribution is. These are called the *exponential dispersion models* (EDMs) [27, pp. 16–25]. The term exponential dispersion model was introduced by Jørgensen [14] but stems from a much older idea [38]. Some of the common distributions amongst the EDMs are normal, Poisson, gamma, and negative binomial. The probability distribution of an EDM is given by

$$f_{Y_i}(y_i; \theta_i, \phi) = \exp\left\{\frac{y_i\theta_i - b(\theta_i)}{\phi/v_i} + c(y_i, \phi, v_i)\right\}. \tag{2.1}$$

Where y_i is the observed outcome of Y_i, θ_i is the canonical parameter, ϕ is the dispersion parameter, and b is the cumulant function. The function c is of little interest in GLM theory [27, p. 17]. The EDMs also have the convenient property that

$$\mathbb{E}[Y] = \mu = b'(\theta), \tag{2.2}$$

$$\mathrm{Var}(Y) = \mathscr{V}(\mu)\phi/v. \tag{2.3}$$

The function $\mathscr{V}(\cdot)$ is known as the *variance function* [27, p. 23] and expresses the relationship between the mean and the variance of a response, Y, following an EDM distribution.

$$\mathscr{V} : \mathbb{R} \to \mathbb{R}_{\geq 0}, \tag{2.4}$$

$$\mu \mapsto \mathscr{V}(\mu) = b''(b'^{-1}(\mu)). \tag{2.5}$$

2.2 Probability Distribution of Claim Frequency

The following presentation coincides with that of Ohlsson and Johansson [27, pp. 18–20]. Let $N(t)$ be the number of claims for an individual policy within the time interval $[0, t]$, with $N(0) = 0$. The counting process $\{N(t); t \geq 0\}$ is called the *claims process*. Cramér [4, pp. 9–11] showed that under certain assumptions, the claims process is a *Poisson process*. The theoretical details of the Poisson process is covered by Levenius [17, p. 19]. Thus, it is reasonable to assume a Poisson distribution for the number of claims of an individual policy during any given period of time. What remains of desire is to express the frequency function in the form of Equation (2.1). If this is possible, it validates that it is an EDM and thus can be modeled using GLMs. Letting Y_i be the number of claims for observation i, with the duration V_i as exposure, the claim frequency is $\tilde{Y}_i = Y_i/V_i$. As a result, the frequency function for $\tilde{Y}_i$ is

$$\begin{aligned} f_{\tilde{Y}_i}(\tilde{y}_i; \mu_i) &= \mathbb{P}(\tilde{Y}_i = \tilde{y}_i) = \mathbb{P}(Y_i = v_i\tilde{y}_i) = e^{v_i\mu_i}\frac{(v_i\mu_i^{v_i\tilde{y}_i})}{(v_i\tilde{y}_i)!} \\ &= \exp\{\frac{\tilde{y}_i\log(\mu_i) - \mu_i}{1/v_i} + v_i\tilde{y}_i\log(v_i) - \log(v_i\tilde{y}_i!)\}, \end{aligned} \tag{2.6}$$

parameterizing with $\theta_i = \log(\mu_i)$ yields

$$f_{\tilde{Y}_i}(\tilde{y}_i; \theta_i, \phi) = \exp\{\frac{\tilde{y}_i\theta_i - e^{\theta_i}}{1/v_i} + v_i\tilde{y}_i\log(v_i) - \log(v_i\tilde{y}_i!)\}. \tag{2.7}$$

This is an EDM with $c(\tilde{y}_i, v_i) = v_i\tilde{y}_i\log(v_i) - \log(v_i\tilde{y}_i!)$, $\theta_i = \log(\mu_i)$ and $b(\theta_i) = e^{\theta_i}$. Here, i denotes tariff cell i having data organized in list form[1].

2.3 Probability Distribution of Claim Severity

By Ohlsson and Johansson [27, pp. 20–21], we drop the index i for simplicity. Here, Y is the total claim cost for a cell, and the exposure, V, is the number of claims. One relevant detail that is not always obvious is that we condition on the number of claims when modeling for claim severity; given this conditioning, V is non-random[2].

The argument for which distribution to assume for claim severity is not as straightforward as in the case of claim frequency. Data tends to be positive and skewed to the right,

[1] List form is explained in Section 2.4

[2] This holds for the remainder of this thesis. Unless stated otherwise, we condition on the exposure, $V = v$.

but several distributions fulfill this behavior. That being said, nowadays it is more or less standard to assume that claim severity follows a gamma distribution [23, p. 10].

Assuming that the cost of an *individual claim* is gamma distributed, i.e., letting $v = 1$, and using index parameter $\alpha > 0$, scale parameter $\beta > 0$ the frequency function for an individual claim is

$$f(y) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} e^{-\beta y};\ y > 0, \tag{2.8}$$

where Γ is the *gamma function* [27, p. 135]. This distribution is denoted Gamma(α, β), having expectation α/β and variance α/β^2. In general, for $v \geq 1$, $Y \sim \text{Gamma}(v\alpha, \beta)$. This is because the sum of independent gamma-distributed random variables with the same scale parameter is itself gamma-distributed with the same scale parameter and the sum of the index parameters as its index parameter, which in this case is $v\alpha$. This means that, given claim severity $\tilde{Y} = Y/v$ (so that $Y = \tilde{Y}v$), the frequency function is

$$f_{\tilde{Y}}(\tilde{y}) = v f_Y(v\tilde{y}) = \frac{(v\beta)^{v\alpha}}{\Gamma(v\alpha)} \tilde{y}^{v\alpha-1} e^{-v\beta\tilde{y}};\ \tilde{y} > 0,$$

yielding the resulting distribution $\tilde{Y} \sim \text{Gamma}(v\alpha, v\beta)$, with expectation α/β. Re-parameterizing with $\mu = \alpha/\beta$ and $\phi = 1/\alpha$ we can express the frequency function as

$$\begin{aligned} f_{\tilde{Y}}(\tilde{y}) = f_{\tilde{Y}}(\tilde{y}; \mu, \phi) &= \frac{(v\beta)^{v\alpha}}{\Gamma(v\alpha)} \tilde{y}^{v\alpha-1} e^{-v\beta\tilde{y}} \\ &= \exp\left\{ \frac{-\tilde{y}/\mu - \log(\mu)}{\phi/v} + c(\tilde{y}, \phi, v) \right\};\ \tilde{y}, \mu, \phi > 0, \end{aligned} \tag{2.9}$$

where $c(\tilde{y}, \phi, v) = \log(v\tilde{y}\pi)v/\phi - \log(\tilde{y}) - \log\Gamma(v/\phi)$. As a final parameter adjustment, we set $\theta = -1/\mu$ in Equation (2.9), where $\theta < 0$ since $\mu > 0$. Re-introducing the indexation, i, the frequency function of the claim severity $\tilde{Y}_i$ is

$$f_{\tilde{Y}_i}(\tilde{y}_i, \theta_i, \phi) = \exp\left\{ \frac{\tilde{y}_i\theta_i + \log(-\theta_i)}{\phi/v} + c(\tilde{y}_i, \phi, v_i) \right\}. \tag{2.10}$$

This aligns with Equation (2.1), indicating that the distribution is an exponential dispersion model (EDM) with $b(\theta_i) = -\log(-\theta_i)$. Therefore, claim severity can be appropriately modeled using GLMs.

2.4 Tabular Form and List Form

Getting back to Table 1.1, in *tabular form* we denote the expected value of key ratio i, j as μ_{ij} where i denotes the group of the first feature, Class, and j denotes the group of the second feature, Age. We assume a multiplicative model structure for the mean:

$$\mu_{i,j} = \gamma_0\gamma_{1i}\gamma_{2j}, \tag{2.11}$$

where γ_{1i} and γ_{2j} correspond to the different parameters for the respective features. Taking the logarithm of this yields

$$\log(\mu_{i,j}) = \log(\gamma_0) + \log(\gamma_{1i}) + \log(\gamma_{2j}). \tag{2.12}$$

By sorting the cells of Table 1.1 in order; $(1,1),(1,2),(1,3),\ldots,(2,4)$, followed by setting some base cell,[3] say $(1,1)$ and letting $\beta_1 = \log(\gamma_0), \beta_2 = \log(\gamma_{12}), \ldots, \beta_5 = \log(\gamma_{24})$ we may express Equation 2.12 in *list form* [27, p. 26]

$$\log(\mu_i) = \sum_{j=1}^{5} x_{ij}\beta_j; \quad i = 1, 2, \ldots, 8. \tag{2.13}$$

The above equation uses x_{ij} as a dummy variable, i.e., $x_{ij} = \mathbb{1}_{\{\beta_j \text{ included in } \mu_i\}}$. We can express this in matrix form, $\log(\boldsymbol{\mu}) = \boldsymbol{X}\boldsymbol{\beta}$, where $\boldsymbol{X}$ is the *design matrix*

$$\log(\boldsymbol{\mu}) = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \\ \mu_5 \\ \mu_6 \\ \mu_7 \\ \mu_8 \end{pmatrix}, \ \boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} \\ x_{61} & x_{62} & x_{63} & x_{64} & x_{65} \\ x_{71} & x_{72} & x_{73} & x_{74} & x_{75} \\ x_{81} & x_{82} & x_{83} & x_{84} & x_{85} \end{pmatrix}, \ \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \end{pmatrix}.$$

2.5 Link Function

Besides allowing the response, Y, to follow a distribution belonging to the EDMs, the mean is not restricted to being a linear function of the features, $\boldsymbol{x}$. Rather, we introduce a monotone and differentiable function, $g(\cdot)$, known as the *link function* that maps the mean to a linear structure.[4] In other words, by using a linear predictor η_i, such that

$$\eta_i = \sum_{j=1}^{r} x_{ij}\beta_j; \ i = 1, 2, \ldots, n.$$

The link function *links* the mean to this linear predictor

$$g(\mathbb{E}[Y_i]) = g(\mu_i) = \eta_i = \sum_{j=1}^{r} x_{ij}\beta_j. \tag{2.14}$$

Since we are working with multiplicative models, i.e., $\mathbb{E}[Y_i] = \exp\{\sum_{j=1}^{r} x_{ij}\beta_j\}$, $g(\cdot)$ is a logarithmic link function, often simply referred to as a *log link*

$$g(\mu_i) = \log(\mu_i). \tag{2.15}$$

[3] By setting $(1,1)$ as base cell, $\gamma_{11} = \gamma_{2,1} = 1$, which is why there are only 5 β-parameters in this case.

[4] Technically, having a linear structure in this context is still considered a link-function, but specifically the identity link.

2.6 Estimating the Parameters

Typically, for a given set of observed data, estimating the $\boldsymbol{\beta}$ is done by finding their *maximum likelihood estimates* (MLEs). Assuming list form and independence between observations, the log-likelihood based on n observations with respect to the parameter vector $\boldsymbol{\theta}$ is

$$\ell(\boldsymbol{\theta}; \phi, \boldsymbol{y}) = \frac{1}{\phi} \sum_{i}^{n} v_i(y_i\theta_i - b(\theta_i)) + \sum_{i}^{n} c(y_i, \phi, v_i). \tag{2.16}$$

We want to express ℓ with respect to $\boldsymbol{\beta}$ rather than $\boldsymbol{\theta}$, this can be done by using the relations $\mu_i = b'(\theta_i)$ as well as the link function $g(\mu_i) = \eta_i = \sum_{j=1}^{r} x_{ij}\beta_j$. Differentiating ℓ with respect to β_j yields

$$\begin{aligned}\frac{\partial \ell}{\partial \beta_j} &= \sum_{i}^{n} \frac{\partial \ell}{\partial \theta_i}\frac{\partial \theta_i}{\partial \beta_j} = \frac{1}{\phi}\sum_{i}^{n}\Big(v_i y_i - v_i b'(\theta_i)\Big)\frac{\partial \theta_i}{\partial \beta_j} \\ &= \frac{1}{\phi}\sum_{i}^{n}\Big(v_i y_i - v_i b'(\theta_i)\Big)\frac{\partial \theta_i}{\partial \mu_i}\frac{\partial \mu_i}{\partial \eta_i}\frac{\partial \eta_i}{\partial \beta_j}.\end{aligned} \tag{2.17}$$

Recalling Equation (2.2), it was stated that $\mu_i = b'(\theta_i)$, as well as $b''(\theta_i) = \mathscr{V}(\mu_i)$. By using this, we observe that $\partial\mu_i/\partial\theta_i = \partial b'(\theta_i)/\partial\theta_i = b''(\theta_i)$. This is used in combination with the inverse relation $\frac{\partial\theta_i}{\partial\mu_i} = \left(\frac{\partial\mu_i}{\partial\theta_i}\right)^{-1}$, consequently, we arrive at $\partial\theta_i/\partial\mu_i = 1/\mathscr{V}(\theta_i)$. As a result of using the same inverse relation, as well as Equation (2.14) which states $\eta_i = g(\mu_i)$, the second partial derivative yields $1/g'(\mu_i)$. As for the final partial derivative, using $\eta_i = \sum_{j}^{r} x_{ij}\beta_j$, we end up with $\partial\eta_i/\partial\beta_j = x_{ij}$. Adding this all up, the concluding form of the differentiation is what is commonly known as the *score function* [27, p. 32]

$$\frac{\partial \ell}{\partial \beta_j} = \frac{1}{\phi}\sum_{i}^{n} v_i \frac{y_i - \mu_i}{\mathscr{V}(\mu_i)g'(\mu_i)}x_{ij}. \tag{2.18}$$

When maximizing ℓ with respect to $\boldsymbol{\beta}$, ϕ is redundant and can therefore be ignored. Furthermore, setting Equation 2.18 equal to zero for $j = 1, ..., r$ yields the desired ML equations

$$\sum_{i}^{n} v_i \frac{y_i - \mu_i}{\mathscr{V}(\mu_i)g'(\mu_i)}x_{ij} = 0,\ j = 1, \ldots, r. \tag{2.19}$$

which when solved with respect to $\boldsymbol{\beta}$ corresponds to obtaining the maximum likelihood estimate, $\hat{\boldsymbol{\beta}}$.

2.7 Optimization Problem

So what do we want to achieve by estimating $\boldsymbol{\beta}$ by maximum likelihood? We return to the empirical estimation of Equation (1.7), where the goal was to find

$$\hat{f}^* = \arg\min_{f \in \mathcal{F}} \frac{1}{n}\sum_{i}^{n} v_i L(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}_i)). \tag{2.20}$$

Both maximum likelihood estimation and empirical loss minimization address the same fundamental objective, assuming a *deviance loss* [27, pp. 39–41] is chosen as a loss function, which is conventional when modeling with GLMs. This is because the deviance loss functions use the negative log likelihood of a given model. In other words, assuming $\mathcal{F}$ is a family of GLM functions satisfying $f(\boldsymbol{x};\boldsymbol{\beta}) = g(\langle \boldsymbol{x}, \boldsymbol{\beta} \rangle)$, and by using parameter vector $\boldsymbol{\theta} = \boldsymbol{\beta}$, the two following problems are the same [37, p. 81]:

$$\hat{\boldsymbol{\beta}}^{MLE} = \arg\max_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta}; \phi, \boldsymbol{y}) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i}^{n} v_i L(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}_i)). \tag{2.21}$$

Chapter 3

Neural Networks

The following chapter introduces neural networks. We present the mathematics and attempt to explain the mechanics intuitively. Neural networks are solely applied in a non-life environment in this thesis. For neural networks applied to mortality forecasting, we recommend [39] by J. Zhan.

