Enrollment No.: 2021MSTA005

Neural Network based enhancement of the Bell regression model for claim frequency data

Project Report

A thesis submitted in fulfillment of the requirements

for the degree

of

Master of Sciences

in

Statistics

Submitted By:

Mr. Debjyoti Mukherjee

Supervisor:

Dr. Deepesh Bhati

Associate Professor

Department of Statistics

School of Mathematics, Statistics and Computational Sciences

Central University of Rajasthan, Ajmer - 305817

2021-2023

Dr. Deepesh Bhati
Associate Professor
Department of Statistics

Email : deepesh.bhati@curaj.ac.in
Website: http://www.curaj.ac.in

राजस्थान केंद्रीय विश्वविद्यालय
Central University of Rajasthan
(संसद के अधिनियम द्वारा स्थापित केंद्रीय विश्वविद्यालय)
(A Central University by an Act of Parliament)
NH-8, Bandarsindri, 305801
Kishangarh (Ajmer), Rajasthan, INDIA
Phone (Office): +91-1463-238755/260200
Telefax: +91-1463-238722

Certificate

This is to certify that the work embodied in the accompanying project report entitled ***"Neural Network based enhancement of the Bell regression model for claim frequency data"*** has been successfully carried by ***Mr Debjyoti Mukherjee***, a IV Semester student of M.Sc. Statistics, of the Department of Statistics, Central University of Rajasthan, under my guidance and supervision.

He worked for about ... weeks and his work carried out is satisfactory.

Place: Cental University of
Rajasthan, Bandarsindri
Date:

Dr. Deepesh Bhati
Associate Professor
Department of Statistics

TO WHOMSOEVER IT MAY CONCERN

This is to certify that ***Mr. Debjyoti Mukherjee***, Enroll. No 2021MSTA005, M.Sc. Statistics student of Department of Statistics has done project work ***"Neural Network based enhancement of the Bell regression model for claim frequency data"*** under the guidance of ***Dr. Deepesh Bhati***, Associate Professor at Department of Statistics, Central University of Rajasthan towards the partial fulfillment of the award of "Master of Science in Statistics" during the period March 2023 to July 2023.

Place: Cental University of
Rajasthan, Bandarsindri
Date:

Dr. JITENDRA KUMAR
Head
Department of Statistics
Central University of Rajasthan

Declaration

I, Debjyoti Mukherjee, hereby declare that the project work entitled **"Neural Network based enhancement of the Bell regression model for claim frequency data"** submitted to Department of Statistics, Central University of Rajasthan as a partial fulfilment of requirements of IV-Semester examination, is a bona fide record of work under taken by me, under the supervision of **Dr. Deepesh Bhati**, Associate Professor at Department of Statistics, Central University of Rajasthan and it is not formed the basis for the award of any other Degree/Associateship/Fellowship by any University.

Signature of Candidate

Debjyoti Mukherjee

Enroll. No.: 2021MSTA005

Place: Bandarsindri

Date: .....

Acknowledgement

I would like to express my sincere gratitude to all those who have supported and guided me throughout the journey of completing this thesis.

First and foremost, I am deeply thankful to my supervisor, **Dr. Deepesh Bhati**, for their invaluable guidance, unwavering support, and insightful feedback. Their expertise and dedication played a pivotal role in shaping the direction of my research and enhancing the quality of this work.

I am also grateful to Ph.D. Scholar, **Mr. Girish Aradhye** who provided huge support and encouragement for learning and research.

My heartfelt thanks go to my family and friends for their constant encouragement and understanding during this challenging phase of my academic journey. Their belief in me kept me motivated to overcome obstacles and strive for excellence.

Lastly, I am indebted to the various resources, libraries, and research materials that have contributed to the depth and authenticity of this thesis.

Thank you all for being a part of this fulfilling endeavor.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many of the times the actuarial claim frequency data exhibits over-dispersion phenomenon, which is not encountered by the conventional Poisson model.Various explanatory variables are available along with the claim count so a regression modes are becoming a practical tool. The conventional GLM is a useful device to study the impact of available co-variates on non-normal response distributions.GLMs are not efficient to capture the non-linear interactions present among the co-variates. Hence, neural network based approach are used to study the non-linear interactions in an efficient manner.Hence, in this work we propose a enhancement of the Bell GLM using feed-forward neural network based approach to (i) automate the pre-processing of the co-variates and (ii) boost the conventional GLM by capturing the non-linear interactions via CANN(Combined Acturial Neural Networks) models. The primary objective of our research was to assess the enhanced predictive capabilities that NN-based models offer in comparison to conventional count regression models, particularly when applied to an extensive real-world dataset within the domain of Claim frequency insurance data. To enhance the accuracy of predictions. In our pursuit of developing accurate and robust models, we also used bias regularization techniques to address common biases that may arise during the fitting of NN models. The outcomes of our study underscore the superiority of NN-based models,in terms of predictive performance.This advancement in predictive accuracy carries significant implications for both personalized insurance strategies and policy-level interventions.

Chapter 2

Model Framework

2.1 Bell Distribution

The probability mass function of the Bell distribution Bell(θ) is

$$P(Y = y) = \frac{\theta^y e^{-e^\theta+1} B_y}{y!} \quad for \qquad y = 0, 1, 2, ... \tag{2.1}$$

where $\theta > 0$, and the coefficient

$$B_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!} \tag{2.2}$$

are the Bell numbers.

Starting with $B_0 = B_1 = 1$,the first few Bell numbers are $B_2 = 2$,$B_3 = 5$,$B_4 = 15$,$B_5 = 52$, $B_6 = 203$, $B_7 = 877$, $B_8 = 4140$, $B_9 = 21147$, $B_{10} = 115975$, $B_{11} = 678570$,$B_{12} = 4213597$ and $B_{13} = 27644437$.

The mean and variance of the Bell distribution are, respectively,

$$E(Y) = \theta e^\theta, \qquad Var(Y) = \theta(1 + \theta)e^\theta \tag{2.3}$$

In a regression model framework, it is typically more useful to model the mean of the response

variable. So, to obtain a regression structure for the mean of the Bell distribution, we shall work with a different parameterization of the Bell probability mass function. Let $\mu = \theta e^{\theta}$, then $\theta = W_0(\mu)$, where $W_0(.)$ is the Lambert function. Then it follows from (2.3)

$$E(Y) = \mu, \qquad Var(Y) = \mu(1 + W_0(\mu)) \tag{2.4}$$

so that $\mu > 0$ is the mean of the response variable Y.

The Bell probability mass function can be written, in the new parameterization, as

$$P(Y = y) = exp(1 - e^{W_0(\mu)})\frac{W_0(\mu)^y B_y}{y!} \qquad y = 0, 1, 2, ... \tag{2.5}$$

where $\mu > 0$, and B_y are the Bell numbers in (2.2) . We have that $W_0(\mu) > 0$ for $\mu > 0$ and, therefore, $Var(Y) > E(Y)$. It implies that the Bell distribution may be suitable for modeling count data with over-dispersion, similar the two-parameter NB distribution.

2.2 Regression model

Assume that the claim numbers, denoted by y_i , are independent and distributed as Bell distribution: $y_i \sim Bell(\mu_i)$

where the mean parameter μ_i depends on the policyholder's characteristics $\mathbf{x}_i$ and an offset of the claims o_i .

For the Bell regression, by choosing the logarithmic link function, we have

$$\hat{\mu} : \mathcal{X} \to \mathcal{R}^+ \qquad \mu_i = exp(o_i + \langle \beta, \mathbf{x}_i \rangle) \tag{2.6}$$

where $\boldsymbol{\beta}$ is the unknown coefficient vector to be estimated by MLE.

Chapter 3

Neural Network

Neural Networks (NNs) are computer systems that are often said to operate in a similar fashion to the human brain. A NN is a series of units called neurons. These are usually simple processing units which take one or more inputs and produce an output. When a NN is trained these weights are adjusted to bring the output as close as possible to that desired.

We focus on the fundamental feed-forward neural network (FFNN) model. Typically, an FFNN consists of several components: an input layer, one or more hidden layers, and an output layer. The input layer is constructed from the feature space $\mathcal{X}$, while each hidden layer comprises a fixed number of neurons. The output of a specific hidden layer serves as the input for the subsequent layer in the network. The output generated by a neuron relies on a linear combination of the preceding layer's output and the activation function associated with the respective layer (for a detailed explanation of activation functions, please refer to Section 3.3). The network's depth, denoted as 'd' and belonging to the set of natural numbers ($d \in \mathcal{N}$), is regarded as a hyperparameter and is commonly referred to as the network's depth. The final layer in this architecture, connected to the last hidden layer, is known as the output layer. In a neural network structure, each layer's function depends on the preceding layer.

