



Detection of interacting variables for generalized linear models via neural networks

Yevhen Havrylenko^{1,2}  · Julia Heger³

Received: 16 September 2022 / Revised: 7 April 2023 / Accepted: 28 August 2023 /
Published online: 1 November 2023
© The Author(s) 2023

Abstract

The quality of generalized linear models (GLMs), frequently used by insurance companies, depends on the choice of interacting variables. The search for interactions is time-consuming, especially for data sets with a large number of variables, depends much on expert judgement of actuaries, and often relies on visual performance indicators. Therefore, we present an approach to automating the process of finding interactions that should be added to GLMs to improve their predictive power. Our approach relies on neural networks and a model-specific interaction detection method, which is computationally faster than the traditionally used methods like Friedman's H-Statistic or SHAP values. In numerical studies, we provide the results of our approach on artificially generated data as well as open-source data.

Keywords Neural network · Model interpretability · Generalized linear model · Interaction detection · Insurance claims prediction

1 Introduction

Insurance companies usually apply generalized linear models (GLMs) to predict insurance claim counts due to the interpretability of these models. GLMs are constantly improved by pricing actuaries via sophisticated choice of variable interactions. This

✉ Yevhen Havrylenko
yh@math.ku.dk

Julia Heger
julia.heger@uni-a.de

¹ TUM School of Computation, Information and Technology, Technical University of Munich, Parkring 11, 85748 Garching, Germany

² Present Address: Department of Mathematical Sciences, University of Copenhagen, Universitetsparken 5, 2100 Copenhagen, Denmark

³ Chair of Analytics and Optimization, University of Augsburg, Universitätsstraße 2, 86159 Augsburg, Germany

process is time-consuming, depends much on expert judgement, and relies on visual performance indicators. These aspects motivate the usage of machine learning (ML) techniques for improving the performance of GLMs by finding the next-best interaction to be added to the GLM. Such automation of the manual and mainly visual process of fine-tuning GLMs could save much time for actuaries, especially in the case of big data sets with dozens of variables, e.g., in motor third-party liability (MTPL) insurance.

In this paper, we propose a methodology for the detection of the next-best interaction that is missing in a benchmark GLM. We aim at improving an arbitrary but fixed existing benchmark GLM instead of creating a new GLM from scratch. Building a new GLM may necessitate drastic changes in the tariff of the MTPL insurance. Large changes in tariffs are not desired by insurance companies for their existing business lines. Instead, GLMs need to be improved gradually.

The approach we suggest has three steps. First, a combined actuarial neural network (CANN) is trained. This model is introduced in the actuarial context in [22] and can be seen as a combination of a benchmark-GLM predictor and a neural-network predictor into a single neural network (NN) using a skip connection for the GLM component. Second, the strength of each pairwise interaction learned by the CANN model is quantified and the interactions are ranked by their strength using a neural interaction detection (NID) algorithm. This algorithm is introduced in [19] for fully connected feed-forward neural networks and adjusted by us to CANN models. Third, the top-ranked interactions are analyzed with the help of mini-GLMs and the next-best interaction to be included in the benchmark GLM is identified.

We show the performance of our approach on an artificially created data set, where the true interactions are known from the data-generation mechanism, and on an open-source French MTPL data set, which has been analyzed in many academic sources, e.g., [16], [12], [3], [8], [23], etc. Finally, we comment on the advantages of our methodology for big MTPL data sets with millions of observations and dozens of features, which are common for larger insurance companies.

Literature overview: GLMs are introduced by [11] as a generalization of a linear regression model with a normally distributed response variable. Since then, GLMs became an important and popular tool in insurance pricing, especially MTPL insurance. For more information, see [2], [14], [23].

The process of finding the best GLM becomes very challenging with the increasing number of variables. As it is not possible to fit and compare all possible models for a large number of variables, three classical approaches have been developed: forward selection, backward selection, and mixed (bi-directional) selection. Forward selection is a greedy approach and might include variables early that later become redundant. Mixed selection can fix this challenge. As an alternative to stepwise variable selection, one can use penalized likelihood estimation, e.g., Least Absolute Selection and Shrinkage Operator (LASSO) introduced by [18], to find the best subset of variables for a GLM.

However, it is even more computationally challenging to use the above methods to search for the best GLM with interacting variables, as the number of possible interactions is too large, whereas re-fitting even one single GLM on a real-world big data set is time-consuming. Therefore, researchers explored the usage of neural-

network based models for predicting claim frequencies and learning from them about the interacting variables, which can be added to a GLM to improve its performance.

First, a neural-network based model is trained. Second, interaction-detection methods are applied. Two types of interaction-detection methods are distinguished—model-agnostic and model-specific ones. Model-agnostic methods do not use a specific structure of the ML model. This class of methods includes Friedman’s H-statistics [4], Greenwell statistics [6], feature interaction in terms of prediction performance [13], and SHAP interaction values [9]. The main drawback of the above-mentioned model-agnostic methods is their high computational cost for big actuarial data sets. Model-specific interaction detection methods rely on the peculiarities of the ML model under consideration. For example, [21] proposes a procedure for determining missing interactions in the benchmark GLM via CANNs. For each interaction of interest, a CANN that uses the interaction of interest and the prediction of the benchmark GLM is trained. If the deviance loss function of the CANN model decreases significantly in comparison to the deviance loss of the benchmark GLM, then this interaction is considered missing. [15] proposes a LocalGLMnet, which retains the additive decomposition of the response variable as in the case of a GLM, but lets the regression coefficients become feature-dependent. Once a LocalGLMnet is trained, one can detect whether there is an interaction between two features by exploring smoothed plots of gradients of the regression coefficients, also called regression attention due to their dependence on the features. How to optimally determine interactions for categorical variables with many levels is still an open question for LocalGLMnets.

In our approach, we also use CANN, as in [16]. We use embedding layers for categorical features with many categories, as it is shown to improve predictive performance on actuarial data sets, see [16], [21]. To extract interactions, we use a model-specific method that is a modification of a method called Neural Interaction Detection (NID). This method is developed in [19]. It computes the strengths of all interactions among input neurons of a neural network very fast, since it uses only trained weights of the neural network, and is extendable to embedding layers and the architecture of a CANN model.

Structure: In Sect. 2, we explain the basics of GLMs. In Sect. 3, we describe in detail our proposed algorithm for detecting the next-best interaction for a benchmark GLM. Each subsection within this section is devoted to a specific part of our algorithm. Section 4 contains case studies, where we apply the proposed algorithm to two data sets—an artificially created one and an open-source one—and briefly comment on its usage for big confidential data sets. We offer our conclusions in Sect. 5. Appendix A contains R-code for the neural interaction detection algorithm.

2 Generalized linear models for modeling insurance claim frequencies

In this section, we briefly describe the basics of a GLM for modeling claim counts. We start with a definition of a GLM without interacting variables and then explain how an interaction is added to a GLM. This section is mainly based on [21]. Since

their introduction in 1972, GLMs have enjoyed great popularity for modeling and forecasting claim frequencies within the insurance sector.

Let a data set be denoted by $\{(N_i, \mathbf{x}_i, v_i)\}_{i=1}^n$, where $n \in \mathbb{N}$ is the number of observations, $v_i \in [0, 1]$ corresponds to the exposure time in years of the i -th observation (the time length in which events occur), $N_i \in \mathbb{N} \cup \{0\}$ refers to the number of claims observed for the observation i within exposure time v_i , $\mathbf{x}_i = (1, x_{i,1}, x_{i,2}, \dots, x_{i,p})^\top \in \mathcal{X} \subset \{1\} \times \mathbb{R}^p$ represents the vector¹ of variables (features, covariates) for the observation i excluding the exposure time, and $p \in \mathbb{N}$ is the number of features. In the Poisson GLM context, $N_i, i = 1, \dots, n$, are assumed to be independent random variables that follow the Poisson distribution. The mean of the distribution of N_i is assumed to depend on the so-called systematic (linear) component $\eta(\mathbf{x}_i) := \langle \boldsymbol{\beta}, \mathbf{x}_i \rangle = \boldsymbol{\beta}^\top \mathbf{x}_i$ and the exposure time v_i as follows:

$$N_i \sim \text{Poisson}(v_i \cdot \exp(\eta(\mathbf{x}_i))),$$

where $\boldsymbol{\beta} \in \mathbb{R}^{p+1}$ is the vector of GLM parameters.

Denote $\lambda^{GLM}(\boldsymbol{\beta}, \mathbf{x}_i) := \exp(\eta(\mathbf{x}_i))$. The vector of parameters $\boldsymbol{\beta}$ is estimated via the maximum likelihood estimation method. We denote the estimated parameters by $\hat{\boldsymbol{\beta}}$ and the estimated (expected) annual number of claims for an observation \mathbf{x}_i by $\hat{\lambda}_i^{GLM} := \lambda^{GLM}(\hat{\boldsymbol{\beta}}, \mathbf{x}_i)$. The weighted average predicted frequency (WAPF) and the weighted average observed frequency (WAOF) are then defined as

$$\text{WAPF} = \frac{\sum_i \hat{\lambda}_i^{GLM} v_i}{\sum_i v_i}, \quad \text{WAOF} = \frac{\sum_i N_i}{\sum_i v_i}. \tag{1}$$

An important requirement for modeling claim counts is the equality of the WAPF and the WAOF. This property is called a balance property. A GLM satisfies this property on the data set used for model fitting. For the proof of this fact, an interested reader is referred to Equation (2.10) in [20].

In GLMs, different choices of variables lead to different predictive performance. The optimal set of variables can be found by fitting GLMs with different subsets of variables and comparing goodness-of-fit measures like the Akaike information criterion (AIC), the Bayesian information criterion (BIC), etc. These model evaluation criteria can be used for the automated selection of the best subset of available variables for a GLM. Popular automated feature selection methods are stepwise backward, forward, and mixed variable selection methods. As mentioned in the introduction section, these methods have huge computational costs for data sets with a large number of variables.