3.1 Different Names, Different Periods

At first glance, neural networks may seem like a cutting-edge innovation of the modern era. However, it might come as a surprise that the foundations of deep learning stretch back as far as the 1940s, albeit under different names and conceptual frameworks. Over the decades, what we now call deep learning has evolved through several distinct phases of development, each marked by unique breakthroughs and influential contributors [8, pp. 13–14]. These major waves in the history of neural networks are:

- *Cybernetics*, starting in the 1940s–1960s by McCulloch & Pitts [21] and Hebb [10].
- *Connectionism* in 1980–1995 with back-propagation by Rumelhart *et al.* [31].
- *Deep learning*, the third and current wave, which began around 2006 by Hinton *et al.* [11], Benigo *et al.* [2], and Ranzato *et al.* [30].

3.2 Deep Feed-forward Networks

Deep feed-forward networks (FFNs), often called feed-forward neural networks (FNNs) or multilayer perceptions (MLPs), are the quintessential deep learning models [8, p. 168]. The term *feed-forward* stems from the fact that information only flows forward, meaning information flows in a strict forward-structure through the neural network, finalizing in the output, $\boldsymbol{y}$. In other words, there is no *feedback* connection, meaning that information is not allowed to flow backwards in the neural network. FNNs are used for both regression and classification tasks, but will only be used for regression in this thesis. An FFN extended to include feedback connections is called a *recurrent neural network* [8, p. 378] but is out of the scope of this thesis.

The idea of FFNs is to, through an *automated process*, take some input $\boldsymbol{x}$ and apply a series of *interconnected layers*. Each layer is represented by a function $\boldsymbol{z}^{(m)}$ where m denotes the mth layer, ranging from 1 to d, assuming the neural network has d layers in total. Using these layer-notations, 1 denotes the input layer $\boldsymbol{x}$ and d denotes the output layer, $\boldsymbol{y}$. Each layer, i.e. each function $\boldsymbol{z}^{(m)}$ consists of q_m nodes, or *neurons*. These are typically referred to as the *dimension* of layer m, or the number of neurons. The layers in between the first and the last layer are what are known as the *hidden layers*. If a neural network only has one hidden layer, it is called a *shallow* neural network. If, on the other hand, it has two or more hidden layers, it is a *deep* neural network.

More formally, we say deep learning uses a finite sequence of functions $(\boldsymbol{z}^{(m)})_{1\leq m\leq d}$ such that

$$\boldsymbol{z}^{(m)}\colon \mathbb{R}^{q_{m-1}} \to \mathbb{R}^{q_m},$$

meaning $\boldsymbol{z}^{(m)} = (z_1^{(m)}, ..., z_{q_m}^{(m)})^T$ maps vectors of dimension q_{m-1} to vectors of dimension $q_m \in \mathbb{N}$. The layers are interconnected, meaning they are a *composite function* of each of the preceding layers. With this composite structure in mind, we define the mth representation of the features $\boldsymbol{x}$ as

$$\boldsymbol{z}^{(m:1)}(\boldsymbol{x}) = \Big(\boldsymbol{z}^{(m)} \circ \cdots \circ \boldsymbol{z}^{(1)}\Big)(\boldsymbol{x}) \ \in \ \mathbb{R}^{q_m}. \tag{3.1}$$

Each node in a layer, excluding the input layer, is connected to nodes in the previous layer through weights $\boldsymbol{W}^{(m)} \in \mathbb{R}^{q_m \times q_{m-1}}$ and a bias term $\boldsymbol{b}^{(m)} \in \mathbb{R}^{q_m}$.[1] The activation in neural networks is done by an *activation function*, $\phi : \mathbb{R} \to \mathbb{R}$. Using the weights, bias, and activation function, each layer $\boldsymbol{z}^{(m)}$ may be expressed as a function of the previous layer

$$\boldsymbol{z}^{(m)} = \phi^{(m)}\Big(\boldsymbol{W}^{(m)} \cdot \boldsymbol{z}^{(m-1)} + \boldsymbol{b}^{(m)}\Big). \tag{3.2}$$

The application of the activation function may also be presented on a neuron level, where neuron j of layer m is computed by

$$z_j^{(m)} = \phi(\langle \boldsymbol{w}_j^{(m)}, \boldsymbol{z}^{(m-1)} \rangle + b_j^{(m)}) = \phi\Big(\sum_{l=1}^{q_{m-1}} w_{l,j}^{(m)} z_l^{(m-1)} + b_j^{(m)}\Big), \ 1 \leq j \leq q_m, \tag{3.3}$$

here $\boldsymbol{w}_j^{(m)} = (w_{l,j}^{(m)})_{1\leq l\leq q_{m-1}} \in \mathbb{R}^{q_{m-1}+1}$, i.e. the weights in vector form and $\langle \cdot, \cdot \rangle$ denotes the *inner product* [7].

We may, out of simplicity include a constant term $z_0^{(m)} = 1$ in the $\boldsymbol{z}$-vectors. Proceeding by including the bias terms in the weights, the expression for the neuron calculations may be simplified to

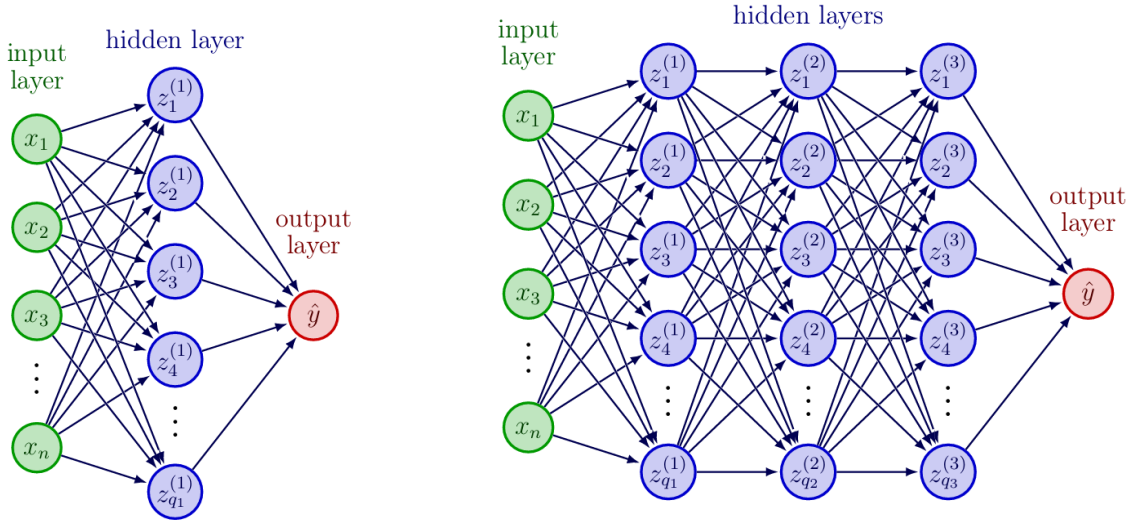
$$z_j^{(m)} = \phi\langle \boldsymbol{w}_j^{(m)}, \boldsymbol{z}^{(m-1)} \rangle, \tag{3.4}$$

which will be the choice of presentation for the rest of the thesis.

Two illustrations are presented in Figure 3.1, showing what a shallow neural network and a deep neural network may look like. The illustrations contain a single-valued output layer,

[1] This architecture of training and sending inputs between layers is where neural networks get their name from, as it mimics that of the human brain. In the human brain, neurons are the information transmitters, connected to one another via synapses, which allow signals to travel between them. When a neuron receives enough of an electrical impulse, it activates and transmits information [22].

i.e. $\hat{y} \in \mathbb{R}$, as neural networks are only used for regression modeling in this thesis, but may also be used for classification.



(a) Shallow neural network.

(b) Deep neural network, three hidden layers.

Figure 3.1: Illustrative example of neural networks.

The shallow neural network has a single layer-dimension of q_1, and maps to the output by $\hat{y} = \boldsymbol{z}^{(1)}(\boldsymbol{x})$. Whereas in the deep neural network, consisting of three hidden layers, each layer has dimension q_1, q_2, and q_3, respectively. This neural network maps to the output by $\hat{y} = (\boldsymbol{z}^{(3)} \circ \boldsymbol{z}^{(2)} \circ \boldsymbol{z}^{(1)})(\boldsymbol{x})$.[2]

3.3 Activation Functions

When working with neural networks, the architecture needs to be pre-determined, i.e. the number of layers as well as how many neurons these layers the neural network is to consist of. An equally important choice to be made is that of the activation function. There are many different activation functions, and the choice of an activation function is in itself something one can delve deep into. We will not be discussing activation functions in depth, we simply note that the activation function is a non-linear and differentiable function that calculates the output of the different neurons based on their individual inputs and weights. Some of the more common activation functions are presented in Table 3.1.

[2]All of the neural network illustrations are made using `TikZ`-code inspired by the open-source code provided at `TikZ`.

Table 3.1: Common activation functions and their derivatives.

Name	Activation Function	Derivative
Sigmoid (logistic)	$\phi(x) = \frac{1}{1+e^{-x}}$	$\phi' = \phi(1-\phi)$
Hyperbolic tangent	$\phi(x) = \tanh(x)$	$\phi' = 1 - \phi^2$
Exponential	$\phi(x) = e^x$	$\phi' = \phi$
Rectified linear unit (ReLU)[1]	$\phi(x) = x\mathbb{1}_{\{x \geq 0\}}$	$\phi' = \mathbb{1}_{\{x>0\}}$

[1] Mathematically speaking, the ReLU function does not have a derivative as the function is not differentiable at 0. However, the convention within machine learning is that its derivative is 0 at this point, making it differentiable.

The need for an activation function to be differentiable will become apparent in later chapters, as we discuss methods of training one's network (Section 4.3). For more intricate details of activation functions, we refer to [8, pp. 191–197].

3.4 Comparing GLMs with FNNs

In terms of traditional GLMs, we saw that the mean is modeled according to Equation (2.14). Expanding on this equation slightly, we use inner products to express the GLM structure

$$\boldsymbol{x} \mapsto g(\mu(\boldsymbol{x})) = \langle \boldsymbol{\beta}, \boldsymbol{x} \rangle. \tag{3.5}$$

This is, in fact, rather similar to what we do with FNNs. When we apply an activation function to the final hidden layer leading to the response, $\hat{y}$, we are effectively applying a GLM. However, rather than having input $\boldsymbol{x}$, we feed this GLM what is sometimes referred to as the *feature extractor*, $\boldsymbol{z}^{(d:1)}(\boldsymbol{x})$ [35, p. 78]. With this in mind, we modify Equation (3.5) by using this feature extractor

$$\boldsymbol{x} \mapsto g(\mu(\boldsymbol{x})) = \langle \boldsymbol{w}^{(d+1)}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x}) \rangle, \tag{3.6}$$

where $\boldsymbol{w}^{(d+1)}$ consists of the weights and biases of the final layer. This small, yet crucial detail is what extends FNNs beyond GLMs. By using the feature extractor, we apply a series of activation functions to the features in the hidden layers, allowing for the input to flow through the network and extracting interactive as well as non-linear structures that are difficult to capture with a GLM.

3.5 An Intuitive Point of View

The activation function is at times referred to as a *ridge function* [26, p. 32]. Applying Equation (3.3) to features $\boldsymbol{x}$, an inner product is first applied, $\langle \boldsymbol{w}_j^{(m)}, \boldsymbol{z}^{(m-1)}(\boldsymbol{x}) \rangle \in \mathbb{R}$. This is a projection and compresses the data, reducing dimensions from q_{m-1} to 1. That is, the q_{m-1}-dimensional feature $\boldsymbol{z}^{(m-1)}$ is in each neuron reduced, or compressed to a real number. Subsequently, a non-linear activation function, ϕ, is applied to the projection.

Because of the dimension reduction, a substantial amount of information is lost. This is why the procedure is performed q_m times for layer $\boldsymbol{z}^{(m)}$, so that each neuron represents a different projection of the input. These neurons are then used as new features in the next layer.

To be able to extract vital feature information from $\boldsymbol{z}$, well-suited weights, $\boldsymbol{w}_j^{(m)}$, are chosen to ensure the best possible explanatory variables are received for the regression task at hand. The methodology for finding $\boldsymbol{w}_j^{(m)}$ is typically some suitable *gradient descent method* and is covered in Section 4.2.

We previously stated that the number of layers and the depth of a neural network, i.e. the number of neurons per layer, must be predetermined. This task is often accomplished through experimentation guided by error measurements [8, p. 198]. However, it is noteworthy that a shallow neural network can sufficiently approximate any function, provided its dimensionality is sufficiently high, according to the *universal approximation theorem* [13]. Nevertheless, the theorem does not specify the required depth of the network, and in some cases, an exponential number of hidden units may be necessary. Such a large number of neurons may, in turn, render the neural network prone to overfitting.

3.6 CANN

The combined actuarial neural network (CANN) produces a *combined* prediction that integrates a regression model, typically a traditional GLM, with an FNN. This is done by analyzing the residuals of the initial model and then boosting this with a second model (FNN) in an attempt to find systematic effects in the residuals that the initial model failed to capture. The prediction is defined as

$$\hat{y}^{CANN} = g^{-1}\Big(\langle \boldsymbol{\beta}, \boldsymbol{x}\rangle + \langle \boldsymbol{w}^{(d+1)}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x})\rangle\Big). \tag{3.7}$$

Note we use g^{-1} as notation here rather than ϕ for the typical activation function. This distinction arises because, in the *final* layer, the activation function corresponds to the inverse log-link, specifically the exponential activation. The first term $\langle \boldsymbol{\beta}, \boldsymbol{x}\rangle$ represents η, the linear predictor of the GLM. The second term, $\langle \boldsymbol{w}^{(d+1)}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x})\rangle$, corresponds exactly to the final layer of the FNN, prior to applying the exponential activation function. With these notations, we may express Equation (3.7) as

$$\hat{y}^{CANN} = \exp\Big\{\langle \boldsymbol{\beta}, \boldsymbol{x}\rangle + \langle \boldsymbol{w}^{(d+1)}, \boldsymbol{z}^{(d:1)}(\boldsymbol{x})\rangle\Big\} = \hat{y}^{IN}\hat{y}^{FNN}. \tag{3.8}$$

The model can be seen as a *fine-tuned* version of a GLM. The process begins by fitting the estimating appropriate parameters $\hat{\boldsymbol{\beta}}$ using MLE as defined in Section 2.6. Having estimated this, it is used as part of the network parameter. However, rather than being connected to the hidden layers, it is instead connected directly to the output node via *skip connection* [12]. The FNN output, $\hat{y}^{FNN}$, serves as an adjustment to this initial prediction. This is illustrated in Figure 3.2.

By keeping the first part, $\langle \boldsymbol{\beta}, \boldsymbol{x}\rangle$ frozen during the boosting step, we are effectively using

the GLM as an *offset*. This is a crucial initialization step, as it ensures we start with $\hat{y}^{IN3}$ and any adjustments made to this initial value will improve upon the initial GLM prediction.