3.1 Neural Network Feature Preparation

3.1.1 Continuous predictors:

Min-Max Scaler

It is necessary for the neural network to have inputs in the same range for faster convergence of the gradient-descent algorithm. The scaler shifts the predictor linearly between -1 and 1, without changing the relative difference between each value to ensure that all the continuous predictors are of similar importance when performing the gradient descent algorithm.

$$x^{(k)} \rightarrow 2\frac{x^{(k)} - min\ x^{(k)}}{max\ x^{(k)} - min\ x^{(k)}} - 1 \tag{3.1}$$

3.1.2 Categorical predictors:

Embedding Layer

A widely-used technique for pre-processing categorical features involves their conversion into numeric vectors. The conventional approach is to represent a feature 'x,' which can take on 'c' distinct categories $(v_1, ..., v_c)$, with a binary vector 'x' of dimension 'c.' In this binary vector, only one coordinate, 'x_i,' is set to 1 if and only if 'x' corresponds to the category 'v_i.' This technique is commonly referred to as 'one-hot encoding. We must question whether the straightforward implementation of categorical feature components using one-hot encoding is the optimal choice, as it appears to introduce an excessive number of network parameters. There is a second reason why one-hot encoding may not be the most suitable option for our objectives. In general, we aim to identify or cluster labels that exhibit similarity in the context of the regression modeling problem. However, one-hot encoding does not facilitate this.

To illustrate, if we consider a set of 11 vehicle brands, denoted as $B = B1, B10, ..., B6$, one-hot encoding assigns a distinct unit vector 'x' in the real numbers $\mathcal{R}$) to each 'VehBrand' within B. For two different brands, 'VehBrand1' and 'VehBrand2' (where VehBrand1 $\neq$ VehBrand2) from the set

B, we always observe , the Euclidean distance between them is the same $\sqrt{2}$.

Here we introduce 'embedding layers' that aim to project categorical feature components into low-dimensional Euclidean spaces. This approach allows for the clustering of labels that exhibit greater similarity within the context of the regression modeling problem. One significant advantage of this approach is that the learned feature representations capture the inherent characteristics of the features. For instance, embeddings find extensive use in natural language processing, where they reflect the semantic similarity between words. An illustrative example is the relationship 'emb(king) - emb(queen) $\sim$ emb(man) - emb(woman),' which demonstrates the ability of embeddings to capture meaningful semantic relationships. Embedding layers aim at mapping every level of categorical feature components to a lower dimensional vector, hence reducing the number of network parameters.

Each label of the categorical predictors will then be represented by a 2-d vector when flowing into the main architecture of the network.

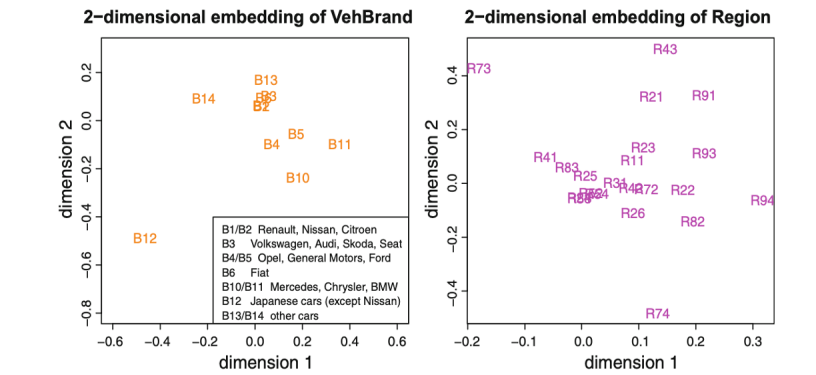


Figure 3.1: Embedding weights for $e^{VehBrand} \in \mathcal{R}^2$ and $e^{Region} \in \mathcal{R}^2$ of the categorical variables VehBrand and region for the embedding dimension $d = 2$

3.2 Neural Network set-up

While defining the architecture of the neural network the choice for the hyperparameters (depth, number of neurons, etc.) should be determined carefully to avoid over-fitting.

3.2.1 Activation Functions

When the activation function is nonlinear, a deep neural network can compute intricate nonlinear functions. Indeed, according to the universal approximation theorem, any continuous function can be approximated with negligible error by a neural network that includes a single hidden layer with a nonlinear activation function. However, such functions are unlikely to generalize effectively because the hidden layer would need to be excessively large. Consequently, they are not practical for learning models that generalize well to new data. Instead, the preference lies in using multilayer networks, where each layer computes specific patterns. These patterns are then integrated by the subsequent layer to create more intricate patterns. In the context of computer vision, one can envision the initial layers of a neural network as detectors of fundamental image features such as edges and contours. Conversely, the subsequent layers amalgamate these fundamental shapes to form more elaborate patterns, which represent various parts of objects. In our case, we choose hyperbolic tangent 'tanh' The choice of non-linear activation functions allows for a non-linear model space and reduce the number of nodes needed. This allows the NN to automatically capture the interaction effect of different features. For output layer: exponential function, the inverse of log This choice is consistent with the link function of the regression model, as in (2.2).

	Activation function	Derivative
Sigmoid (logistic) activation	$\phi(x) = (1 + e^{-x})^{-1}$	$\phi' = \phi(1 - \phi)$
Hyperbolic tangent activation	$\phi(x) = \tanh(x)$	$\phi' = 1 - \phi^2$
Exponential activation	$\phi(x) = \exp(x)$	$\phi' = \phi$
Step function activation	$\phi(x) = \mathbb{1}_{\{x \geq 0\}}$	
Rectified linear unit (ReLU) activation	$\phi(x) = x\mathbb{1}_{\{x \geq 0\}}$	

Figure 3.2: Various activation functions and their derivatives

Having simple derivatives is an advantage in gradient descent algorithms for model fitting. The hyperbolic tangent activation function is anti-symmetric w.r.t. the origin with range (-1,1). This anti-symmetry and boundedness is an advantage in fitting deep FN network architectures. For this reason we usually prefer the hyperbolic tangent over other activation functions.

3.2.2 Hidden Layers, Batch size and epochs

- Number of hidden layers: the number of hidden layers was kept at three.

- Other main model attributes that need to be determined are the batch size, and epochs. Due to the computational burden of considering a large data set at once, during the training of a neural network, the data in the training set $\mathcal{D}^{(-)}$ are considered in smaller batches The batch size b refers to the size of the smaller batches created. Epochs give the number of times that the full learning data set is iteratively considered during training. The ideal choice of batch size must be determined in conjunction with the epochs as it determines the total number of gradient descent steps undertaken during model training, which impacts the model performance. Due to the size of the data in this work, we determined the batch size and epochs using trial and error. The were considered using a Bell NN with several different model architectures of varying complexity. For all architectures, three hidden layers were used (100,75,50), with (75,50,25), (50,35,25), (35,25,20), (25,20,15), (20,15,10), and (15,10,5) neurons in each of the three hidden layers. or the dataset we used, the following combination gives relatively good performance:

  Depth: $d = 3$,

  Number of Neurons in each layer: 100, 75, 50

  batch size = 100000

  epochs = 1000

3.2.3 Gradient descent method(GDM)

The training of the neural network employs a gradient descent optimization algorithm to estimate the model weights. In their study, Ferrario et al. (2020) conducted a comparison of various gradient descent methods in terms of their performance and identified the Nesterov-accelerated adaptive moment estimation (Nadam) method as the superior choice when compared to other similar methods. Consequently, we have also adopted the Nadam algorithm as our preferred gradient descent

method.

3.2.4 Loss function : Deviance Loss

The loss function serves as the objective function minimized by the Gradient Descent Optimization Algorithm (GDM) to estimate the model weights (Goodfellow et al., 2016). When it comes to selecting the appropriate loss function, there are several options available. Notable choices for loss functions in regression problems include Mean Squared Error (MSE), Mean Absolute Error (MAE), and Deviance Loss. In our specific context, we have opted for the Deviance Loss as our chosen loss function. The rationale behind this choice lies in the fact that minimizing the Deviance Loss is equivalent to maximizing the corresponding log-likelihood function, which yields the Maximum Likelihood Estimate (MLE). The Deviance Loss is defined as the disparity between the log-likelihood of the saturated or full model and that of the fitted model. For a dataset of sample size n the Bell Deviance Loss can be expressed as:"

$$\mathcal{D}(\mathbf{Y}, \hat{\boldsymbol{\mu}}) = \begin{cases} \frac{2}{n}\sum_{i=1}^{n}(e^{W_0(\hat{\mu_i})} - 1) & y_i = 0 \\ \frac{2}{n}\sum_{i=1}^{n}[e^{W_0(\hat{\mu_i})} - e^{W_0(y_i)} + y_i.log(\frac{W_0(y_i)}{W_0(\hat{\mu_i})})] & y_i > 0 \end{cases} \tag{3.2}$$

This formula could also be derived form the exponential form of the bell distribution.This makes sure that we can make the Bell regression model comparable to the above discussed neural network models.