Next, we explain what an interaction means in the context of a GLM. For simplicity, we focus on pairwise interactions, i.e., those between pairs of variables. However, the concepts below can be extended to higher-order interactions, i.e., those among more than two variables. In a GLM, a pairwise interaction is an additional term $I(\cdot, \cdot)$ in the GLM's systematic component $\eta(\mathbf{x})$ and this term is a function that cannot be

¹ For each vector of variables, its first component is always 1 and serves the purpose of modeling an intercept component of a GLM.

expressed as a sum of two separate functions of the corresponding single variables. For example, let x_1 and x_2 be two numerical variables and $\beta_{1,2}$ be some parameter to be estimated. Then adding the term $\beta_{1,2} \cdot x_{\cdot,1} \cdot x_{\cdot,2}$ to $\eta(\mathbf{x})$ is considered adding a pairwise interaction, where $x_{\cdot,1}$ denotes a generic observation of feature x_1 and $x_{\cdot,2}$ denotes a generic observation of feature. However, including the term $\beta_{1,2} \cdot \ln(x_{\cdot,1} \cdot x_{\cdot,2}) = \beta_{1,2} \cdot \ln(x_{\cdot,1}) + \beta_{1,2} \cdot \ln(x_{\cdot,2})$ does not mean adding an interaction, but instead adding two transformed variables.

The parametric form of an interaction term $I(\cdot, \cdot)$ depends on the type of the corresponding covariates that interact. In the following, we provide three common parametric forms of $I(\cdot, \cdot)$ depending on the type of variables.

– *Interaction between two numerical covariates:*

Let x_1 and x_2 be two numerical variables. For an observation i , the term modeling the interaction² between them can be given by

$$I(x_{i,1}, x_{i,2}) = \beta_{1,2} \cdot x_{i,1} \cdot x_{i,2},$$

where $\beta_{1,2}$ is a parameter to be estimated.

– *Interaction between one numerical and one categorical covariate:*

Let x_1 be a numerical variable and x_2 be a categorical variable with J categories, where the last one serves as a reference category (also called a base level). For an observation i , the interaction between them can be modeled by

$$I(x_{i,1}, x_{i,2}) = \sum_{j=1}^{J-1} \beta_j \cdot x_{i,1} \cdot \mathbb{1}_{\{x_{i,2}=j\}},$$

where β_j are parameters to be estimated, $\mathbb{1}_{\{x_{i,2}=j\}} = 1$ if the observation i of variable x_2 is a category j , 0 otherwise.

– *Interaction between two categorical covariates:*

Let x_1 and x_2 be two categorical variables with R and S categories respectively, where the last one each serves as a reference category. For an observation i , the interaction between features $x_{\cdot,1}$ and $x_{\cdot,2}$ is modeled by

$$I(x_{i,1}, x_{i,2}) = \sum_{r=1}^{R-1} \sum_{s=1}^{S-1} \beta_{r,s} \cdot \mathbb{1}_{\{x_{i,1}=r\}} \cdot \mathbb{1}_{\{x_{i,2}=s\}},$$

where $\beta_{r,s}$ are parameters to be estimated, $\mathbb{1}_{\{x_{i,1}=r\}} = 1$ if the observation i of variable x_1 is a category r and 0 otherwise and likewise $\mathbb{1}_{\{x_{i,2}=s\}} = 1$ if the observation i of variable x_2 is a category s and 0 otherwise.

The search of important interactions is more challenging than the search of the best subset of variables for a GLM, since the number of all possible combinations of

² This is a simple parametric form of an interaction. In general, transformations of involved variables may be needed, e.g., raising to a power or taking a logarithm.

interacting variables is usually larger than the number of variables.³ Therefore, actuaries often use their expert knowledge to decrease the number of pairwise interactions to analyze in detail. The interactions to be analyzed are mainly explored in a visual manner, e.g., by evaluating plots that indicate the (weighted) average of the response variable for each unique combination of values of variables (or their binned versions).

In the next section, we describe in detail our suggested approach to detecting important pairwise interactions. It is faster than the majority of methods proposed in the literature and, thus, may save actuaries time to focus on other challenging tasks.

3 Algorithmic detection of the strongest interaction missing in a GLM

From now on we refer to the GLM that is to be improved as the benchmark GLM. To detect the next-best interaction for the benchmark GLM, we suggest an algorithm that consists of three steps:

1. Outperform the benchmark GLM using a CANN model.
2. Rank the interactions learned by the CANN according to their strength.
3. Determine the next-best interaction among top-ranked ones using mini-GLMs.

We refer to the model developed in Step 1 as the competitor model. Note that CANNs and other ML models cannot yet replace benchmark GLMs used by insurance companies in production for various reasons: lack of interpretability, etc.

3.1 Outperforming the benchmark GLM via CANN

As previously mentioned, GLMs have been a traditional technique for modeling and forecasting claim frequencies within the insurance sector. However, according to [20] their performance is limited in comparison to models based on NNs, which by their design learn non-linear interactions among variables. Thus, in Step 1 of the suggested approach, we train a CANN that can be seen as a boosting step for the benchmark GLM and is, essentially, a NN that uses the predictions of the benchmark GLM while learning additional interactions among variables to improve the predictive power of the overall model. Before explaining CANNs in more detail, we provide the basics of NNs.

Consider a fully-connected feed-forward NN with $d \in \mathbb{N}$ hidden layers and one neuron in the output layer. Denote by $q_l \in \mathbb{N}$ the number of neurons in the l -th hidden layer, $l = 1, \dots, d$. $q_0 \in \mathbb{N}$ denotes the number of neurons in the input layer. In our application of NNs, q_0 will be equal to the number of neurons needed to encode p variables in the original data set and may be larger than p since each categorical variable usually needs more than one neuron in the input layer. Denote by $\tilde{\mathbf{x}} \in \mathbb{R}^{q_0}$ the vector of pre-processed features that serve as input to the NN. Denote by $W^{(l)} \in \mathbb{R}^{q_l \times q_{l-1}}$ the weight matrices and by $\mathbf{b}^{(l)} \in \mathbb{R}^{q_l}$ the bias vectors, $l = 1, \dots, d$. Denote by $\mathbf{w}^y \in \mathbb{R}^{q_d}$ and by $b^y \in \mathbb{R}$ the coefficients vector and bias for the output neuron. Denote by $\phi_l(\cdot)$ the activation function of neurons in the l -th layer $l = 1, \dots, d + 1$

³ For p variables, there are $p(p - 1)/2$ possible pairwise interactions.

and $\vec{\phi}_l(\boldsymbol{\xi}) = (\phi_l(\xi_1), \dots, \phi_l(\xi_{q_l}))^\top$ for any $\boldsymbol{\xi} \in \mathbb{R}^{q_l}$. Then the hidden layers $\mathbf{z}^{(l)}$ and the output layer consisting of one neuron y (the NN’s prediction) can be expressed as follows:

$$y = \phi_{d+1} \left((\mathbf{w}^y)^\top \mathbf{z}^{(d)} + b^y \right), \quad \mathbf{z}^{(l)} = \vec{\phi}_l \left(W^{(l)} \mathbf{z}^{(l-1)} + \mathbf{b}^{(l)} \right), \quad l = 1, \dots, d,$$

with $\mathbf{z}^{(0)} := \tilde{\mathbf{x}}$. Let $\phi_{d+1}(z) = z$ and denote the regression function of a NN by

$$\lambda^{\text{NN}}(\tilde{\mathbf{x}}) := (\mathbf{w}^y)^\top \left(\mathbf{z}^{(d)} \circ \mathbf{z}^{(d-1)} \circ \dots \circ \mathbf{z}^{(1)} \right) (\tilde{\mathbf{x}}) + b^y. \tag{2}$$

A combined actuarial neural network (CANN) for claim counts satisfies two model assumptions:

1. $N_i \sim \text{Poisson}(v_i \cdot \lambda^{\text{CANN}}(\tilde{\mathbf{x}}_i))$ with the regression function λ given by

$$\tilde{\mathbf{x}}_i \mapsto \ln \left(\lambda^{\text{CANN}}(\tilde{\mathbf{x}}_i) \right) = \ln(\hat{\lambda}_i^{\text{GLM}}) + \lambda^{\text{NN}}(\tilde{\mathbf{x}}_i) \tag{3}$$

2. The regression function in (3) is initialized with weights $\mathbf{w}^y = (0, 0, \dots, 0)^\top \in \mathbb{R}^{q_d}$ and $b^y = 0$.

The first structural assumption means that the NN part of CANN boosts the benchmark GLM. The second structural assumption implies that at the beginning of the training phase, the Poisson deviance of a CANN model equals the Poisson deviance of the benchmark GLM. If the Poisson deviance loss is used as the objective function for training a CANN, then during training the gradient-descent algorithm explores the NN architecture for an additional model structure that is not present in the benchmark GLM and that further decreases the CANN’s Poisson deviance.

It is not necessary to know the structure of the benchmark GLM, only its predictions are used by a CANN model. So the weights of the benchmark GLM are non-trainable and the implementation of a Poisson CANN can be simplified by merging the annualized predictions of the benchmark GLM with the given volumes. In particular, as an alternative to the first structural assumption of a Poisson CANN, we can consider $N_i \sim \text{Poisson}(v_i^{\text{GLM}} \cdot \lambda^{\text{NN}}(\tilde{\mathbf{x}}_i))$ with modified exposure $v_i^{\text{GLM}} := v_i \cdot \hat{\lambda}_i^{\text{GLM}}$.

The architecture of a CANN is illustrated in Fig. 1. The neuron marked blue takes as input the modified exposure v_i^{GLM} and passes it directly to the output neuron marked green, whereas the corresponding red-marked connection has a non-trainable weight 1 and a bias coefficient 0. The neurons marked black and connections among them constitute the NN component of a CANN. In the NN component of the CANN model shown in Fig. 1, $q_0 = 8, q_1 = 6, q_2 = 4, q_3 = 1$. The output λ^{NN} of the NN component is passed via the red-marked connection (with a non-trainable weight 1) to the green-marked output neuron. The output neuron sums two incoming values and applies the exponential activation function on the result. The CANN model is trained using the Poisson deviance loss function.