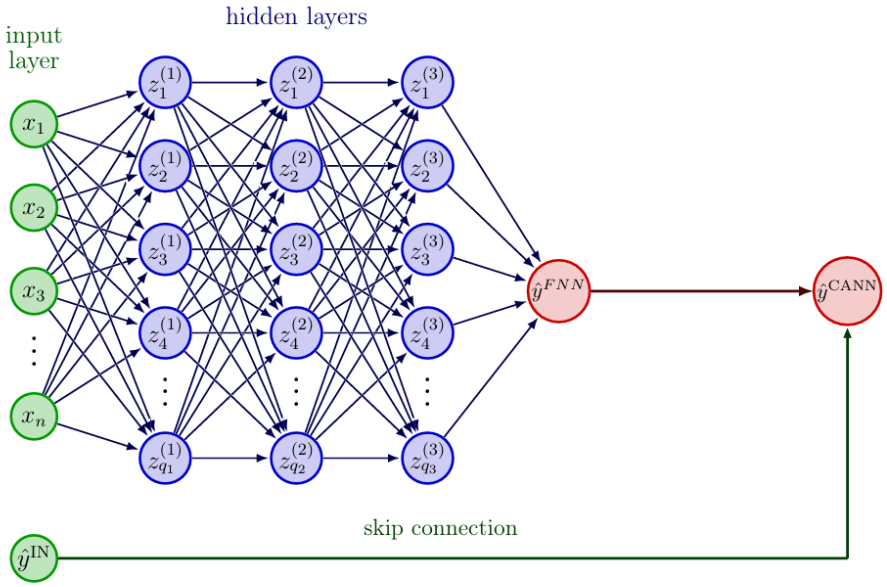


Figure 3.2: A CANN model, with skip connection.

[3]We denote the response of the boosted model as $\hat{y}^{IN}$ since in general, this may be obtained using other models than GLMs.

Chapter 4

Training the Network

4.1 Challenges When Compared With GLMs

The objective of model selection within neural networks consists of minimizing the loss function with respect to the network parameter, $\boldsymbol{\theta}$ and corresponds to obtaining the MLE (Section 2.6), when working with GLMs. However, due to the inherent non-linearity of neural networks, the MLE cannot typically be determined explicitly, as the model selection/fitting problem is very high-dimensional (complex) and non-convex and usually, at best, the found "solutions" are close to local optimums [35, p. 83]. Still, the optimization problem we wish to solve remains the same as in Section 2.7, meaning we want to find a function that minimizes the error in predictions

$$f^* = \arg\min_f \mathbb{E}[L(\boldsymbol{Y}, f(\boldsymbol{X}))]. \tag{4.1}$$

Based sample data, we estimate f^* empirically with some function $f(\boldsymbol{x}; \boldsymbol{\theta}) \in \mathcal{F}$, where $\mathcal{F}$ is a sufficiently large set of functions. So, whether it be a GLM or an FNN, we are fundamentally attempting to solve the same optimization problem.

That being said, there is a fundamental distinction between neural networks and GLMs. In the case of neural networks, even if the MLE was attainable, it would likely result in a highly over-fitted model, i.e. one that fits the training data so well that it not only catches the systematic effects but also noise and randomness. Such a model tends to generalize poorly to test data and more often than not results in low in-sample loss but also poor predictive power.

Because of this, the arguably more challenging part of the training process of neural networks is calibrating the network parameter in such a way that one attains a balance between minimizing in-sample loss and minimizing over-fitting, more on this in Section 4.4.

4.2 Gradient Descent Methods

The methods of choice used to minimize the loss function are the gradient descent methods (GDMs). These methods of training use iterative, gradient-based *optimizers* that drive the loss function close to local minima, rather than finding an explicit solution as one does for traditional regression models. GDMs calculate the gradient of some pre-determined loss

function, $L(\mathcal{D}, \boldsymbol{\theta})$, with respect to the network parameter, $\boldsymbol{\theta}$, and a partition of the total data, $\mathcal{D}$, known as the training set. The parameter is then adjusted in the *opposite* direction of the gradient, hence the name gradient descent. In other words, in each step $t \to t+1$ we adjust $\boldsymbol{\theta}$ in the direction in which $L(\mathcal{D}, \boldsymbol{\theta})$ experiences the steepest descent. If these steps are made by a sufficiently small perturbation of $\boldsymbol{\theta}$, we may approximate this by a first-order Taylor expansion[1]

$$L(\mathcal{D}, \boldsymbol{\theta}^{(t+1)}) \approx L(\mathcal{D}, \boldsymbol{\theta}^{(t)}) + \nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}^{(t)})^T (\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}), \tag{4.2}$$

assuming $\boldsymbol{\theta}^{(t+1)}$ and $\boldsymbol{\theta}^{(t)}$ are close to one another. The right-hand side of the approximation in Equation (4.2) is minimized when the second term is as negative as possible, which is why we update $\boldsymbol{\theta}$ in the opposite direction of the gradient. This leads to the so called *standard gradient descent update* [35, p. 84]

$$\boldsymbol{\theta}^{(t)} \to \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \rho_{t+1} \nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}), \tag{4.3}$$

where $\rho_{t+1} > 0$ is the learning rate, or step size. The learning rate, ρ_{t+1}, can not be too small, as this will result in too many steps in the gradient descent, making the computations very heavy. At the same time, it needs to be small enough to ensure we stay in a neighborhood of $\boldsymbol{\theta}^{(t)}$, to be able to perform Taylor expansion.

4.3 Back-Propagation

When using GDMs, the gradient of the loss function, $\nabla L(\mathcal{D}, \boldsymbol{\theta})$, must be computed. Because the loss function is generally high-dimensional, this tends to be a computationally intensive process. In theory, one could compute the gradient by standard calculus; however, it would likely be an exceedingly tedious process.

In order to make these gradient computations more efficient, the *back-propagation* method is used. This method was first introduced by Rumelhart *et al.* (1986) [31]. The idea of this method is to compute all of the neuron values, propagating forward in the network. At the output layer, the loss function is computed. From the output layer, the gradients are then computed in the network by propagating backward, hence the name, into the network by use of the chain rule of calculus. The method enables us to express the partial derivative $\partial L/\partial w$ of the loss function with respect to any weight, w, and $\partial L/\partial b$ with respect to any bias, b, in the network. It then uses matrix operations for its computations, in particular, the *Hadamard product* [25, p. 43].

Given two vectors $\boldsymbol{w}$, $\boldsymbol{t} \in \mathbb{R}^s$ the Hadamard product operation, denoted by $\odot$, is the element-wise product of the two vectors such that $(\boldsymbol{w} \odot \boldsymbol{t})_j = w_j t_j$. We illustrate with a simple example

$$\text{let } \boldsymbol{w} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \text{ and } \boldsymbol{t} = \begin{bmatrix} 4 \\ 2 \end{bmatrix},$$

then the Hadamard product, $\boldsymbol{w} \odot \boldsymbol{t}$ is

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} \odot \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \cdot 4 \\ 3 \cdot 2 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}. \tag{4.4}$$

[1] Assuming $L(\boldsymbol{\theta}, \mathcal{D})$ is differentiable with respect to $\boldsymbol{\theta}$.

The actual method works as follows: we introduce an equation for the *error* [25, p. 44] in output-layer d, which we denote by $\delta^{(d)}$

$$\delta^{(d)} = \nabla_{\boldsymbol{z}^{(d)}} L \odot \phi'(\tilde{\boldsymbol{z}}^{(d)}).^2 \tag{4.5}$$

Here, $\tilde{\boldsymbol{z}}^{(d)}$ denotes a vector of neurons of the output layer *prior* to applying the activation function, i.e. we compute the neurons according to Equation (3.4) but without applying ϕ. The error term acts as an intermediate quantity that will allow us to relate to ∇L in a back-propagating manner.

Next, we express the error of some layer m, $\delta^{(m)}$ in terms of the error in the next layer, $\delta^{(m+1)}$ by

$$\delta^{(m)} = ((\boldsymbol{W}^{(m+1)})^T \delta^{(m+1)}) \odot \phi'(\tilde{\boldsymbol{z}}^{(m)}). \tag{4.6}$$

The interpretation of Equation (4.6) is as follows, suppose we know $\delta^{(m+1)}$ at the $(m+1)$th layer. When we apply the transposed weight matrix, $(\boldsymbol{W}^{(m+1)})^T$, we can think of this as moving the error *backward* through the network giving us some sort of measure of the error at the *output* of layer m. We then take the Hadamard product, $\odot \tilde{\boldsymbol{z}}^{(m)}$. This moves the error backward through the activation function in layer m, giving us the error $\delta^{(m)}$ in the weighted input to layer m.

By combining Equation (4.5) with Equation (4.6), we are able to compute the error of any layer in the network. We start by computing $\delta^{(d)}$ according to Equation (4.5), this is then used in Equation (4.6) to compute the error of layer $d-1$, which in turn can be re-applied in the same equation to obtain the error of layer $d-2$, and so on.

These error terms are then used to compute the gradients with respect to the weights

$$\nabla_{\boldsymbol{W}^{(m)}} L = \boldsymbol{z}^{(m-1)}((\boldsymbol{W}^{(m+1)})^T \delta^{(m+1)}) \odot \phi'(\tilde{z}^{(m)}) = \boldsymbol{z}^{(m-1)} \delta^{(m)}. \tag{4.7}$$

4.4 Early Stopping

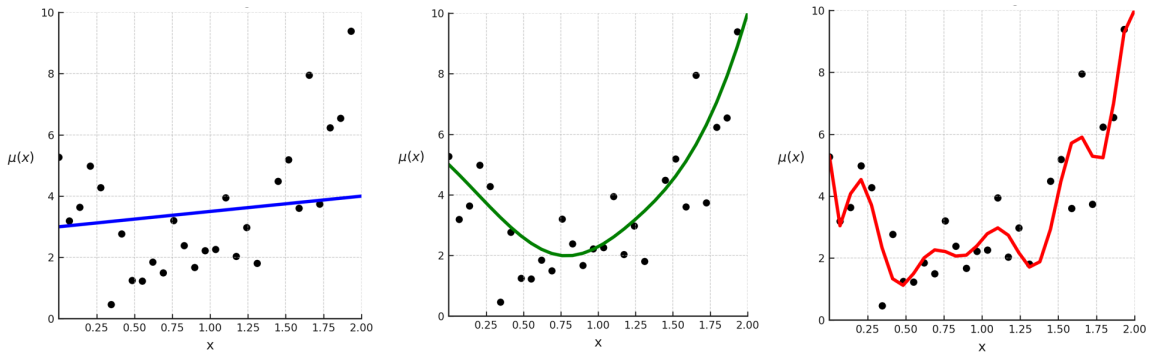
As mentioned in Section 4.1, over-fitting is generally of greater concern when modeling with FFNs compared to GLMs. In light of this, we stated in Section 4.1 we are not interested in finding explicit solutions (even if it was attainable) to our loss function, $L(\boldsymbol{\theta}, \mathcal{D})$ as this would likely lead to over-fitting. Be that as it may, we still want the loss to be as small as possible in order to find a good network parameter, $\hat{\boldsymbol{\theta}}$. Within machine learning, the methods used to treat over-fitting are called *regularization*. A common regularization method is the so called *early stopping* method [35, p. 88] during the gradient descent training.

Before explaining how early stopping works, we re-introduce the *volume-scaled empirical loss* [35, p. 28]

$$L(\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \frac{v_i}{\phi} L(y_i, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_i)). \tag{4.8}$$

Here, $\mathcal{D} = (y_i, \boldsymbol{x}_i, v_i)_{i=1}^n$ and v_i is the volume measurement, or weight as defined in Section 1.4. The dispersion parameter, ϕ, is the same as the one in Section 2.1.

[2] In order to use the convenient notation with the Hadamard product, we need to express the biases and weights separately accordingly with Equation (3.2).



(a) Under-fitting. (b) Appropriate degree of fit. (c) Over-fitting.

Figure 4.1: Three plots depicting different degrees of fitting.

Figure 4.1 provides an illustrative take on what varying degrees of model complexity may look like.[3] Here, Figure 4.1a has intentionally been made too simplistic, avoiding noise but also most of the systematic effects in the data. Figure 4.1b depicts what would be a reasonable amount of fitting. A high degree of systematic effects would be caught without a concerning amount of noise. Finally, Figure 4.1c shows clear indications of over-fitting, which may occur by allowing too much training. Having trained a network parameter to this extent will likely catch more than just systematic effects and will generalize poorly to test data.

The gradient of the volume-scaled empirical loss is computed by

$$\nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \frac{v_i}{\phi} \nabla_{\boldsymbol{\theta}} L(y_i, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_i)), \tag{4.9}$$

where the right-hand side is a sum of n individual gradients, computed for every i. Each computation is a step in the gradient descent method, and in each step, the network parameter is updated in such a way as to provide the most significant update. The systematic effects of the training data are, as the name suggests, more persistent across the data and are a more predictable distortion as compared to noise. Therefore, prior to finding these effects, they will dominate the steps in the gradient descent. As we continue iterating, however, more and more of these effects will have been found, and the relative impact of the noise (random effects) will start to increase. It is at this point that the early stopping is to be implemented.

To ensure we perform early stopping at the correct time, we adjust the partitioned data. Previously, we partitioned the data into two parts. The training part, $\mathcal{D}$, and the test part, $\mathcal{T}$. Now, the training data is split into two parts, $\mathcal{U}$ and $\mathcal{V}$ where $\mathcal{U}$ is treated identically to the previous training data and $\mathcal{V}$ becomes a separate set of data called the validation set.

In this way, the gradient descent steps are computed using this new training set, $\mathcal{U}$, followed by an instantaneous validation analysis on $\mathcal{V}$, which now is an out-of-sample set (completely separate from the training set). The validation set is typically 10% or 20% of the previous training data, $\mathcal{D}$.

[3] These figures do not represent actual training data related to this thesis, but only serve for the illustrational purpose of what could be feasible by exposing one's model to varying amounts of training.

With this in mind, we only compute the gradient descent with respect to $\mathcal{U}$

$$\nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}) = \sum_{i \in \mathcal{U}} \frac{v_i}{\phi} \nabla_{\boldsymbol{\theta}} L(y_i, \mu_{\boldsymbol{\theta}}(\boldsymbol{x}_i)), \tag{4.10}$$

followed by an out-of-sample computation of the loss function with respect to the validation set, called the validation loss, $L(\mathcal{V}, \boldsymbol{\theta}^{(t)})$ at each step t. As long as we are learning the systematic effects, the validation loss decreases alongside the training loss. Once we reach the point where we try to learn noise, however, the validation error will start to increase. The step in the iteration when this occurs is denoted as t^*, and is the precise moment of implementing the early stopping, and will be the choice of estimate for the network parameter, such that $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(t^*)}$. Figure 4.2 displays how the partition is made.