3.2.5 Overfitting Problem

Over-fitting is a very common problem in case of neural networks. To prevent this we used validation split. Typically, a neural network model tends to experience over-fitting after a certain number of epochs. The underlying rationale behind the early-stopping technique is to pinpoint the ideal number of epochs beyond which the model enters the over-fitting phase. In simpler terms, the objective is to find the epoch count at which the validation loss begins to rise, as an increase in the validation loss serves as an indicator of over-fitting.

Numerous methodologies for implementing early stopping are available,in our specific implementation, we employ a callback approach. Initially, the model is trained for a large number of epochs. Following the completion of training, we retrieve the model weights that correspond to the lowest value of the loss function on the validation set. In the training of neural network models, it is necessary to further partition the learning dataset(90 percent of the whole dataset as 10 percent is preserved as testing data) into two subsets: a training set, denoted as $D^{(-)}$, and a validation dataset, denoted as V. The validation dataset plays a crucial role in the iterative process of estimating the model weights. To be precise, V serves as an evaluation set during the training process to monitor the potential occurrence of overfitting of the model to $D^{(-)}$. In the context of the network-based models under consideration here, we have chosen an 80:20 split ratio for the training and validation datasets. Upon the completion of the training phase, the ultimate performance of the fitted model is assessed using the testing dataset.

Chapter 4

Regression Networks and CANN models

4.1 Neural Networks for Regression

For neural network fitting, we replace the predictor of regression model

$$\hat{\mu} : \mathcal{X} \to \mathcal{R}^{+} \qquad \mu_i = exp(o_i + \langle \beta, \mathbf{x}_i \rangle) \tag{4.1}$$

by the neural network predictor

$$\hat{\mu}_{NN} : \mathcal{X} \to \mathcal{R}^{+} \qquad \mu_i = exp(o_i + \langle W^{(d+1)}, (z^{(d)} \circ ... \circ z^{(1)})(x_i) \rangle) \tag{4.2}$$

where d is the depth of the network and $W^{(d+1)}$ is the weights which maps the neurons of the last hidden layer $z^{(d)}$ to the output layer $\mathcal{R}^{+}$.

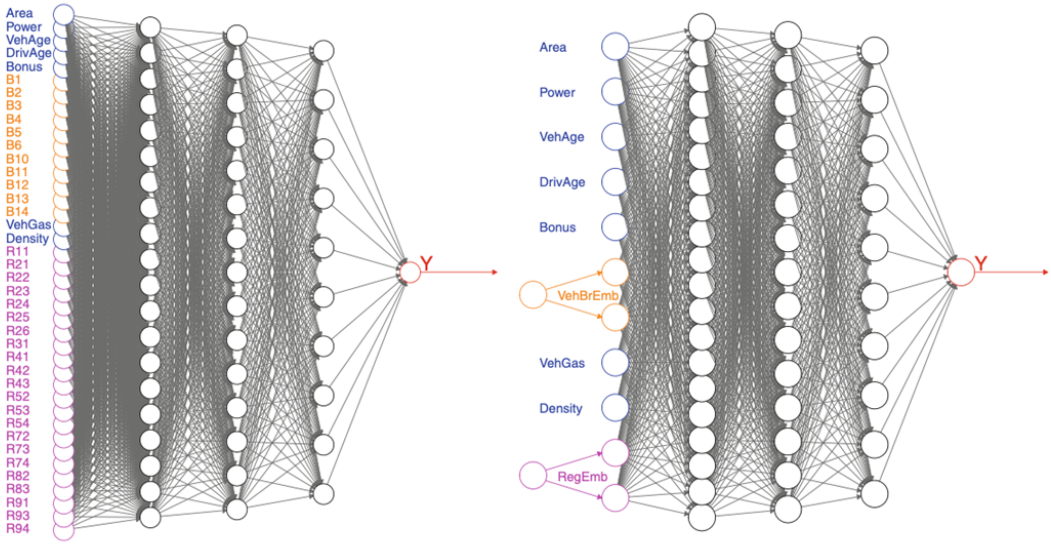


Figure 4.1: (lhs) One-hot encoding with $q_0 = 40$ and (rhs) embedding layer for VehBrand and region with embedding dimension 2 and network depth 3

4.2 Combined Neural Network for Bell Regression(CANN)

The CANN model has an additional regression function nested into the model predictor using a skip connection. Following Wuthrich ad Merz's work "Yes, we CANN!", we define a combined neural network - Bell regression model under the Bell distributional assumption, Under the Bell distributional assump- tion with depth $d \in \mathcal{N}$,

$$\hat{\mu}_{CANN} : \mathcal{X} \to \mathcal{R}^{+} \qquad \mu_i = exp(o_i + \langle \beta, \mathbf{x}_i \rangle + \langle W^{(d+1)}, (z^{(d)} \circ ... \circ z^{(1)})(x_i) \rangle) \tag{4.3}$$

The first term in (4.3) is the regression function (also called the skip connection), and the second term in (4.3) is the neural network function. The weights of the last layer of the network output is assigned such that the inital output of the neural network mathces with that of the Bell regression. To do this , we set the weights of the last layer to zero for all the weights and biases. Also we set the final exponentiation part weight and bias to be 1 and 0 respectively, and making these weights non-trainable. There can be two variants of this CANN model based on wether the regression estimates

are kept trainable or not. If they are kept trainable then the model corresponds for interations between the linear and non-linear sections.

In our cases we have assumed so such interactions are present. Also we are using the same deviance loss function in the CANN models so we can also say that the network is such designed that the the initial loss of the network is same as that of the basic Bell regression. This can be verified by setting epoch to 1 and batch size to 1, thus each sample will be treated as a batch and will produce the same loss as that of Bell regression.