As mentioned above, the original vector of variables $\mathbf{x} \in \mathbb{R}^p$ does not enter the input layer of a NN, but its pre-processed version $\tilde{\mathbf{x}} \in \mathbb{R}^{q_0}$ does. In particular, all features that appear in the input layer of a NN, must not contain missing values, should be

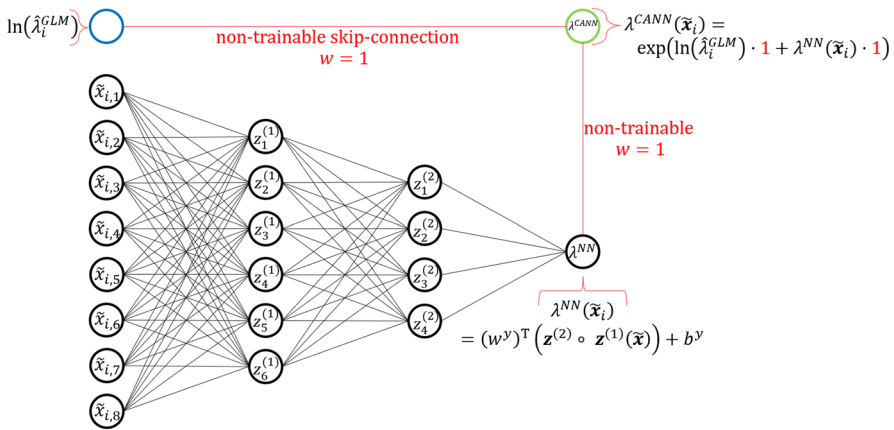


Fig. 1 Architecture of a CANN model

numerical, and ideally have the same range. Therefore, we use min-max-scaling to all numerical features used for training a NN. As for categorical features, we recommend using one-hot encoding for features with a low number of unique categories, e.g., below 5, and use the embedding layers technique for those with a larger number of unique categories. These techniques are recommended in [21].

An embedding e of a categorical feature with k distinct categories $\{a_1, \dots, a_k\}$ is a mapping

$$e : \{a_1, \dots, a_k\} \rightarrow \mathbb{R}^g, a \mapsto e(a),$$

with $g \in \mathbb{N}$ denoting the dimension of the embedding. This dimensionality parameter is chosen by the user, whereby typically $g \ll k$. The components

$$e(1)_1, \dots, e(1)_g, \dots, e(k)_1, \dots, e(k)_g$$

of such an embedding of k categories constitute additional NN weights that are learned during training. So an embedding layer of dimension g results in additional $g \cdot k$ embedding weights. The embedding representation of an embedded feature, i.e., the output of the embedding layer, equals the embedding weights. A NN with the above-mentioned peculiarities is schematically illustrated in Fig. 2.

To reduce the risk of overfitting, we recommend using a dropout technique and an early stopping of the NN training process. According to the drop-out technique proposed by [17], a pre-specified percentage (a so-called dropout rate) of neurons randomly selected in each layer is “switched off” and not updated in a NN training step. According to the early-stopping method, the NN training is stopped as soon as a significant deterioration or no significant improvement in the model performance is observed within a predefined period of time.

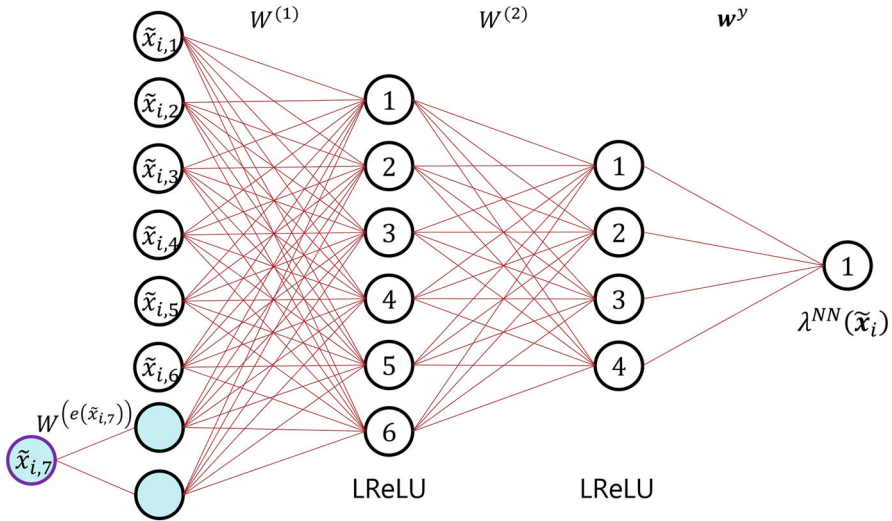


Fig. 2 Example of the NN part of a CANN model that uses a 2-dimensional embedding layer (in light blue) encoding a categorical feature $\tilde{x}_{.,7}$

A Poisson CANN model does not satisfy the balance property (1). As we will see in the numerical studies and as it is also found in the numerical studies of [16], the violation of the balance property is very small, since a CANN model uses the predictions of a GLM that fulfills this property. In the view of the interaction detection as the main focus of our paper, the violation of the balance property is negligible. Readers interested in enforcing the balance property on neural networks are referred to [20] and [21].

Before training the above-described CANN, one has to specify certain hyper-parameters, e.g., the embedding dimension, the number of hidden layers, the number of neurons per layer, the dropout rates, the activation functions, loss functions and the optimizer, the batch size, the number of epochs, or the usage of early stopping, etc. Our experiments on both artificial and real data sets show that choosing 3 hidden layers, an embedding dimension of 2, the Poisson deviance as a loss function, and a dropout rate of 10% has very high chances for a CANN to outperform sophisticated benchmark GLMs. If one would like to further improve the performance of the ML model, then one should explore ML models with different values of hyper-parameters. The search for the optimal values of hyper-parameters can be done either via a grid search or a genetic algorithm. The latter approach is more time-consuming, but it can yield a better model performance.

To compare the performance of a ML model (for short competitor) and a benchmark GLM (for short benchmark) in the context of MTPL insurance claim counts, we recommend using so-called double lift plots on the test data set. These plots are of high practical importance for actuaries, see, e.g., Section 7.2.2 in [5]. A double lift-plot requires predictions of each of the two models and the true observed values of the response variable. A double lift plot is created in the following way:

1. Determine the deviance δ_i (also called the sort ratio), which is the relative difference between the competitor model and the benchmark GLM:

$$\delta_i = \frac{\hat{\lambda}_i^{\text{competitor}}}{\hat{\lambda}_i^{\text{benchmark}}} - 1,$$

where $\hat{\lambda}_i^{\text{competitor}}$ denotes the i -th prediction of the competitor model and $\hat{\lambda}_i^{\text{benchmark}}$ refers to the i -th prediction of the benchmark GLM.

2. Sort the observations based on δ_i , from smallest to largest.
3. Bucket the observations into predetermined bins in an interval of interest, e.g., bins $(-\infty, -0.5]$, $(-0.5, -0.48]$, $(-0.48, -0.46]$, \dots , $(0.48, 0.5]$, $(0.5, +\infty)$.
4. For each bin, calculate the exposure, WAOF, WAPF of the competitor model, and WAPF of the benchmark model.
5. For each bin, plot the quantities calculated in Step 4. The left y-axis refers to the WAOF or WAPF that are marked by dots in the double lift plot. The right y-axis refers to the exposure that is depicted by bars below the dots.

An example of a lift plot with predetermined binning can be seen in the left sub-figure of Fig. 3. As an alternative to bucketing the observations based on the predetermined binning, one can use quantile-based binning. A double-lift plot of this type can be seen in the right sub-figure of Fig. 3. In this chart, each bin has the same number of observations and is determined based on the quantiles of the distribution of δ_i .

Obviously, the evaluation of these lift plots is based on visual perception. In order to allow for a purely quantitative model evaluation, we construct KPIs reflecting the information captured in these lift plots and, thus, not requiring visual evaluation of lift plots.

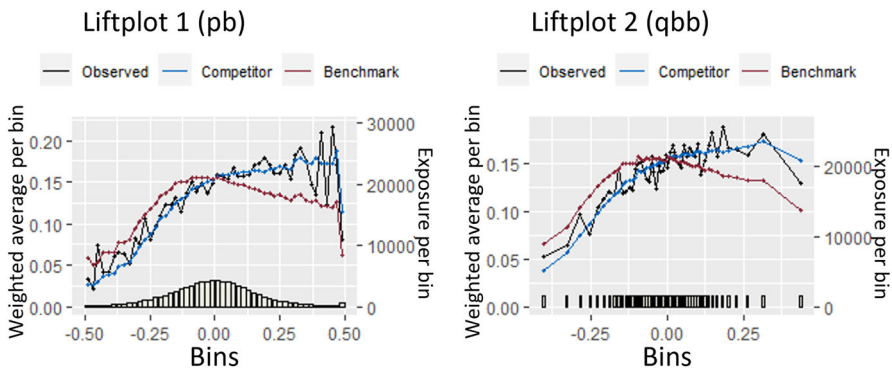


Fig. 3 Lift plot 1 has predetermined bins (PB). Lift plot 2 has quantile-based bins (QBB)

Let $B = \{1, \dots, |B|\}$ be the set of bins in a lift plot and $b \in B$ an index of a certain bin. Define the weighted exposure w_b per bin as follows:

$$u_b = \frac{\sum_{x_i \in \mathcal{X}_b} v_i}{\sum_{x_i \in \mathcal{X}} v_i},$$

where \mathcal{X}_b is the set of features vectors that correspond to observations in bin $b \in B$. The numerator equals the total exposure of observations in a bin b and the denominator is equal to the total exposure in the whole data set used for calculating this KPI. Using this weighted exposure per bin, the mean absolute error based on the lift plot bins is given by

$$\text{mae_lift_...} = \sum_{b \in B} u_b |\text{WAPF}_b - \text{WAOF}_b|.$$

All in all, we thereby construct a selection of numerical KPIs, namely

- `mae_lift_pb` and `mae_lift_pb_benchmark`, which are based on the lift plot with predetermined bins and use the predictions of the competitor model and the benchmark model respectively
- `mae_lift_qbb` and `mae_lift_qbb_benchmark`, which are based on the lift plot with quantile-based bins and use the predictions of the competitor model and the benchmark model respectively

The smaller the value of `mae_lift_...`, the better the model. For example, if `mae_lift_pb` is smaller than `mae_lift_pb_benchmark`, the competitor model outperforms the benchmark model based on the lift-plot with predetermined bins. The same reasoning holds for the KPIs using quantile-based binning.