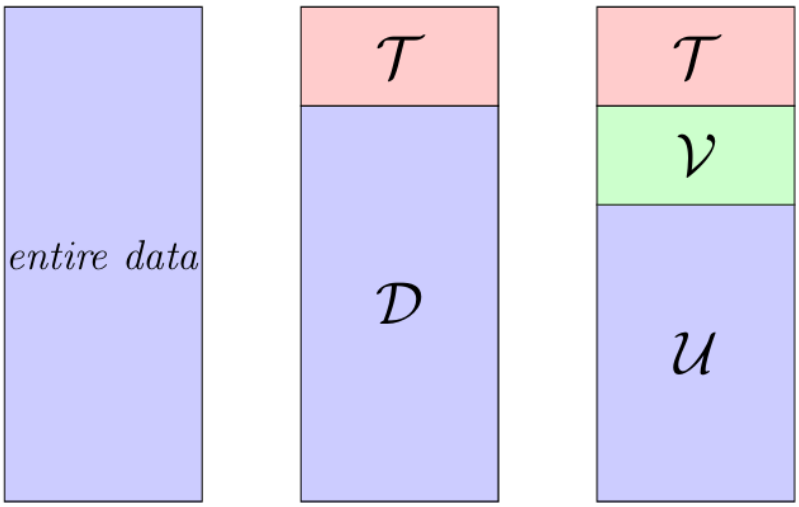


Figure 4.2: Data partition, inspired by [35, p. 89].

4.5 Stochastic Gradient Descent

A common gradient descent method is the *stochastic gradient descent* method (SGD). The concept is rather simple. When dealing with high dimensionality and large samples of data, computation of the loss in Equation (4.10) can get very cumbersome. A clever way of speeding up this process is to *randomly* partition $\mathcal{U}$ into mini-batches of similar sizes, $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_{n/s}$. These batches are then used cyclically for each step in the gradient descent steps $t \to t+1$

$$\boldsymbol{\theta}^{(t)} \to \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \rho_{t+1} \nabla_{\boldsymbol{\theta}} L(\mathcal{U}_k, \boldsymbol{\theta}). \tag{4.11}$$

The updating process of the right-hand side of Equation (4.11) uses these batches and is what is referred to as the SGD.

4.6 Momentum-Based Gradient Descent Methods

We saw the updated schematic of the standard gradient descent in Section 4.2. In this section, we introduce *momentum-based gradient descent methods*, which update the gradient using momentum. We present three common methods and their update schematics for the network parameters. For further details, see [37, pp. 285–288].

- `rmsprop` stands for root mean square propagation, and has the update schematic

$$\boldsymbol{\theta}^{(t)} \mapsto \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\gamma}{\sqrt{\varepsilon + \boldsymbol{\nu}^{(t+1)}}} \odot \nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}^{(t)}), \tag{4.12}$$

  where $\gamma > 0$ is the global decay rate, $\boldsymbol{\nu}$ the moving average of the squared gradients and $\varepsilon > 0$ is some small constant. The moving average of the squared gradients, $\boldsymbol{\nu}$, is updated by

$$\boldsymbol{\nu}^{(t)} \mapsto \boldsymbol{\nu}^{(t+1)} = \alpha \boldsymbol{\nu}^{(t)} + (1-\alpha) \left(\nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}^{(t)}) \odot \nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}^{(t)}) \right),$$

  for a given weight $\alpha \in (0, 1)$.

- `adam` combines `rsmprop` with the moving average of the gradients, $\boldsymbol{r}$, and has the update schematic

$$\boldsymbol{\theta}^{(t)} \mapsto \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\gamma}{\varepsilon + \sqrt{\frac{\boldsymbol{\nu}^{(t+1)}}{1-\alpha^t}}} \odot \frac{\boldsymbol{r}^{(t+1)}}{1-\beta^t}, \tag{4.13}$$

  for given weights $\alpha, \beta \in (0, 1)$. The moving average of the gradients, $\boldsymbol{r}$, is updated by

$$\boldsymbol{r}^{(t)} \mapsto \boldsymbol{r}^{(t+1)} = \beta \boldsymbol{r}^{(t)} + (1-\beta) \nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}^{(t)}).$$

- `nadam` is a Nestorov-accelerated version of `adam`,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\varepsilon + \sqrt{\frac{\boldsymbol{\nu}^{(t+1)}}{1-\alpha^t}}} \odot \frac{\beta \boldsymbol{r}^{(t+1)} + (1-\beta) \nabla_{\boldsymbol{\theta}} L(\mathcal{D}, \boldsymbol{\theta}^{(t)})}{1-\beta^t}, \tag{4.14}$$

where, for each of the optimizers, the operations performed on the right-hand side prior to the Hadamard product are done element-wise. Furthermore, $\boldsymbol{\nu}$ and $\boldsymbol{r}$ are both initialized by $\boldsymbol{\nu}^{(0)} = \boldsymbol{r}^{(0)} = 0$, for relevant optimizers.

Chapter 5

Data, Coding, and Pre-Processing

At this point, a sufficient theoretical basis for the application of neural networks as well as GLMs has been established. In this chapter, we examine the data, pre-process it and build our GLMs and FNNs. Having built our GLMs, we walk through the coding of FNNs as well as CANNs and the parameter tuning. We only model on claim frequency, assuming a Poisson distribution in accordance with the theory in Section 2.2.

5.1 Coding

The coding for this thesis is written in `R`, and is influenced by the articles *Case Study: French Motor Third-Party Liability Claims* [26] as well as *Nesting Classical Actuarial Models into Neural Networks* [32]. The neural networks are designed using the `keras3`[1] and `tensorflow` packages in `R`. The `keras3` package works as a high-level neural network API. Essentially, `keras3` runs on top of some back-end package, such as, in our case, `tensorflow`, which can be viewed as the engine that runs behind the scenes. `keras3` is then used in a user-friendly manner to design and work with the neural networks. Some figures and plots are aesthetically enhanced using the `TikZ` package in LaTeX.

5.2 Data

The data we use is the French motor third-party liability data, `freMTPL2freq`, which is a part of a large collection of datasets, originally from *Computational Actuarial Science with R* [3]. These actuarial datasets are frequently used in academia and research and are contained in the `R` package `CASdatasets` [6]. The data consists of claim counts observed over one calendar year and is a widely used open-source dataset when modeling claim frequency. A total of 678013 observations, or car insurance policies, are contained in the data, and for each of the observations, there are 12 features, which are presented in Table 5.1.

[1] The `keras3` package has a rather extensive range of applications as well as hyper-parameter fine tuning. To navigate this, the guide [15], written by Tomasz Kalinowski *et al.* is a helpful and recommended companion.

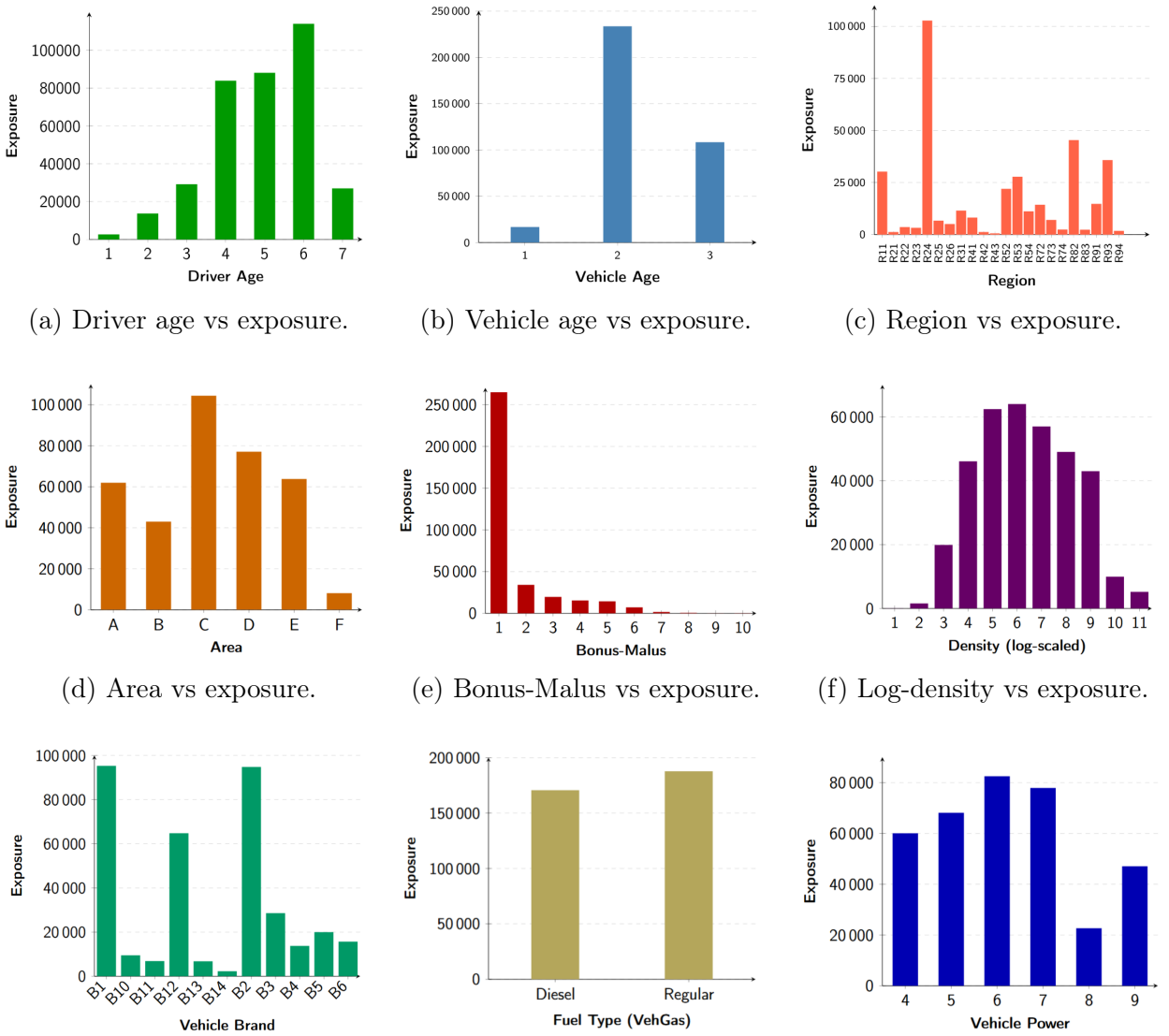
Table 5.1: The 12 features of `freMTPL2freq`.

Feature	Description
`IDpol`	Policy number of each policy
`ClaimNb`	Number of claims on the given policy
`Exposure`	Total exposure in yearly units
`Area`	Area code
`VehPower`	Power of the car
`VehAge`	Age of the car in years
`DrivAge`	Age of the driver in years
`BonusMalus`	Bonus-malus level, capped at 150.
`VehBrand`	Car brand
`VehGas`	Diesel or regular fuel
`Density`	Log-density of inhabitants per km^2 in the city of the living place of the driver
`Region`	Regions in France[1]

[1] The regions of `Region` are prior to 2016.

The dataset is rather well-polished, however, some corrections need to be made. As the data is collected over one calendar year, any `Exposure` value above 1 is likely caused by some data error. The dataset contains 1224 values exceeding 1, which have been capped at 1. Furthermore, 9 observations had `ClaimNb` values exceeding 4. These are also likely caused by some error and have been capped at 4.

The feature `BonusMalus` is commonly used in insurance and adjusts the premium paid by a customer according to their individual claim history. Typically, a bonus discount is given on the renewal of the policy if no claim was made the previous year. Note that `Exposure` is the total amount of claims given a specific observation of a feature. We model using the 9 features `Area-Region` and present their distributions against `Exposure` in Figure 5.1.



(a) Driver age vs exposure. (b) Vehicle age vs exposure. (c) Region vs exposure.

(d) Area vs exposure. (e) Bonus-Malus vs exposure. (f) Log-density vs exposure.

(g) Vehicle brand vs exposure. (h) Fuel type vs exposure. (i) Vehicle power vs exposure.

Figure 5.1: Exposure distribution across all features.

We refrain from analyzing the data too extensively, however, we look at the correlation between the features. Two types of correlations are measured, the *Pearson's correlation* [29]:

$$\rho_p = \frac{Cov(x_i, x_j)}{\sigma_{x_i}\sigma_{x_j}}, \tag{5.1}$$

As well as the *Spearman's rank correlation* [33]:

$$\rho_s = \frac{Cov(R(x_i), R(x_j))}{\sigma_{R(x_i)}\sigma_{R(x_j)}}. \tag{5.2}$$

Pearson's correlation is a linear correlation measurement, while Spearman's measures the between-rank correlation; thus, measuring both of them gives a broader view of the behavior of the features. The results are presented in Table 5.3.

Table 5.3: Bottom-left shows Spearman's ρ, top-right shows Pearson's ρ.

	Area	VehPower	VehAge	DrivAge	BonusMalus	Density
Area	**1.00**	0.00	-0.10	-0.05	0.12	0.59
VehPower	-0.01	**1.00**	-0.01	0.03	-0.08	0.04
VehAge	-0.10	0.00	**1.00**	-0.06	0.08	-0.09
DrivAge	-0.05	0.04	-0.08	**1.00**	-0.48	0.00
BonusMalus	0.14	-0.07	0.08	-0.57	**1.00**	0.08
Density	0.98	-0.01	-0.10	-0.04	0.14	**1.00**

If no significant loss of predictiveness is made by excluding one of the features, it suggests we may want to exclude one of them from our GLM. The Spearman's ρ as well as the Pearson's ρ indicate `Area` and `Density` are highly correlated. We bear this in mind when building and analyzing our GLMs.

5.3 Loss Functions

A loss function needs to be selected for implementation in the gradient descent method, and there is a range of functions to choose from. Some of the common ones are the *mean absolute error* (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|, \tag{5.3}$$

and the *mean square error* (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2. \tag{5.4}$$

As we are modeling for claim frequency, the *Poisson deviance* [27, p. 40] is a suitable loss function and is used throughout this thesis. In particular, we use the average Poisson deviance:

$$D_{Poisson} = \frac{200}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i + y_i \log\frac{y_i}{\hat{y}_i}\right). \tag{5.5}$$

The scaling factor of 100 is purely for a visually more appealing and easy-to-read result. We call the Poisson deviance obtained on training data, $\mathcal{D}$, our *in-sample loss*, and that of the test data, $\mathcal{T}$, our *out-of-sample loss*. When working with GLMs, no validation partitions are made. See Figure 4.2.

5.4 GLM Feature Pre-Processing

Some pre-processing of the data is made before creating our GLMs, and is presented in Table 5.4.

Table 5.4: GLM pre-processing.