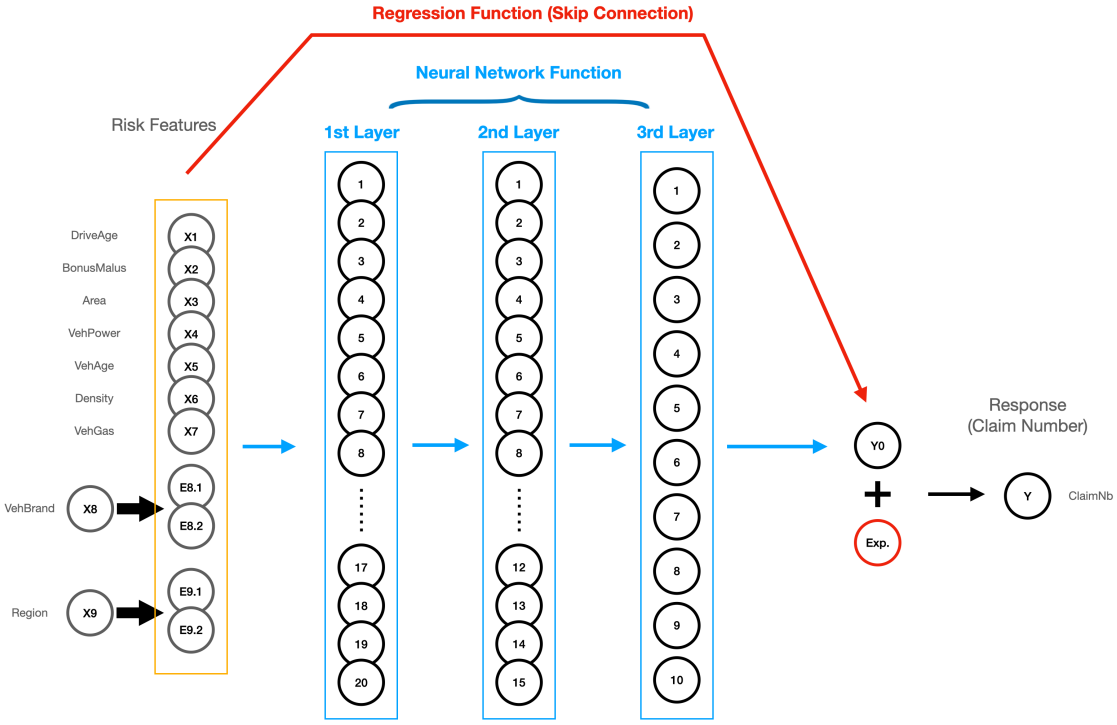


Figure 4.2: CANN architechture with Skip connection

Chapter 5

Data Analysis

Data Description

We used claim frequency data from a French Motor Third-Party Liability (freMTPL) dataset in the R package CASdatasets.

```
> str(data)
'data.frame':   678013 obs. of  12 variables:
 $ IDpol     : num  1 3 5 10 11 13 15 17 18 21 ...
 $ ClaimNb   : num [1:678013(1d)] 1 1 1 1 1 1 1 1 1 1 ...
  ..- attr(*, "dimnames")=List of 1
  .. ..$ : chr [1:678013] "139" "414" "463" "975" ...
 $ Exposure  : num  0.1 0.77 0.75 0.09 0.84 0.52 0.45 0.27 0.71 0.15 ...
 $ Area      : Factor w/ 6 levels "A","B","C","D",..: 4 4 2 2 2 5 5 3 3 2 ...
 $ VehPower  : int  5 5 6 7 7 6 6 7 7 7 ...
 $ VehAge    : int  0 0 2 0 0 2 2 0 0 0 ...
 $ DrivAge   : int  55 55 52 46 46 38 38 33 33 41 ...
 $ BonusMalus: int  50 50 50 50 50 50 50 68 68 50 ...
 $ VehBrand  : Factor w/ 11 levels "B1","B10","B11",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ VehGas    : chr  "Regular" "Regular" "Diesel" "Diesel" ...
 $ Density   : int  1217 1217 54 76 76 3003 3003 137 137 60 ...
 $ Region    : Factor w/ 22 levels "R11","R21","R22",..: 18 18 3 15 15 8 8 20 20 12 ...
```

Figure 5.1: Strucutre of the dataset

We briefly describe this data provided here. We have 678'013 individual car insurance policies and for each policy we have 12 variables:

- IDpol: policy number (unique identifier);
- ClaimNb: number of claims on the given policy;
- Exposure: total exposure in yearly units;

- Area: area code (categorical, ordinal);
- VehPower: power of the car (categorical, ordinal);
- VehAge: age of the car in years;
- DrivAge: age of the (most common) driver in years;
- BonusMalus: bonus-malus level between 50 and 230 (with reference level 100);
- VehBrand: car brand (categorical, nominal);
- VehGas: diesel or regular fuel car (binary);
- Density: density of inhabitants per km2 in the city of the living place of the driver;
- Region: regions in France (prior to 2016), these are illustrated in Figure 1 (categorical).

Table 5.1: Split of the porfolio w.r.t. the number of claims

0	1	2	3	4	5	6	8	9	11	16
643953	32178	1784	82	7	2	1	1	1	3	1

which shows that nearly 95 percent of the data have 0 claim count , hence the data is overdispersed. A detaile descriptive analysis of this dataset is provided in the tutorial of Noll. et al.[3]. The analysis in that reference also includes a minor data cleaning part on the original data which is used here for further analysis of GLM and neural Networks.

Chapter 6

Bias Regularization Method

A significant criticism faced by neural network models is the failure of the balance property to hold on a population level. Although these models provide accurate outcomes when dealing with individual-level data, they fall short in terms of unbiasedness, or the balance property, when applied at the portfolio level. In actuarial applications, this discrepancy is a notable concern, as it has the potential to result in significant mispricing on a population scale.

The underlying issue can be attributed to the limited number of steps in gradient descent algorithms, which may prevent parameter estimates from converging to critical points of the Bell deviance loss function. For models operating under the Bell distributional assumption, the utilization of a log link function (which deviates from the canonical link function for the Bell distribution) can also contribute to bias in the results.

These two reasons are discussed below:

- Portfolio level: The Bell regression model provides unbiased estimator only if the canonical link is chosen. However, the choice of link function, the logarithm, is not the canonical link of the Bell distribution. (The usage of canonical link will cause issues in the MLE process, especially for complex feature spaces.)

- The fitting process of neural network model and combined model utilizes an early-stopping

algorithm to prevent from over-fitting issues on the individual policy level at the cost of bias on the portfolio level.

To tackle this problem, we apply a simple bias regularization method from Wuthrich and Merz's book. We adjust the intercept in the last layer of the neural network, or the regression coefficient, to shift the biased results.This approach involves adjusting the intercept β^{d+1} in the linear function originating from the final layer of the neural network. Since the logarithm is the link function, the adjustment of the intercept in the linear function is simply to multiply the results by a fixed coefficient c, given by

$$c = \frac{\bar{\mu}}{\hat{\mu}}$$

where $\bar{\mu}$ is the mean of the observed claim numbers Y_i and $\hat{\mu}$ is the mean of the predicted values $\hat{y_i}$.

This helps since for the first epoch :

$$\frac{1}{n}\sum c\hat{Y} = c\frac{1}{n}\sum \hat{Y} = \frac{\bar{\mu}}{\hat{\mu}}\hat{\mu} = \bar{\mu}$$

Chapter 7

Results

Results for the freMTPL dataset

Table 7.1: Training loss, testing loss and portfolio average for the Bell Case(in order of 10^2)

model	Training Loss	Testing Loss	Portfolio Average
Data			0.053
Bell GLM	28.82	28.36	0.056
Network Emb (d =1)	27.6	27.225	0.007
Network Emb (d =1) w/ reg.	27.673	27.251	0.006
Network Emb (d =2)	27.884	27.442	0.006
Network Emb (d =2) w/ reg.	27.74	27.322	0.06
CANN (d =1)	27.92	27.562	0.055
CANN (d =1) w\ reg.	27.693	27.361	0.055
CANN (d =2)	27.636	27.316	0.055
CANN (d =2) w\ reg.	27.388	27.105	0.054

Table 7.2: Training loss, testing loss for Poisson case (in order of 10^2)

model	Training Loss	Testing Loss
Poisson GLM	31.26	32.15
Network Emb (d =1)	30.24	31.51
Network Emb (d =2)	30.17	31.45
CANN (d =1)	30.40	31.50
CANN (d =2)	30.48	31.57

Figure 7.1: Plot of the Training and Validation Loss

We observe that the best results are obtained for the **Bell CANN model with bias regularization and 2-dimensional embeddings**.

For comparison we have used the Poisson model with various combinations of networks and embeddings. We can see that the the Poisson and Bell models differ significantly in terms of loss values. This mostly attributes to the over-dispersed property of the bell distribution , which makes it more suitable for this dataset. CANN models allows us to identify missing structure in GLMs (more) explicitly.An additional GLM step allows us to satisfy the balance property. CANN model combined with bias regularization methods give better fit to the data (both training and testing) and also handles the portfolio average quiet well. CANN also allows us to learn across different portfolios.