In summary, actuaries often rely on lift plots when evaluating model performance. However, the visual interpretation of such lift plots may be rather subjective. Hence, the transformation of the lift plot into a numeric KPI and using it along with Poisson deviance for model selection may enhance reliability and the objectivity of the performance evaluation. For example, it may happen⁴ that Poisson deviance is the same for two models, but one model is convincingly better than the other one according to lift-plot based KPIs.

3.2 Opening the black box: ranking learned interactions

Having found a well-performing CANN, the next step is to find the most significant pairs of interacting variables learned by the model. Here, “significant” means that those pairs of interacting variables that are captured by the CANN model are likely to strongly improve the predictive power of the benchmark GLM if included in it. In the CANN model, its NN component learns non-linear interactions among the input features. To quantify the significance of each of the learned interactions, we apply

⁴ We have observed this when applying our methodology to a big proprietary data set.

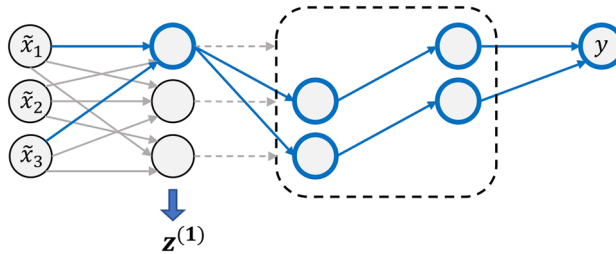


Fig. 4 Generation of interactions in the first hidden layer and propagation of these interactions through the neural network (figure adapted from [19])

a fast model-specific interaction-detection method. The method can be seen as an adjustment of a technique called Neural Interaction Detection (NID), proposed by [19] for fully-connected feed-forward NNs.

The original NID algorithm is based on the assumption that feature interactions are created in the first hidden layer of a neural network. Note that learning interactions in the first hidden layer is possible due to the usage of non-linear activation functions. Moreover, [19] provides an empirical evidence that considering the first hidden layer is indeed sufficient for determining interactions. These interactions are then propagated through the whole network and influence the final prediction. This concept is exemplified by Fig. 4. As can be seen, the first neuron in the first hidden layer $z^{(1)}$ (highlighted in blue) takes inputs \tilde{x}_1 and \tilde{x}_3 and thereby creates an interaction between them if the activation function of that neuron is non-linear.⁵ The strength of this interaction is evaluated based on both incoming weights as well as the outgoing paths from the neuron to the output neuron y as colored in blue in Fig. 4. The higher the incoming weights and the higher the impact of the considered neuron on the final output, the stronger the interaction. The strength of an interaction is quantified by an interaction strength score.

Let \mathcal{I} be a pair of input neurons. The interaction between these input neurons happens at each neuron of the first hidden layer. Denote by $s_j(\mathcal{I})$ the strength of an interaction between input neurons in \mathcal{I} measured at the j -th neuron in the first hidden layer, $j = 1, \dots, q_1$. It is quantified as follows:

$$s_j(\mathcal{I}) = \zeta_j^{(1)} \cdot \mu(|W_{j,\mathcal{I}}^{(1)}|), \quad s_j(\mathcal{I}) \in \mathbb{R}, \quad (4)$$

where $\zeta_j^{(1)}$ represents the influence of neuron j in the first hidden layer on the model prediction, $|W_{j,\mathcal{I}}^{(1)}|$ denotes the absolute value of the incoming weights from features in \mathcal{I} to neuron j in the first hidden layer, and $\mu(\cdot)$ represents a so-called generalized surrogate function used to capture the strength of the interaction based on the relevant

⁵ A superposition of linear functions is a linear function. Since the argument of a neuron's activation function is a linear function of the activation values of neurons in the previous layer of a NN, the usage of linear activation functions leads to an output that is linear in the values of neurons in the input layer of an NN. Hence, linear activation functions are not suitable for capturing interactions, as interactions are dependencies among variables that cannot be represented as a sum of functions of the corresponding single variables.

incoming weights. In our notation, $|\cdot|$ applied to a matrix means that the absolute value is taken element-wise, i.e., for all matrix elements.

As per [19], the generalized surrogate function $\mu(\cdot)$ should be such that interaction strength is

1. Quantified as zero when the interaction does not exist;
2. Non-decreasing in the magnitude of feature weights;
3. Less sensitive to changes in large feature weights.

The third property mitigates the impact of situations, when the weight of the connection from one input neuron has much higher magnitude than the weight of connection from another input neuron. If the large weight grows in magnitude, then interaction strength should not change much, but if instead the smaller (in magnitude) weight grows at the same rate, then interaction strength should increase. Thus, maximum, root mean square and arithmetic mean are not suitable candidates for $\mu(\cdot)$. [19] empirically investigates a selection of possible surrogate functions and concludes that the minimum is the best-performing function that recovers the highest number of true interactions in their experiments. The second-best choice for $\mu(\cdot)$ was the harmonic mean function. Therefore, we choose $\mu(\cdot)$ as *minimum* in all our experiments.

The influence $\zeta^{(1)}$ on the network prediction is calculated as the following matrix product of the absolute weight matrices:

$$\zeta^{(1)} = |\mathbf{w}^y|^\top \cdot |W^{(d)}| \cdot |W^{(d-1)}| \cdot \dots \cdot |W^{(2)}|, \zeta^{(1)} \in \mathbb{R}^{q_1}. \tag{5}$$

In this case, q_1 denotes the number of neurons in the first hidden layer, $W^{(m)}$ each represents the weight matrix connecting the neurons in hidden layers $m - 1$ and m , whereas \mathbf{w}^y denotes the vector of weights connecting the last hidden layer and the output neuron. Note that $\zeta^{(1)}$ results in a vector where the j -th index corresponds to the influence of a neuron j of the 1-st hidden layer on the output neuron of a NN. According to Lemma 3 in [19], if all activation functions in a NN are 1-Lipschitz continuous, then expression (5) is an upper bound for the gradient magnitudes of neurons in the first hidden layer, i.e., if $\left| \frac{\partial \phi(x)}{\partial x} \right| \leq 1$, then $\left| \frac{\partial y}{\partial z_j^{(1)}} \right| \leq \zeta_j^{(1)}$ for all $j = 1, \dots, q_1$. Common activation functions such as rectified linear unit, hyperbolic tangent and sigmoid are 1-Lipschitz continuous.

After having extracted the incoming weights of the NN as well as the importance (w.r.t. the influence on the NN’s output) of each neuron in the first hidden layer, the strength of a (local) interaction between a subset of input neurons can be computed for each neuron of the first hidden layer. Subsequently, the final interaction strength score for this subset of input neurons is equal to the sum of local interaction strength scores across all q_1 neurons in the first hidden layer:

$$s(\mathcal{I}) = \sum_{j=1}^{q_1} s_j(\mathcal{I}),$$

which is exemplary illustrated in Fig. 5.

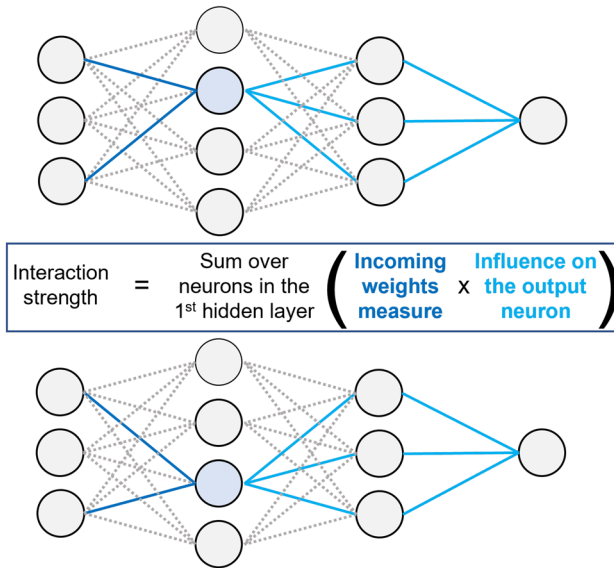


Fig. 5 Illustration of the interaction-strength calculation

Recall that a CANN model is a combination of a NN and a benchmark-GLM prediction with a skip-connection to the output neuron of the model, where the NN component is a feed-forward fully-connected neural network. Therefore, we can apply NID to the NN component of CANN model to quantify the strength of all pairwise⁶ interactions among features. We provide the R-code for the described NID approach in Listing 5 in Appendix A.

The original NID method evaluates the strengths of interactions among input neurons. However, the encoding of categorical features may require several neurons in the input layer of a NN, e.g., in case of one-hot encoding one needs the same number of input neurons as the number of categories. If an actuary wants to detect interactions on a per-category level, then it is not necessary to aggregate NID scores related to neurons encoding a categorical variable. However, to obtain the interaction-strength scores related to a categorical feature taken as a whole, one has to aggregate the scores on a per-neuron basis using some aggregation function like mean, minimum, maximum. If an actuary is interested in finding interactions where the majority of categories of the categorical feature of interest are strongly interacting with another variable, then *min* is recommended. If the aim is to find categorical variables whose categories have on average high interaction-strength scores with the other variable, then *mean* is a good choice for aggregation. Finally, if an actuary is interested in finding categorical variables where one category is especially strongly interacting with the other variable, then we recommend to use *max* as the aggregation function. For example, to get the strength of an interaction between a categorical one-hot encoded feature and a numer-

⁶ The strengths of interactions of the order higher than 2 can be also computed via the NID approach. In that case, one can use a greedy approach to speed up calculations. For more information, see Section 4.2 in [19].

ical feature, one can take the maximum of all $s(\mathcal{I})$ where \mathcal{I} contains an input neuron encoding a category of the categorical feature of interest and an input neuron encoding the numerical feature of interest.

3.3 Identification of the next-best interaction for a GLM

After extracting the most significant interactions, the final step is to determine the next-best interaction for the benchmark GLM. This step is necessary for several reasons. First, the inclusion of any interaction in a GLM requires a parametric specification $I(\cdot, \cdot)$ of the interaction. This is also important for preserving the interpretability of the benchmark GLM. Second, it may happen that several top-ranked interactions have very similar interaction-strength scores according to NID, which is why choosing the next-best interaction may become ambiguous. In this case, an actuary may want to estimate the improvement of the benchmark GLM for each of the top-ranked interactions and afterwards decide which one to include and retrain the benchmark model with the found interaction.