Feature	Feature structure for GLM	Baseline group
`Area`	Continuous log-linear, $\{\texttt{A}, \ldots, \texttt{F}\} \mapsto \{1, \ldots, 6\}$	-
`VehPower`	Categorical with 6 groups $\{4, \ldots, 9\}$, values above 9 rounded down to 9.	`4`
`VehAge`	Categorized into 3 groups $[0, 1)$, $[1, 10)$, $(10, \infty)$	`2`
`DrivAge`	Categorized into 7 groups $[18, 21)$, $[21, 26)$, $[26, 31)$, $[31, 41)$, $[41, 51)$, $[51, 71)$, $[71, \infty)$	`5`
`BonusMalus`	Continuous log-linear, capped at 150	-
`VehBrand`	Categorical with 11 groups, $\{\texttt{B1}, \ldots, \texttt{B14}\}$	`B1`
`VehGas`	Binary, Diesel or Regular fuel	`1`
`Density`	Continuous log-linear feature based on population density (log-scale)	-
`Region`	Categorical with 22 levels representing French regions	`R24`

5.5 The GLMs

As we want a competitive GLM to compare with the neural network models, we prefer one with the best in-sample and out-of-sample deviance losses, as this will be our key measurement of performance. Commencing with a full model, we call this `GLM_Full`:

$$\texttt{ClaimNb} \sim \texttt{VehPower}+\texttt{VehAge}+\texttt{BonusMalus}+\texttt{VehBrand}+\texttt{VehGas}+\texttt{Density}+\texttt{Region}+\texttt{Area}+\texttt{DrivAge}.$$

We suspect this model can be improved upon however, we add non-linear regression to `DrivAge`, in terms of polynomial regression as well as `log(DrivAge)`, in accordance with [32, p. 5]. We call this model `GLM_Poly`:

$$\texttt{DrivAge} \mapsto \beta_l + \beta_{l+1} \log(\texttt{DrivAge}) + \sum_{j=2}^{4} \beta_{l+j} (\texttt{DrivAge})^j.$$

A `Drop1`-test is performed on `GLM_Poly`, showing all features are significant as presented in Table 5.6.

Table 5.6: Results of `Drop1`-test on `GLM_Poly` model.

Variable	Df	Deviance	AIC	LRT	$\Pr(>\chi^2)$	Signif.
`<none>`	–	147072	192820	–	–	
`VehPowerGLM`	5	147153	192891	81.1	4.97×10^{-16}	***
`VehAgeGLM`	1	147226	192976	153.2	$< 2 \times 10^{-16}$	***
`BonusMalusGLM`	1	150696	196712	3894.1	$< 2 \times 10^{-16}$	***
`VehBrand`	10	147308	193035	235.6	2.2×10^{-16}	***
`VehGas`	1	147093	192842	21.6	3.3×10^{-6}	***
`DensityGLM`	1	147082	192828	10.0	1.39×10^{-3}	**
`Region`	21	147264	192967	192.1	2.2×10^{-16}	***
`AreaGLM`	6	147076	192860	4.1	4.17×10^{-2}	*
`DrivAge`	1	147222	192972	149.8	$< 2 \times 10^{-16}$	***
`log(DrivAge)`	1	147194	192944	121.4	$< 2 \times 10^{-16}$	***
`I(DrivAge^2)`	1	147193	192943	118.2	$< 2 \times 10^{-16}$	***
`I(DrivAge^3)`	1	147161	192911	86.5	$< 2 \times 10^{-16}$	***
`I(DrivAge^4)`	1	147138	192888	65.7	5.33×10^{-16}	***

We delve deeper, analyzing the `summary` of `GLM_Poly`. Several groups of `Region` as well as `VehBrand` are showing highly insignificant values. We perform merging on `Region`, merging `R23, R25, R53, R11, R21, R42, R83` and `R91` into the baseline, `R24`. Furthermore, we merge the groups `B2` and `B10` of `VehBrand` into `B1`. This merged GLM will be called `GLM_Merged` and is our best performing model in terms of in-sample and out-of-sample deviance, meaning this is the most competitive GLM; the results are presented in Table 5.7.

Table 5.7: Our three best performing GLMs and their losses.

Model	In-sample loss	Out-of-sample loss
`GLM_Full`	24.113	24.167
`GLM_Poly`	24.102	24.133
`GLM_Merged`	**24.102**	**24.129**

Returning to the correlations we saw in Table 5.3, we point out GLMs excluding `Area` or `Density` have been analyzed but were getting outperformed by `GLM_Merged` and so are not discussed in detail in this section.

5.6 FNN Feature Pre-Processing

Before we can start building our neural networks, the features need some treatment that differs from that of the GLMs. For neural networks to work, all features need to be transformed into numerical values in some way. For starters, we have two categorical features, `Region` and `VehBrand`, that need to be treated. There are multiple ways of treating categorical features, we use *one-hot encoding* [35, pp. 38–39].

One-hot encoding means we map each group of a categorical feature to a basis vector in $\mathbb{R}^k$, where k is the number of groups of the feature. In doing so, we end up with a unit matrix of size $k \times k$. Figure 5.2 illustrates one-hot encoding for `VehBrand`.

VehBrand
B1
B10
B11
B12
B13
B14
B2
B3
B4
B5
B6

One-hot encoding
$\Longrightarrow$

B1	1	0	0	0	0	0	0	0	0	0	0
B10	0	1	0	0	0	0	0	0	0	0	0
B11	0	0	1	0	0	0	0	0	0	0	0
B12	0	0	0	1	0	0	0	0	0	0	0
B13	0	0	0	0	1	0	0	0	0	0	0
B14	0	0	0	0	0	1	0	0	0	0	0
B2	0	0	0	0	0	0	1	0	0	0	0
B3	0	0	0	0	0	0	0	1	0	0	0
B4	0	0	0	0	0	0	0	0	1	0	0
B5	0	0	0	0	0	0	0	0	0	1	0
B6	0	0	0	0	0	0	0	0	0	0	1

Figure 5.2: One-hot encoding of `VehBrand`.

Several techniques exist for the treatment of categorical variables, some of which are *embedding* [35, p. 40] and *large language models* (LLMs) [1]. These are, however, not discussed in this thesis.

Next, the numerical features need pre-processing. When working with neural networks, or rather gradient descent methods, it is important that the features are on a similar scale. If the features have different scales, the steps in the updating of the gradient descent may become unstable and too large in some directions or too small in others. Therefore, two transforms are commonly used. The first one is *min-max scaling*:

$$X_{i,j} \mapsto 2\frac{X_{i,j} - \min(\boldsymbol{X}_i)}{\max(\boldsymbol{X}_i) - \min(\boldsymbol{X}_i)} - 1, \tag{5.6}$$

where i denotes the ith feature and j the jth observation. The second is the *standardization*:

$$X_{i,j} \mapsto \frac{X_{i,j} - \widehat{m}(\boldsymbol{X}_i)}{\widehat{s}(\boldsymbol{X}_i)}, \tag{5.7}$$

where $\widehat{m}$ is the empirical mean, and $\widehat{s}$ the empirical standard deviation. We use standardization as a transformation for our numerical features.

5.7 FNN Coding

In this section, we explain some of the codes used for our neural networks. The format and functional construction of these networks are inspired by those of M. V. Wüthrich *et al.* [35]. Some of the code is included in the section, and some is found in the Appendix.

Our first step is to perform one-hot encoding as well as standardization on our features, as shown in Listing A.1, found in the Appendix. We proceed by transforming the data in the training set, $\mathcal{D}$, and the test set $\mathcal{T}$, see Listing 5.1.

Listing 5.1: Data partitioning.

```
## load and pre-process data
train <- Features.PreProcess(train_dat)
test <- Features.PreProcess(test_dat)
```

We are now ready to define the architecture of our network, as presented in Listing 5.2. The most important components to fine-tune are:

- The activation functions
- The number of neurons of each layer
- The number of hidden layers
- Possible regularization techniques, such as *dropout* [8, pp. 258–268]

We start with an FNN consisting of three hidden layers. The `keras3` and `tensorflow` packages have their own internal randomness function, and so we need to set a separate seed for these packages. As the weights of each layer are typically initialized randomly, this is a crucial detail in the code.

The `FNN()` function in Listing 5.2 takes two arguments, one is the seed and the other is a vector representing the dimensions of the layers, `layer_dims`. The vector `layer_dims` defines the number of units, or neurons, of each of the hidden layers as well as the input layer of the network. The network uses *dense layers*, meaning all the neurons in the current layer receive input from all the neurons in the previous layer. We use *tanh* as our activation function for all but the final layer, connecting to the response. To mimic the GLM, we apply an *exponential* activation function here. The `Volume` matrix consists of the exposure data. In terms of regularization, attempts at different degrees of dropout have been made, but worsened the performance, so no results using dropout are presented.

Listing 5.2: FNN architecture, 3 hidden layers.

```
FNN <- function(seed, layer_dims){
    tf$keras$backend$clear_session() # Clears previous Keras models
    set.seed(seed)
    set_random_seed(seed) # seed for Keras/TensorFlow
    Design  <- layer_input(shape = c(q0[1]), dtype = 'float32')
    Volume  <- layer_input(shape = c(1), dtype = 'float32')
    #
    Network = Design %>% # Tanh as activation in hidden layers
          layer_dense(units=q0[2], activation='tanh') %>%
          layer_dense(units=q0[3], activation='tanh') %>%
          layer_dense(units=q0[4], activation='tanh') %>%
          layer_dense(units=1, activation='exponential')
          # Output layer uses exponential activation function to mimic GLM
    #
    Response = list(Network, Volume) %>% layer_multiply()
```

```
    #
    keras_model(inputs = c(Design, Volume), outputs = c(Response))
    }
```

Next, the necessary matrices are made for the networks, see Listing A.2 in the appendix. The code is fairly straightforward. We also note there are two GLM matrices in the code, these will be used in place of the `Volume` when performing the CANN. Proceeding, we construct our FNN in Listing 5.3.

Listing 5.3: Constructing our FNN, 3 layers.

```
FNNneurons <- c(input_layer, c(20,15,10)) # The number of neurons for
   input- and hidden layers
seed <- 100
FNN3_model <- FNN(seed, FNNneurons)
FNN3_model
```

If we print `FNN3_model`, it yields Table 5.8. This table displays the number of parameters as well as the overall structure. For example, if we look at `Dense`, it *densely* connects the input layer to the first hidden layer meaning we get a total of $40 \cdot 20 + 20 = 820$ parameters.[2]

Table 5.8: `FNN_model3` layer structure.

Layer (type)	Output Shape	Param #	Connected to
InputLayer	(None, 40)	0	-
Dense	(None, 20)	820	InputLayer
Dense_1	(None, 15)	315	Dense
Dense_2	(None, 10)	160	Dense_1
Dense_3	(None, 1)	11	Dense_2
InputLayer_1	(None, 1)	0	-
Multiply	(None, 1)	0	Dense_3, InputLayer_1

Total params: 1,306 **Trainable params:** 1,306

Moving to Listing 5.4, we begin training our network. We start by creating a file path where we store our weights.. These are used in the `callback` function, which dictates the early stopping. The optimal network parameters are stored in `path1`, and we monitor the validation loss, `val_loss`.

Next, we compile the model. We choose nadam as our optimizer, and we may adjust the learning rate used for the optimization. However, the standard learning rate for nadam is 0.002, and we chose not to alter this. The actual training begins with the `fit` function.

Key parameters to tune `fit()` are:

- Validation set size
- Batch size

[2]Since each neuron of a layer has its own bias term, we add another 20 parameters on top of the already 800.

- Number of epochs
- Callbacks (for early stopping)

We set the validation set, $\mathcal{V}$, to be 10% of the training set. The batch size sets the size of each of the mini-batches in the gradient descent method, and we set this to 5000. Attempts at other batch sizes have been made, specifically 64, 1024, and 2048, but unsuccessfully improved the out-of-sample losses. In general, a smaller batch size may improve generalization but is slower and requires a larger amount of parameter network updates. The number of epochs dictates a upper bound for the number of training iterations, and within each epoch, the gradient descent updates our weights via back propagation. These steps can be summarized as follows:

1. Data propagates forward in the network, resulting in a prediction in the output
2. The loss is computed
3. Backpropagation computes gradients with respect to the weights
4. The Nadam updates the weights via the backpropagation

Within *each* epoch, these steps are executed in batches of 5000 observations until every batch has been trained on; this is repeated for every epoch. Finally, the losses for each epoch are stored for visualization.

Listing 5.4: Network parameter training.

```
# Early stopping
if (!dir.exists("./Networks")){dir.create("./Networks")}
path1 <- paste0("./Networks/FNN3_",seed,".weights.h5") # Save the weights
    here
CBs <- callback_model_checkpoint(path1, monitor = "val_loss", verbose = 0,
      save_best_only = TRUE, save_weights_only = TRUE)
# Compile with Poisson deviance loss and nadam optimizer(Gradient descent
    method)
FNN_model3 %>% compile(loss = 'poisson', optimizer = 'nadam')
# Begin the fitting procedure, batch size might be tweaked. Epochs seem
    fine at 200 as
# We usually stop somewhere around 50-60 epochs due to early stopping
{t1 <- proc.time()
  fit <- FNN_model3 %>% fit(list(Xtrain, Vtrain),  Ytrain,
                    validation_split=0.1, batch_size=5000, epochs=200,
                    verbose=1, callbacks=CBs)
(proc.time()-t1)[3]}

which.min(fit[[2]]$val_loss) # Provide the early stopping epoch
# Optimal network parameter obtained after 68 epochs.
# We store the training values in a .csv file so that we can present a
    polished plot in the thesis:
FNNloss3_df <- as.data.frame(fit[[2]])
write.csv(FNNloss3_df, "FNN3_loss_data.csv", row.names = FALSE)
```

Chapter 6

Application

It is now time to test our models, we evaluate the FNNs as well as the CANNs with hopes of promising results. We finalize by boosting a more complex model and present our results. Figures illustrating the training process are presented, as well as figures comparing `GLM_Poly` to our complex models. The latter are found in Appendix B. These comparison figures do not necessarily tell us how well a model performs, but rather how the predictions relate to the GLM.

6.1 FNN Models

In this section, we present the best-performing FNN. As mentioned in Section 5.7, we have attempted a number of batch sizes, as well as adding dropout layers to our network. Numerous architectural tweaks were also made with the intent of improving the predictive performance. The code for this network is found in Section 5.7.[1] This network maintains the same architecture as that of M. V. Wüthrich and M. Merz [37, pp. 298–305].

We allow for training over a total of 200 epochs; however, by early stopping, we reach our best performing network parameter after just 68 epochs. The training is illustrated in Figure 6.2, where the red line depicts at what epoch early stopping halts the training. We note after this epoch, there is a clear increase in validation loss, suggesting the decrease in training loss is likely noise. This model achieves an in-sample loss of 23.714 and an out-of-sample loss of 23.852. We call this model `3FNN`.

[1] The networks presented are all drawn using draw.io.

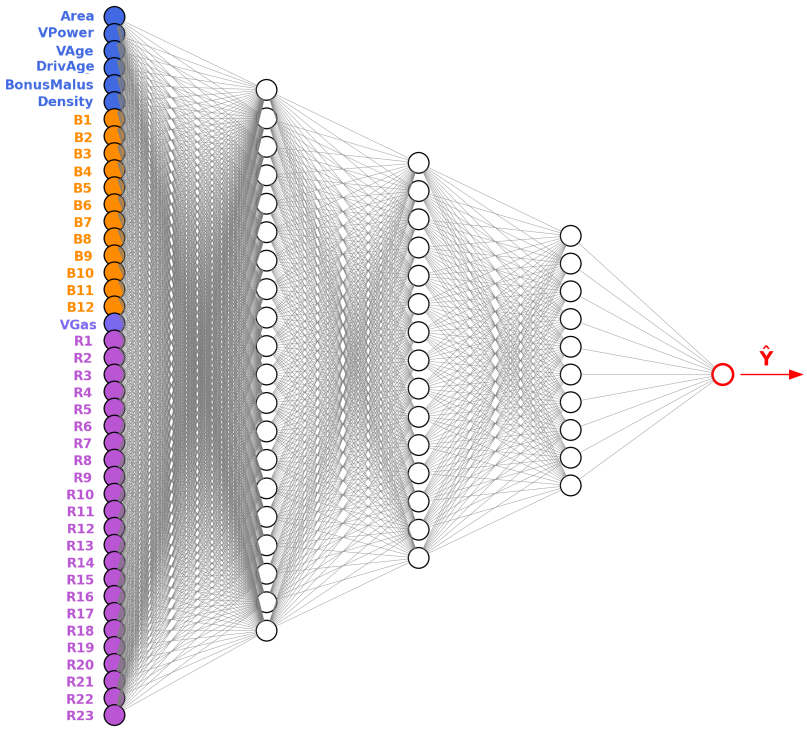


Figure 6.1: Three-layer FNN with 20,15, and 10 neurons per hidden layer.