Chapter 8

Appendix

A1. R codes for GLM

```
rm(list = ls())

library(LambertW)

library(dplyr)

library(keras)

library(nloptr)

data = read.csv('/Users/debjyoti_mukherjee/Downloads/freMTPL2freq.csv',

header = TRUE, stringsAsFactors = TRUE)

################################################################################

str(data)

summary(data)

## Min exposure cooresponds to 0.00273 yrs or 1 day

data$ClaimNb = pmin(data$ClaimNb,4)

data$Exposure = pmin(data$Exposure,1)


# Feature pre-processing for GLM as numeric  :

data3 = data
```

```
data3$AreaGLM = as.numeric(data3$Area)

data3$VehPowerGLM = as.numeric(pmin(data3$VehPower,9))

VehAgeGLM = cbind(c(0:110), c(1, rep(2,10), rep(3,100)))

data3$VehAgeGLM = as.numeric(VehAgeGLM[data3$VehAge+1,2])

# data3[,"VehAgeGLM"] =relevel(data3[,"VehAgeGLM"], ref="2")

DrivAgeGLM = cbind(c(18:100), c(rep(1,21-18), rep(2,26-21),

rep(3,31-26), rep(4,41-31), rep(5,51-41), rep(6,71-51), rep(7,101-71)))

data3$DrivAgeGLM = as.numeric(DrivAgeGLM[data3$DrivAge-17,2])

# data3[,"DrivAgeGLM"] = relevel(data3[,"DrivAgeGLM"], ref="5")

data3$BonusMalusGLM = as.numeric(pmin(data3$BonusMalus, 150))

data3$DensityGLM = as.numeric(log(data3$Density))

data3$RegionGLM = as.numeric(data3$Region)

data3$VehGasGLM = as.numeric(data3$VehGas)

data3$VehBrandGLM = as.numeric(data3$VehBrand)

head(data3)

################################################################################

## Splitting into training and testing set


set.seed(100)

ll = sample(c(1:nrow(data3)), round(0.9*nrow(data3)), replace = FALSE)

learn <- data3[ll,]

test <- data3[-ll,]

X_train <- learn[c('VehPowerGLM', 'VehAgeGLM', 'DrivAgeGLM',

'BonusMalusGLM','VehBrandGLM', 'VehGasGLM','DensityGLM', 'RegionGLM','AreaGLM')]

y_train = learn$ClaimNb

Exposure_train = learn$Exposure
```

```
n_train = nrow(X_train)

p = ncol(X_train)


X_test <- test[c('VehPowerGLM', 'VehAgeGLM', 'DrivAgeGLM',

'BonusMalusGLM','VehBrandGLM', 'VehGasGLM','DensityGLM', 'RegionGLM','AreaGLM')]

y_test = test$ClaimNb

Exposure_test = test$Exposure

n_test= nrow(X_test)

################################################################################

## choose training or testing (0 train, 1 test )

a = 0

if(a ==0) {X = X_train;y = y_train;Exposure = Exposure_train;n = n_train}

else { X = X_test;y = y_test;Exposure = Exposure_test;n = n_test}

################################################################################

# testing with a small sample of size 300

# sample_begin = cbind(Exposure,y,X_train)

# sample= slice_sample(.data = sample_begin, n= 300)

# n = 300

# y = sample$y

# X = subset(sample, select = -c(Exposure ,y))

# Exposure = sample$Exposure

################################################################################

## Stirling Numbers of the 2-nd kind

Stirling2 <- function(n,m)

{

  if (0 > m || m > n) stop("'m' must be in 0..n !")
```

```
  k <- 0:m
  sig <- rep(c(1,-1)*(-1)^m, length= m+1)# 1 for m=0; -1 1 (m=1)
  ga <- gamma(k+1)
  round(sum( sig * k^n /(ga * rev(ga))))
}
## Bell numbers
fBell <- function(x){
  if (x == 0) return(1)
  vBell <- numeric()
  for(j in 1:x){
    vBell[j] = Stirling2(x,j)
  }
  return(sum(vBell))
}


## Bell pmf
Bell_pmf = function( mu ,y){
  (exp(1-exp(W(mu)))*((W(mu))^y)*fBell(y))/ factorial(y)
}


ll = function(pars){
  mu = exp(log(Exposure) + as.matrix(X)%*%pars)
  ll = c()
  for (i in 1:n){
    ll[i] = log(Bell_pmf(mu[i], y[i]))
  }
```

```
  return(-sum(ll))
}

{t1 <- proc.time()
  opt = optim(par = c(rep(0.001,5),-1, rep(-0.001,3)),fn = ll, method = "BFGS")
  (proc.time()-t1)[3]}
################################################################################
# Poission glm on same data

## poi pmf
POI_pmf = function(lambda , y){
  exp(-lambda)*(lambda^y)/ factorial(y)
}

ll = function(pars){
  lambda = exp(log(Exposure) + as.matrix(X)%*%pars)
  ll = c()
  for (i in 1:n){
    ll[i] = log(POI_pmf(lambda[i], y[i]))
  }
  return(-sum(ll))
}
opt = optim(par =c(rep(0.001,5),-1, rep(-0.001,3)),fn = ll, method = "BFGS")
################################################################################
# > opt Bell case. (results should improve on dummy-coding)
# $par
```

```
# [1] -0.058907266 -0.825711077 -0.069375417  0.012374439 -0.023627158
-0.036957226 -0.186467433 -0.007628798
# [9]  0.257658974
#
# $value
# [1] 129394.8
#
# $counts
# function gradient
# 112       14
# pars = c(-0.058907266, -0.825711077, -0.069375417  ,0.012374439,
-0.023627158 ,-0.036957226, -0.186467433, -0.007628798,0.257658974)
################################################################################
# > opt Poisson case  (results should improve on dummy-coding)
# $par
# [1] -0.059662959 -0.820658904 -0.069075686  0.012225486 -0.023556506
-0.042905475 -0.187115136 -0.007633975
# [9]  0.257655488
#
# $value
# [1] 129830.4
#
# $counts
# function gradient
# 116       14
################################################################################
```

```
## Bell deviance loss:
Bell.Deviance = function(obs, pred)
  {
  loss = rep()
  for (i in 1:length(obs)){
    if (obs[i]==0)
    {
      loss[i] = 2*(-1+exp(W(pred[i]))) }
      else {
        loss[i] = 2*((exp(W(pred[i])))-(exp(W(obs[i])))+(log((W(obs[i])/W(pred[i]))^(obs[i])))) }
  }
  return(mean(loss))
}
       # 2*(sum(exp(W(pred)))-sum(exp(W(obs)))+sum(log((W(obs)/W(pred))^(obs))))/length(pred)


## Poisson deviance
Poisson.Deviance = function(obs,pred){2*(sum(pred)-
sum(obs)+sum(log((obs/pred)^(obs))))/length(pred)}
################################################################################
# predictions and deviances
#bell
pars = c(-0.058907266, -0.825711077, -0.069375417  ,0.012374439,
-0.023627158 ,-0.036957226, -0.186467433, -0.007628798,0.257658974)
pred_train = exp(log(Exposure_train) + as.matrix(X_train)%*%pars)
pred_test = exp(log(Exposure_test) + as.matrix(X_test)%*%pars)
Bell.Deviance(obs = y_train, pred_train) # in-sample
```

```
Bell.Deviance(obs = y_test, pred_test)  # out-sample


#poisson
pars = c(-0.059662959, -0.820658904 ,-0.069075686 , 0.012225486
,-0.023556506, -0.042905475, -0.187115136 ,-0.007633975, 0.257655488)
pred_train = exp(log(Exposure_train) + as.matrix(X_train)%*%pars)
pred_test = exp(log(Exposure_test) + as.matrix(X_test)%*%pars)
Poisson.Deviance(obs = y_train, pred_train)  # in-sample
Poisson.Deviance(obs = y_test, pred_test)    # out-sample


################################################################################
# Bell prediction column addition -> predict for learn and test and combine
the two vectors and add as a col to data
head(data3)


X_data = data3[c('VehPowerGLM', 'VehAgeGLM', 'DrivAgeGLM',
'BonusMalusGLM','VehBrandGLM', 'VehGasGLM','DensityGLM', 'RegionGLM','AreaGLM')]
y_data = data3$ClaimNb
Exposure_data = data3$Exposure
n_data = nrow(X_data)
pars = c(-0.058907266, -0.825711077, -0.069375417  ,0.012374439,
-0.023627158 ,-0.036957226, -0.186467433, -0.007628798,0.257658974)
pred_data= exp(log(Exposure_data) + as.matrix(X_data)%*%pars)
Bell.Deviance(obs = y_data, pred_data) # whole data
data$bellGLM = pred_data[,1]
################################################################################
```

```
## pre-processing and splitting section for ANN, and CANN

# min-max-scaler:

PreProcess.Continuous = function(var1, data2){

  names(data2)[names(data2) == var1]  = "V1"

  data2$X = as.numeric(data2$V1)

  data2$X = 2*(data2$X-min(data2$X))/(max(data2$X)-min(data2$X))-1

  names(data2)[names(data2) == "V1"]  = var1

  names(data2)[names(data2) == "X"]  <- paste(var1,"X", sep="")

  data2

}

# pre-procecessing function:

Features.PreProcess = function(data2){

  data2 = PreProcess.Continuous("Area", data2)

  data2 = PreProcess.Continuous("VehPower", data2)

  data2$VehAge = pmin(data2$VehAge,20)

  data2 = PreProcess.Continuous("VehAge", data2)

  data2$DrivAge = pmin(data2$DrivAge,90)

  data2 = PreProcess.Continuous("DrivAge", data2)

  data2$BonusMalus = pmin(data2$BonusMalus,150)

  data2 = PreProcess.Continuous("BonusMalus", data2)

  data2$VehBrandX = as.integer(data2$VehBrand)-1

  data2$VehGasX <- as.integer(data2$VehGas)-1.5

  data2$Density <- round(log(data2$Density),2)

  data2 <- PreProcess.Continuous("Density", data2)

  data2$RegionX <- as.integer(data2$Region)-1  # char R11,,R94 to number 0,,21

  data2
```

```
}

data2 = Features.PreProcess(data)  # keep original variables and bellGLM (CANN)

set.seed(100)

ll = sample(c(1:nrow(data2)), round(0.9*nrow(data2)), replace = FALSE)

learn = data2[ll,]

test = data2[-ll,]

write.csv(learn, file = "/Users/debjyoti_mukherjee/Downloads/bell_learn.csv")

write.csv(test, file = "/Users/debjyoti_mukherjee/Downloads/bell_test.csv")

################################################################################
```