To decide which interaction to add to the benchmark GLM, we suggest to predict the observed claim counts via “mini-GLMs” that use the predictions of the benchmark GLM and the top-ranked interactions. This approach can also be interpreted as freezing the coefficients of the benchmark GLM and adding one interaction on top to better predict the claim counts. The approach works as follows:

1. For each (x_j, x_k) from the list of top-ranked interactions and for each relevant parametric form of $I(\cdot, \cdot)$

(a) Fit a mini-GLM assuming:

$$N_i \sim \text{Poisson} \left(v_i \hat{\lambda}_i^{\text{benchmark}} \cdot \exp \left(I(x_{i,j}, x_{i,k}) \right) \right), \quad i = 1, 2, \dots, n.$$

(b) Calculate the KPIs of interest, e.g., AIC, residual deviance, etc.

2. Recommend as the next-best interaction the one that corresponds to the mini-GLM with the best KPIs.

Remarks

1. The word “relevant” refers to the fact that the exact functional form $I(x_{\cdot,j}, x_{\cdot,k})$ of the interaction between variables x_j and x_k depends on the types of these variables (see the Sect. 2). The expressions $x_{\cdot,j}$ and $x_{\cdot,k}$ denote the respective values (realizations) of variables x_j and x_k in a generic observation.
2. If at least one of the interacting variables is continuous, one has multiple options for choosing the parametric form of the interaction:
 - (a) Consider several continuous transformations of the continuous feature(s) of interest. For example, an actuary may consider only parametric interactions of the form of $I(x_{\cdot,j}, x_{\cdot,k}) = \beta_{j,k} \cdot x_{\cdot,j}^a \cdot x_{\cdot,k}^b$ for $a \in \{1, 2, 3\}$ and $b \in \{1, 2, 3\}$. The form that leads to a mini-GLM with the best KPI is chosen.
 - (b) Bin the continuous feature(s) of interest and include the interaction between the binned versions of those features. A simple binning procedure can be based on

the quantiles of their distribution. A more advanced binning procedure can be based on fitting a generalized additive model (GAM) that uses only a smooth version of the interaction of interest and the predictions of the benchmark-GLM as offset. Afterwards one trains a regression tree that predicts the GAM-captured interaction effect using the interacting features and concludes the optimal binning from the splits of the regression tree. For more information on this method, see Section 4.2 in [7].

3. It may be computationally challenging⁷ to fit a mini-GLM for categorical features with a large number of categories, e.g., postcode. For these cases, we recommend clustering categories of such variables based on the embedded representations of those variables.

4 Case studies

In this section, we summarize the results of several case studies, which we conduct on a computer with 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz processor, 32 GB RAM, Intel(R) Iris(R) Xe Graphics, and Windows 10 Enterprise operating system. In the first case study, we work with an artificially generated data set, where we know the true interactions among variables in the data set. The aim of this case study is to show that our methodology detects and recommends the true interaction. In the second case study, we work with an open-source data set, where the true interactions in the data are not known. In the third case study, we briefly discuss the benefits of our methodology for big data sets, since big insurers have millions of observations and keep track of tens of variables.

4.1 Artificial data set

In this subsection, we apply the previously described methodology to an artificially created data set. We start with generating vectors of covariates $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,10})^\top \in \mathbb{R}^{10}$, $i = 1, \dots, 2 \cdot 10^6$, such that each vector is independent of other vectors. Variables x_1, \dots, x_8 are numerical and come from a multivariate normal distribution with zero mean and unit variance, as in [15], namely $(x_1, \dots, x_8)^\top \sim N(0, \Sigma)$ with Σ being an identity matrix with an additional entry of 0.5 in the cells (2, 8) and (8, 2). Covariates x_9 and x_{10} are categorical and come from a binomial distribution. To be specific, the variable $x_9 \sim \text{Binomial}(2, 0.3)$ with three categories $\{0, 1, 2\}$ and is independent of other variables. The covariate $x_{10} \sim \text{Binomial}(5, 0.2)$ has six categories $\{0, 1, 2, 3, 4, 5\}$ and is independent of other covariates. For simplicity, we assume that $v_i = 1$, $i = 1, \dots, 2 \cdot 10^6$.

Based on the above-generated features and exposure, we generate the claim counts as follows. First, we calculate

⁷ Some versions of the actuarial software Emblem have a technical limit on the number of categories in categorical variables used in a GLM.

Table 1 Distribution of the number of claims

		Number of observed claims							Claim total
		0	1	2	3	4	5	6	
Full	Num.	1887159	105560	6516	645	103	15	2	112841
	%	94.3580	5.2780	0.3258	0.0323	0.0052	0.0075	0.0001	100.00
Train	Num.	1509486	84518	5250	540	88	11	2	90409
	%	94.3491	5.28272	0.3281	0.0337	0.0055	0.0007	0.0001	80.12
Val.	Num.	189254	10441	630	56	7	1	0	11135
	%	94.4433	5.2104	0.3144	0.0279	0.0035	0.0005	0.0000	9.87
Test	Num.	188419	10601	636	49	8	3	0	11297
	%	94.3435	5.3080	0.3185	0.0245	0.0040	0.0015	0.0000	10.01

$$\begin{aligned}
 \mathbf{x} \in \mathbb{R}^{10} \mapsto \mu(\mathbf{x}) = & \exp(-3 + 0.5 \cdot x_1 - 0.25 \cdot x_2^2 + 0.5 \cdot |x_3| \cdot \sin(2 \cdot x_3) + 0.5 \cdot x_4 x_5 \\
 & + 0.125 \cdot x_5^2 x_6 - 0.1 \cdot 1_{\{x_9=1\}} - 0.2 \cdot 1_{\{x_9=2\}} + 0.1 \cdot 1_{\{x_{10}=1\}} \\
 & + 0.2 \cdot 1_{\{x_{10}=2\}} + 0.3 \cdot 1_{\{x_{10}=3\}} + 0.4 \cdot 1_{\{x_{10}=4\}} + 0.5 \cdot 1_{\{x_{10}=5\}}).
 \end{aligned}$$

For a small number of feature vectors it holds $\mu(\mathbf{x}) > 1$. In those cases we set $\mu(\mathbf{x}) = 1$ to avoid unrealistically large number of claim counts for those vectors. In the final step of the data generation process, we obtain claim counts by generating them as follows:

$$N_i \sim \text{Poisson}(v_i \cdot (\mu(\mathbf{x}_i))), \quad i = 1, \dots, 2 \cdot 10^6.$$

The structure of the resulting data set is summarized in Listing 1.

Listing 1 Structure of the artificial data set

```

'data.frame':  2000000 obs. of  14 variables:
 $ claim_total_nb : int  0 0 0 1 0 1 0 0 0 0 ...
 $ annual_exposure: num  1 1 1 1 1 1 1 1 1 1 ...
 $ x_1 : num  -0.185 1.131 1.305 1.049 0.355 ...
 $ x_2 : num  0.465 0.444 -1.664 -1.004 0.99 ...
 $ x_3 : num  0.2259 -0.4888 0.0332 0.2362 1.0171 ...
 $ x_4 : num  0.696 -0.537 -3.043 1.849 -1.085 ...
 $ x_5 : num  1.932 0.276 -0.41 1.086 1.257 ...
 $ x_6 : num  0.716 0.11 -0.214 1.137 0.63 ...
 $ x_7 : num  -0.533 -1.466 -0.276 -1.457 -1.446 ...
 $ x_8 : num  0.5058 -0.0455 -1.0353 0.8823 -1.2143 ...
 $ x_9 : Factor w/ 3 levels "0","1","2": 1 3 1 2 2 1 1 2 ...
 $ x_10 : Factor w/ 6 levels "0","1","2","3",...: 2 2 1 1 3 ...
    
```

We split the data set as follows: 80% for training, 10% for validation, 10% for testing. This is a rule for splitting data according to [1]. The training set is used for fitting the model, the validation set is utilized for fine-tuning the hyper-parameters of the ML model, and the test set is used for evaluating the final out-of-sample performance of the chosen best-performing ML model. This results in the claim distributions shown in Table 1.

To fit a benchmark GLM, we use both training and validation data. In this GLM, we include terms $x_1, x_2^2, x_3, x_3^2, x_9, x_{10}$, which appeared in the data generation process. However, we do not include in the benchmark GLM the interactions between features x_4 and x_5 and between features x_5 and x_6 , which are the true interactions according to the process of the artificial data generation. If our interaction-detection methodology works correctly, one of these interactions will be recommended as the next-best one to be included in the benchmark GLM.

4.1.1 Step 1: training CANN

We conduct the following data pre-processing steps prior to training the CANN model:

- Use one-hot encoding for the categorical feature x_9 .
- Use a 2-dimensional embedding layer for the categorical feature x_{10} .
- Apply min-max scaling to all numerical features x_1, \dots, x_8 :

$$\tilde{x}_{:,j} = \frac{2 \cdot (x_{:,j} - \min(x_{:,j}))}{\max(x_{:,j}) - \min(x_{:,j})} - 1, \quad j = 1, \dots, 8.$$

To fit a CANN, we use the R package *keras*. The search for the optimal hyper-parameters of a CANN is based on the KPIs introduced in Subsection 3.1. To find the best CANN model, we search for the best combination of hyper-parameters along a pre-defined grid of hyper-parameters. We focus on the leaky rectified linear unit (LReLU), sigmoid (σ) and hyperbolic tangent (TanH) activation functions that are defined as

$$\begin{aligned} \text{LReLU}(w, \alpha) &= \max(w, \alpha \cdot w), \\ \sigma(w) &= \frac{1}{1 + e^{-w}}, \\ \text{TanH}(w) &= 2\sigma(2w) - 1, \end{aligned}$$

with w denoting the weighted sum of the inputs of the neuron to which the activation function is applied and $\alpha \in (0, 1)$ is a parameter, which we set to 0.3 in all our case studies. We use the Poisson deviance loss function, which is minimized via the RMSProp optimizer. To prevent overfitting, we use drop out rate of 5% and early stopping of model training when the value of the loss function is not improved 5 epochs in a row. We set the dimension of embedding layers to 2 and the number of neurons in the first, second and third hidden layers to $q_1 = 20, q_2 = 15, q_3 = 10$ respectively. In addition, the batch size is set to 1000 and the number of epochs to 100.