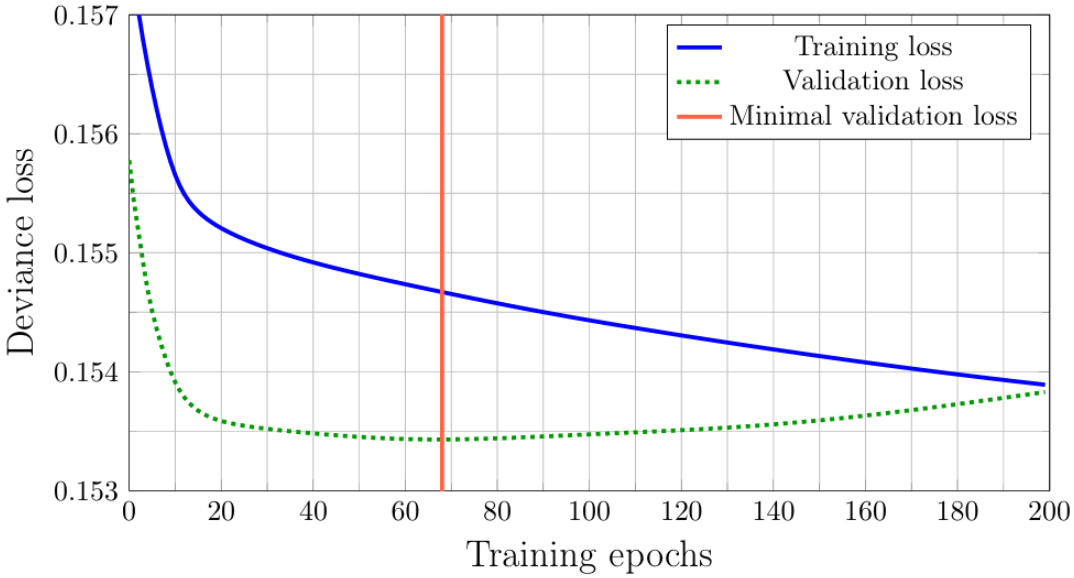


Figure 6.2: Training `3FNN` over 200 epochs.

We compare `3FNN` to `GLM_Poly` in Figure B.1. The black, dashed line represents the line `3FNN`=`GLM_Poly`. If predictions are stacked along this line, it suggests the models are

predicting similarly. The red line represents a linear regression of `3FNN` using `GLM_Poly`. Having many predictions above this line suggests `3FNN` overestimates in relation to `GLM_Poly`. On the other hand, if many predictions fall below this line, it suggests `3FNN` underestimates in relation to `GLM_Poly`. The figure suggests a fairly balanced ratio, meaning there appears to be no obvious overestimation or underestimation in relation to `GLM_Poly`.

A clear indication of overestimation, when compared with `GLM_Poly`, is illustrated in Figure B.2. This figure compares a FNN consisting of 5 hidden layers, `5FNN`, with `GLM_Poly`. The model is our worst-performing stand-alone model and performed poorly in terms of prediction. However, in Section 6.2, when boosting with this model, promising results are achieved.

6.2 GLM-CANNs

We have now reached the CANN, which means we boost GLMs with FNNs. In terms of coding, the CANNs are made with very similar coding when compared with the FNNs. There is, however, one key difference: the GLMs. Getting back to Section 3.6, we initialize the FNN with a network parameter consisting of our GLM prediction. This is then used as an offset rather than the volume matrices and is not altered, but the FNN attempts to find more signal in the data and acts in a complementary manner. This means, rather than using `Vtrain` and `Vtest`, we use `GLM_train` and `GLM_test` in Listing A.2 when fitting the CANN. The GLM we choose to boost is `GLM_Poly`, and we present the first CANN in Figure 6.3. The architecture of this CANN is the same as that of the FNN in Figure 6.3, with the exception of the skip connection of the GLM. This model is called `3CANN`.

A comparison of `3CANN` and `GLM_Poly` is provided in Figure B.3. This figure shows no indication of clear underestimation or overestimation. Overall, less scatter can be seen, suggesting relatively more balanced predictions.

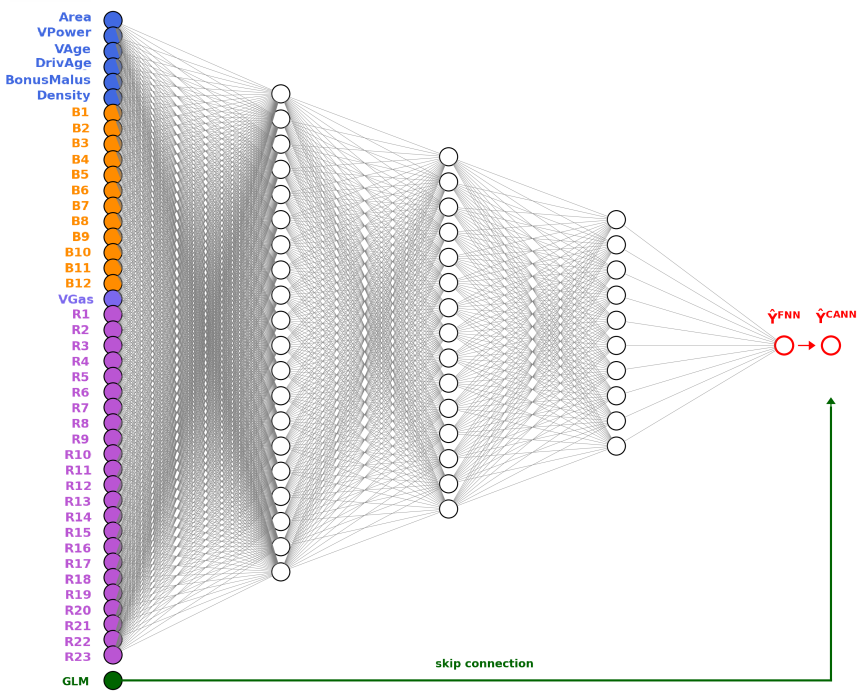


Figure 6.3: Three-layer CANN with GLM applied via skip connection.

We train using `3CANN` and achieve an in-sample loss of 23.735 and an out-of-sample loss of 23.823, which is an improvement over `3FNN`. The predictive performance is not only improved, but also achieved at an earlier epoch, 52. This is expected, as we start the training process with a well-performing GLM. At this point, a satisfactory result has been achieved. We have outperformed a rather polished GLM, `GLM_Merged`, using `3FNN`. We then managed to achieve even better results by boosting `GLM_Poly` using `3CANN`, see Figure 6.4.

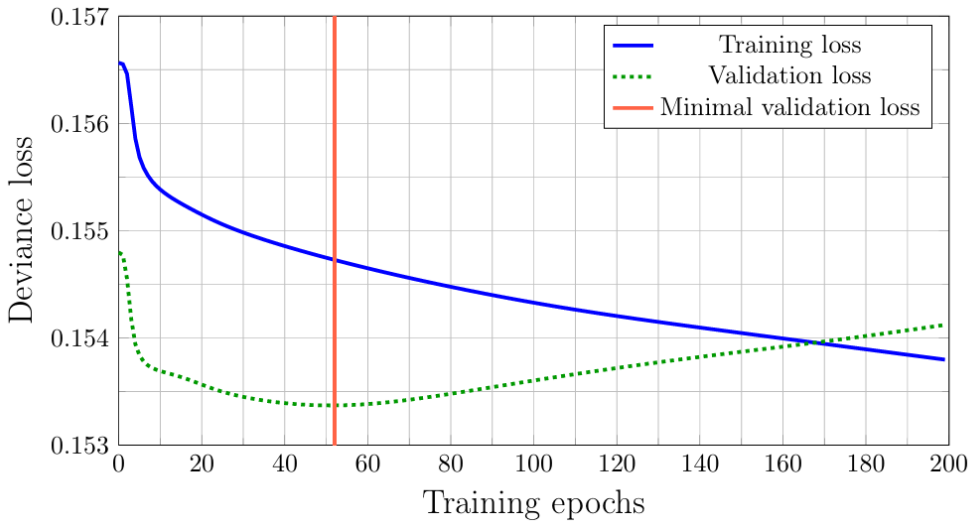


Figure 6.4: Training `3CANN` over 200 epochs.

The question remains, can we do even better?

As mentioned, a number of architectural as well as hyperparameter tweaks have been evaluated. We find that a five-layer FNN would allow us to improve upon `GLM_Poly` even further. This is a rather large model, and has 40,60,30,20,10 nodes per respective hidden layer. We illustrate the training of this model in Figure 6.5, and call this model `5CANN`.

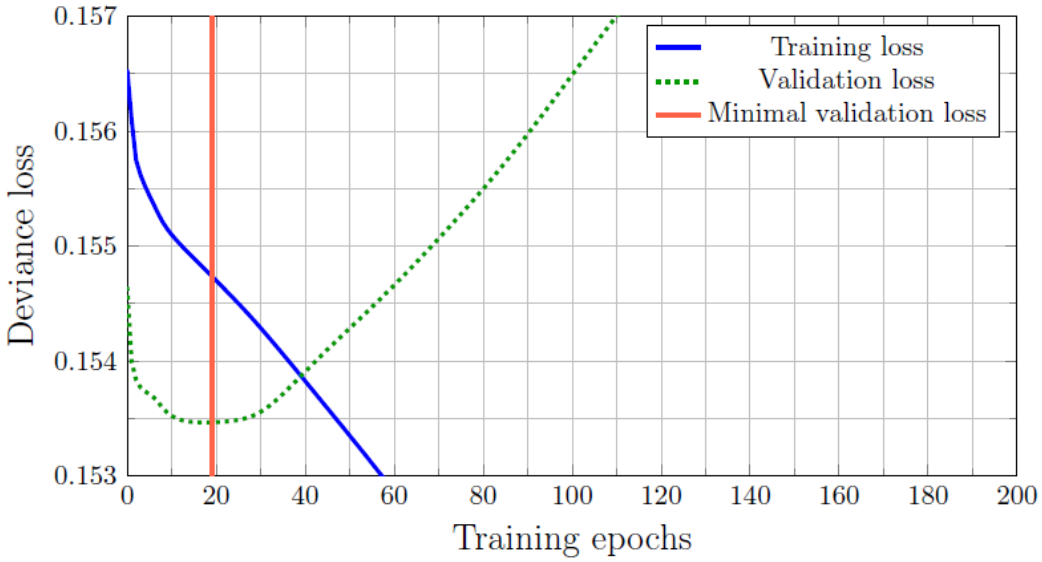


Figure 6.5: Training `5CANN` over 200 epochs.

Early stopping is exercised earlier than that of `3FNN`, this time at epoch 19. This is because a network of this size tends to achieve better training results per epoch, but also over-fits much faster. This is depicted in the figure, as the validation loss has a rather steep incline shortly after epoch 19. `5CANN` had an in-sample loss of 23.731 and an out-of-sample loss of 23.810, making it the best model so far. A comparison is found in Figure B.4, suggesting `5CANN` overestimates more than `3CANN` in relation to `GLM_Poly`.

6.3 GBM-CANNs

Having achieved promising results boosting GLMs, a curious thought would be to apply the same boosting method to other variants of regression models. Intuitively, it is not as straightforward what one benefits from performing boosting on more flexible models however. The core concept of CANN is to start with a highly interpretable, but rather non-flexible model, and boost this model in order to find more complex signal in data. Nonetheless, it is interesting to see what one could achieve out of a predictive point of view.

We therefore attempt to boost a *gradient boosting machine* (GBM) [35, pp. 111–122] to see if we can achieve similar improvements. In particular, *tree-based gradient boosting machines* are used, meaning we use regression trees as learners. We refrain from an in-depth theoretical presentation of GBMs, but point out that a series of regression trees are used where each new tree attempts to improve upon the previous tree in terms of errors. The

trees are often shallow. Similarly to neural networks, this process minimizes a loss function by use of gradient descent. The code for our GBM is provided in Listing A.3. Setting `n.trees` to 1000, we allow for training on 1000 trees, and `interaction.depth` is set to 1, meaning we use shallow trees. The same loss function is used as for previous models, the GBM achieves an in-sample loss of 23.849 and an out-of-sample loss of 23.882, which is much better than our GLMs but not quite at par with our FNNs. We call this model `GBM_Model` and a comparison with `GLM_Poly` is illustrated in Figure B.5.

Proceeding with the boosting, we first boost `GBM_Model` using `3FNN`. We call this model `GBM_3CANN`, and the training of this model is illustrated in Figure 6.6.

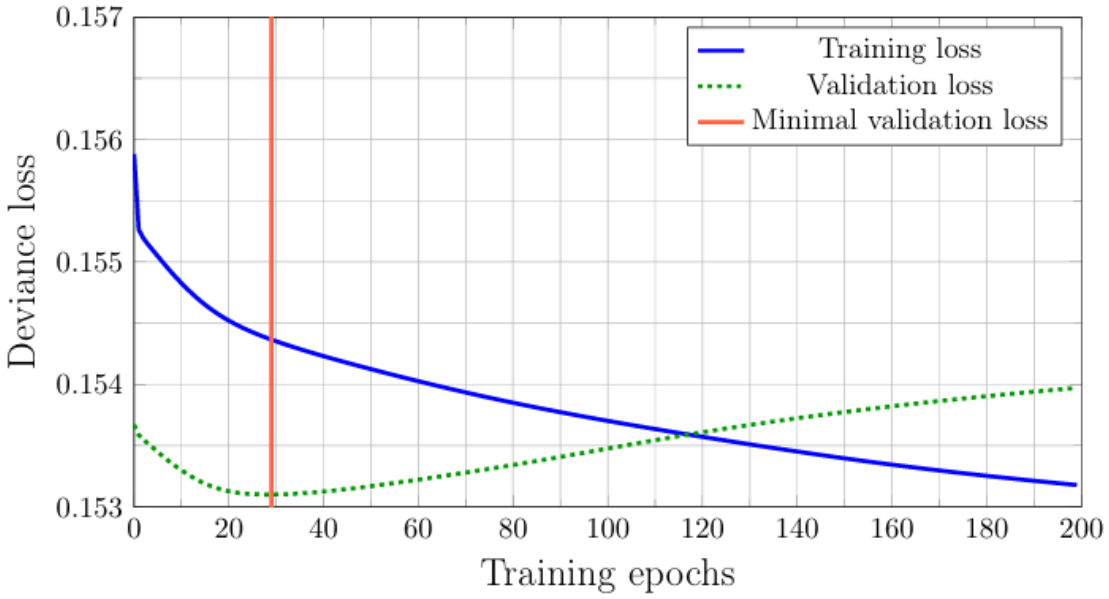


Figure 6.6: Training `GBM_3CANN` over 200 epochs.