A2. Python codes for Neural Network

In [ ]:
```
## imports

import numpy as np
import pandas as pd
from scipy.special import lambertw as W1
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import keras
from keras.layers.core import Dense, Dropout
from keras.layers import Input , Embedding
from keras.models import Model
from keras.models import Sequential
# import tensorflow as tf
# from keras.optimizers.legacy.adam import adam
import matplotlib.pyplot as plt
import seaborn as sns
from keras import callbacks
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from keras.initializers import initializer
from keras.optimizers import Nadam
import keras.backend as K
import tensorflow as tf
from tensorflow.keras.losses import Loss
import tensorflow_probability as tfp
from tensorflow.keras.regularizers import l2
# from tensorflow.keras import backend as K
```

In [ ]:
```
## importing pre-processed data

learn = pd.read_csv('/Users/debjyoti_mukherjee/Downloads/bell_learn.csv'
                    , index_col= 0)
test = pd.read_csv('/Users/debjyoti_mukherjee/Downloads/bell_test.csv'
                   , index_col = 0)

# learn = learn.sample(500)
```

In [ ]:
```
learn.shape, test.shape
```

Out[ ]:
```
((610212, 22), (67801, 22))
```

In [ ]:
```
data = pd.concat([learn, test], axis=0)
```

In [ ]:
```
np.mean(data['bellGLM']), np.mean(data['ClaimNb'])
```

Out[ ]:
```
(0.05572372657312568, 0.05317892134811574)
```

In [ ]:
```
## final Bell Deviance:

def Bell_Deviance(y_true, y_pred):
    eps =0    #1e-8

    y_true = tf.reshape(y_true, (-1,1))
    y_pred = tf.reshape(y_pred, (-1,1))

    obs_zero = tf.boolean_mask(y_true, tf.equal(y_true, 0))
    pred_zero = tf.boolean_mask(y_pred, tf.equal(y_true, 0))

    obs_nonzero = tf.boolean_mask(y_true, tf.not_equal(y_true, 0))
    pred_nonzero = tf.boolean_mask(y_pred, tf.not_equal(y_true, 0))

    loss_zero = 2 * (-1 + tf.exp(tfp.math.lambertw(tf.cast(pred_zero,
                                            dtype = tf.float64))))

    loss_nonzero = 2 * (
    tf.exp(tfp.math.lambertw(tf.cast(pred_nonzero, dtype=tf.float64)))
    - tf.exp(tfp.math.lambertw(tf.cast(obs_nonzero, dtype=tf.float64))) +
    tf.math.log(
        (tfp.math.lambertw(tf.cast(obs_nonzero, dtype=tf.float64))/
        (tfp.math.lambertw(tf.cast(pred_nonzero, dtype=tf.float64))+eps))
        tf.cast(obs_nonzero, dtype=tf.float64)
    )
    )
    loss = tf.concat([loss_zero, loss_nonzero], axis=0)

    mean_loss = tf.reduce_mean(loss)

    return mean_loss
```

In [ ]:
```
Bell_Deviance(learn['ClaimNb'],learn['bellGLM'])
```

```
2023-09-10 21:47:57.170504: I tensorflow/core/common_runtime/pluggable_d
evice/pluggable_device_factory.cc:305] Could not identify NUMA node of p
latform GPU ID 0, defaulting to 0. Your kernel may not have been built w
ith NUMA support.
2023-09-10 21:47:57.171870: I tensorflow/core/common_runtime/pluggable_d
evice/pluggable_device_factory.cc:271] Created TensorFlow device (/job:l
ocalhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical Plu
ggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
Metal device set to: Apple M1

systemMemory: 8.00 GB
maxCacheSize: 2.67 GB
```

Out[ ]:
```
<tf.Tensor: shape=(), dtype=float64, numpy=0.28819543059850866>
```

In [ ]:
```
Bell_Deviance(test['ClaimNb'],test['bellGLM'])
```

Out[ ]:
```
<tf.Tensor: shape=(), dtype=float64, numpy=0.28359761726147104>
```

In [ ]:
```
q0 = 7
Xdata = data[['AreaX','VehPowerX','VehAgeX','DrivAgeX','BonusMalusX'
              ,'VehGasX','DensityX']]
Xlearn  = learn[['AreaX','VehPowerX','VehAgeX','DrivAgeX','BonusMalusX'
                 ,'VehGasX','DensityX']]
Xtest  = test[['AreaX','VehPowerX','VehAgeX','DrivAgeX','BonusMalusX'
               ,'VehGasX','DensityX']]
```

In [ ]:
```
## Choosing the right volume for EmbNN and CANN model:
Vlearn = np.log(learn['Exposure'])
Vtest = np.log(test['Exposure'])
lambda_hom = sum(learn['ClaimNb']/sum(learn['Exposure']))

CANN = 1        ## 0 =  EmbNN, 1 = CANN

if (CANN == 1):
    Vlearn = np.log(learn['bellGLM'])
    Vtest = np.log(test['bellGLM'])
    lambda_hom = sum(learn['ClaimNb']/sum(learn['bellGLM']))

lambda_hom
```

Out[ ]:
```
0.9578401201961692
```

In [ ]:
```
## Hyperparameter setup:

Brlabel = len(learn['VehBrandX'].unique())
Relabel = len(learn['RegionX'].unique())
q1 = 100
q2 = 75
q3 = 50
d = 2   ## Vary 1 or 2
p = 0
```

In [ ]:
```
## Embedding layer for categorical variables
# the network architechture:

Design =  keras.layers.Input(shape=(q0,), name= 'design')
VehBrand = keras.layers.Input(shape= (1,))
Region = keras.layers.Input(shape= (1,))
LogVol = keras.layers.Input(shape= (1,), name='LogVol')
#
BrandEmb = keras.layers.Embedding(input_dim= Brlabel ,
        output_dim= d, input_length = 1, name = 'BrandEmb')(VehBrand)
Brand_Flat = keras.layers.Flatten(name = 'Brand_Flat')(BrandEmb)
RegionEmb = keras.layers.Embedding(input_dim= Relabel , output_dim= d,
                        input_length = 1, name = 'RegionEmb')(Region)
Region_flat = keras.layers.Flatten(name = 'Region_Flat')(RegionEmb)
#
concate = keras.layers.Concatenate()([Design, Brand_Flat, Region_flat])
hidden1 = keras.layers.Dense(q1, activation='tanh')(concate)
hidden2 = keras.layers.Dense(q2, activation = 'tanh')(hidden1)
hidden3 = keras.layers.Dense(q3, activation = 'tanh')(hidden2)
Network = keras.layers.Dense(1, activation ='linear', use_bias=True,
        weights= [np.zeros((q3,1)),  np.array((lambda_hom,))])(hidden3)

Add= keras.layers.Add()([Network,LogVol])
Response = keras.layers.Dense(1, activation = keras.activations
                .exponential, dtype= "float64",trainable= False,
            weights= [np.ones((1,1)), np.zeros(1,)])(Add)
model = keras.Model(inputs = [Design, VehBrand, Region, LogVol]
                    , outputs = Response)
model.compile(optimizer=keras.optimizers.Nadam(0.001)
              , loss = Bell_Deviance)

model.summary()
```

```
Model: "model"
__________________________________________________________________________
_____________________________
 Layer (type)                Output Shape              Param #     Connect
ed to
==========================================================================
==========================
 input_1 (InputLayer)        [(None, 1)]               0           []

 input_2 (InputLayer)        [(None, 1)]               0           []

 BrandEmb (Embedding)        (None, 1, 2)              22          ['input
_1[0][0]']

 RegionEmb (Embedding)       (None, 1, 2)              44          ['input
_2[0][0]']

 design (InputLayer)         [(None, 7)]               0           []

 Brand_Flat (Flatten)        (None, 2)                 0           ['Brand
```

```
Emb[0][0]']

 Region_Flat (Flatten)          (None, 2)            0           ['Regio
nEmb[0][0]']

 concatenate (Concatenate)      (None, 11)           0           ['desig
n[0][0]',
                                                                 'Brand
_Flat[0][0]',
                                                                 'Regio
n_Flat[0][0]']

 dense (Dense)                  (None, 100)          1200        ['conca
tenate[0][0]']

 dense_1 (Dense)                (None, 75)           7575        ['dense
[0][0]']

 dense_2 (Dense)                (None, 50)           3800        ['dense
_1[0][0]']

 dense_3 (Dense)                (None, 1)            51          ['dense
_2[0][0]']

 LogVol (InputLayer)            [(None, 1)]          0           []

 add (Add)                      (None, 1)            0           ['dense
_3[0][0]',
                                                                 'LogVo
l[0][0]']

 dense_4 (Dense)                (None, 1)            2           ['add[0
][0]']

==================================================================================
Total params: 12,694
Trainable params: 12,692
Non-trainable params: 2
__________________________________________________________________________________
```