According to both Poisson deviance and the lift-plot-based KPIs `mae_lift_pb` and `mae_lift_qbb`, the best CANN model (among those we tested) for the artificial data has LReLU activation function in all neurons of all hidden layers. This architecture is summarized in Fig. 6.

As can be seen, the input layer is composed of 13 neurons corresponding to the 8 numeric features and the one-hot encoded feature capturing 3 categories. Moreover, the categorical feature x_{10} is encoded via an embedding layer of dimension two. The input

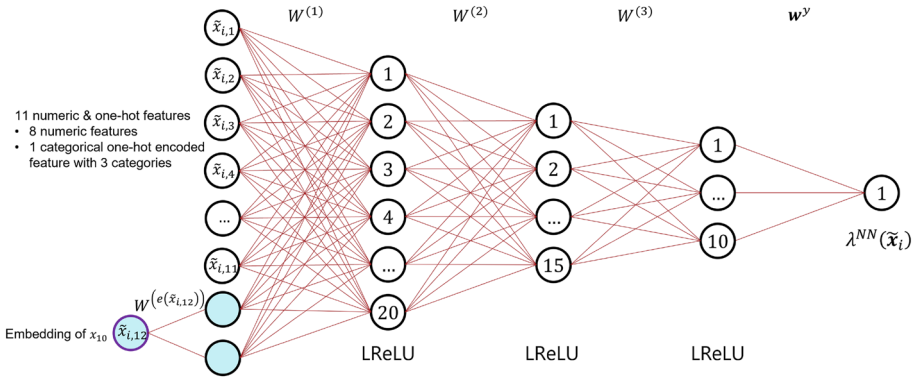


Fig. 6 The architecture of the NN component of the best-performing CANN model

Table 2 KPIs on the test data for the best-performing CANN model

Pois. dev	Pois. dev. bench	Lift_pb	Lift_pb_benchm	Lift_qbb	Lift_qbb_benchm
0.3049	0.3314	0.0033	0.0277	0.0042	0.0278

layer is connected to the first hidden layer via the weight matrix $W^{(1)}$. Similarly, the first and the second hidden layers are connected via weight matrix $W^{(2)}$, the second and the third hidden layers are connected via weight matrix $W^{(3)}$. The third (last) hidden layer is connected to the output layer via the vector of weights w^y .

The KPIs for the best CANN are summarized in Table 2.

4.1.2 Step 2: ranking of learned interactions via neural interaction detection

From the best-performing CANN model, we extract the weight matrices $W^{(1)}$, $W^{(2)}$, $W^{(3)}$ as well as the vector w^y . The weight matrices can be extracted using the *get_weights* function in R. The structure of the resulting output is shown in Listing 2. The first element of the list corresponds to the embedding weight matrix $W^{(e(\tilde{x}_{i,12}))} \in \mathbb{R}^{7 \times 2}$ depicted in Fig. 6. The second list element represents the transposed version of the weight matrix $W^{(1)}$ connecting the input layer consisting of 13 neurons (11 numeric & one-hot features + 1 · 2 neurons of the embedding layer related to x_{10}) with the first hidden layer, which has 20 neurons. Likewise, the fourth element of the list and the sixth one correspond to the transposed versions of the weight matrices $W^{(2)}$ and $W^{(3)}$ respectively. The eighth element of the list is the vector w^y that connects the last hidden layer of the NN component with its output neuron y . The third, the fifth, the seventh and the ninth element of the list represent bias vectors corresponding to the three hidden layers and the output layer of the NN component of a CANN, respectively. The tenth and eleventh element of the list are the non-trainable weights of the NN component (see red connections in Fig. 1). The twelfth (last) element of the list is the (non-trainable) bias element related to the output neuron of CANN.

Table 3 Top 5 pairwise interactions according to the CANN + NID approach

Interaction rank	Feature 1	Feature 2	Interaction strength score
1	x_4	x_5	70.0263
2	x_5	x_6	37.3492
3	x_4	x_6	34.7608
4	x_5	x_{10}	24.3280
5	x_4	x_{10}	23.9654

Listing 2 Weight matrices extracted from the CANN model

```

> model_weights <- get_weights(model)
> str(model_weights)
List of 12
 $ : num [1:7, 1:2] 0.0358 0.063 0.0683 0.0792 0.0507 ...
 $ : num [1:13, 1:20] 0.0656 0.0494 -0.9771 0.3547 ...
 $ : num [1:20(1d)] 0.049 -0.0885 -0.1021 0.0257 ...
 $ : num [1:20, 1:15] 0.287 -0.234 0.371 -0.104 0.799 ...
 $ : num [1:15(1d)] -0.0183 -0.02 -0.1593 0.0207 0.0435 ...
 $ : num [1:15, 1:10] 0.2464 -0.0183 -0.4323 0.5566 ...
 $ : num [1:10(1d)] 0.083 -0.0328 0.0171 0.0682 0.0687 ...
 $ : num [1:10, 1] -1.165 0.674 0.579 0.641 -0.802 ...
 $ : num [1(1d)] 0.138
 $ : num [1, 1] 1
 $ : num [1, 1] 1
 $ : num [1(1d)] 0

```

Next, we apply the modified NID to calculate the strength of interactions for each pair of features. Following the recommendation of [19], we use $\min(\cdot)$ as a surrogate function $\mu(\cdot)$. Having obtained the strength of interactions for each pair of input neurons, we apply the aggregation procedure for categorical features, as proposed in Subsection 3.2. In particular, we use minimum as the aggregation function.⁸ Finally, we sort the resulting list and provide top 5 entries in Table 3.

As can be seen, our modified NID procedure ranks the interactions between features x_4 and x_5 and between features x_5 and x_6 as the first and the second respectively. Interestingly, the NID procedure suggests that the third-ranked interaction happens between x_4 and x_6 . The reason for it is that x_5 appears in two interactions: $0.5 \cdot x_4 x_5$ and $0.125 \cdot x_5^2 x_6$. The strength of interactions among other variables is quantified as much lower.

Next we compare our method with another approach used by practitioners, namely training a gradient boosting machine (GBM) and calculating Friedman's H-statistic for each pair of features. Training one GBM model takes around 120 seconds. Calculating Friedman's H-statistic is very time-consuming for the whole data set. Therefore, we consider only a small portion of data, namely 10^4 observations, which is 0.5% share of all data. In this case, the calculation takes about 40 seconds. We report the results in Table 4.

⁸ Using the arithmetic average as the aggregation function for categorical variables does not change the ranking of top 5 interactions.

Table 4 Top 5 pairwise interactions according to GBM + Friedman H-statistic approach

Interaction rank	Feature 1	Feature 2	Interaction strength score
1	x_4	x_5	0.8495
2	x_5	x_6	0.2223
3	x_3	x_5	0.0156
4	x_3	x_6	0.0055
5	x_3	x_4	0.0001

According to Table 4, the true interactions have the largest H-statistic and are, thus, the strongest ones according to the method of training a GBM model and calculating Friedman's H-statistic for all possible pairs of variables. However, a different amount of data may lead to a different computation time and may result in a different ranking. For example, the calculation of this interaction-strength measure for the same GBM model but using 5% of data (10^5 observations) took about 350 seconds and indicated a few strong but false interactions, e.g., interactions between variables x_1 and x_2 , x_7 and x_8 had the H-statistic of 1.

We would like to close this subsection with a brief comparison of two methods. According to [10], Friedman's H-Statistic:

1. Can be applied to any model;
2. Is defined through the partial dependence decomposition and calculates the share of variance that is explained by the interaction;
3. Is usually (but not always) between 0 and 1, which allows for comparison across different models;
4. Detects all forms of interactions, independently of their specific structure;
5. Can be used for quantifying the strength of higher-order interactions, i.e., the interaction among 3 or more features
6. Is computationally time-consuming;
7. May lead to unstable results if not all data points are used, as the estimates also vary from run to run, which is why it is recommend to compute the H-statistic multiple times;
8. Does not provide a clear answer whether the interaction is statistically significant and it is not clear whether H-statistic is large enough to consider an interaction "strong";
9. Does not give the functional form of the interaction;
10. Has the assumption that features can be shuffled independently, which is, however, violated if features are strongly correlated;
11. May yield unexpected results for small amount of data.

Our approach of applying NID method to the NN component of a CANN model:

1. Is model specific and works only for feed-forward NNs with some regularity conditions on activation functions;
2. Is based on the decomposition of the strength of interaction between input neurons into two parts: the strength of connections from those input-layer neurons to the

- neurons in the first hidden layer, the influence of neurons in the first-hidden layer on the output neuron of the NN;
3. Does not lead to the interaction-strength score that is normalized between 0 and 1, which makes it challenging to compare the scores across different NNs;
 4. Detects all forms of interactions learned by the NN, independently of their specific structure;
 5. Can be used for quantifying the strength of higher-order interactions, i.e., the interaction among 3 or more features;
 6. Is computationally fast, since it requires only cumulative matrix multiplications of the matrices with absolute values of trained weights in the NN;
 7. Always leads to the same result, given that the NN is fixed, since the method does not explicitly use data points;
 8. Does not provide a clear answer whether the interaction is statistically significant and it is not clear whether the NID score is large enough to consider an interaction “strong”;
 9. Does not give the functional form of an interaction;
 10. Has the assumption that the interactions learned by the neural network are captured in the first hidden layer.

4.1.3 Step 3: recommendation of the next-best interaction

As described in Subsection 3.3, for each interaction from Table 3 we fit a mini-GLM and keep track of the corresponding KPIs.

The mini-GLM based on the interaction between features x_4 and x_5 has the lowest AIC and the lowest residual deviance among all 5 mini-GLMs. Therefore, it is selected as the next-best interaction to be included in the benchmark GLM.