Having trained `GBM_3CANN`, the best results out of all models was achieved, with an in-sample loss of 23.663 and an out-of-sample loss of 23.764 after 29 epochs. We compare `GBM_3CANN` with `GLM_Poly` in Figure B.6. The final model we evaluate is `GBM_Model` boosted with `5FNN`; we call this model `GBM_5CANN`. This model performed similarly to that of `GBM_3CANN` but with slightly higher losses at 23.776 and 23.766, see Figure 6.7. Early stopping was exercised at a very early epoch, 10, followed by a very steep incline in the validation loss curve, indicating this model over-fits very rapidly.

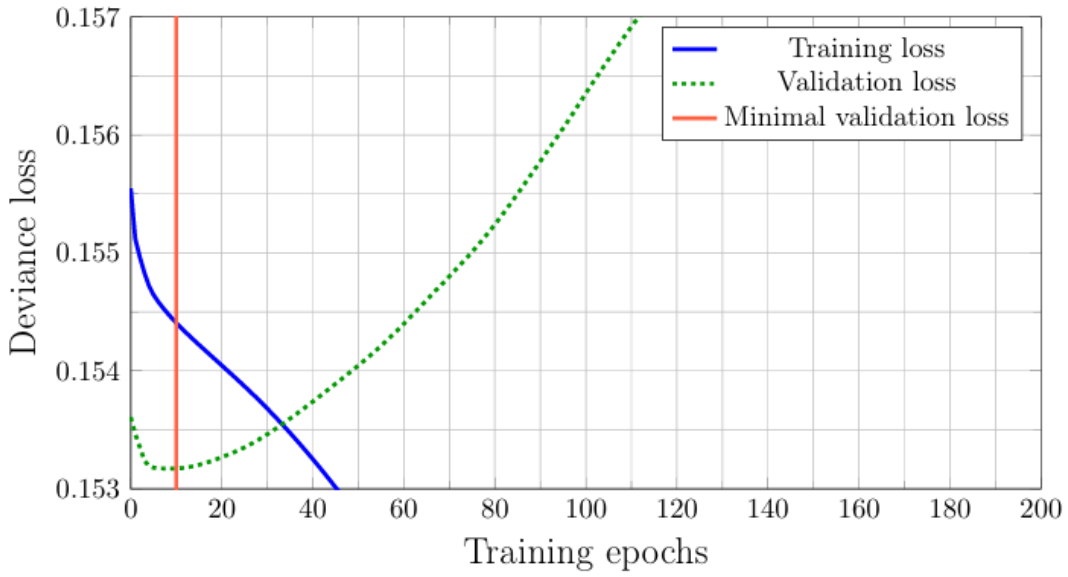


Figure 6.7: Training `GBM_5CANN` over 200 epochs.

6.4 Results

We end this chapter by summarizing all of our results, see Table 6.1. The poorest performance was obtained using `GLM_Merged`, and the best performance was obtained using `GBM_3CANN`.

Table 6.1: Deviance losses of the GLM, FNN, GBM, and CANN.

Model	In-sample loss	Out-of-sample loss	Epochs
`GLM_Merged`	24.102	24.129	–
`3FNN`	23.714	23.852	68
`3CANN`	23.735	23.823	52
`5CANN`	23.731	23.810	19
`GBM_Model`	23.849	23.882	–
`GBM_3CANN`	**23.663**	**23.764**	**29**
`GBM_5CANN`	23.667	23.766	10

Chapter 7

Conclusion

The purpose of this thesis is to study, learn, and apply the theory of feed-forward neural networks (FNNs). In particular, the end goal is to boost generalized linear models (GLMs) with FNNs using a combined actuarial neural network (CANN). This is applied to real French car insurance data, which is available as open-source content. A well-performing GLM (`GLM_Merged`) is used as a benchmark model to ensure we put the CANN to the test.

We conclude that the CANN is a powerful tool, achieving positive results and successfully boosting our GLM. The CANN models outperform both the GLMs but also the FNNs. We also manage to boost a gradient boosting machine (GBM) in a similar fashion, also with successful results.

Some of the difficulties of the CANN, alongside other machine learning methods, are to interpret the results. Exactly what interactions are we finding? Further research is recommended to gain a better understanding of this.

Bibliography

[1] Balona, C., "Operationalizing LLMs - A Guide for Actuaries", *Society of Actuaries*, 2025.

[2] Benigo, Y., Lamblin, P., Popovici, D., and Larochelle, H., "Greedy Layer-Wise Training of Deep Networks", *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pp. 153–160, 2006.

[3] Charpentier, A., *Computational Actuarial Science with R*. Chapman & Hall, 2016.

[4] Cramér, H., "Bidrag till utjämningsförsäkringens teori", in *Försäkringsbolags fondbildning och riskutjämning*. Stockholm: Försäkringsinspektionen, 1919, vol. III.

[5] Dorwart, R. A., "The Earliest Fire Insurance Company in Berlin and Brandenburg, 1705–1711", *The Business History Review*, vol. 32, no. 2, pp. 192–203, 1958.

[6] Dutang, C., Charpentier, A., Gallic, E., and Siharath, J., "CASdatasets: Insurance Datasets", version 1.2-0, 2024.

[7] Friedberg, S. H., Insel, A. J., and Spence, L. E., *Linear Algebra*, 4th ed. Pearson, 2014.

[8] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*. MIT Press, 2016.

[9] Hastie, T. and Tibshirani, R., "Generalized Additive Models", *Statistical Science*, vol. 1, no. 3, pp. 297–310, 1986.

[10] Hebb, D. O., *The Organization of Behavior*. John Wiley & Sons, 1949.

[11] Hinton, G. E., Osiendero, S., and Teh, Y.-W., "A Fast Learning Algorithm for Deep Belief Nets", *Neural Computation*, vol. 18, no. 7, 2006.

[12] Holvoet, F., Antonio, K., and Henckaerts, R., "Neural Networks for Insurance Pricing With Frequency And Severity Data", *arXiv*, 2024.

[13] Hornik, K., Stinchcombe, M., and White, H., "Multilayer Feedforward Netowrks are Universal Approximators", *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[14] Jørgensen, B., *The Theory of Dispersion Models*. Chapman & Hall, 1997.

[15] Kalinowski, T., Falbel, D., Allaire, J., *et al.*, "Package "keras"", 2024.

[16] Legendre, A. M., *Nouvelles méthodes pour la détermination des orbites des comètes*. 1805.

[17] Levenius, L. G., *Den Cramérska assuransmatematiken*. Stockholm University, 2025.

[18] Lindholm, M., Lindskog, F., and Palmquist, J., "Local Bias Adjustment, Duration-Weighted Probabilities, and Automatic Construction of Tariff Cells", *Scandianvian Actuarial Journal*, vol. 2023, no. 10, pp. 946–973, 2023.

[19] Lindskog, F., "Non-life Pricing Essentials", 2024.

[20] McCullagh, P. and Nedler, J. A., *Generalized Linear Models*. Chapman & Hall, 1983.

[21] McCulloch, W. and Pitts, W., "A Logical Calculus of Ideas Immmanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[22] R. Morris and M. Fillenz, Eds., *Neuroscience: Science of the Brain*. The British Neuroscience Association, 2003.

[23] Murphy, K. P., Brockman, M. J., and Lee, P. K. W., *Using Generalized Linear Models to Build Dynamic Pricing Systems for Personal Lines Insurance*. Casualty Actuarial Society Forum, 2000, pp. 107–140.

[24] Nedler, J. A. and Wedderburn, R. W. M., "Generalized Linear Models", *Journal of the Royal Statistical Society. Series A*, vol. 135, no. 3, pp. 370–384, 1972.

[25] Nielsen, M., *Neural Networks and Deep Learning*. 2019.

[26] Noll, A., Salzmann, R., and Wüthrich, M. V., "Case Study: French Motor Third-Party Liability Claims", *Swiss Association of Actuaries SAV*, 2020.

[27] Ohlsson, E. and Johansson, B., *Non-Life Insurance Pricing with Generalized Linear Models*. Springer, 2015.

[28] ——, *Gradient Boosting Machines and Non-Life Insurance Pricing*. Stockholm University, 2022.

[29] Pearson, K., "On a Mathematical Theory of Determinal Inheritance from Suggestions and Notes of the Late W.F.R Weldon", *Biometrika*, vol. 6, no. 1, pp. 80–93, 1908.

[30] Ranzato, M., Poultney, C., Chopra, S., and Cun, Y. L., "Efficient Learning of Sparse Representations with an Energy-Based Model", *Advances in Neural Information Processing Systems*, pp. 1137–1144, 2006.

[31] Rumelhart, D., Hinton, G., and Williams, R., "Learning Representations by Back-Propagating Errors", *Nature*, vol. 323, pp. 533–536, 1986.

[32] Schelldorfer, J. and Wüthrich, M. V., "Nesting Classical Actuarial Models into Neural Networks", *Swiss Association of Actuaries SAV*, 2019.

[33] Spearman, C., "The Proof and Measurement of Association between Two Things", *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.

[34] Stigler, S. M., *The History of Statistics*. Harvard University Press, 1986.

[35] Wüthrich, M. V., Lindholm, M., Richman, R., *et al.*, "AI Tools for Actuaries", 2025.

[36] Wüthrich, M. V. and Merz, M., "Editorial: Yes, We CANN!" *ASTIN Bulletin*, vol. 49, no. 1, pp. 1–3, 2018.

[37] ——, *Statistical Foundations of Actuarial Learning and its Applications*. Springer, 2023.

[38] Zakrisson, H., *Tree-Based Machine Learning Methods With Non-life Insurance Applications*. Stockholm University, 2024.

[39] Zhan, J., *Neural Netwerking Beyond Lee-Carter: A Song of Mortality Forecasting and Deep Learning*. Stockholm University, 2025.

Appendix A

Code Listings

Listing A.1: Feature pre-processing code.

```
# Pre-processing function for FNN - One-hot encoding and standardization

# One-hot encoding function
PreProcess.OneHot <- function(var1, name, dat2){
   names(dat2)[names(dat2) == var1]  <- "V1"
   XX <- data.frame(to_categorical(as.integer(dat2$V1)))
   colnames(XX) <- paste0(name, c(1:ncol(XX)))
   names(dat2)[names(dat2) == "V1"]  <- var1
   cbind(dat2, XX)
   }
# Standardization function
PreProcess.Continuous <- function(var1, dat2){
   names(dat2)[names(dat2) == var1]  <- "V1"
   dat2$X <- as.numeric(dat2$V1)
   dat2$X <- (dat2$X-mean(dat2$X))/sd(dat2$X)
   names(dat2)[names(dat2) == "V1"]  <- var1
   names(dat2)[names(dat2) == "X"]  <- paste0(var1,"X")
   dat2
   }
# Apply to data
Features.PreProcess <- function(dat2){
   dat2 <- PreProcess.Continuous("Area", dat2)
   dat2 <- PreProcess.Continuous("VehPower", dat2)
   dat2$VehAge <- pmin(dat2$VehAge,20)
   dat2 <- PreProcess.Continuous("VehAge", dat2)
   dat2$DrivAge <- pmin(dat2$DrivAge,90)
   dat2 <- PreProcess.Continuous("DrivAge", dat2)
   dat2$BonusMalus <- pmin(dat2$BonusMalus,150)
   dat2 <- PreProcess.Continuous("BonusMalus", dat2)
   dat2 <- PreProcess.OneHot("VehBrand", "B", dat2)
   dat2$VehGasX <- as.integer(dat2$VehGas)-1
   dat2$Density <- round(log(dat2$Density),2)
   dat2 <- PreProcess.Continuous("Density", dat2)
   dat2 <- PreProcess.OneHot("Region", "R", dat2)
   dat2
    }
```

Listing A.2: Matrix preparation for the FNN.

```
    # Constructing the needed matrices

features <- c("AreaX", "VehPowerX", "VehAgeX", "DrivAgeX", "BonusMalusX",
              "VehGasX", "DensityX", paste0("B", c(1:11)), paste0("R", c
   (1:22)))
(input_layer <- length(features))
Xtrain <- as.matrix(train[, features])  # design matrix training sample
Xtest  <- as.matrix(test[, features])   # design matrix test sample

Vtrain <- as.matrix(train$Exposure)     # This is the offset for FNN
Vtest  <- as.matrix(test$Exposure)      #

GLM_train <- as.matrix(trainGLM$GLM)    # We use the GLM as offset for
   CANN
GLM_test <- as.matrix(testGLM$GLM)      # We use the GLM as offset for
   CANN

Ytrain <- as.matrix(train$ClaimNb)      # Response(train)
Ytest  <- as.matrix(test$ClaimNb)       # Response(test)
```

Listing A.3: GBM code.

```
gbm_model <-  gbm(ClaimNb ~ VehPowerGLM + VehAgeGLM + BonusMalusGLM
                        + VehBrand + VehGas + DensityGLM + Region +
   AreaGLM +
                        DrivAgeGLM + offset(log(Exposure)),
                      distribution = "poisson",
                      data = trainGLM,
                      n.trees = 1000,
                      cv.folds = 5,
                      interaction.depth = 1)
```

Appendix B

Figures

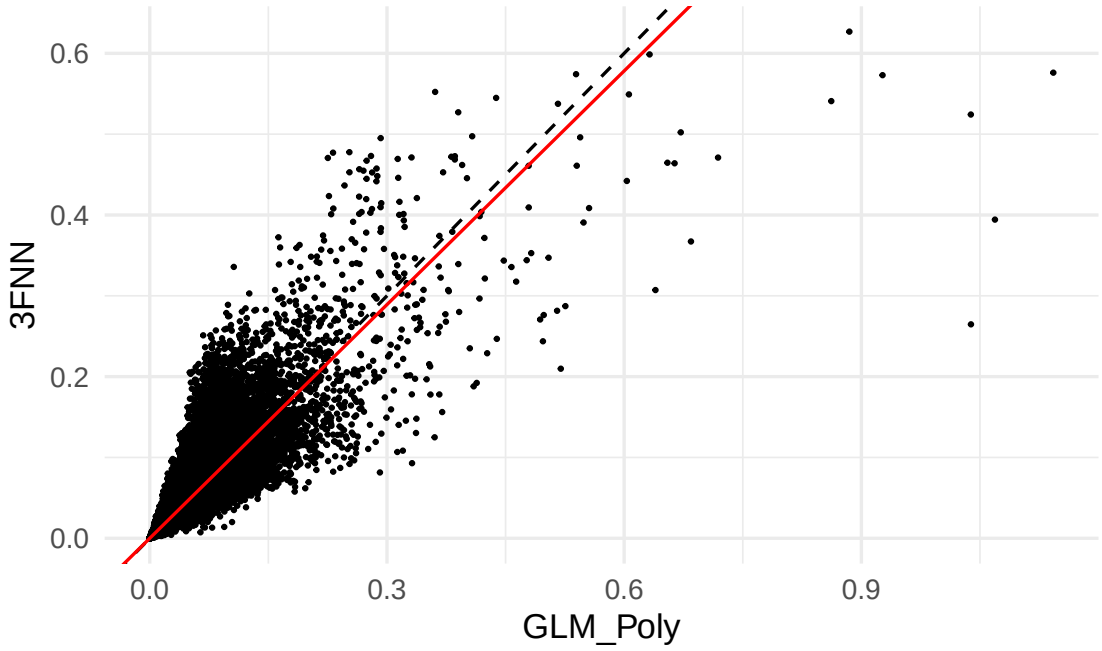


Figure B.1: `GLM_Poly` against `3FNN`.