In [ ]:
```
cbs =  callbacks.EarlyStopping(monitor ="val_loss",  mode ="min",
                        patience = 5, restore_best_weights = True)
history = model.fit([Xlearn, learn['VehBrandX'], learn['RegionX'],
          Vlearn], (np.array(learn['ClaimNb'])).reshape(-1),
             epochs=1000,batch_size= 100000, verbose = 2
            , validation_split=0.2,callbacks=cbs)
```

```
Epoch 1/1000
```

```
2023-09-10 21:48:12.998700: W tensorflow/core/platform/profile_utils/cpu
_utils.cc:128] Failed to get CPU frequency: 0 Hz
2023-09-10 21:48:14.803748: I tensorflow/core/grappler/optimizers/custom
_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU i
s enabled.
2023-09-10 21:48:17.469182: I tensorflow/core/grappler/optimizers/custom
_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU i
s enabled.
5/5 - 5s - loss: 0.3502 - val_loss: 0.3473 - 5s/epoch - 960ms/step
Epoch 2/1000
5/5 - 1s - loss: 0.3379 - val_loss: 0.3272 - 1s/epoch - 234ms/step
Epoch 3/1000
5/5 - 1s - loss: 0.3146 - val_loss: 0.3035 - 1s/epoch - 222ms/step
Epoch 4/1000
5/5 - 1s - loss: 0.2945 - val_loss: 0.2916 - 1s/epoch - 218ms/step
Epoch 5/1000
5/5 - 1s - loss: 0.2864 - val_loss: 0.2891 - 1s/epoch - 225ms/step
Epoch 6/1000
5/5 - 1s - loss: 0.2850 - val_loss: 0.2891 - 1s/epoch - 213ms/step
Epoch 7/1000
5/5 - 1s - loss: 0.2849 - val_loss: 0.2891 - 1s/epoch - 215ms/step
Epoch 8/1000
5/5 - 1s - loss: 0.2848 - val_loss: 0.2889 - 1s/epoch - 218ms/step
Epoch 9/1000
5/5 - 1s - loss: 0.2846 - val_loss: 0.2887 - 1s/epoch - 229ms/step
Epoch 10/1000
5/5 - 1s - loss: 0.2844 - val_loss: 0.2886 - 1s/epoch - 213ms/step
Epoch 11/1000
5/5 - 1s - loss: 0.2843 - val_loss: 0.2885 - 1s/epoch - 225ms/step
Epoch 12/1000
5/5 - 1s - loss: 0.2842 - val_loss: 0.2884 - 1s/epoch - 217ms/step
Epoch 13/1000
5/5 - 1s - loss: 0.2842 - val_loss: 0.2883 - 1s/epoch - 223ms/step
Epoch 14/1000
5/5 - 1s - loss: 0.2841 - val_loss: 0.2882 - 1s/epoch - 227ms/step
Epoch 15/1000
5/5 - 1s - loss: 0.2840 - val_loss: 0.2882 - 1s/epoch - 217ms/step
Epoch 16/1000
5/5 - 1s - loss: 0.2839 - val_loss: 0.2881 - 1s/epoch - 224ms/step
Epoch 17/1000
5/5 - 1s - loss: 0.2839 - val_loss: 0.2880 - 1s/epoch - 226ms/step
Epoch 18/1000
5/5 - 1s - loss: 0.2838 - val_loss: 0.2879 - 1s/epoch - 231ms/step
Epoch 19/1000
5/5 - 1s - loss: 0.2837 - val_loss: 0.2878 - 1s/epoch - 237ms/step
Epoch 20/1000
5/5 - 1s - loss: 0.2836 - val_loss: 0.2877 - 1s/epoch - 220ms/step
Epoch 21/1000
5/5 - 1s - loss: 0.2835 - val_loss: 0.2875 - 1s/epoch - 299ms/step
Epoch 22/1000
5/5 - 1s - loss: 0.2834 - val_loss: 0.2874 - 1s/epoch - 270ms/step
Epoch 23/1000
5/5 - 1s - loss: 0.2833 - val_loss: 0.2873 - 1s/epoch - 234ms/step
```

```
Epoch 183/1000
5/5 - 1s - loss: 0.2740 - val_loss: 0.2788 - 1s/epoch - 225ms/step
Epoch 184/1000
5/5 - 1s - loss: 0.2741 - val_loss: 0.2782 - 1s/epoch - 219ms/step
Epoch 185/1000
5/5 - 1s - loss: 0.2739 - val_loss: 0.2786 - 1s/epoch - 220ms/step
Epoch 186/1000
5/5 - 1s - loss: 0.2739 - val_loss: 0.2783 - 1s/epoch - 218ms/step
Epoch 187/1000
5/5 - 1s - loss: 0.2739 - val_loss: 0.2782 - 1s/epoch - 222ms/step
Epoch 188/1000
5/5 - 1s - loss: 0.2738 - val_loss: 0.2783 - 1s/epoch - 223ms/step
Epoch 189/1000
5/5 - 1s - loss: 0.2739 - val_loss: 0.2781 - 1s/epoch - 213ms/step
Epoch 190/1000
5/5 - 1s - loss: 0.2737 - val_loss: 0.2782 - 1s/epoch - 222ms/step
Epoch 191/1000
5/5 - 1s - loss: 0.2742 - val_loss: 0.2785 - 1s/epoch - 221ms/step
Epoch 192/1000
5/5 - 1s - loss: 0.2739 - val_loss: 0.2780 - 1s/epoch - 221ms/step
Epoch 193/1000
5/5 - 1s - loss: 0.2738 - val_loss: 0.2778 - 1s/epoch - 220ms/step
Epoch 194/1000
5/5 - 1s - loss: 0.2737 - val_loss: 0.2780 - 1s/epoch - 217ms/step
Epoch 195/1000
5/5 - 1s - loss: 0.2738 - val_loss: 0.2783 - 1s/epoch - 217ms/step
Epoch 196/1000
5/5 - 1s - loss: 0.2737 - val_loss: 0.2779 - 1s/epoch - 221ms/step
Epoch 197/1000
5/5 - 1s - loss: 0.2737 - val_loss: 0.2785 - 1s/epoch - 221ms/step
Epoch 198/1000
5/5 - 1s - loss: 0.2735 - val_loss: 0.2778 - 1s/epoch - 222ms/step
Epoch 199/1000
5/5 - 1s - loss: 0.2735 - val_loss: 0.2779 - 1s/epoch - 219ms/step
Epoch 200/1000
5/5 - 1s - loss: 0.2735 - val_loss: 0.2778 - 1s/epoch - 228ms/step
Epoch 201/1000
5/5 - 1s - loss: 0.2736 - val_loss: 0.2777 - 1s/epoch - 217ms/step
Epoch 202/1000
5/5 - 1s - loss: 0.2734 - val_loss: 0.2777 - 1s/epoch - 228ms/step
Epoch 203/1000
5/5 - 1s - loss: 0.2734 - val_loss: 0.2779 - 1s/epoch - 215ms/step
Epoch 204/1000
5/5 - 1s - loss: 0.2734 - val_loss: 0.2776 - 1s/epoch - 259ms/step
Epoch 205/1000
5/5 - 1s - loss: 0.2735 - val_loss: 0.2778 - 1s/epoch - 290ms/step
Epoch 206/1000
5/5 - 1s - loss: 0.2734 - val_loss: 0.2778 - 1s/epoch - 276ms/step
Epoch 207/1000
5/5 - 1s - loss: 0.2740 - val_loss: 0.2783 - 1s/epoch - 225ms/step
Epoch 208/1000
5/5 - 1s - loss: 0.2736 - val_loss: 0.2775 - 1s/epoch - 222ms/step
Epoch 209/1000
```

```
5/5 - 1s - loss: 0.2733 - val_loss: 0.2777 - 1s/epoch - 217ms/step
Epoch 210/1000
5/5 - 1s - loss: 0.2733 - val_loss: 0.2780 - 1s/epoch - 216ms/step
Epoch 211/1000
5/5 - 1s - loss: 0.2737 - val_loss: 0.2776 - 1s/epoch - 217ms/step
Epoch 212/1000
5/5 - 1s - loss: 0.2736 - val_loss: 0.2775 - 1s/epoch - 215ms/step
Epoch 213/1000
5/5 - 1s - loss: 0.2731 - val_loss: 0.2773 - 1s/epoch - 227ms/step
Epoch 214/1000
5/5 - 1s - loss: 0.2731 - val_loss: 0.2782 - 1s/epoch - 219ms/step
Epoch 215/1000
5/5 - 1s - loss: 0.2734 - val_loss: 0.2775 - 1s/epoch - 210ms/step
Epoch 216/1000
5/5 - 1s - loss: 0.2732 - val_loss: 0.2773 - 1s/epoch - 217ms/step
Epoch 217/1000
5/5 - 1s - loss: 0.2733 - val_loss: 0.2777 - 1s/epoch - 210ms/step
Epoch 218/1000
5/5 - 1s - loss: 0.2731 - val_loss: 0.2774 - 1s/epoch - 221ms/step
Epoch 219/1000
5/5 - 1s - loss: 0.2730 - val_loss: 0.2773 - 1s/epoch - 218ms/step
Epoch 220/1000
5/5 - 1s - loss: 0.2730 - val_loss: 0.2774 - 1s/epoch - 216ms/step
Epoch 221/1000
5/5 - 1s - loss: 0.2730 - val_loss: 0.2774 - 1s/epoch - 221ms/step
```