The addition of the interaction between features x_4 and x_5 to the benchmark GLM improves the performance of the benchmark GLM. Its residual deviance drops from 596992 to 561969 and its AIC decreases from 804445 to 769424, implying that the model with this interaction should be favored. The Poisson deviance on the test data drops from 0.3314 to 0.3134.

After the benchmark GLM has been updated by adding the recommended interaction between x_4 and x_5 , we can repeat the whole process. Namely, training a new CANN model that uses the predictions of the updated benchmark GLM and applying the NID method to the NN component of the trained CANN model, we obtain the ranking of learned interactions as shown in Table 5. We see that the true interaction between features x_5 and x_6 is ranked as the strongest one. It has a much higher score than others. For each of the 5 top-ranked interactions, we train a mini-GLM with simple parametric forms of the interaction, which are described at the end of Sect. 2. As expected, the winning mini-GLM is related the true interaction between features x_5 and x_6 . This model has an AIC of 769406 and a residual deviance of 561960 on 1800737 degrees of freedom. The Poisson deviance on the test set is 0.3134.

If we train mini-GLMs with a larger class of parametric forms for interactions, namely, $I(x_{\cdot,j}, x_{\cdot,k}) = \beta_{j,k} \cdot x_{\cdot,j}^a \cdot x_{\cdot,k}^b$ for $a \in \{1, 2\}$ and $b \in \{1, 2\}$, the best-performing mini-GLM corresponds to the interaction of the form $I(x_{\cdot,5}, x_{\cdot,6}) = \beta_{5,6} \cdot x_{\cdot,5}^2 \cdot x_{\cdot,6}$.

Table 5 Top 5 pairwise interactions in according to the CANN + NID approach

Interaction rank	Feature 1	Feature 2	Interaction strength score
1	x_5	x_6	25.1675
2	x_3	x_{10}	18.8742
3	x_3	x_5	17.8092
4	x_3	x_6	15.6298
5	x_6	x_{10}	14.8482

This mini-GLM has an AIC of 763910 and a Poisson deviance of 0.3105 on the test set. Adding this interaction to the benchmark GLM leads to an AIC of 763805 and a Poisson deviance of 0.3104 on the test set.

To justify that our approach does not only work as desired but is additionally way more time efficient, we measure the time required for executing the above described steps of training the CANN model, applying the NID technique, and fitting mini-GLMs. This yields on average approximately 170.3 seconds for the training of one CANN architecture, 1.19 seconds for the application of NID and 6.7 seconds for fitting one mini-GLM.

In this case study, we have verified that our methodology leads to a correct recommendation of the next-best interaction for the benchmark GLM and showed that NID is faster than Friedman's H-statistic. In the next case study, we work with a real-world open-source data set that has more features than in the toy example considered before.

4.2 Open-source data set *freMTPL2freq*

In this subsection, we work with an open-source data set *freMTPL2freq*, which is a part of the R package *CASdatasets*. We choose this data set, since it has been analysed in several papers, e.g., [16], [20], [21], [23]. We take [16] as the main reference and use the benchmark GLM as indicated on page 5 there. Afterwards, we apply our interaction-detection methodology and compare our results with those stated in Section 3.5 of [16].

The data set consists of 678013 observations. Listing 3 provides a glimpse on the data.

Listing 3 Structure of the data set

```
> str(freMTPL2freq)
' data.frame ': 678013 obs. of 12 variables:
 $ IDpol      : num  1 3 5 10 11 13 15 17 18 21 ...
 $ ClaimNb    : num  1 1 1 1 1 1 1 1 1 1 ...
 $ Exposure   : num  0.1 0.77 0.75 0.09 0.84 0.52 0.45 ...
 $ VehPower   : int   5 5 6 7 7 6 6 7 7 7 ...
 $ VehAge     : int   0 0 2 0 0 2 2 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 ...
 $ VehGas     : Factor w/ 2 levels "Diesel", "Regular": 2 2 1
 1 1 ...
```

```

$ Area      : Factor w/ 6 levels "A","B","C","D",...: 4 4 2
              2 2 ...
$ Density   : int 1217 1217 54 76 76 3003 3003 137 137 ...
$ Region    : Factor w/ 21 levels "Alsace","Aquitaine",...:
              21 21 ...

```

We conduct data pre-processing as in Section 1.3. of [16] and split the data into training data (80%), validation data (10%), and data for testing (10%). Next we train the benchmark GLM, referred to as GLM2 in Section 1.3 of the mentioned paper. The resulting benchmark GLM is summarized in Listing 4.

Listing 4 Model summary of the benchmark GLM

```

> summary(benchmark.GLM)

Call:
glm(formula = ClaimNb ~ VehPowerGLM + VehAgeGLM +
     BonusMalusGLM + VehBrand + VehGas + DensityGLM +
     Region + AreaGLM + DrivAge + log(DrivAge) +
     I(DrivAge^2) + I(DrivAge^3) + I(DrivAge^4), family =
     poisson(), data = data.trainval, offset =
     log(Exposure))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9626  -0.3782  -0.2891  -0.1629   6.7970

...

Null deviance: 200978 on 610346 degrees of freedom
Residual deviance: 190836 on 610300 degrees of freedom
AIC: 253210

Number of Fisher Scoring iterations: 6

```

4.2.1 Step 1: training CANN

As in the first case study, we conduct the following data pre-processing steps prior to training CANNs:

- Use one-hot encoding for all categorical features with 5 or fewer categories.
- Use embedding layers for all categorical features with more than 5 categories.
- Apply min-max scaling to all numerical features.

We focus on CANNs with three hidden-layers such that $q_1 = 20$, $q_2 = 15$, $q_3 = 10$, and use the same grid of hyper-parameters as the one in the case study with artificially generated data.

The best-performing CANN model has LReLU activation function in all hidden layers. The KPIs of this model on the test data are summarized in Table 6.

On the test data set, the best-performing CANN model outperforms the benchmark GLM in terms of all considered KPIs. This is an indication that the NN component that boosts the benchmark GLM may have found some interactions missing in the benchmark GLM.

Table 6 KPIs on the test data for the best-performing CANN

Pois. dev	Pois. dev. bench	Lift_pb	Lift_pb_bench	Lift_qbb	Lift_qbb_bench
0.3636	0.3749	0.0067	0.0326	0.0096	0.0317

Table 7 Top 10 pairwise interactions according to the CANN + NID approach

Rank	Feature 1	Feature 2	NID interaction-strength score
1	VehAge	BonusMalus	34.5956
2	VehAge	VehGas	25.8459
3	BonusMalus	VehGas	25.5106
4	BonusMalus	Area	24.5642
5	BonusMalus	Region	24.4570
6	BonusMalus	VehBrand	23.2495
7	VehAge	VehBrand	22.9104
8	VehAge	Region	21.9878
9	DrivAge	BonusMalus	21.1584
10	VehAge	Area	20.8879

4.2.2 Step 2: ranking of learned interactions

After training the CANN model, we apply the NID algorithm to calculate the strengths of pairwise interactions that were learned by the NN component, as described in Subsection 3.2. Similar to the case study with an artificial data set, we use minimum as a surrogate function and minimum as an aggregation function. Table 7 summarizes the resulting strongest 10 interactions.

According to Table 7, the interaction between variables VehAge and BonusMalus is much stronger than all other pairwise interactions. The other 4 interactions have a comparable magnitude and do not exhibit a clear “winner” among them.

Next we relate our results to those of [16] by reporting the interactions the researchers identified and indicating their interaction-strength rank according to our methodology: (VehPower; VehAge) with NID rank of 22, (VehPower; VehBrand) with NID rank of 26, (VehAge; VehBrand) with NID rank of 7, (VehAge; VehGas) with NID rank of 2, (DrivAge; BonusMalus) with NID rank of 9. Interestingly, the interaction between BonusMalus and regional variables Area or Region was not detected by the methodology proposed in [16], neither was detected the interaction between VehAge and BonusMalus.

Finally, we compare our results to the method based on GBMs and Friedman’s H-statistic. We choose the following grid of hyper-parameters to search for the best-performing GBM

- Number of trees 100, 200, 300;
- Minimal number of observations in a node 10, 25, 50;
- Shrinkage parameter 0.01, 0.05, 0.1,

Table 8 KPIs of the best-performing GBM model on the test data

Pois. dev	Pois. dev. bench	lift_pb	lift_pb_bench	lift_qbb	lift_qbb_bench
0.3431	0.3749	0.0493	0.0718	0.0512	0.0595

Table 9 Top 8 pairwise interactions according to the GBM + Friedman's H-statistic approach

Rank (H-stat.)	Feature 1	Feature 2	Friedman H-statistic	Rank (NID)
1	VehAge	VehGas	0.3436	2
2	DrivAge	Region	0.2728	15
3	DrivAge	BonusMalus	0.1660	9
4	BonusMalus	VehBrand	0.0997	6
5	VehAge	BonusMalus	0.0729	1
6	VehGas	Region	$1.9 \cdot 10^{-14}$	12
7	VehBrand	Region	$1.6 \cdot 10^{-14}$	18
8	VehAge	DrivAge	$6.7 \cdot 10^{-15}$	20

and train the corresponding 27 GBM models with the benchmark-GLM prediction as an offset. Training one GBM takes on average 80 seconds for the data under consideration. The best-performing GBM in terms of Poisson deviance has 100 trees, 50 as the minimal number of observations in a node, the shrinkage parameter of 0.1, and the bag-fraction parameter 0.5. The KPIs of this model are reported in Table 8. Interestingly, the best-performing GBM model has a better Poisson deviance than the best-performing CANN model, but its lift-plot based KPIs are worse.

When the whole data set is used, the calculation of Friedman's H-statistic for each pair of variables takes around 5 minutes. We report the corresponding strongest 8 pairwise interactions in Table 9. The H-statistic for each of the remaining pairwise interactions is 0.

We see that the first-strongest pairwise interaction according to the GBM + Friedman's H-statistic is the second strongest interaction according to the CANN + NID. The first-strongest pairwise interaction according to our approach is ranked as the fifth strongest according to GBM + Friedman's H-statistic. Interestingly, the pairwise interaction between BonusMalus and regional variables is not captured by the approach of GBM + Friedman's H-statistic.