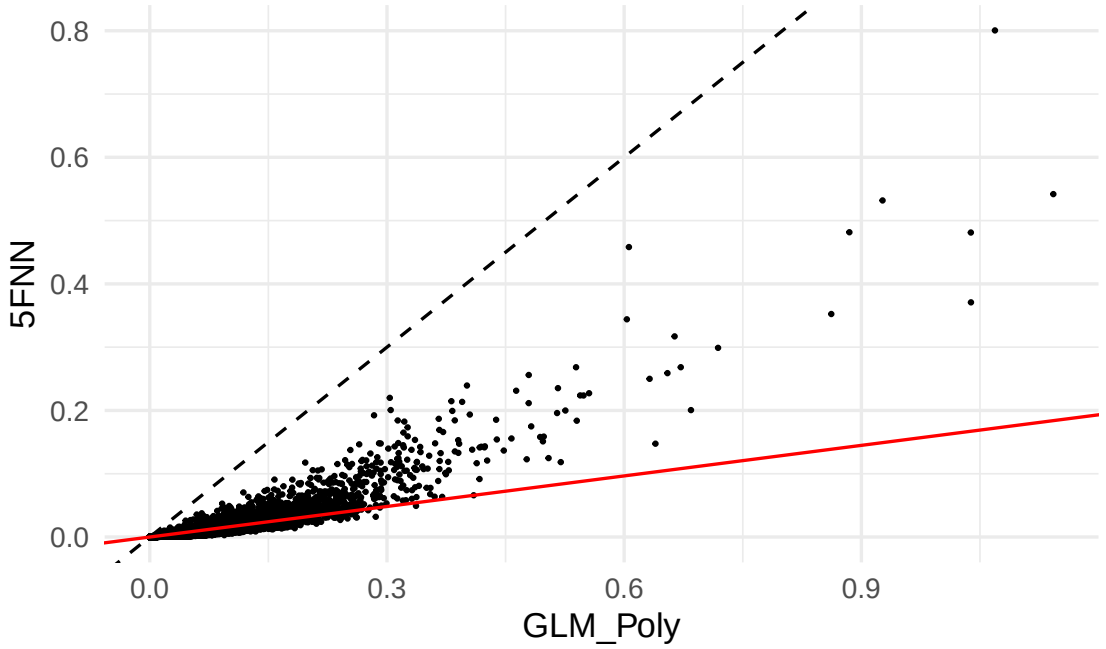


Figure B.2: `GLM_Poly` against `5FNN`.

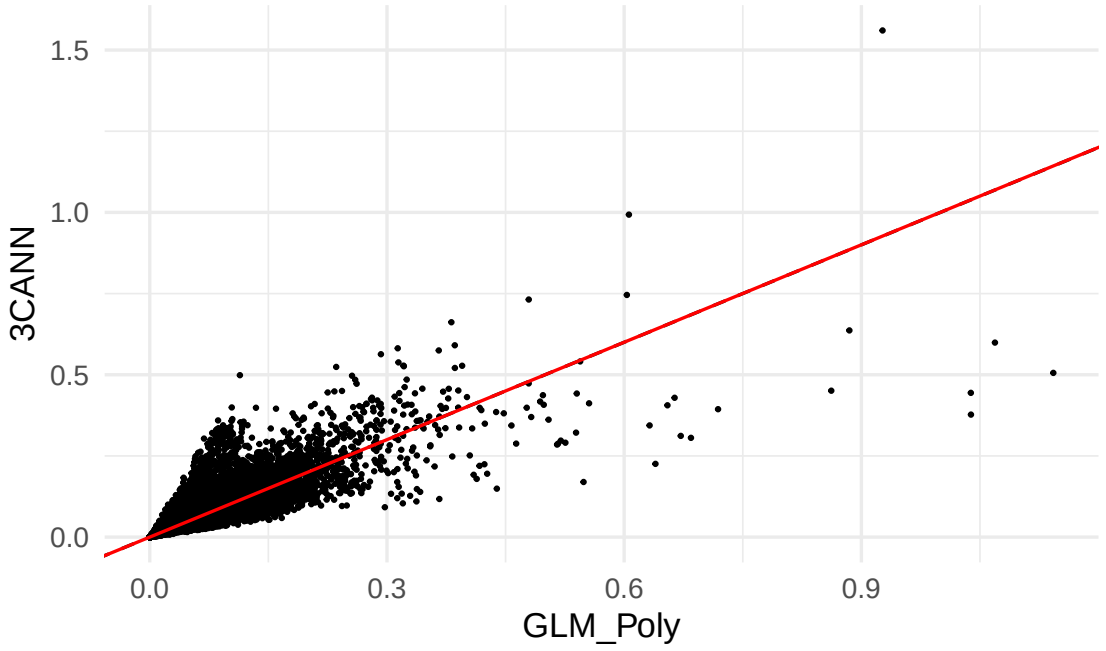


Figure B.3: `GLM_Poly` against `3CANN`.

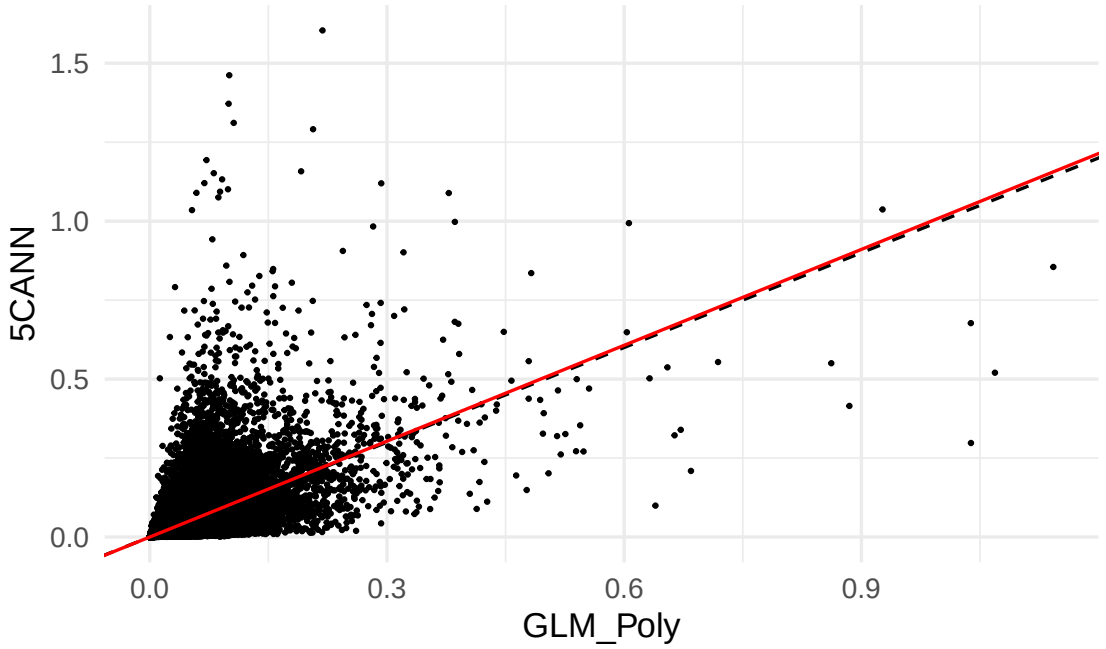


Figure B.4: `GLM_Poly` against `5CANN`.

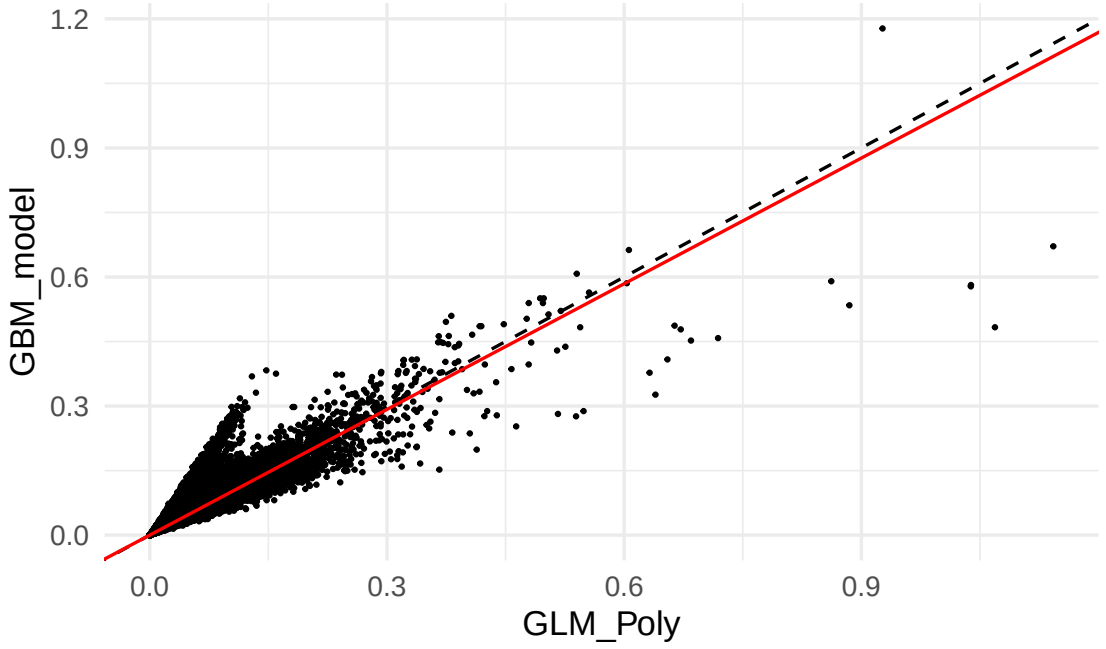


Figure B.5: `GLM_Poly` against `GBM_Model`.

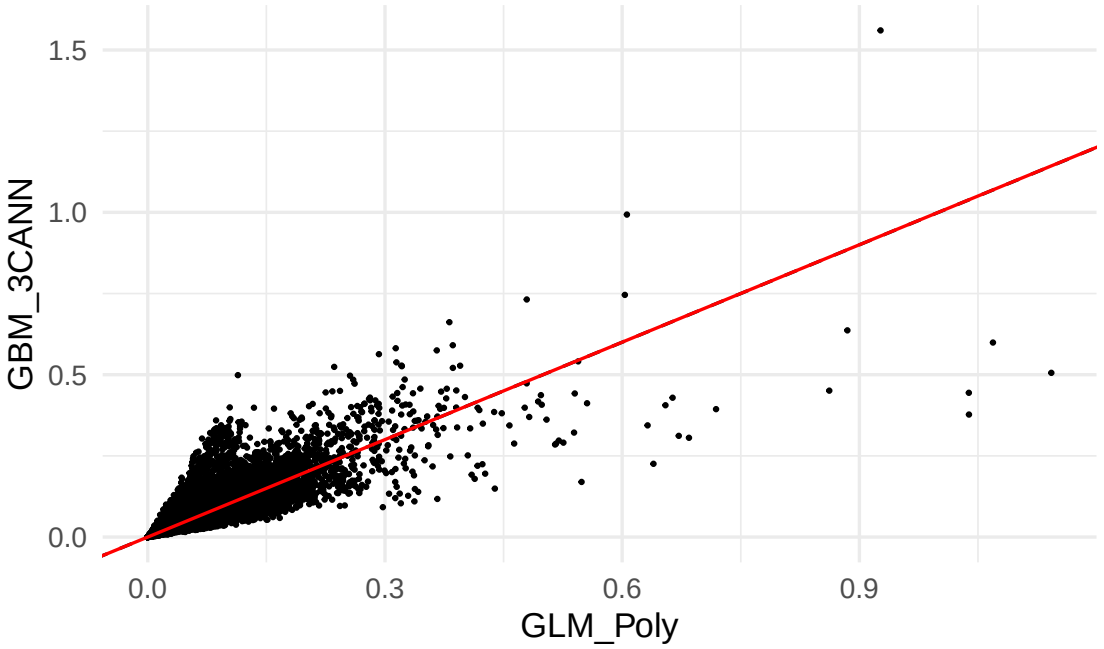


Figure B.6: `GLM_Poly` against `GBM_3CANN`.

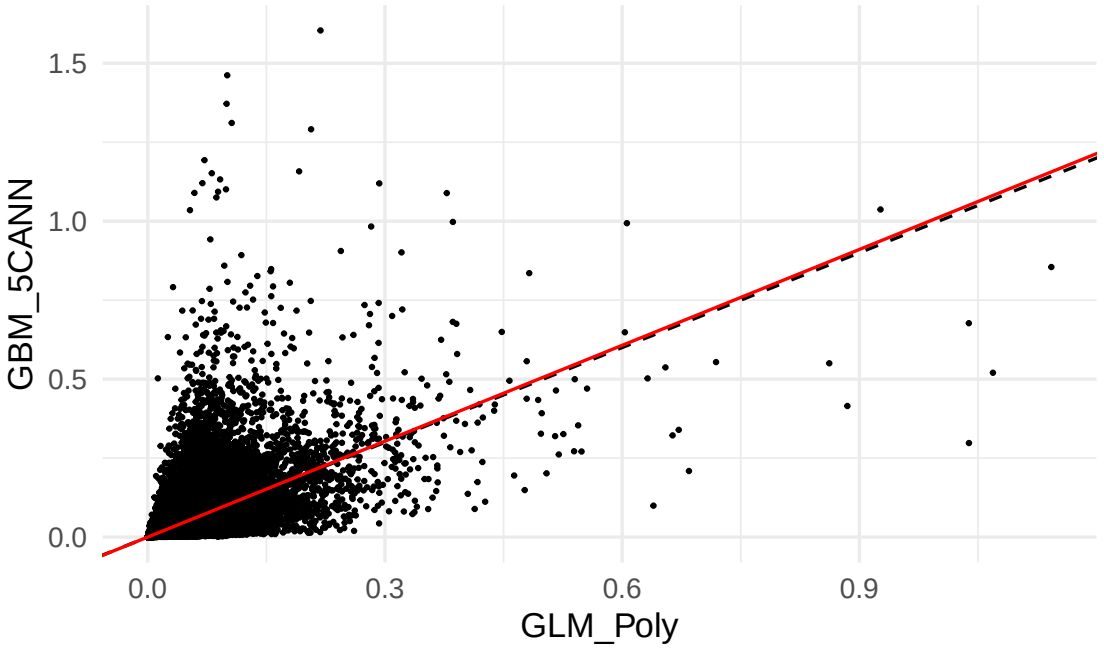


Figure B.7: `GLM_Poly` against `GBM_5CANN`.