In [ ]:
```
model.evaluate([Xlearn, learn['VehBrandX'], learn['RegionX'],
                Vlearn], (np.array(learn['ClaimNb'])).reshape(-1)) ,

model.evaluate([Xtest, test['VehBrandX'], test['RegionX'],
                Vtest], (np.array(test['ClaimNb'])).reshape(-1))
```

```
19070/19070 [==============================] - 363s 19ms/step - loss: 0.
2739
2119/2119 [==============================] - 40s 19ms/step - loss: 0.271
1
```

Out[ ]:
```
(0.2738856077194214, 0.27105283737182617)
```

In [ ]:
```
y_pred_data = (model.predict([Xdata , data['VehBrandX']
            , data['RegionX'], np.log(data['bellGLM'])]))
```

```
   21/21188 [..............................] - ETA: 1:52
2023-09-10 22:03:55.679352: I tensorflow/core/grappler/optimizers/custom
_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU i
s enabled.
21188/21188 [==============================] - 89s 4ms/step
```

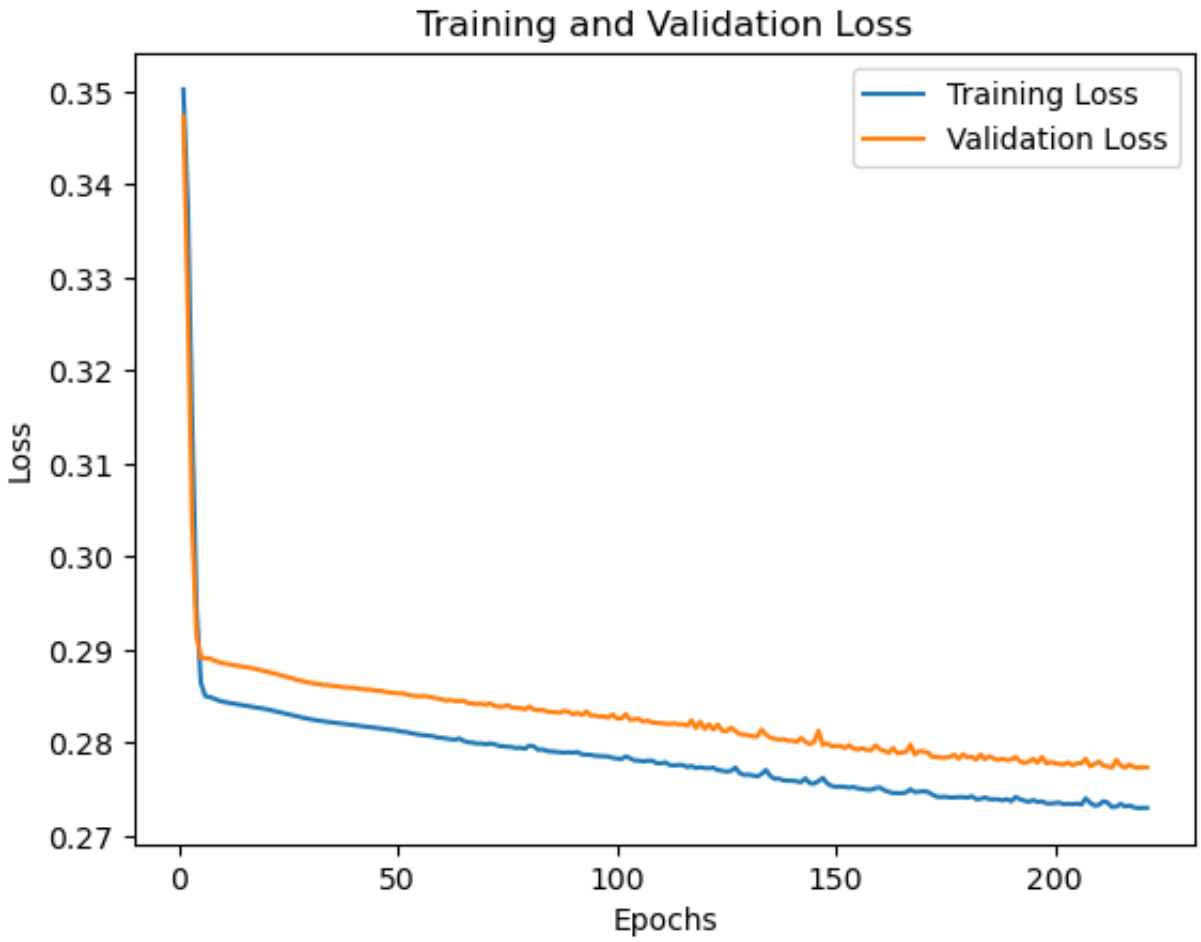
In [ ]:
```
Bell_Deviance(data['ClaimNb'], y_pred_data)
```

Out[ ]:
```
<tf.Tensor: shape=(), dtype=float64, numpy=0.27360195990960207>
```

In [ ]:
```
np.mean(y_pred_data)
```

Out[ ]: 0.055199446975010516

In [ ]:
```
import matplotlib.pyplot as plt

# Assuming you've already trained your model
# and have the 'history' object

# Access training and validation loss from the history object
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Generate a sequence of integers to represent the epoch numbers
epochs = range(1, len(train_loss) + 1)

# Plot and label the training and validation loss values
plt.plot(epochs, train_loss, label='Training Loss')
plt.plot(epochs, val_loss, label='Validation Loss')

# Add in a title and axes labels
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')

# Display the plot
plt.legend(loc='best')
plt.show()
```


Training and Validation Loss
Training Loss
Validation Loss
Loss
0.35
0.34
0.33
0.32
0.31
0.30
0.29
0.28
0.27
0
50
100
150
200
Epochs

Bibliography

[1] Ferrario, A., Noll, A., Wuthrich, M. V. (2020). Insights from inside neural networks. *Available at SSRN 3226852*.

[2] Jose, A., Macdonald, A. S., Tzougas, G., Streftaris, G. (2022). A combined neural network approach for the prediction of admission rates related to respiratory diseases. *Risks, 10(11)*, 217.

[3] Noll, A., Salzmann, R., Wuthrich, M. V. (2020). Case study: French motor third-party liability claims. *Available at SSRN 3164764*.

[4] Schelldorfer, J., Wuthrich, M. V. (2019). Nesting classical actuarial models into neural networks. *Available at SSRN 3320525*.

[5] Tzougas, G., & Kutzkov, K. (2023). Enhancing Logistic Regression Using Neural Networks for Classification in Actuarial Learning. *Algorithms*, 16(2), 99.

[6] Wuthrich, M. V., & Merz, M. (2019). Yes, we CANN!. *ASTIN Bulletin: The Journal of the IAA*, 49(1), 1-3.

[7] Wüthrich, M. V., Merz, M. (2023). *Statistical foundations of actuarial learning and its applications* (p. 605). Springer Nature.