4.2.3 Step 3: Recommendation of the next-best interaction

As described in Subsection 3.3, for each interaction from Table 7 we fit a mini-GLM and keep track the KPIs of interest. All mini-GLMs lead to the Poisson deviance of 0.3696 on the test set. Based on the AIC, the winning mini-GLM achieves the lowest AIC of 279859.2 and corresponds to the interaction between BonusMalus and Region. This interaction is then recommended to an actuary for improving the benchmark GLM.

If an actuary prefers to use another performance measure, it may well be that another interaction is recommended as the next-best one. For example, using BIC for evaluating mini-GLMs, our methodology would suggest the interaction between VehAge and VehGas, since the corresponding mini-GLM has the lowest BIC (279941.9).

In contrast to the case study with the artificial data set, we do not know the true functional form of the interaction between variables. Therefore, one may want to explore more sophisticated pairwise interaction terms, as mentioned in Section 3.3 in Remark 2. All in all, the determination of the optimal functional form of the next-best interaction is beyond the scope of this paper. The final decision is to be made by the actuaries.

4.3 Brief discussion on proprietary data sets

Data sets of large insurance companies contain millions observations (policy snippets) with dozens of features.⁹ Some of the categorical features, e.g., postcode or vehicle model, have a high number of categories. In such cases, our methodology is especially powerful. Due to a very large number of possible pairwise interactions, comparing all of them by training as many mini-models or refitting as many times the benchmark GLM would come with huge computation-time costs. An alternative method of finding the best-performing GBM model that uses the benchmark-GLM predictions as offset and then evaluating the strength of all interactions via Friedman's H-statistic is very time-consuming, as we already saw in the previous case studies for smaller open-source data sets. Our approach to interaction detection is instantaneous, once the CANN model is trained. Moreover, embedding layers in the trained CANN model allow to efficiently cluster categories of categorical variables with a large number of categories (e.g., postcodes, car brands) to be able to include them in the benchmark GLM.

5 Conclusion

In this paper, we propose an approach to detect the next-best interaction to be added to an arbitrary but fixed benchmark GLM in the context of claim counts modeling. First, we trained a CANN model, which can be seen as boosting the benchmark GLM by a neural network. Second, we applied a fast model-specific method called Neural Interaction Detection to quantify the strength of interactions between each pair of features and ranked interactions by their strength. Third, we identified the next-best interaction by comparing a small number of mini GLMs that corresponded to the top-ranked interactions. In the case studies, we validated the approach on two different data sets and discussed its usage for large proprietary data sets.

There are two advantages of our methodology. First, it is a fully automatable way of enhancing a benchmark GLM by including the next-best interaction missing in

⁹ For example, we had a chance to work with approximately 11 million observations and over 50 features. The calculation of Friedman's H-statistic was computationally expensive, whereas our methodology was fast.

it. Second, our methodology is faster than other approaches based on Friedman's H-statistic. Therefore, our methodology is especially suitable for big data sets with dozens of features and millions of observations. As a result, it can substantially decrease the amount of time that pricing actuaries spend on searching for interactions to improve their GLMs, which is usually visual and time-consuming.

The interaction-detection procedure we have introduced has several degrees of freedom, e.g., the encoding of features, hyper-parameters of the NN, the KPIs for selecting the best-performing CANN, and those for comparing mini-GLMs. Therefore, it would be interesting to analyze how sensitive our approach is to different choices for each degree of freedom.

Acknowledgements We acknowledge the support of ERGO Center of Excellence in Insurance, funded by the ERGO Group AG. We thank Kay Adam for providing the data as well as for valuable suggestions and Frank Ellgring for the opportunity to gain practical insights in actuarial pricing at Global P&C Pricing Department at ERGO Group AG. We acknowledge the support of Noel Stein, Samarth Mehrotra, Mario Ponce-Martinez, and Yichen Lou in the preparation phase of this project.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability statement The R code for the generation of the artificial data described and analyzed in Subsection 4.1 of this paper is available on GitHub in the repository *interactions-for-glm-via-nn*, <https://github.com/Y-I-H/interactions-for-glm-via-nn/tree/main/code>. The *freMTPL2freq* data set described and analyzed in Subsection 4.2 of this paper is available as a part of the Rpackage *CASdatasets*, <http://cas.uqam.ca/>. The proprietary data set mentioned in Subsection 4.3 of this paper is confidential and cannot be publicly shared.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Appendix A A code for NID algorithm

Listing 5 Code for executing NID algorithm to the NN component of a CANN

```
# extract weights from CANN
model_weights <- get_weights(CANN_model)
# IL = input layer, OL = output layer, HL = hidden layer
# Index of weight matrix from IL to HL 1
layer_1_index <- num.embed.features + 1
# Index of weight matrix from HL1 to HL2
layer_2_index <- layer_1_index + 2
# Index of weight matrix from HL2 to HL3
layer_3_index <- layer_2_index + 2
```

```

# Index of weight matrix from HL3 to OL
layer_4_index <- layer_3_index + 2
# transpose(W^(1)), take abs. vals. of elements
m1_matrix <- model_weights[layer_1_index][[1]]
m1_matrix_abs <- abs(m1_matrix)
# transpose(W^(2)), take abs. vals. of elements
m2_matrix <- model_weights[layer_2_index][[1]]
m2_matrix_abs <- abs(m2_matrix)
# transpose(W^(3)), take abs. vals. of elements
m3_matrix <- model_weights[layer_3_index][[1]]
m3_matrix_abs <- abs(m3_matrix)
# w^y, take abs. vals. of elements
m4_matrix <- model_weights[layer_4_index][[1]]
m4_matrix_abs <- abs(m4_matrix)

# Compute the influence vector
influence_matrix <- m2_matrix_abs %*% m3_matrix_abs
%*% m4_matrix_abs

# Compute the interaction strength score
# between the input neurons i and j
calc_interaction_strength_ij <- function(i, j,
m1_matrix_abs, influence_matrix){
  # Input: indices of input neurons, weight matrix (m1),
  # influence matrix
  # Outut: score of the interaction strength
  # between input neurons i and j
  total_interaction <- 0

# iterate through neurons of first hidden layer,
# compute interaction strength scores
# at each neuron in the 1st HL, sum up scores
for (m in 1:length(influence_matrix)){
  w_i <- m1_matrix_abs[i,m]
  w_j <- m1_matrix_abs[j,m]
  interaction_strength <- min(w_i,w_j)
  interaction_influence <- interaction_strength *
  influence_matrix[m]
  total_interaction <- total_interaction +
  interaction_influence
}
return(total_interaction)
}

```

In *keras* the matrices extracted by the function `get_weights` are the transposed versions of the matrices used in our notation as well as the notation by Tsang et al. [19]. Therefore, in the code above `influence_matrix` corresponds to $(\zeta^{(1)})^\top = |W^{(2)}|^\top \cdot |W^{(3)}|^\top \cdot |w^y|$ in the notation of our paper, $d = 3$.

References

1. Burkov A (2020) Machine learning engineering. True Positive Inc
2. Denuit M, Walhin JF, Pitrebois S, Maréchal X (2007) Actuarial modelling of claim counts: risk classification. Credibility and bonus-malus systems. Wiley

3. Ferrario A, Noll A, Wüthrich MV (2018) Insights from inside neural networks. Available at SSRN 3226852
4. Friedman JH, Popescu BE (2008) Predictive learning via rule ensembles. *Ann Appl Stat* 2:916–954
5. Goldburd M, Khare A, Tevet D, Guller D (2020) Generalized Linear Models for Insurance Rating. *Casualty Actuarial Society*
6. Greenwell B, Boehmke B, McCarthy A (2018) A simple and effective model-based variable importance measure. [arXiv:1805.04755](https://arxiv.org/abs/1805.04755)
7. Henckaerts R, Antonio K, Clijsters M, Verbelen R (2018) A data driven binning strategy for the construction of insurance tariff classes. *Scand Actuar J* 2018(8):681–705. <https://doi.org/10.1080/03461238.2018.1429300>
8. Henckaerts R, Antonio K, Côté MP (2022) When stakes are high: balancing accuracy and transparency with model-agnostic interpretable data-driven surrogates. *Expert Syst Appl* 202:117230. <https://doi.org/10.1016/j.eswa.2022.117230>
9. Lundberg S, Lee SI (2017) A unified approach to interpreting model predictions. <https://doi.org/10.48550/ARXIV.1705.07874>
10. Molnar C (2019) Interpretable machine learning. Lulu.com
11. Nelder JA, Wedderburn RWM (1972) Generalized linear models. *J R Stat Soc* 135(3):370–384. <https://doi.org/10.2307/2344614>
12. Noll A, Salzmann R, Wüthrich MV (2020) Case study: French motor third-party liability claims. Available at SSRN 3164764
13. Oh S (2019) Feature interaction in terms of prediction performance. *Appl Sci* 9(23):5191. <https://doi.org/10.3390/app9235191>
14. Ohlsson E, Johansson B (2010) Non-life insurance pricing with generalized linear models. Springer Berlin Heidelberg, Heidelberg
15. Richman R, Wüthrich MV (2023) LocalGLMnet: interpretable deep learning for tabular data. *Scand Actuar J* 2023(1):1–25. <https://doi.org/10.1080/03461238.2022.2081816>
16. Schelldorfer J, Wüthrich MV (2019) Nesting classical actuarial models into neural networks. Available at SSRN 3320525
17. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res* 15(56):1929–1958
18. Tibshirani R (1996) Regression shrinkage and selection via the lasso. *J Roy Stat Soc Ser B (Methodol)* 58(1):267–288. <https://doi.org/10.1111/j.2517-6161.1996.tb02080.x>
19. Tsang M, Cheng D, Liu Y (2018) Detecting statistical interactions from neural network weights. *ICLR* 2018
20. Wüthrich MV (2019) Bias regularization in neural network models for general insurance pricing. *Eur Actuar J* 10:179–202
21. Wüthrich MV (2020) From generalized linear models to neural networks, and back. Available at SSRN 3491790
22. Wüthrich MV, Merz M (2019) Editorial: Yes, we can! *ASTIN Bull.* <https://doi.org/10.1017/asb.2018.42>
23. Wüthrich MV, Merz M (2022) Statistical foundations of actuarial learning and its applications. Springer. <https://doi.org/10.1007/978-3-031-12409-9>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.